Formally Modeling, Analyzing, and Designing Network Protocols — A Case Study on Retransmission-Based Reliable Multicast Protocols

by

Carolos Livadas

M.Eng. in Electrical Engineering and Computer Science, MIT (1997)
S.M. in Aeronautics and Astronautics, MIT (1996)
S.B. in Computer Science and Engineering, MIT (1993)
S.B. in Aeronautics and Astronautics, MIT (1993)

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

September 2003

Author ____
Department of Electrical Engineering and Computer Science
August 1, 2003

Certified by ____
Professor Nancy A. Lynch
Department of Electrical Engineering and Computer Science
NEC Professor of Software Science and Engineering

Accepted by ____
Professor Arthur C. Smith
Department of Electrical Engineering and Computer Science
Chairman, Department Committee on Graduate Theses

Formally Modeling, Analyzing, and Designing Network Protocols — A Case Study on Retransmission-Based Reliable Multicast Protocols
by
Carolos Livadas

Submitted to the Department of Electrical Engineering and Computer Science on August 1, 2003, in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

In this thesis, we conduct an extensive case study on formally modeling, analyzing, and designing retransmission-based reliable multicast protocols. We first present an abstract model of the communication service that several reliable multicast protocols [12, 13, 32–34] strive to provide. This model precisely specifies i) what it means to be a member of the reliable multicast group, ii) which packets are guaranteed delivery to which members of the group, and iii) how long it takes for a packet to be reliably multicast to the appropriate members of the reliable multicast group.

We then model and analyze the correctness and performance of three retransmission-based reliable multicast protocols, namely the Scalable Reliable Multicast (SRM) protocol [12, 13], the novel Caching-Enhanced Scalable Reliable Multicast (CESRM) protocol [24], and the Light-weight Multicast Services (LMS) router-assisted protocol [32–34]. We show the each such protocol is correct by proving that it is a faithful implementation of our reliable multicast service model. These correctness proofs ensure the equivalence of the protocols in the sense that they guarantee the delivery of the same packets to the same members of the reliable multicast group.

Under some timeliness assumptions and presuming a fixed number of per-recovery packet drops, we show that our model of SRM guarantees the timely delivery of packets. Our timeliness analysis of SRM reveals that the careless selection of SRM's scheduling parameters may introduce superfluous recovery traffic and may undermine the loss recovery process. This is an important observation that has, to date, been overlooked.

CESRM augments SRM with a caching-based expedited recovery scheme that exploits packet loss locality in IP multicast transmissions by attempting to recover from losses in the manner in which recent losses were recovered. We analytically show that the worst-case recovery latency for successful expedited recoveries in CESRM is roughly 1 round-trip time (RTT) where as that of successful first-round recoveries in SRM is 4 RTT (for typical scheduling parameter settings). Moreover, trace-driven simulations, which exhibit the packet loss locality of actual IP multicast transmissions, reveal that CESRM reduces the average recovery latency of SRM by roughly 50% and incurs less overhead in terms of recovery traffic.

Finally, although LMS recovers promptly from packets in static membership and topology environments, we demonstrate several dynamic scenarios in which LMS does not perform well. Thus, CESRM is a preferable reliable multicast protocol to both SRM and LMS; CESRM inherits SRM's robustness to dynamic environments and, thanks to its caching-based expedited recovery scheme, drastically reduces the average recovery latency of SRM in static environments.

Thesis Supervisor: Professor Nancy A. Lynch
Title: NEC Professor of Software Science and Engineering

Acknowledgments

My long journey as a graduate student has finally come to an end. This journey would not have been either possible or enjoyable without the support, help, and friendship of many people. To begin, I would like to thank my thesis committee members Nancy A. Lynch, Idit Keidar, and Dave Clark.

- Nancy has been my research advisor for numerous years. Her insight, her thorough reviews, and her wise suggestions have been an invaluable asset to my research. I also would like to thank her for supporting and letting me pursue research that was not mainstream within the Theory of Distributed Systems group.
- Idit has been very supportive of my research and of the idea that lead to the novel Caching-Enhanced Scalable Reliable Multicast protocol. I am also thankful that she insisted on simulating CESRM. These simulations demonstrated that the protocol works well and gave it credibility. I also thank her for making the trip to Boston for my defense.
- Dave was adventurous enough to agree to be on my thesis committee. While seeing the value of our formal approach to modeling, analyzing, and designing network protocols, he acted as the networking community conscience of my work.

I would also like to thank Nancy, Idit, and John Lygeros for writing recommendation letters for me.

It has been a pleasure to be a graduate student at MIT, the EECS department, the Theory of Computation group, and the Theory of Distributed Systems group. I would like to thank my academic advisor Daniel Jackson for always being available to discuss academic and professional issues and encouraging me throughout my graduate studies. I feel very lucky to have been assigned to him. I would also like to thank Marilyn Pierce for bearing with me throughout the years as an eternal graduate student and for nudging me along.

I would also like to thank several people from the 3rd floor of LCS who have made my graduate studies a pleasurable experience. Joanne Talbot Hanley, the administrative assistant of our group, has always been very helpful and has taken care of all the issues that have come up throughout the years. Be Blackburn has been instrumental in making the 3rd floor a nice place to work in; the numerous party's and unending supply of sweets has been much appreciated. Finally, our administrative assistants William Ang, Greg Shomo, Matt McKinnon, and Michael Vezza, have always promptly addressed all my computer related needs. They are a great sysadmin team.

My officemates over the years have made working in the TDS group a pleasurable and highly educational experience. Roger Khazan, Roberto De Prisco, Rui Fan, Sayan Mitra, and Victor Luchangco have all been good friends and I wish them well in their professional and academic careers and endeavors. I also thank my latest officemate, Martin Demaine, for welcoming me into his office during the last stages of my thesis.

During my many years as a graduate student I was very fortunate for having a large number of friends. I thank them for making my life outside of school enjoyable and for encouraging me during my arduous graduate student years. I thank Chris Hadjicostis, Pani Pyla, Nicolas Hadjiconstantinou, Olga Simek, Andreas Savvides, Tony Ezzat, Amir R. Amir, Paris Smaragdis, Karrie Karahalios, Petros Boufounos, Giorgos Zacharia, and Giorgos Kotsalis. I would also thank my other set of friends Yannis Paschalidis and Gina Mourtzinou, Antonis Armoundas and Fania Mela, Dimitris Bertsimas and Georgia Perakis, and John Lygeros for the many enjoyable get-togethers.

Above all, I would like to thank Maria Kartalou for her love, support, and encouragement throughout the years. She has always been there for me, has helped me through tough times, and has always encouraged and believed in me. Maria, thank you for being there; I can only hope that I have given you as much love, support, and companionship as you have given me throughout the years.

Finally, this thesis would not have been possible without the love and support of my parents Georgios-Argyrios Livadas and Nancy Weston Livadas. They have always encouraged me, looked after me, and helped me in every possible way. I hope they now rejoice in the completion of this thesis.

Contents

List of Figures

List of Tables

Nomenclature

Latin Abbreviations

cf.	*confer*; Latin for "compare".
e.g.	*exempli gratia*; Latin for "for example".
et al.	*et alii*; Latin for "and others".
etc.	*et cetera*; Latin for "and so forth".
i.e.	*id est*; Latin for "that is".
ib. or *ibid.*	*ibidem*; Latin for "in the same work/place".
n.b.	*nota bene*; Latin for "take special note of".
op. cit.	*opere citato*; Latin for "in the work/text cited".
v.g.	*verbi gratia*; Latin for "for example".
v.i.	*vide infra*; Latin for "see below".
v.s.	*vide supra*; Latin for "see above".
viz.	*videlicet*; Latin for "that is to say" or "namely".
vs.	*versus*; Latin for "against".

Mathematical Notation

$\emptyset$	The empty (or null) set.
$\mathbb{N}^+$	The set of positive natural numbers, *i.e.*, the set $\{1, 2, 3, \ldots\}$.
$\mathbb{N}$	The set of natural numbers, *i.e.*, the set $\{0, 1, 2, 3, \ldots\}$.
$\mathbb{R}, \mathbb{R}^{\geq 0}, \mathbb{R}^+$	The set of all, non-negative, and positive real numbers.
$\mathbb{Z}, \mathbb{Z}^{\geq 0}, \mathbb{Z}^+$	The set of all, non-negative, and positive integers.

Chapter 1

Introduction

To date, communication protocols are designed and analyzed using predominantly non-rigorous techniques. Protocols are usually specified by informal descriptions, their correctness is validated through informal reasoning and simulations, if at all, and their performance is evaluated through statistical and simulation-based analyses. This informal approach to designing and analyzing communication protocols has and continues to serve the networking community well. Novel protocol ideas are presented without worrying about infrequent and exceptional behavior and protocol simulations serve to weed out and refine promising ideas.

However, this informal design and analysis approach has some disadvantages. In many cases, the informal protocol descriptions are imprecise, incomplete, and have unclear or lacking assumptions about the environment in which the protocols are presumed to operate. Moreover, due to their complexity, statistical analysis techniques have predominantly been used to analyze the behavior of protocols in simple settings. As the complexity of either the protocols or the settings increases, statistical analysis techniques become increasingly complex and unwieldy to use. Similarly, simulation-based analysis techniques usually observe a protocol's average performance under normal operating conditions. Thus, the complete behavior of a protocol and its performance under all operating conditions is seldom evaluated. As protocol complexity increases, due for instance to the onset of host mobility and wireless connections, such techniques may fail to evaluate a protocol's complete behavior and, thus, expose its weaknesses.

In contrast to these traditional network protocol design and analysis techniques, we advocate the use of a formal approach to modeling, analyzing, and designing network protocols. The first step in this approach is to produce precise and complete specifications of both the protocol and the communication service that the protocol intends to provide. The communication service specifications provide an abstract description of the protocol's external behavior, which may specify both a protocol's correctness and performance guarantees. The specification of the protocol involves a precise description of the protocol's complete functionality. The protocol's correctness is shown by proving that the protocol is a faithful implementation of the abstract communication service. Its performance is shown either by reasoning about the protocol's behavior explicitly, or by proving that the protocol faithfully implements a communication service that imposes the appropriate performance guarantees.

In this thesis, we demonstrate the use of this formal approach to modeling, analyzing, and designing network protocols through an extensive case study in the area of *reliable multicast* — the reliable transmission of packets in the multicast (either one-to-many, or many-to-many) communication setting. We proceed by describing our formal modeling approach, and giving an overview of our case study. We conclude the chapter by presenting how the rest of the thesis is organized.

1.1 Modeling Framework

1.1.1 Formal Model

In our work, we model systems using *I/O automata* [25] and their timed extension *timed I/O automata* [27]; formal specification models that produce simple, precise, and unambiguous descriptions of complex system behavior and component interactions and lend themselves to formal correctness and performance analyses. The use of I/O automata affords several benefits. Formal specifications constitute precise system descriptions that can be used to rigorously reason about a system's behavior. An invaluable side-effect of producing these formal specifications is the exposure of hidden system and modeling assumptions that may otherwise be overlooked. Moreover, formal notions of *composition* and *refinement* enable the modeling and analysis of very complex systems. Systems may be decomposed into distinct parts, which may subsequently be analyzed in isolation. Composition allows the extension of component-wise properties to the system as a whole. Refinement is used to model systems at varying *levels of abstraction*. Reasoning about a system's behavior while keeping track of implementation details is often too cumbersome and overwhelming. However, implementation details and technicalities may be abstracted away by describing the system functionality at a high or abstract level. Reasoning about a system's behavior at an abstract level is simpler and more tractable. High-level system specifications may subsequently be refined to describe the low-level implementation details. This refinement process may lead to several increasingly detailed system specifications, each suitable for showing distinct sets of system properties.

I/O automata, and their timed extension, are accompanied by formal correctness and performance analysis techniques. Two such techniques are *invariant assertions* and *simulation relations*. Invariant assertions are used to systematically prove system properties. Simulation relations are used to show that more refined system specifications actually implement their more abstract counterparts. Once a simulation relation is demonstrated between the abstract and refined system specifications, the properties shown to be true for the abstract system specifications extend to their more refined counterparts without additional proof obligations.

1.1.2 Protocol Correctness And Performance Analyses

Once a protocol and the communication service it provides have been formally specified, the protocol's correctness is shown by proving that it is a faithful implementation of the communication service. In some cases, a protocol may implement the intended communication service only under particular assumptions. In these cases, the protocol's correctness proof involves precisely specifying the assumptions under which the protocol is a faithful implementation of the communication service. This process is often invaluable in understanding the behavior of the protocol and in exposing the implicit assumptions made during the protocol's design.

A protocol's performance is quantified by proving conditional performance guarantees; that is, absolute claims that a protocol achieves particular levels of performance under particular assumptions. The art in this type of performance analysis lies in weakening the assumptions and strengthening the performance guarantees involved in the conditional performance claims. The performance of two protocols that implement the same communication service may be compared by stating comparative performance claims; that is, claims that juxtapose the performance of the two protocols. Such claims are particularly useful when comparing the performance of two protocols, where one is an optimization of the other.

1.2 Our Case Study in Reliable Multicast

Our case study in reliable multicast involves the modeling, analysis, and design of retransmission-based reliable multicast protocols. We begin by informally describing our precise specification for a reliable multicast communication service that provides eventual delivery with, possibly, some timeliness guarantees. We proceed to specify, prove the correctness of, and analyze the timeliness of the Scalable Reliable Multicast (SRM) [12, 13] protocol. We then design, specify, and analyze the Caching-Enhanced Scalable Reliable Multicast (CESRM) protocol. This protocol enhances SRM by a caching-based expedited recovery scheme that attempts to exploit the packet loss locality exhibited by IP multicast transmission losses. In addition to the analytical correctness and performance analyses, we also evaluate CESRM using trace-driven simulations. We conclude our case study by specifying and informally analyzing the behavior of the router-assisted reliable multicast protocol based on the Light-weight Multicast Services (LMS) [34].

1.2.1 The Reliable Multicast Service

Reliability in the multicast setting has assumed many meanings, ranging from in-order eventual delivery to timely delivery where a small percentage of packet losses is tolerable. The many notions of reliability stem from the varying assumptions regarding the communication environment and the goals and requirements of the applications to which particular reliable multicast protocols cater. Most often, the behavior of reliable multicast protocols is described informally. Moreover, a protocol's description is seldom accompanied by a precise definition of its reliability guarantees. In its simplest form, reliability is informally defined as the eventual delivery of all multicast packets to all group members; other notions of reliability may include ordering, no-duplication, and, possibly, timeliness guarantees.

Although intuitive, this simplistic reliability definition does not precisely specify which packets are guaranteed delivery to which members of the group. This is especially the case when the group membership is dynamic. Moreover, protocol descriptions put little emphasis on the behavior, or the analysis of the behavior, of the protocols when the group membership is dynamic, either due to failures or frequent joins and leaves. With the proliferation of mobile hosts and wireless connections, a better understanding of the behavior of such services and protocols in highly dynamic and faulty environments is increasingly important.

We begin our case study in reliable multicast by presenting a formal model of the reliable multicast service, which we henceforth refer to as the *reliable multicast specification* (RMS). Specifying the reliable multicast service is not straightforward. The plethora of reliable multicast protocols cater to diverse applications that impose diverse correctness and performance requirements. Clearly, capturing the functionality of all reliable multicast protocols using a single specification would be quite complex and unwieldy.

Our reliable multicast service specification formalizes the behavior of a number of protocols, such as SRM [12, 13] and LMS [34], that strive to provide eventual delivery with, possibly, some timeliness guarantees. We stipulate that, in the context of dynamic group membership, membership is intrinsically intertwined with reliability; that is, membership and reliability must be addressed together. Thus, our reliable multicast specification dictates precisely what it means to be a member of a reliable multicast group and which packets are guaranteed delivery to which members of the reliable multicast group. We parameterize our specification with a delivery latency bound. This bound specifies the worst-case latency incurred for reliably delivering multicast packets. Thus, our reliable multicast specification may be used to model the behavior of a collection of reliable multicast protocols, some with loose and others with potentially stringent timeliness guarantees.

1.2.2 Scalable Reliable Multicast (SRM) [12, 13]

The Scalable Reliable Multicast (SRM) [12,13] protocol is a simple and robust retransmission-based protocol. SRM uses IP multicast to transmit packets to the members of the reliable multicast group. Packet recovery in SRM is initiated when a receiver detects a loss and schedules the transmission of a *request*; an error control packet requesting the retransmission of the missing packet. If a request for the same packet is received prior to the transmission of this local request, then the local request is rescheduled by performing an exponential backoff. When a group member receives a request for a packet that it has already received, the group member schedules a *reply*; a retransmission of the requested packet. If a reply for the same packet is received prior to the transmission of this local reply, then the local reply is canceled. Using this scheme, all session members participate in the packet recovery process and share the associated overhead.

SRM minimizes the transmission of duplicate request and reply packets through *deterministic* and *probabilistic* suppression. These suppression techniques prescribe how requests and replies should be scheduled so that only few requests and replies are transmitted for each loss. Unfortunately, suppression introduces a tradeoff between the number of duplicate requests and replies and the recovery latency — the scheduling of requests and replies are delayed sufficiently so as to minimize the number of duplicate requests and replies.

Our formal model of SRM specifies precisely the behavior of the SRM protocol. This behavior includes not only the behavior of the reliable multicast group members but also the behavior of the underlying IP multicast communication service. We prove the correctness of SRM by showing that it is a faithful implementation of our reliable multicast service specification without any timeliness guarantees. Moreover, under certain timeliness assumptions and assuming that the number of losses suffered per recovery is bounded, we show that SRM guarantees the timely delivery of packets; that is, that the worst-base time to recover from any loss is bounded. This timeliness guarantee is shown by bounding the number of recovery rounds that may fail prior to recovering a packet.

Our timeliness analysis of SRM reveals that choosing SRM's scheduling parameters arbitrarily may result in either superfluous recovery traffic or the failure of particular recovery rounds due to scheduling issues rather than losses. This observation illustrates that formal protocol analysis may help to better understand and, potentially, redesign a protocol's behavior. Moreover, our analysis gives rise to several constraints on SRM's scheduling parameters. These constraints constitute guidelines for choosing SRM's scheduling parameters so that scheduling issues do not induce superfluous traffic and recovery round failure.

1.2.3 Caching-Enhanced Scalable Reliable Multicast (CESRM) [23, 24]

SRM, as do most retransmission-based reliable multicast protocols, treats losses independently and blindly repeats the recovery process for each loss. This blind repetition of the recovery process wastes resources and, potentially, unduly delays packet recovery. This is especially the case when IP multicast transmission losses exhibit locality — the property that losses suffered by a receiver at proximate times often occur on the same link of the IP multicast tree.

We claim that packet loss locality in IP multicast transmissions can be exploited by simple caching schemes. In such schemes, receivers cache information about the recovery of recently recovered packets and use this information to expedite the recovery of subsequent losses. We present a methodology for estimating the potential effectiveness of caching within multicast loss recovery. We use this methodology on the IP multicast transmission traces of Yajnik *et al.* [41] and observe that IP multicast losses exhibit substantial locality and that caching can be very effective.

Motivated by this expected effectiveness of caching in multicast loss recovery, we design and analyze the Caching-Enhanced Scalable Reliable Multicast (CESRM) protocol. CESRM opportunistically attempts to recover from losses in the manner in which recent losses were recovered. In so doing, CESRM attempts to exploit packet loss locality and to reduce the recovery latency and overhead of SRM.

CESRM augments the functionality of SRM by a caching-based expedited recovery scheme, which operates in parallel to SRM's recovery scheme. In this scheme, reliable multicast group members cache the requestor/replier pairs that carried out the recovery of recent losses. Based on this cached information, receivers attempt to expeditiously recover losses in the manner (*i.e.*, the requestor/replier pair) in which the plurality of a fixed number of recent losses were recovered — this fixed number constitutes the size of the cache. Expedited requests are unicast to the appropriate replier and, upon receiving this request, this replier multicasts the requested packet.

Expedited requests and replies are not delayed for purposes of suppression. Thus, successful expedited recoveries incur minimum recovery latency. CESRM uses SRM as a fall-back recovery scheme. When the expedited recovery scheme fails to recover a loss, either due to losses or because the replier has also shared the loss of the particular packet, then the packet is recovered, in due time, through SRM's recovery scheme.

Our formal model of CESRM extends that of SRM by specifying CESRM's expedited recovery scheme. Moreover, we extend our model of the IP communication service to provide both unicast and multicast transmission capability. As in the case of SRM, we formally analyze the performance of CESRM by showing that it is a faithful implementation of both an eventual and a timely reliable multicast service specification. Furthermore, we analytically show that the worst-case recovery latency for successful expedited recoveries in CESRM is roughly 1 round-trip time (RTT) where as that of successful first-round recoveries in SRM is 4 RTT (for typical scheduling parameter settings). Finally, trace-driven simulations reveal that CESRM reduces the average recovery latency of SRM by roughly 50% and incurs less overhead in terms of recovery traffic.

1.2.4 Light-Weight Multicast Services (LMS) [32–34]

The reliable multicast protocol based on the Light-weight Multicast Services [32–34], which we will henceforth refer to as LMS, enhances the functionality of the underlying IP multicast tree routers so as to enable the intelligent forwarding of recovery packets and, hence, enable local packet loss recovery. The idea behind LMS, as well as other similar router-assisted reliable multicast protocols (*e.g.*, [19]), is to appoint particular members (repliers) of the reliable multicast group to be responsible for replying to requests originating within particular subtrees of the underlying IP multicast tree. In the case of LMS, this is achieved by having each router maintain a replier link onto which it forwards requests that originate within the subtree rooted at the given router. Thus, requests originating within each subtree are forwarded to the appropriate replier by the routers at the root of the given subtree. Subsequently, the replies to such requests are unicast to the aforementioned routers which in turn subcast the replies downstream. The traffic pertaining to the recovery of a particular packet is thus contained within the subtree of the IP multicast tree affected by the given loss.

We precisely specify the behavior of LMS by refining our earlier models of the IP multicast service so as to describe the enhanced router functionality introduced by LMS. In particular, we model the IP multicast routers, their replier state, and the manner in which this state is maintained. Using our precise model of LMS, we carefully reason about its behavior in dynamic and faulty environments. This reasoning exposes several scenarios in which packet loss recovery in LMS may be prolonged and even inhibited due to either changes in the reliable multicast group membership, changes in the replier hierarchy, or replier failures. With the proliferation of host mobility and wireless connections, a protocol's performance in dynamic environments becomes increasingly important. LMS's weaknesses suggests that future protocols should be designed to perform better in these highly dynamic and faulty environments. Perhaps LMS's lack of robustness to highly dynamic environments tilts the scale in favor of CESRM — CESRM inherits SRM's robustness to dynamic environments and, thanks to its caching-based expedited recovery scheme, takes advantage of packet loss locality and affords good recovery latency in static environments.

1.3 Thesis Organization

In Chapter 2, we present some background material pertaining to the work presented in this thesis. We start by presenting the *timed input/output (I/O) automaton* (TIOA) model which we use to model and analyze the various protocols considered in this thesis. We then describe briefly the manner in which IP multicast is implemented and give a brief overview of the area of reliable multicast, in general, and retransmission-based reliable multicast protocols, in particular.

In Chapter 3, we present a formal model of a reliable multicast service. This model specifies i) what it means to be a member of the reliable multicast group, ii) which packets are guaranteed delivery to which members of the group, and iii) how long it takes for a packet to be reliably multicast to the appropriate members of the reliable multicast group.

In Chapter 4, we present a formal model of the Scalable Reliable Multicast (SRM) protocol of Floyd *et al.* [13]. Moreover, we prove that our model of SRM is a correct implementation of the reliable multicast service and that, under certain timeliness and faultiness assumptions, guarantees the timely delivery of reliable multicast packets.

In Chapter 5, we make the case for exploiting packet loss locality within the loss recovery schemes of reliable multicast protocols, such as SRM [13]. Packet loss locality in IP multicast transmissions can be exploited by simple caching schemes, in which receivers cache information about the recovery of recently recovered packets and use this information to expedite the recovery of subsequent losses. We present a methodology for estimating the potential effectiveness of caching within multicast loss recovery. By applying this methodology to the IP multicast transmission traces of Yajnik *et al.* [41] and observing that IP multicast losses exhibit substantial locality, we establish that caching can be very effective.

In Chapter 6, we present, model, and analyze the correctness and performance of the novel Caching-Enhanced Scalable Reliable Multicast (CESRM) protocol. The correctness analysis states that CESRM is a correct implementation of the reliable multicast service. The timeliness analysis states that, under certain timeliness and faultiness assumptions, CESRM guarantees the delivery of the appropriate packets to the appropriate members of the reliable multicast group within a finite amount of time. We also use trace-driven simulations to evaluate CESRM's performance and compare it to that of SRM.

In Chapter 7, we model and informally analyze the performance of LMS. Our informal performance analysis of LMS involves: i) stating the worst-case recovery latency of LMS when recoveries proceed smoothly, ii) stating the worst-case recovery latency of LMS in scenarios that demonstrate LMS's lack of robustness to highly dynamic and faulty environments, and iii) comparing its performance to that of both SRM and CESRM.

In Chapter 8, we give a brief summary of the thesis, state its contributions, and present an interesting direction in which the CESRM protocol may be extended to limit the exposure of expedited recoveries by exploiting some of the light-weight router functionality enhancements introduced by LMS.

Chapter 2

Background

In this chapter, we present some background material pertaining to the work presented in this thesis. We begin by giving a brief overview of the *timed input/output (I/O) automaton* (TIOA) model (introduced as the *general timed automaton* model in Ref. 25), the framework that we use to specify and analyze protocols. We then overview the functionality of IP multicast. This overview includes a description of the network of IP multicast capable routers and how they are interconnected, a description of the various protocols that collectively implement the IP multicast service, and a summary of the results of several research studies investigating the correlation characteristics of losses in IP multicast transmissions. We continue by introducing the various approaches to providing reliability in the multicast communication setting and by focusing and describing in detail the issues afflicting retransmission-based schemes. We conclude by describing the functionality of two representative examples of application-layer and router-assisted reliable multicast protocols, namely the Scalable Reliable Multicast (SRM) protocol [12,13] and the reliable multicast protocol based on the Light-weight Multicast Services (LMS) [32–34], respectively.

2.1 The Timed I/O Automaton Model

In this thesis, we use the *timed input/output (I/O) automaton* (TIOA) modeling framework (introduced as the *general timed automaton* model in Ref. 25); a framework for modeling timed systems. A timed I/O automaton A is a simple state-machine in which transitions are labeled by *actions*. The actions of A, denoted $acts(A)$, are partitioned into *input* ($in(A)$), *output* ($out(A)$), *internal* ($int(A)$), and *time-passage* sets. Time-passage actions model the passage of time. The input and output actions of A are collectively referred to as *external* and denoted as $ext(A)$. Input, output, and time-passage actions are collectively referred to as *visible* and denoted as $vis(A)$.

A timed I/O automaton A is defined by its *signature* (input, output, internal, and time-passage actions), states ($states(A)$), start states ($start(A)$), and state-transition relation ($trans(A)$). The state transition relation of A is a subset of the cross product of states, actions, and states, *i.e.*, $trans(A) \subseteq states(A) \times acts(A) \times states(A)$, and dictates A's allowable transitions.

A *timed execution fragment* of A is a finite or infinite alternating sequence, $\alpha = s_0\pi_1s_1\pi_2s_2\ldots$, of states and actions consistent with A's state-transition relation. A *timed execution* of A is a timed execution fragment of A that begins in one of A's start states. A timed execution fragment of A is *admissible* if an infinite amount of time elapses within the particular fragment. An admissible timed execution α of A is *fair* when no action is enabled onwards of a particular state within α without appearing within the suffix of α starting at that state. The time of occurrence of an action π_k, for $k \in \mathbb{N}^+$, within a timed execution fragment α of A is the time elapsing within α prior to the occurrence of π_k. The *timed trace* of a timed execution fragment α of A is the sequence of visible actions in α, each paired with its time of occurrence. We let $aexecs(A)$ denote the set of all admissible timed executions of A, $attraces(A)$ denote the timed traces of all executions in $aexecs(A)$, $fair\text{-}aexecs(A)$ denote the set of all fair admissible timed executions of A, and $fair\text{-}attraces(A)$ denote the timed traces of all executions in $fair\text{-}aexecs(A)$.

The composition of compatible timed I/O automata yields a timed I/O automaton. The *hiding* operation reclassifies output actions of a timed I/O automaton as internal. Letting A, B be timed I/O automata with the same external interface, B *implements* A, denoted $B \leq A$, when its external behavior is allowed by A; that is, when $attraces(B) \subseteq attraces(A)$. The implementation relation between two timed I/O automata is often shown by defining a *timed simulation relation*; that is, relating states of B to states of A and showing that for any step of B there is a timed execution fragment of A that preserves the state relation and whose trace matches that of the step in B.

We use a *precondition-effect* style notation to define the state-transition relations of timed I/O automata. The syntax and the semantics of this notation are described in detail in Ref. 14. We complement this notation with the following notational shorthand. For any variable s and any set variables S_1 and S_2, we use the notation $S_1 \cup= S_2$ as shorthand for $S_1 := S_1 \cup S_2$, $S_1 \setminus= S_2$ as shorthand for $S_1 := S_1 \cup S_2$, $S_1 :\subseteq S_2$ as shorthand for the assignment of an arbitrary subset of S_2 to S_1, and $s :\in S_1$ as shorthand for the assignment of an arbitrary element of S_1 to s. Moreover, for any state u of a timed I/O automaton A and any action `foo` of A, we use the notation $u.Pre(\texttt{foo})$ to denote the valuation of the precondition of the action `foo` in state u.

2.2 IP Multicast

IP multicast is the IP primitive for providing multi-party best-effort communication. A certain subset of the IP address space is reserved for multicast communication. Individual hosts may choose to subscribe and unsubscribe to messages addressed to such multicast addresses, thus forming multicast groups. Packets addressed to such multicast addresses are disseminated to all subscribers of the particular multicast address in a best-effort manner. In this section we describe the various aspects of the IP multicast service. We begin by describing the *multicast backbone* (MBone); the virtual network that is overlaid on portions of the Internet and used to disseminate multicast packets. We then proceed to briefly describe the various protocols that are involved in providing the IP multicast service.

2.2.1 Multicast Backbone Topology

As described by Casner [6] and summarized by Yajnik [41], the *multicast backbone* (MBone) is a virtual network that is layered over portions of the Internet so as to support the transmission of IP multicast traffic. Not all Internet routers are capable of handling IP multicast traffic. Thus, the MBone is comprised of a set of IP multicast routers, referred to as *islands*, which are interconnected by virtual point-to-point connections, referred to as *tunnels*. Islands are interconnected through a combination of mesh and star configurations. Core MBone routers, which are used to provide IP multicast connectivity to distinct geographical regions, are interconnected by a mesh of tunnels. Redundant interconnections within this mesh protect the MBone against network failures. Within each region, a star topology is used to connect the region's backbone router to all local routers that wish to participate in IP multicast sessions. Additional tunnels may also branch out from these local routers to include IP multicast capable routers on local area networks (LANs). MBone routers disseminate IP multicast traffic by encapsulating packets into ordinary IP unicast packets and transmitting them through the tunnels to their neighboring MBone routers. The use of encapsulation allows the transmission of multicast traffic through intermediary routers not capable of handling multicast traffic.

Some protocols make use of the *time-to-live* (TTL) field of IP packets to estimate the hop distance between senders and receivers. Although the use of this field in the multicast setting is also tempting, it is not as straightforward. Since IP multicast packets are encapsulated prior to being transmitted through the MBone tunnels, their TTL field may not get decremented when traversing the intermediary tunnel routers. Thus, the TTL field may underestimate the hop-count to the source and cease to be an accurate hop-count estimate. To make things worse, the semantics pertaining to the TTL field of IP multicast packets may be neither well-defined, nor consistent among the various underlying multicast routing protocols [15].

2.2.2 Protocols

In order for a host to receive messages of a particular IP multicast session, it must join the IP multicast session using the Internet Group Management Protocol (IGMP) [4,8,11]. As summarized by Semeria and Maufer [37], local MBone routers maintain a list of all IP multicast sessions that each of their network interfaces is interested in receiving. This list is updated according to the join and report messages sent by hosts wishing to subscribe and remain subscribed, respectively, to particular IP multicast sessions. Hosts inform their local MBone routers of their wish to receive traffic addressed to an IP multicast session by sending a join message. This message alerts the local MBone router of the existence of a host on the particular network interface that wants to receive the packets pertaining to the particular IP multicast session. Keeping a list of the IP multicast sessions that each of its network interfaces is interested in, the local MBone router is able to correctly forward IP multicast traffic on its network interfaces. This list is also updated by report messages that are sent by hosts in response to query messages sent by their local MBone router. In order to refresh the information in its list, the local MBone router sends out query messages to the hosts reachable by each of its network interfaces. Upon receiving such a query, a host sends a report message to the local MBone router for each IP multicast session it is still interested in. Unless the local MBone router receives a report for an IP multicast session from a network interface, the forwarding of packets of the particular IP multicast session on that network interface is ceased. Thus, in order to stop receiving packets addressed to a particular IP multicast session, a host simply refrains from acknowledging the query messages for the particular IP multicast session. IGMPv2 [11] augments the functionality of IGMP by introducing group-specific queries for local MBone routers and leave messages for hosts. Thus, hosts may expedite leaving a particular IP multicast session by sending a leave message which, in turn, induces the local MBone router to send a group-specific query message. If no hosts respond to this query message, the local MBone router ceases to forward packets of the particular IP multicast session down the given network interface. IGMPv3 [4] augments the functionality of IGMPv2 by introducing group-source report messages. Such messages enable hosts to instruct their local MBone routers to begin or cease forwarding IP multicast packets sent by particular members of particular IP multicast sessions. Bandwidth may thus be conserved by allowing hosts to refine the set of packets they are interested in receiving and by minimizing the extent of IP multicast trees pertaining to particular session and source pairs.

The dissemination of IP multicast traffic among the local MBone routers is carried out by an IP multicast routing protocol [9, 37]. Most such protocols save memory, computation, and bandwidth resources by arranging a set of the MBone routers into a spanning tree that is subsequently used to forward packets pertaining to a particular IP multicast session. Such trees can be either shared by all the sources of the particular IP multicast session or specific to each source (referred to as *source-based* IP multicast trees). The advantage of shared trees is that MBone routers store per IP multicast session state only. The disadvantage is that traffic is concentrated on particular interconnections of the MBone and that the point-to-point distance between sources and receivers may not be optimal. Conversely, source-based trees better utilize the network by distributing the load among more links and guarantee optimal routing between the sources and the receivers. However, the use of source-based trees requires MBone routers to store per source state for each IP multicast session, which may be prohibitive for IP multicast sessions involving numerous sources.

Several protocols make use of multicast messages to ascertain timing and topology information regarding the underlying IP multicast trees. While doing so may often result in invaluable information, collecting such information must be done cautiously. An important issue that is often overlooked is that in the case of source-based trees, packets multicast by distinct members to the same multicast group are forwarded on different IP multicast trees; that is, the paths traversed by packets exchanged by two sources are not necessarily the same. Thus, the collective use of timing and topology information gathered by packets multicast by different members may not be straightforward.

2.2.3 Loss Characteristics

There have been several research studies regarding the location, the cause, and the statistical characteristics of IP multicast transmission losses. The motivation behind such studies lies in the promise that better insight into the characteristics of losses can guide the design of more effective multicast communication applications and services. This rationale applies in particular to applications and services relating to reliable multicast communication.

In a study of audio packet losses, Bolot *et al.* [1] report that in real-time audio transmissions in both unicast and multicast settings the loss burst lengths are small, especially when the network load is low. In particular, the authors observe that the probability distribution of loss burst length decreases geometrically with the length of the loss burst. Bolot *et al.* conclude that, since losses in real-time audio transmissions are predominantly solitary and prompt recovery is essential, *forward error correction* (FEC) and *error-concealment* techniques are more suitable for error control in real-time audio and video transmissions than their retransmission-based counterparts.

Yajnik *et al.* [41] analyze the *spatial* and *temporal correlation* of losses in constant bit-rate IP multicast transmissions among 17 research community hosts. In their work, *spatial correlation* is defined as the correlation of packet losses across receivers, *i.e.*, the degree to which the losses are shared among receivers. *Temporal correlation* is defined as the correlation of packet losses at each receiver, *i.e.*, the burstiness of packet losses. As to the location of losses, Yajnik *et al.* observe that losses are rare within the MBone core, except for some occasional long loss periods on individual links. Furthermore, losses are negligible at the receiving interface, *i.e.*, packets are seldom lost during the delivery from the MBone router on a LAN to the receiving hosts of that LAN. Yajnik *et al.* also report that the pairwise spatial correlation among receivers is low, except for the losses that occur close to the source and are thus shared by all receivers. Moreover, the occasional long loss periods on the MBone seem to contribute heavily to the spatial correlation observed. In terms of temporal correlation, losses are predominantly solitary and the lengths of loss bursts are small, except again for occasional long loss bursts.

Handley [15] extends the work of Yajnik *et al.* by studying the multicast communication on a broader scale, in terms of the number of receivers, the type of receivers, and the rate of transmission. In particular, his experiments involve IP multicast transmissions from a single variable-rate video source to a few hundred widespread receivers. Compared to the 17 research community receivers used by Yajnik *et al.*, hundreds of widespread receivers that are not necessarily part of the research community are intended to faithfully represent the network characteristics of a more typical IP multicast transmission. Moreover, a variable-rate video session is used to draw some conclusions relating to the characteristics of potential congestion control mechanisms. Handley concludes that 80% of all receivers report loss rates of less than 20%, periodic bursts in losses occur at roughly every 30 seconds, and packet losses are not independent. Handley observes that there were a small number of particularly lossy links and a large number of slightly lossy links. Handley also observes that the probability that a packet is received by all receivers is very low as the session size increases; in particular, on the order of 3–6% for sessions of a few hundred receivers. Thus, Handley concludes that any viable error control scheme for large multicast sessions must use either a FEC scheme, a retransmission-based loss recovery scheme that achieves localized loss recovery, or, more appropriately, a combination of the two. Solely using an FEC scheme would require excessive amounts of redundancy to cater to loss bursts. Moreover, since losses are predominantly due to congestion, the required redundancy would worsen the congestion and induce additional losses. Similarly, solely using a retransmission-based scheme would require the retransmission of almost all packets.

In a subsequent study, Yajnik *et al.* [42] further study the temporal correlation of packet loss in both unicast and multicast constant bit-rate transmissions. Their work uses 128 hours of transmission traces represented either as binary time series indicating whether particular packets were received or as alternating sequences of reception and loss burst lengths. These traces reveal significant non-stationary effects as to the mean loss-rates. In particular, gradual, abrupt, and dramatic changes as well as spikes in the mean loss-rate are observed. Nonetheless, trace sections amounting to 76 hours were identified as stationary and used to evaluate the temporal correlation of losses. Yajnik *et al.* confirmed their earlier results that losses are predominantly solitary, with autocorrelation time-scales of less that 1 second, and that loss burst lengths are geometrically distributed. The authors were also able to faithfully model the observed loss patterns using Markov chain models of varying orders.

2.3 Reliable Multicast

Reliable multicast refers to the service of providing reliable communication in the one-to-many and many-to-many communication settings. Due to network congestion, queue overflow, and processing overload at routers and hosts, packet losses are inevitable. The design of an efficient and scalable error control scheme for multicast communication has been the focus of much research. Reliable multicast surveys [10, 17, 18, 31, 40] group the various approaches to providing reliable multicast communication into the following categories: i) retransmission-based, *e.g.*, [13], ii) forward error correction (FEC)-based, *e.g.*, [5,30,36], and iii) error concealment-based, *e.g.*, [36]. Retransmission-based schemes recover from losses by detecting and promptly retransmitting missing packets. FEC-based schemes proactively encode data packets with enough redundancy to tolerate a certain number of packet losses. Using such encodings, the original data packets may be reconstructed by decoding the subset of packets received by each receiver. The advantage of FEC-based schemes is that the redundant encoding allows the recovery from different losses at different receivers. Lastly, error concealment-based schemes, which are mostly used for audio and video transmissions, attempt to faithfully recreate missing packets by duplicating, interpolating, and, otherwise, processing the packets received.

In this thesis, we focus our attention on retransmission-based reliable multicast protocols. Such schemes may be further split into application-layer and router-assisted protocols. Application-layer protocols often use the underlying IP multicast service as a black box, or primitive, and build upon it. Router-assisted protocols break the IP multicast abstraction and enlist the help of the routers to intelligently forward error control and retransmission packets. In the rest of this section, we describe some of the correctness and performance issues pertaining to retransmission-based reliable multicast protocols, we present the performance metrics that have been used to date to evaluate such protocols, and we conclude with a description of two representative examples of application-layer and router-assisted protocols, namely the Scalable Reliable Multicast (SRM) protocol [12,13] and the reliable multicast protocol based on the Light-weight Multicast Services (LMS) [32–34], respectively.

In the rest of this section, we use the term *data packets* to refer to originally transmitted packets that comprise the data of the application using the reliable multicast service. We use the term *retransmissions* to denote subsequent transmissions of data packets. We use the term *control packets* to denote packets used by the particular reliable multicast protocol to coordinate the recovery of losses. We use the term *recovery round* to refer to the process of recovering from a particular packet loss, *i.e.*, the sequence of control packets exchanged by the members of the reliable multicast group in an attempt to recover from a particular loss. Finally, we use the term *recovery latency* to refer to the time needed by a particular member of the reliable multicast group to recover from a particular loss, *i.e.*, the time elapsing from the time at which the member detects the loss to the time at which it receives the packet.

2.3.1 Retransmission-Based Reliable Multicast

Retransmission-based reliable multicast protocols recover from losses by detecting and promptly retransmitting missing packets. In a one-to-one communication setting, reliable transport is achieved by having the receiver acknowledge the reception of packets and the sender retransmit any packets not acknowledged by the receiver. This approach does not extend well to the one-to-many and many-to-many communication settings. In such settings, an acknowledgment-based error control scheme induces acknowledgment (ACK) implosion; that is, senders get swamped by the duplicate ACKs sent by the numerous receivers. A variety of approaches have been proposed in the literature to solve this problem. One such approach is to limit the number of receivers that send ACKs. This is achieved by *a priori* designating such receivers either arbitrarily, randomly, or by arranging the receivers in a hierarchy [16, 28]. Another approach is to have receivers send negative acknowledgements (NACKs) upon detecting packet losses, instead of acknowledging all packets received. While the use of NACKs tends to alleviate ACK-implosion, there is still the possibility of NACK-implosion, especially in large sessions with either high loss-rate, or losses with high spatial correlation. In combating NACK-implosion, researchers have resorted to multicasting NACKs and having receivers abstain from sending NACKs for packets for which a NACK has already been overheard — otherwise referred to as *duplicate suppression*.

As described by Levine *et al.* [17,18], the use of NACKs leads to the memory deallocation problem. In an ACK-based error control scheme, the receivers acknowledge the reception of all packets. The sender may thus determine whether a packet has been received by all session members and release it from memory. In a NACK-based error control scheme, the sender is not capable of determining whether a packet has been received by all session members. Thus, memory may not be released. Levine *et al.* [17, 18] show that NACK-based schemes work correctly only with infinite memory and may lead to a deadlock when constrained by finite memory. While some applications, such as the distributed white-board application, inherently need to store all multicast packets, the data multicast by other applications is ephemeral. Thus, storing all packets transmitted would constitute a waste of memory and, potentially, render the error control scheme impractical.

The approach of multicasting error control and retransmission packets to the whole multicast group leads to the problem of *exposure*. In large multicast sessions, it is common for losses to be concentrated in particular regions of the underlying multicast tree. Thus, the use of global error control and retransmission packets wastes memory, processing, and bandwidth resources in regions of the IP multicast tree that are not affected by the losses. *Local error recovery*, or otherwise *recovery isolation*, is the obvious solution to this problem. It's simplest form involves using the TTL field to limit the scope of error control and retransmission packets. More complicated forms include arranging the members of a group in a hierarchy that aggregates error control and retransmission messages, or similarly the use of representatives, *e.g.*, [16, 28]. Router-assisted protocols are particularly well suited for local error recovery, as routers may be used to limit and focus the scope and direction, respectively, of error control and retransmission packets.

The performance of multicast communication depends heavily on the network topology and the packet loss characteristics. To make things worse, multicast session membership and the underlying topology are dynamic — members may join and leave the particular multicast session and network failures may cause the underlying multicast tree to change. To cater to their unknown and dynamic environment, reliable multicast protocols either use active services to gain up-to-date information as to the network topology and the packet loss characteristics, or enlist the network routers to help with sending ACKs/NACKs and retransmitting packets. Active services may use multicast messages in order to deduce the inter-host round-trip times (RTTs) [12, 13] and additional IP multicast tools, such as *mtrace*, to gain knowledge of the underlying IP multicast tree topology and estimate the loss characteristics of its links [15]. Although these active services obtain invaluable information as to the cause and location of packet losses, they increase a protocol's complexity and introduce additional overhead. On the other hand, router-assisted approaches take advantage of the network routers to achieve efficient and localized loss recovery. Although the router-assisted approaches result in reliable multicast protocols that are more scalable and have better performance, the viability of their deployment is questionable. Despite the fact that the lack of a viable deployment strategy may impede a protocol's adoption, the issue of a protocol's deployment has, until recently, rarely been addressed in the literature [33].

2.3.2 Protocol Correctness and Performance Analysis

Following its design, a reliable multicast protocol may be analyzed in terms of both correctness and performance [17, 18, 29, 40]. A protocol's correctness may be shown by proving that the protocol faithfully implements the communication service it is intended to provide. Unfortunately, a protocol's correctness is rarely analyzed; in fact, a precise definition of the reliable multicast service the protocol is intended to provide is seldom specified and a protocol's correctness is verified only informally.

A protocol's performance may be evaluated with respect to several quantitative and qualitative metrics. The quantitative metrics include the protocol's recovery latency and overhead. Loss recovery latency is defined as the time that elapses from the moment a loss is detected to the moment a retransmission of the given packet is received. Recovery overhead refers to the memory, processing, and bandwidth resources used by the reliable multicast protocol to recover from a loss. Not surprisingly, the goal of a reliable multicast protocol is to minimize recovery latency while limiting the recovery overhead.

Since loss recovery latency is different for each receiver and each loss, simulations are used to measure a protocol's average recovery latency. A protocol's overhead is analyzed both statistically and through simulations. Statistical analyses neglect the observed temporal and spatial characteristics of packet loss and assume that packet losses are mutually independent, packet losses are independent among receivers, and ACKs and NACKs are never lost. Then, a protocol's overhead is obtained by calculating the expected number of messages exchanged while recovering from a loss and the associated overhead incurred for each message. Simulations are used to observe a protocol's average recovery overhead.

Apart from these quantitative performance metrics, reliable multicast protocols are analyzed qualitatively based on scalability to large sessions, adaptability to topology and membership changes, fault-tolerance, deployment, and infrastructure requirements. With the advent of applications involving large numbers of sources and receivers, the scalability of reliable multicast protocols is a highly desirable, if not required, property. Moreover, a reliable multicast protocol must be dynamic, fault-tolerant, and adapt to changes in the reliable multicast group membership and changes to the underlying network due to failures and congestion. Deployment is an issue that is often ignored. In order for a protocol to be adopted, either an immediate or an incremental deployment strategy is required. Finally, the underlying services on which a reliable multicast protocol relies must be examined and analyzed with respect to all aforementioned metrics. In particular, when analyzing the overhead of a protocol, the contribution of all supporting services must be included. For example, application-layer protocols rely on IP multicast and possibly additional external services. The quantitative and qualitative performance of such services must also be included is a thorough performance analysis.

2.3.3 Scalable Reliable Multicast (SRM) [12, 13]

SRM is an application-layer reliable multicast protocol that uses the IP multicast service as its communication primitive. SRM recovers from losses through (multicast) retransmissions. These retransmissions are instigated by the (multicast) transmission of retransmission requests (NACKs). Duplicate requests and retransmissions are limited through delay-based suppression schemes. Since SRM was initially designed for a distributed white-board application, in which receivers archive all packets, the infinite memory requirements put forth by Levine *et al.* [17, 18] does not apply. We proceed to give a more detailed description of SRM and several proposed extensions to SRM that attempt to alleviate some of its shortcomings.

The SRM protocol consists of two distinct components: i) *session message* exchange, and ii) *error repair*. Session messages are used to exchange state and timing information; state aids in the detection of losses and timing aids in the suppression of duplicate error control and retransmission packets. Losses in the middle of a sequence of packets are detected upon receiving data packets with subsequent sequence numbers. However, when the last packet in a sequence is lost and the size of the sequence is unknown *a priori*, as is the case in the white-board application, members may be unaware of losses. Session messages, which are periodically multicast by each session member, contain the sequence number of the last packet received from each source by the respective member. Members use this up-to-date transmission state information to detect packet losses. In terms of timing, session messages are used to estimate the *round-trip time* (RTT) among receivers. In view of avoiding congestion, the frequency of session messages is adjusted to comprise a fixed percentage of the total bandwidth used by the reliable multicast session. Thus, assuming a fixed session bandwidth allocation, the frequency of session messages is reduced as the session size grows.

Error repair in SRM is initiated when receivers detect losses and schedule the transmission of a *repair request*; an error control packet requesting the retransmission of the missing packet. A repair request is scheduled by setting a repair request timer. Upon its expiration, the repair request is multicast. If a repair request is overheard prior to the expiration of the request timer, then the request is rescheduled by performing an exponential backoff. When a host receives a repair request for a packet that is has either sent or received, it schedules a *repair reply*; a retransmission of the requested packet. A repair reply is scheduled by setting a repair reply timer. Upon its expiration the repair reply is multicast. If a repair reply is overheard prior to the expiration of the reply timer, then the reply is canceled. Using this scheme, all session members participate in the error repair by sending repair requests and replies and the retransmission load is shared among all members of the reliable multicast group.

SRM reduces duplicate error control and retransmission traffic through *deterministic* and *probabilistic* suppression. These suppression techniques prescribe how repair requests and replies should be scheduled so that only few requests and replies are transmitted for each loss. Deterministic suppression prescribes that request and reply timers be set proportionately to the distance from the source and the requestor, respectively. In the case of requests, receivers further away from the source will schedule their requests later in time and, presumably, their requests will suppress those of their descendants in the IP multicast tree. The rationale is analogous for the case of scheduling replies. Probabilistic suppression prescribes that members that are equidistant from the source and the requestor spread out their requests and replies, respectively. This is done by scheduling requests and replies within intervals whose widths are proportional to the distance from the source and the requestor, respectively. Using this approach, members that are equidistant from the source and the requestor, respectively, are given the opportunity to suppress each other.

Although SRM is highly robust to changes in the reliable multicast group and the IP multicast tree topology, it suffers from scaling problems. First, hosts participating in a reliable multicast session must maintain a table of RTT estimates of all other members of the session — a storage requirement that grows linearly with the size of the session. Second, since each member of the reliable multicast group periodically sends session messages, the number of the session messages grows linearly with the session size. Thirdly, presuming a fixed bandwidth constraint on reliable a multicast session, as the session grows in the number of members, the frequency of the exchange of session messages drops. This results in poor performance in terms of both detecting packet losses and updating RTT estimates. Finally, since requests and replies are transmitted to the whole IP multicast group, even very localized losses consume bandwidth, memory, and computation resources in regions of the network that are not affected by the losses.

Sharma *et al.* [38, 39] describe a scheme in which session members are organized into a dynamically self-configuring hierarchy, thus disseminating timing and session state information more efficiently. In particular, scoped session messages are used to exchange timing and state information within local neighborhoods and neighborhood representatives are used to exchange such info among neighborhoods. Neighborhoods and representatives dynamically reconfigure so as to keep the hierarchy well populated; that is, members self-appoint (and self-denounce) themselves representatives so as to ensure that representatives are spread apart, close to the members they represent, and represent numerous members. The benefits of this approach are several. First, members store timing information pertaining only to neighborhood representatives and local neighborhood members, thus conserving memory. Second, local session messages contain less timing and state information and are transmitted only within their respective neighborhood. Similarly, global session messages contain timing information pertaining only to each representative and aggregate state information for their respective neighborhood. Thus, bandwidth is conserved both within the neighborhoods and among them, session messages may be transmitted more frequently, and losses may be detected sooner. Finally, the self-configuring hierarchy introduces no performance degradation in terms of number of requests and replies per loss and recovery latency.

Liu *et al.* [21, 22] address the absence of local error recovery in SRM. The authors propose two distinct approaches to limiting the error recovery overhead incurred by wasteful exposure. The *hop-scoped* approach limits the scope of repair requests and replies using the TTL field of IP multicast packets. Inter-receiver hop-count information is piggybacked on session messages. Thus, the scope of repair requests and replies is adjusted to reach the closest IP multicast group member capable of servicing the request and all the receivers that shared the original loss, respectively. The overhead of this approach is minimal since the required hop-count information can be piggybacked onto the session messages. The drawback of this approach is that while the scope of recovery messages may be adjusted, the direction of message dissemination may not. The *group-scoped* approach limits the scope and the direction of recovery messages using distinct local recovery groups. Local IP multicast groups are set-up based on the degree to which receivers share losses. Sequences of a fixed number of packet losses, referred to by Liu *et al.* as *error fingerprints*, are used to determine the degree of loss sharing among members. At the extreme, one such group is created for each lossy link and thus the recovery overhead is constrained to the set of hosts that share all the losses due to the particular lossy link. The overhead of this approach heavily depends on the number of local groups needed to achieve good performance and the efficiency with which such groups are created, maintained, and dissolved. Both hop-scoped and group-scoped approaches reduce the error control overhead. The hop-scoped approach outperforms the group-scoped approach in reducing the request overhead, except in the case of star topologies. The group-scoped approach outperforms the hop-scoped approach in reducing the reply overhead. This constitutes a performance advantage since replies carry payload and are thus costlier to transmit. Issues that are left for future research include the measurement of the overhead incurred by maintaining the local groups in the group-scoped approach, the convergence time of both approaches when the environment is dynamic, and the combination of both approaches in a group-scoped approach where the scope of local group messages are limited further by hop-count. The combination of both approaches is promising since group-scoped local recovery performs well but may not be scalable to large sessions due to the overhead of the numerous local groups required.

2.3.4 Light-Weight Multicast Services (LMS) [32–34]

LMS is one of the router-assisted reliable multicast protocols that have recently been proposed. In its simplest form, LMS assumes that the underlying IP multicast routing protocol builds source-based trees. Thus, each router has a clear notion of an upstream interface — the interface that leads to the source of the given source-based multicast tree.

In LMS, each router selects one of its descendant members to conduct transport layer duties on behalf of the subtree originating at the respective router. This member is denoted the *replier* of the respective subtree and router. The replier selection process is carried out as follows. Members who are willing to perform transport duties advertise themselves to the MBone routers. Among all such willing members, the routers select the best candidate based on its distance and load; the closer the member and the lighter its load the better. Following this selection process, the router stores the interface leading to its replier.

Upon detecting a loss, receivers multicast a NACK with a hop-by-hop designation such that all intermediate routers process it. Routers process NACKs according to the interface on which they arrive. If a NACK arrives on the upstream interface, then the router knows that the NACK is destined for its replier and forwards the NACK along the replier interface. If a NACK arrives on the replier interface, then the router forwards the NACK upstream toward the source — the replier must have not received the particular packet and is sending a NACK upstream in an effort to reach either a replier responsible for an encompassing subtree or the source of the packet. Finally, if a NACK arrives at any other interface, the router forwards the NACK along the replier interface thus calling upon the replier to perform its transport layer duties. In this case, the router annotates the forwarded NACK with fields containing the router's IP address and the interface on which the router received the NACK. Papadopoulos *et al.* call this router the *turning point* because it is at this location within the multicast tree where the NACK stops moving upstream toward the source and starts moving downstream toward the replier.

Upon receiving a NACK, a replier does one of two things. If the replier has not received the requested packet, it ignores the NACK since it will send (or has already sent) a NACK for the given packet. If the replier has received the requested packet, it encapsulates it and unicasts it to the turning point — the encapsulated packet is also annotated with a field containing the interface on which the original NACK had arrived at the turning point. Upon receiving this unicast packet, the turning point router unwraps the unicast packet and subcasts the missing packet along the interface provided within the interface field of the encapsulated packet.

As expected, LMS improves the performance and reduces the overhead in comparison to application-layer reliable multicast protocols. In particular, LMS limits the exposure of error recovery within a subtree which is capable of recovering from the loss and thus achieves low exposure, overhead, and recovery latency. LMS, however, inherits the Achilles heal of router-assisted protocols: the issue of deployment. In a preliminary study of the effect of various deployment schemes on LMS's performance, Papadopoulos and Laliotis [33] observe that a partial deployment of LMS has a significant impact on its performance. Although recovery latency is lightly affected by a partial deployment, exposure and peak NACK load are heavily affected. This impact varies greatly depending on the deployment strategy used. LMS performs better if deployed in contiguous regions rather than in dispersed patches across the MBone. The best performing deployment strategy was to deploy LMS at the core routers. LMS performed well when the protocol was deployed on paths from the source to several of the receivers. Finally, deploying LMS at the border routers performed better than a random deployment strategy. This result is quite promising since border deployment may constitute perhaps the only viable first stage of an incremental deployment strategy. The study of the impact of partial deployment schemes is particularly important because it may guide future deployment efforts in harnessing LMS's performance potential early on in its incremental deployment schedule.

6

Chapter 3

The Reliable Multicast Service

With the increasing use of the Internet, multi-party communication and collaboration applications are becoming mainstream. One such service or application is reliable multicast; that is, the reliable transmission of packets in the one-to-many and many-to-many communication settings. In the recent past, there have been a slew of protocols and schemes that strive to efficiently multicast packets reliably [13, 16, 19, 20, 34, 35]. However, reliability in the multicast setting has assumed many meanings, ranging from in-order eventual delivery to timely delivery where a small percentage of packet losses is tolerable. The many notions of reliability stem from the varying assumptions regarding the communication environment and the goals and requirements of the applications to which particular reliable multicast protocols cater.

In our work, we focus on the eventual delivery notion of reliability and ignore additional transmission guarantees such as ordering and no-duplication; that is, we focus on the notion of reliability that is informally defined in the literature as the eventual delivery of all multicast packets to all group members. Although intuitive, this simplistic definition of reliability in the multicast setting is imprecise and vague. It specifies neither the assumptions regarding the environment, nor the meaning of reliability in the context of a dynamic group membership. For instance, it is not clear what types of faults are allowed/tolerated and which packets are guaranteed delivery to hosts that join the multicast group while a particular transmission is already in progress.

In this chapter, we present a formal model of a reliable multicast service. This model precisely describes the service that several reliable multicast protocols, such as SRM [13] and LMS [32–34], strive to provide. Our reliable multicast service model includes a precise definition of what it means to be a member of the reliable multicast group and of which packets are guaranteed delivery to each reliable multicast group member. We begin the chapter by a brief modeling overview that describes the physical system at hand and our reliable multicast service model. Then, we present a timed I/O automaton model of the reliable multicast service and its environment. We conclude the chapter by stating the various transmission properties that our formal specification of the reliable multicast service provides.

3.1 Modeling Overview

3.1.1 The Physical System

We abstractly model the physical system as an infinite set of hosts that interact through an underlying network. This network involves a set of interconnected routers. Each host is connected to a particular router of the underlying network; for each host, we refer to this particular router as the *gateway router* of the particular host. Hosts and routers are connected among themselves through bi-directional communication links.

We assume that all hosts are of comparable processing power and storage resources. Resident on each host are a set of processes. We assume that hosts are symmetric in the sense that the same set of processes reside on each host. The set of processes on each host consists of a single application process and several additional communication service processes. Henceforth, we refer to the application process at each host as the *client* at the given host. The communication service processes, either individually or collectively, provide the communication services required by the client. For instance, the IP unicast service may be modeled as a set of processes, one such process for each host. Clients may thus exchange IP unicast packets through their respective IP unicast processes.

In terms of system faults, we consider only host crashes and packet drops on the communication links. Once a host crashes it remains crashed thereafter. A host is said to be *operational* prior to crashing and to have *crashed* thereafter. All the processes on each host are *fate-sharing*; that is, if a host crashes, then all of its processes crash. Router failures and network partitions are assumed to be ephemeral. Such failures are modeled as numerous packet drops.

Our assumption of an infinite set of hosts simplifies the modeling of host restarts. In particular, hosts restart by taking on the identity of another host that has up to that point in time been idle; that is, hosts restart by being reincarnated as completely new hosts. This modeling simplification is equivalent to having hosts choose a unique host identifier each time they restart; presuming of course the existence of an infinite set of such host identifiers. For instance, such an identifier could involve a processor identifier and an infinite reincarnation counter that is stable across host failures and gets incremented each time the processor crashes and restarts.

3.1.2 The Reliable Multicast Service and its Environment

We abstractly model the reliable multicast service as a single component that interacts with a potentially infinite set of clients. In terms of the above description of the physical system, the reliable multicast service encapsulates the behavior of all communication service processes at all hosts and the underlying network. The clients correspond to the client processes at each host. For simplicity, we assume that there is a single reliable multicast group. Since we assume a single client per host and a single reliable multicast group, we do not distinguish among the client process and the host when considering reliable multicast group membership. In fact, we often use the terms client and host interchangeably.

Group Membership

The reliable multicast service maintains the set of hosts that comprise the reliable multicast group. Hosts initiate the process of joining and leaving the reliable multicast group by issuing join and leave requests to the reliable multicast service. A host becomes a member of the reliable multicast group upon the acknowledgment of an earlier join request. Hosts may send and receive packets through the reliable multicast service only while they are both operational and members of the reliable multicast group.

A host initiates the process of leaving the reliable multicast group by issuing a leave request. Once a host issues a request to leave the reliable multicast group, it relinquishes its right to receive any more multicast packets. A host ceases to be a member of the reliable multicast group upon the acknowledgment of an earlier leave request. Once a host leaves the reliable multicast group, it may later rejoin the reliable multicast group.

Hosts may crash at any point in time. Following a crash, a host may restart by taking on the identity of a host that has up to that point in time been idle; that is, a host restarts by being reincarnated as a completely new host. This modeling simplification is equivalent to having hosts choose a unique host identifier each time they restart; presuming of course the existence of an infinite set of such host identifiers. For instance, such an identifier could involve a processor identifier and an infinite reincarnation counter that is stable across host failures and gets incremented each time the processor crashes and restarts.

Packet Naming Scheme

Floyd *et al.* [13] propose that, in the multicast setting, the application (the clients) should divide the data to be multicast into segments, called *application data units* (ADUs), and assign unique and persistent identifiers to each such segment. Floyd *et al.* argue that such a naming scheme is preferable to the use of shared communication state among the senders and the receivers, as is predominantly done in the unicast communication setting. An ADU-based scheme ensures unique and persistent naming of the data, which is desirable in the multicast setting. We proceed by giving a simple example that compares the two naming schemes.

For the purposes of illustration, consider the multicast transmission of a file named `foo`. In the case of the ADU-based naming scheme, the file `foo` is split up by the application into segments that are enumerated by consecutive sequence numbers. Thus, each data segment of the file `foo` is identified by the file name `foo` and its sequence number. Presuming that the file name `foo` is unique and persistent, this naming scheme identifies data segments uniquely and persistently. That is, the identifiers of the data segments remain the same no matter when and by which host the data segments are transmitted. Moreover, ADU names are persistent across host failures. In contrast, in the case of shared sender/receiver communication state, the data segments of the file `foo` are identified by ephemeral sequence numbers that pertain to a particular transmission of the file `foo` from a particular host. Although this scheme is simple and has been very successful in the unicast communication setting, it is not well suited for the multicast setting. This is the case because data segments may be named differently whenever they are retransmitted either by the source or by any other host. Thus, it is very hard to keep track of which data segments have actually been received by each receiver and to distribute the recovery overhead among the reliable multicast group members.

In their presentation of SRM [13], Floyd *et al.* use a simple ADU-based naming scheme in which each host assigns unique sequence numbers to each packet it multicasts. These sequence numbers are assigned in a continuous fashion as hosts join, leave, and rejoin the reliable multicast group; that is, consecutive packets sent by each host are assigned consecutive sequence numbers. Thus, packets are uniquely and persistently identified by a pair involving their source host and their sequence number. Throughout our treatment of reliable multicast, we adopt this naming scheme.

Reliability Guarantee

In this subsection, we describe the reliability guarantees provided by our reliable multicast service. As noted above, we focus on the eventual delivery aspects of reliability and do not consider any ordering and no-duplication guarantees. Thus, reliability entails specifying precisely which packets are guaranteed delivery to which members of the reliable multicast group.

We say that a member h of the reliable multicast group has *delivered* the packet p if it has either sent or received the packet p. We say that a member h of the reliable multicast group is *aware of* p if it has delivered either p or a packet p' that is sent earlier than p from the source of p; that is, the sequence number of p' is smaller than that of p. Moreover, we say that a packet p is *active* if at least one host that is operational, is a member of the reliable multicast group, and is aware of p, has delivered it.

We argue that once a host joins the reliable multicast group, the issue of catching up on any of the packets multicast earlier is orthogonal to the reliable transmission of future packets through the reliable multicast service. Once a host joins the reliable multicast group, the first packet it receives from a particular source dictates the set of packets whose delivery will be guaranteed to the given host; that is, earlier packets will not be delivered to the given host and later packets will be delivered provided they remain active after being sent and the host remains a member of the reliable multicast group. The host may catch up on the earlier packets from the given source through a separate service. The rationale behind this choice is that the recovery of a large number of earlier packets may strain the reliable multicast service and wastefully expose the recovery of these earlier packets to all or a subset of the reliable multicast group. Alternatively, the earlier packets may be requested directly from the source through a unicast communication channel.

Our reliable multicast service guarantees that if a packet p remains active forever after its transmission then any member of the reliable multicast group that becomes aware of p and remains operational and a member of the reliable multicast group thereafter, delivers p. Equivalently, if two members become aware of a packet p, remain members forever thereafter, and one member delivers p, then the other member delivers p also. It is important to note that a host is not required to remain a member of the reliable multicast group indefinitely in order for the packets it multicasts to be received by hosts that are aware of them; the eventual reception of packets is guaranteed to all hosts that are aware of them provided that the packets remain active forever after they are sent.

Although possibly not apparent at first glance, the above notion of reliability captures the reliability notion adopted by several reliable multicast protocols including SRM [13] and LMS [32–34]. For example, consider the simple scenario in which a particular host joins the reliable multicast group, starts multicasting packets, and remains a member of the group forever thereafter. Then, according to the above definition, the reliable multicast service eventually delivers to all the hosts, that join the reliable multicast group and remain members forever thereafter, all the packets that they become aware of; that is, each member delivers a particular suffix of the stream of packets multicast from the given source — the first packet of each such suffix is the first packet from the given source delivered by each member.

3.2 Formal Model

We formally specify the reliable multicast service and each of the client processes using timed I/O automata. The automaton $\mathrm{RM}(\Delta)$, for $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, models the reliable multicast service. $\mathrm{RM}(\Delta)$ defines what it means to be a member of the reliable multicast group and specifies precisely which packets are guaranteed delivery to each member of the reliable multicast group. The parameter Δ specifies an upper bound on the amount of time required by the reliable multicast service to reliably deliver each packet. The automaton RM-Client_h models the client at the host h. We let RM-Clients denote the composition of all client automata and $\mathrm{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, denote the composition of the reliable multicast service and all client automata; that is, $\mathrm{RM}_S(\Delta) = \mathrm{RM}(\Delta) \times \text{RM-Clients}$. Figure 3.1 depicts the interaction of the $\mathrm{RM}(\Delta)$ and RM-Client_h, for $h \in H$, automata.

Figure 3.1 Diagram of Reliable Multicast Service Architecture

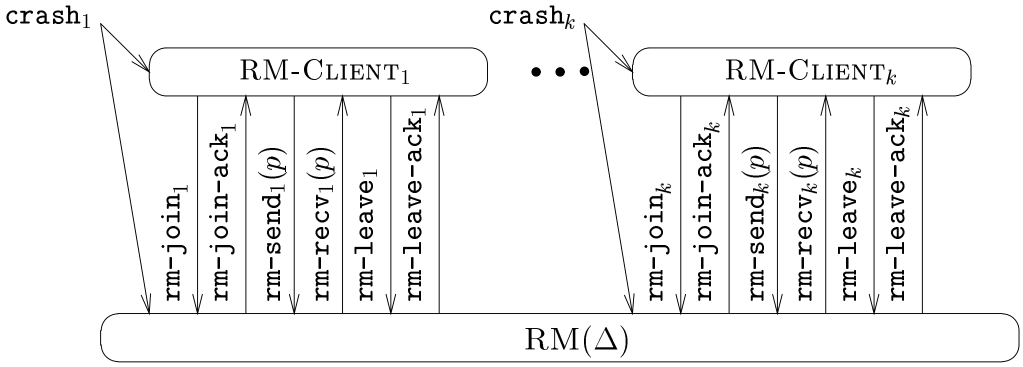


Figure 3.2 Reliable Multicast Specification Definitions

H Set of all hosts.

$Status = \{\texttt{idle}, \texttt{joining}, \texttt{leaving}, \texttt{member}, \texttt{crashed}\}$

$P_{\text{RM-CLIENT}}$ = Set of packets such that $\forall\, p \in P_{\text{RM-CLIENT}}$
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$data(p) \in \{0,1\}^*$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$suffix(p) = \{\langle s, i\rangle \in H \times \mathbb{N} \mid source(p) = s \wedge seqno(p) \le i\}$

3.2.1 Preliminary Definitions

Figure 3.2 includes several set definitions pertaining to our reliable multicast service specification. H is the set of all hosts that could potentially participate in the reliable multicast communication.

The set *Status* consists of all possible valuations of the reliable multicast membership status of a host. The value `idle` indicates that the host is *idle* with respect to the reliable multicast group; that is, it is neither a member, nor in the process of joining or leaving the reliable multicast group. The value `joining` indicates that the host is in the process of joining the reliable multicast group; that is, the client has issued a request to join the reliable multicast group and is awaiting an acknowledgment of this join request from the reliable multicast service. The value `leaving` indicates that the client is in the process of leaving the reliable multicast group; that is, the client has issued a request to leave the reliable multicast group and is awaiting an acknowledgment of this leave request from the reliable multicast service. The value `member` indicates that the client is a member of the reliable multicast group. The value `crashed` indicates that the host has crashed.

The set $P_{\text{RM-CLIENT}}$ represents the set of packets that may be transmitted by the client processes using the reliable multicast service. According to the ADU naming scheme described above, data segments are identified by their original source and a sequence number. Thus, for any packet $p \in P_{\text{RM-CLIENT}}$ the operations $source(p)$, $seqno(p)$, and $data(p)$ extract the source, sequence number, and data segment corresponding to the packet p. The operation $id(p)$ extracts the source and sequence number pair corresponding to the packet p. Such pairs comprise unique packet identifiers. We also define the $suffix(p)$ to be the subset of $P_{\text{RM-CLIENT}}$ comprised of all packets whose source is that of p and whose sequence number is greater than or equal to that of p.

Figure 3.3 The RM-CLIENT$_h$ Automaton

Parameters:

$h \in H$

Actions:

Input:
`crash`$_h$
`rm-join-ack`$_h$
`rm-leave-ack`$_h$
`rm-recv`$_h(p)$, for all $p \in P_{\text{RM-CLIENT}}$

Output:
`rm-join`$_h$
`rm-leave`$_h$
`rm-send`$_h(p)$, for all $p \in P_{\text{RM-CLIENT}}$

Time Passage:
$\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status \in Status$, initially $status =$ `idle`
$seqno \in \mathbb{N} \cup \perp$, initially $seqno = \perp$

Discrete Transitions:

input `crash`$_h$
eff $status :=$ `crashed`

input `rm-join-ack`$_h$
eff **if** $status =$ `joining` **then**
$status :=$ `member`

input `rm-leave-ack`$_h$
eff **if** $status =$ `leaving` **then**
$status :=$ `idle`

input `rm-recv`$_h(p)$
eff None

output `rm-join`$_h$
pre $status =$ `idle`
eff $status :=$ `joining`

output `rm-leave`$_h$
pre $status =$ `member`
eff $status :=$ `leaving`

output `rm-send`$_h(p)$
pre $status =$ `member` $\wedge\ source(p) = h$
$\wedge(seqno = \perp \vee seqno(p) = seqno + 1)$
eff $seqno := seqno(p)$

time-passage $\nu(t)$
pre None
eff $now := now + t$

3.2.2 The RM-CLIENT$_h$ Automaton

Figure 3.3 presents the signature, the variables, and the discrete transitions of RM-CLIENT$_h$. The RM-CLIENT$_h$ automaton models a *well-behaved* client; that is, a client that: i) transmits packets only when it is a member of the reliable multicast group, ii) transmits packets in ascending and contiguous sequence number order, iii) issues join requests only when it is idle with respect to the reliable multicast group, and iv) issues leave requests only when it is a member of the reliable multicast group.

Variables The variable $now \in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of RM-CLIENT$_h$. The variable $status \in Status$ denotes the membership status of the host h. It takes on one of the following values: `idle`, `joining`, `leaving`, `member`, and `crashed`. These values indicate whether the host h either is idle, joining, leaving, a member of the reliable multicast group, or has crashed, respectively. We say that a host h is *operational* if it has not crashed. After a host h crashes, none of the input actions affect the state of RM-CLIENT$_h$ and none of the locally controlled actions, except the time passage action, are enabled. The variable $seqno \in \mathbb{N} \cup \perp$ indicates the sequence number of the last packet to have been transmitted by RM-CLIENT$_h$ — the value $\perp$ indicates that RM-CLIENT$_h$ has yet to transmit a packet using the reliable multicast service. The $seqno$ variable is initialized to $\perp$.

Input Actions The input action `crash`$_h$ models the crashing of the host h. The effects of `crash`$_h$ are to record that the host h has crashed by setting the $status$ variable to `crashed`.

The input action $\texttt{rm-join-ack}_h$ acknowledges the client's join request at h. If the client is in the process of joining the reliable multicast group, *i.e.*, $status = \texttt{joining}$, then the $\texttt{rm-join-ack}_h$ action sets the *status* variable to $\texttt{member}$ so as to indicate that the client at h has become a member of the reliable multicast group.

The input action $\texttt{rm-leave-ack}_h$ acknowledges the client's leave request at h. If the client is in the process of leaving the reliable multicast group, *i.e.*, $status = \texttt{leaving}$, then the $\texttt{rm-leave-ack}_h$ action sets the *status* variable to $\texttt{idle}$ so as to indicate that the client at h has become idle with respect to the reliable multicast group.

The input action $\texttt{rm-recv}_h(p)$ models the delivery of the packet p to the client at h. This action has no effects.

Output Actions The output action $\texttt{rm-join}_h$ is performed by the client to initiate the process of joining the reliable multicast group. This action is enabled only while the client is idle with respect to the reliable multicast group. Its effects are to set the *status* variable to $\texttt{joining}$ so as to indicate that the client at h has initiated the process of joining the reliable multicast group.

The output action $\texttt{rm-leave}_h$ is performed by the client so as to initiate the process of leaving the reliable multicast group. This action is enabled only while the client is a member of the reliable multicast group. Thus, the client waits for join requests to complete prior to issuing leave requests. Its effects are to set the *status* variable to $\texttt{leaving}$ so as to indicate that the client at h has initiated the process of leaving the reliable multicast group.

The output action $\texttt{rm-send}_h(p)$ models the client's transmission of the packet p using the reliable multicast service. The $\texttt{rm-send}_h(p)$ action is enabled when the client is a member of the reliable multicast group and the packet p is either the first or the next packet in the sequence of packets to be transmitted by the client at h; that is, $status = \texttt{member}$, $source(p) = h$, and either $seqno = \perp$ or $seqno(p) = seqno + 1$. The effects of the $\texttt{rm-send}_h(p)$ action are to set *seqno* to $seqno(p)$ (or, equivalently, to increment *seqno*), thus recording the transmission of the packet p.

Time Passage The action $\nu(t)$ models the passage of t time units. It is enabled at any point in time and increments the variable *now* by t time units.

3.2.3 The RM Automaton

The RM automaton specifies the reliable multicast service as a whole. Figures 3.4 and 3.5 present the signature, the variables, and the discrete transitions of RM.

Parameters

The RM automaton is parameterized by a time bound, $\Delta \in \mathbb{R}^{\geq 0} \cup \{\infty\}$, which specifies the maximum delay in delivering each packet sent to the appropriate members of the reliable multicast group. The value ∞ corresponds to the case in which the reliable multicast service guarantees the eventual delivery of all packets to the appropriate members of the reliable multicast group. An instance of the RM automaton is denoted by RM(Δ).

Variables

The variable $now \in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of RM. Each variable $status(h) \in Status$, for $h \in H$, denotes the status of the host h. Each of its

Figure 3.4 The RM Automaton — Signature

Parameters:	
$\Delta \in \mathbb{R}^{\geq 0} \cup \{\infty\}$	
Actions:	
Input:	**Output:**
$\texttt{crash}_h$, for $h \in H$	$\texttt{rm-join-ack}_h$, for $h \in H$
$\texttt{rm-join}_h$, for $h \in H$	$\texttt{rm-leave-ack}_h$, for $h \in H$
$\texttt{rm-leave}_h$, for $h \in H$	$\texttt{rm-recv}_h(p)$, for $h \in H, p \in P_{\text{RM-CLIENT}}$
$\texttt{rm-send}_h(p)$, for $h \in H, p \in P_{\text{RM-CLIENT}}$	**Time Passage:**
	$\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

valuations is described in the definition of the set *Status*. We say that the host h is *operational* if it has not crashed. After a host h crashes, none of the input actions pertaining to h affect the state of RM and none of the locally controlled actions pertaining to h are enabled.

Each variable $\textit{trans-time}(p) \in \mathbb{R}^{\geq 0} \cup \perp$, for $p \in P_{\text{RM-CLIENT}}$, denotes the transmission time of the packet p; that is, the time the packet p was sent by its source. Prior to the transmission of p, $\textit{trans-time}(p)$ is equal to $\perp$. Each variable $\textit{expected}(h, h') \subseteq H \times \mathbb{N}$, for $h, h' \in H$, is the set comprised of the identifiers of the packets from h' that the host h is aware of since it last joined the reliable multicast group and, consequently, expects to deliver. Each variable $\textit{delivered}(h, h') \subseteq H \times \mathbb{N}$, for $h, h' \in H$, is the set comprised of the identifiers of the packets from h' that the host h has delivered.

Derived Variables

The derived variable $\textit{idle} \subseteq H$ is a set of hosts that is comprised of all the hosts that are idle with respect to the reliable multicast group. The derived variable $\textit{joining} \subseteq H$ is a set of hosts that are in the process of joining the reliable multicast group. The derived variable $\textit{leaving} \subseteq H$ is a set of hosts that are in the process of leaving the reliable multicast group. The derived variable $\textit{members} \subseteq H$ is a set of hosts that are members of the reliable multicast group.

The derived variable $\textit{intended}(p)$, for each $p \in P_{\text{RM-CLIENT}}$, is the set of hosts that are expecting the delivery of the packet p. We henceforth refer to the set $\textit{intended}(p)$ as the intended delivery set of p. The derived variable $\textit{completed}(p)$, for each $p \in P_{\text{RM-CLIENT}}$, is the set of hosts that have delivered the packet p. Recall that we say that a host has delivered a packet p if it has either sent or received p. We henceforth refer to the set $\textit{completed}(p)$ as the completed delivery set of p. The derived variable *sent-pkts* is the set of packets that have been sent since the beginning of the given execution of the RM(Δ) automaton. The derived variable *active-pkts* is the set comprised of the sent packets that have been delivered by at least one of the hosts in their respective intended delivery sets.

Input Actions

Each input action $\texttt{crash}_h$, for $h \in H$, models the crashing of the host h. The effects of $\texttt{crash}_h$ are to record that the host h has crashed by setting the variable $\textit{status}(h)$ to the value `crashed`. Furthermore, the $\texttt{crash}_h$ action resets the set of packets that the host h is expecting from each source and the set of packets it has delivered from each source. Thus, the RM automaton is released of the obligation to deliver any of the active packets to the host h.

The input action $\texttt{rm-join}_h$ models the client's request at the host h to join the reliable multicast group. The $\texttt{rm-join}_h$ action is effective only while the host h is idle with respect to the reliable multicast group. When effective, the $\texttt{rm-join}_h$ action sets the $\textit{status}(h)$ variable to `joining` so as to record that the host h has initiated the process of joining the reliable multicast group. If

Figure 3.5 The RM(Δ) Automaton — Variables and Discrete Transitions

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status(h) \in Status$, for all $h \in H$, initially $status(h) = \texttt{idle}$, for all $h \in H$
$trans\text{-}time(p) \in \mathbb{R}^{\geq 0} \cup \perp$, for all $p \in P_{\text{RM-CLIENT}}$, initially $trans\text{-}time(p) = \perp$, for all $p \in P_{\text{RM-CLIENT}}$
$expected(h, h') \subseteq H \times \mathbb{N}$, for all $h, h' \in H$, initially $expected(h, h') = \emptyset$, for all $h, h' \in H$
$delivered(h, h') \subseteq H \times \mathbb{N}$, for all $h, h' \in H$, initially $delivered(h, h') = \emptyset$, for all $h, h' \in H$

Derived Variables:

$idle = \{h \in H \mid status(h) = \texttt{idle}\}$
$joining = \{h \in H \mid status(h) = \texttt{joining}\}$
$leaving = \{h \in H \mid status(h) = \texttt{leaving}\}$
$members = \{h \in H \mid status(h) = \texttt{member}\}$
$intended(p) = \{h \in H \mid id(p) \in expected(h, source(p))\}$, for all $p \in P_{\text{RM-CLIENT}}$
$completed(p) = \{h \in H \mid id(p) \in delivered(h, source(p))\}$, for all $p \in P_{\text{RM-CLIENT}}$
$sent\text{-}pkts = \{p \in P_{\text{RM-CLIENT}} \mid trans\text{-}time(p) \neq \perp\}$
$active\text{-}pkts = \{p \in P_{\text{RM-CLIENT}} \mid p \in sent\text{-}pkts \wedge intended(p) \cap completed(p) \neq \emptyset\}$

Discrete Transitions:

input $\texttt{crash}_h$

eff $status(h) := \texttt{crashed}$
  foreach $h' \in H$ **do:**
    $expected(h, h') := \emptyset$
    $delivered(h, h') := \emptyset$

input $\texttt{rm-join}_h$

eff **if** $h \in idle$ **then**
    $status(h) := \texttt{joining}$

input $\texttt{rm-leave}_h$

eff **if** $h \in joining \cup members$ **then**
    $status(h) := \texttt{leaving}$
    foreach $h' \in H$ **do:**
      $expected(h, h') := \emptyset$
      $delivered(h, h') := \emptyset$

input $\texttt{rm-send}_h(p)$

eff **if** $h \in members \cap \{source(p)\}$ **then**
    if $expected(h, h) = \emptyset$ **then**
      $expected(h, h) := suffix(p)$
    if $id(p) \in expected(h, h)$ **then**
      $trans\text{-}time(p) := now$
      $delivered(h, h) \cup= \{id(p)\}$

output $\texttt{rm-join-ack}_h$

pre $h \in joining$
eff $status(h) := \texttt{member}$

output $\texttt{rm-leave-ack}_h$

pre $h \in leaving$
eff $status(h) := \texttt{idle}$

output $\texttt{rm-recv}_h(p)$

pre $h \in members \backslash \{source(p)\}$
  $\wedge p \in sent\text{-}pkts$
  $\wedge (expected(h, source(p)) = \emptyset$
    $\Rightarrow now \leq trans\text{-}time(p) + \Delta)$
  $\wedge (expected(h, source(p)) \neq \emptyset$
    $\Rightarrow id(p) \in expected(h, source(p)))$
eff **if** $expected(h, source(p)) = \emptyset$ **then**
    $expected(h, source(p)) := suffix(p)$
  $delivered(h, source(p)) \cup= \{id(p)\}$

time-passage $\nu(t)$

pre $\forall\, p \in active\text{-}pkts$,
    $now + t \leq trans\text{-}time(p) + \Delta$
    $\vee intended(p) \subseteq completed(p)$
eff $now := now + t$

the client is either a member of or in the process of joining the reliable multicast group, then the $\texttt{rm-join}_h$ action is superfluous. If the client is already in the process of leaving the group, then the $\texttt{rm-join}_h$ action is discarded so as to allow the process of leaving the reliable multicast group to complete.

The input action $\texttt{rm-leave}_h$ models the client's request at the host h to leave the reliable multicast group. The $\texttt{rm-leave}_h$ action is effective only while the host h is a member of or in the process of joining the reliable multicast group. When effective, the $\texttt{rm-leave}_h$ action sets the $status(h)$ variable to $\texttt{leaving}$ so as to record that the host h has initiated the process of leaving the reliable multicast group. Moreover, the $\texttt{rm-leave}_h$ action initializes the set of packets that the host h is expecting from each source and the set of packets it has delivered from each source. Thus, the RM automaton is released of the obligation to deliver any of the active packets to the host h. Leave requests overrule join requests; that is, when a $\texttt{rm-leave}_h$ action is performed while the host h is in the process of joining the reliable multicast group, its effects are to abort the process of joining and to initiate the process of leaving the reliable multicast group. If the client is either idle or already in the process of leaving the reliable multicast group, then the $\texttt{rm-leave}_h$ action is superfluous.

The client at h sends the packet p using the reliable multicast service through the input action

$\texttt{rm-send}_h(p)$. The $\texttt{rm-send}_h(p)$ action is effective only when the host h is both a member of the reliable multicast group and the source of the packet p. If p is the first packet sent by the host h, then the $\texttt{rm-send}_h(p)$ action initializes the set of packets expected by h from h to the set $\mathit{suffix}(p)$; that is, all packets whose source is h and whose sequence number is greater or equal to that of p. Then, if p is in the expected set of packets of h from h, the $\texttt{rm-send}_h(p)$ records the transmission time of p by setting the variable $\mathit{trans\text{-}time}(p)$ to now and adds the packet p to the set of packets from the host h that the host h has delivered.

Output Actions

The output action $\texttt{rm-join-ack}_h$ acknowledges the join request of the client at h. The action $\texttt{rm-join-ack}_h$ is enabled when the host h is in the process of joining the reliable multicast group. Its effects are to set the $\mathit{status}(h)$ variable to `member` so as to indicate that the client at h has become a member of the reliable multicast group.

The output action $\texttt{rm-leave-ack}_h$ acknowledges the leave request of the client at h. The action $\texttt{rm-leave-ack}_h$ is enabled when the host h is in the process of leaving the reliable multicast group. Its effects are to set the $\mathit{status}(h)$ variable to `idle` so as to indicate that the client at h has become idle with respect to the reliable multicast group.

The output action $\texttt{rm-recv}_h(p)$ models the delivery of the packet p to the client at h. The $\texttt{rm-recv}_h(p)$ action is enabled when the host h is a member of the reliable multicast group, the host h is not the source of p, and p is an active packet. Moreover, if the expected deliver set of h with respect to the source of p is undefined, then the delivery deadline $\mathit{trans\text{-}time}(p) + \Delta$ of p must not have expired; that is, the first packet from any source to be delivered to any client must be delivered prior to its delivery deadline. If the expected deliver set of h with respect to the source of p has already been defined, then p must be expected by h. The effects of the $\texttt{rm-recv}_h(p)$ action are: i) to define the expected delivery set of h with respect to the source of p to the set $\mathit{suffix}(p)$, unless already defined, and ii) to add the host h to the completed delivery set of p.

Time Passage

The action $\nu(t)$ models the passage of t time units. Time is prevented from elapsing past the delivery deadline of any active packet that has yet to be delivered to all the hosts in its intended delivery set. Thus, prior to allowing time to elapse past the delivery deadline of an active packet, all the hosts in its intended delivery set must either send or receive the packet, leave the reliable multicast group, or crash.

3.3 Properties of the Reliable Multicast Service

In this section, we present various properties of the $\mathrm{RM}(\Delta)$, for $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, the RM-$\textsc{Client}_h$, for $h \in H$, and the $\mathrm{RM}_S(\Delta) = \mathrm{RM}(\Delta) \times \textsc{rmClients}$ automata. We begin the section by defining various notions and specifying some preliminary properties of the aforementioned automata. We conclude the section by defining the reliability properties exhibited by the $\mathrm{RM}_S(\Delta)$ automaton; that is, the reliable multicast service interacting with well-behaved clients.

3.3.1 Preliminary Properties and Definitions

The automaton RM-$\textsc{Client}_h$, for any $h \in H$, satisfies *transmission correctness*, *transmission uniqueness*, and *in order transmission*. Transmission correctness is the property that clients only transmit packets for which they are actually the source. Transmission uniqueness is the property that no two packets transmitted by a client share the same identifier. Finally, in order transmission is the property that each client transmits packets through the reliable multicast group in ascending sequence number order.

Lemma 3.1 (Transmission Correctness) *Let β be any timed trace of* RM-CLIENT$_h$*, for any $h \in H$. If β contains the action* $\texttt{rm-send}_h(p)$*, for some $p \in P_{\text{RM-CLIENT}}$, then the host h is the source of p; that is, $h = source(p)$.*

Proof: Follows directly from the precondition of the action $\texttt{rm-send}_h(p)$. ❒

Lemma 3.2 (Transmission Uniqueness) *Let β be any timed trace of* RM-CLIENT$_h$*, for any $h \in H$. For any packet identifier $\langle s, i\rangle \in H \times \mathbb{N}$, at most one packet $p \in P_{\text{RM-CLIENT}}$ is transmitted within β; that is, β contains at most one action* $\texttt{rm-send}_h(p)$*, for $p \in P_{\text{RM-CLIENT}}$, such that $id(p) = \langle s, i\rangle$.*

Proof: Let α be any timed execution of RM-CLIENT$_h$ such that $\beta = ttrace(\alpha)$. Within α each action $\texttt{rm-send}_h(p')$, for $p' \in P_{\text{RM-CLIENT}}$ such that $source(p') = h$, transmits the packet p' whose sequence number is equal to *seqno* and increments the variable *seqno*. Since no other actions affect the variable *seqno* it follows that *seqno* monotonically increases each time a packet is transmitted. Thus, β does not contain the transmission of more than one packets sharing the same sequence number. ❒

Lemma 3.3 (In Order Transmission) *Let β be any timed trace of* RM-CLIENT$_h$*, for $h \in H$, that contains the actions* $\texttt{rm-send}_h(p)$ *and* $\texttt{rm-send}_h(p')$*, for $p, p' \in P_{\text{RM-CLIENT}}$, such that $h = source(p) = source(p')$ and $seqno(p) < seqno(p')$. Then, the action* $\texttt{rm-send}_h(p)$ *precedes the action* $\texttt{rm-send}_h(p')$ *in β.*

Proof: The effects of any $\texttt{rm-send}_h(p'')$, for $p'' \in P_{\text{RM-CLIENT}}$, are to increment the variable RM-CLIENT$_h$.*seqno*. Moreover, no other action affects the variable RM-CLIENT$_h$.*seqno*. Thus is, the variable RM-CLIENT$_h$.*seqno* is monotonically non-decreasing in any execution of RM-CLIENT$_h$.

The actions $\texttt{rm-send}_h(p)$ and $\texttt{rm-send}_h(p')$ are enabled only when $seqno(p) =$ RM-CLIENT$_h$.*seqno* and $seqno(p') =$ RM-CLIENT$_h$.*seqno*, respectively. It follows that $\texttt{rm-send}_h(p)$ precedes the action $\texttt{rm-send}_h(p')$ in any timed execution of RM-CLIENT$_h$ such that $\beta = ttrace(\alpha)$. ❒

The automaton $\text{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$ satisfies *transmission integrity.* Transmission integrity it the property that, within a timed trace of $\text{RM}_S(\Delta)$, the reception of a packet must be preceded by the particular packet's transmission.

Lemma 3.4 (Transmission Integrity) *Let β be any timed trace of $\text{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$. For $h, h' \in H$ and $p \in P_{\text{RM-CLIENT}}$, such that $h \neq h'$ and $h = source(p)$, it is the case that any* $\texttt{rm-recv}_{h'}(p)$ *action is preceded in β by a* $\texttt{rm-send}_h(p)$ *action.*

Proof: Let α be any timed execution of $\text{RM}_S(\Delta)$ such that $\beta = ttrace(\alpha)$. It suffices to show that any $\texttt{rm-recv}_{h'}(p)$ action is preceded by a $\texttt{rm-send}_h(p)$ action within α. This follows directly from the precondition of the action $\texttt{rm-recv}_{h'}(p)$. In particular, the precondition of the action $\texttt{rm-recv}_{h'}(p)$ requires that there is a tuple in *pkts* corresponding to the packet p. However, such a tuple may be added to *pkts* only by the occurrence of the action $\texttt{rm-send}_h(p)$. Thus, the occurrence of any action $\texttt{rm-recv}_{h'}(p)$ within α is preceded by the occurrence of the action $\texttt{rm-send}_h(p)$. ❒

We proceed by defining the set of *members* of the reliable multicast group following a finite timed trace of $\mathrm{RM}_S(\Delta)$.

Definition 3.1 (Membership) *Let β be any timed trace of $\mathrm{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$. We define the* ***members of*** *β, denoted $members(\beta)$, to be the set of all hosts $h \in H$ such that β contains a $\texttt{rm-join-ack}_h$ action that is not succeeded by either an $\texttt{rm-leave}_h$ or a $\texttt{crash}_h$ action. If a host $h \in H$ is in the set $members(\beta)$, then we say that h is a reliable multicast group* member *of β.*

The following lemma relates the set $members(\beta)$ of Definition 3.1 to the derived variable *members* of the automaton RM.

Lemma 3.5 *Let $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$ and α be any finite timed execution of $\mathrm{RM}_S(\Delta)$. Letting s be the last state in α and β be the timed trace of α, it is the case that $s.members = members(\beta)$.*

Proof: Follows directly from the definitions of $s.members$ and $members(\beta)$. ❒

Lemma 3.6 *Let $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, $h \in H$, and α be any timed execution of $\mathrm{RM}_S(\Delta)$ such that $h \in members(ttrace(\alpha))$. Letting s be any state following the last occurrence of the $\texttt{rm-join-ack}_h$ action in α, it is the case that $h \in s.members$.*

Proof: Let α', α'' be the execution fragments of $\mathrm{RM}_S(\Delta)$ such that $\alpha'\alpha'' = \alpha$ and the last action in α' is the last occurrence of the $\texttt{rm-join-ack}_h$ action in α. Letting $s' = \alpha'.lstate$, the effects of the $\texttt{rm-join-ack}_h$ action imply that $s'.status(h) = \texttt{member}$. By the definition of $members(ttrace(\alpha))$, it follows that α'' contains neither a $\texttt{rm-leave}_h$ or a $\texttt{crash}_h$ action.

The rest of the proof involves showing that for any prefix α_n of α'' of length $n \in \mathbb{N}$, such that $s_n = \alpha_n.lstate$, it is the case that $h \in s_n.members$. This follows by a simple induction on the length n of α_n. For the base case, consider α_0. Since $\alpha_0 = s'$ and $s'.status(h) = \texttt{member}$, it follows that $s_0.status(h) = \texttt{member}$, as required. For the inductive step, consider α_{k+1}. Let $s_{k+1} = \alpha_{k+1}.lstate$, let α_k be the prefix of α_{k+1} involving its first k steps, and $s_k = \alpha_k.lstate$. The induction hypothesis is the assertion that $s_k.status(h) = \texttt{member}$. Since α'' contains neither a $\texttt{rm-leave}_h$ or a $\texttt{crash}_h$ action, the $k+1$-st step of α_{k+1} is neither an $\texttt{rm-leave}_h$ or a $\texttt{crash}_h$ action. Moreover, since $s_k.status(h) = \texttt{member}$, the $k+1$-st step of α_{k+1} is neither an $\texttt{rm-join}_h$, $\texttt{rm-join-ack}_h$, nor $\texttt{rm-leave-ack}_h$ action. The remaining actions do not affect the $status(h)$ variable. Thus, it follows that $s_{k+1}.status(h) = \texttt{member}$, as required. ❒

We proceed by defining the *intended and completed delivery* sets of a packet within a timed trace of $\mathrm{RM}_S(\Delta)$.

Definition 3.2 (Intended Delivery Set) *Let β be any timed trace of $\mathrm{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, containing the transmission of a packet $p \in P_{\text{RM-CLIENT}}$. We define the* ***intended delivery set of*** *p* ***within*** *β, denoted $intended(p, \beta)$, to be the members of β that have delivered either the packet p or an earlier packet from the source of p since they last joined the reliable multicast group; that is, $h \in intended(p, \beta)$ if and only if $h \in members(\beta)$ and the last $\texttt{rm-join-ack}_h$ action in β is succeeded by either a $rm\text{-}send_h(p')$ or a $rm\text{-}recv_h(p')$ action, where $source(p') = source(p)$ and $seqno(p') \leq seqno(p)$.*

Lemma 3.7 *Let β be any finite timed trace of* $\mathrm{RM}_S(\Delta)$*, for any* $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$*, containing the transmission of a packet* $p \in P_{\text{RM-CLIENT}}$*. Then, it is the case that* $\mathit{intended}(p, \beta) \subseteq \mathit{members}(\beta)$*.*

Proof: Follows directly from Definition 3.2. ❒

The following lemma relates the intended delivery set of a packet p within a timed trace β defined in Definition 3.2 to the derived variable $\mathit{intended}(p)$ of the RM automaton.

Lemma 3.8 *Let* $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$*,* $p \in P_{\text{RM-CLIENT}}$*, and α be any finite timed execution of* $\mathrm{RM}_S(\Delta)$ *that contains the transmission of p. Letting* $s = \alpha.\mathit{lstate}$ *and* $\beta = \mathit{ttrace}(\alpha)$*, it is the case that* $s.\mathit{intended}(p) = \mathit{intended}(p, \beta)$*.*

Proof: Follows directly from the definition of the derived variable $\mathit{intended}(p)$ and Definition 3.2. ❒

Definition 3.3 (Completed Delivery Set) *Let β be any timed trace of* $\mathrm{RM}_S(\Delta)$*, for any* $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$*, containing the transmission of a packet* $p \in P_{\text{RM-CLIENT}}$*. We define the* ***completed delivery set of** p **within*** β*, denoted* $\mathit{completed}(p, \beta)$*, to be the members of β that have delivered the packet p since they last joined the reliable multicast group; that is,* $h \in \mathit{completed}(p, \beta)$ *if and only if* $h \in \mathit{members}(\beta)$ *and the last* `rm-join-ack`$_h$ *action in β is succeeded by either a* $\mathit{rm\text{-}send}_h(p)$ *or a* $\mathit{rm\text{-}recv}_h(p)$ *action.*

The following lemma relates the completed delivery set of a packet p within a timed trace β defined in Definition 3.3 to the derived variable $\mathit{completed}(p)$ of the RM automaton.

Lemma 3.9 *Let* $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$*,* $p \in P_{\text{RM-CLIENT}}$*, and α be any finite timed execution of* $\mathrm{RM}(\Delta) \times \mathrm{RMCLIENTS}$ *that contains the transmission of p. Letting* $s = \alpha.\mathit{lstate}$ *and* $\beta = \mathit{ttrace}(\alpha)$*, it is the case that* $s.\mathit{completed}(p) = \mathit{completed}(p, \beta)$*.*

Proof: Follows directly from the definition of the derived variable $\mathit{completed}(p)$ and Definition 3.3. ❒

We continue by defining the set of active packets within a timed trace of $\mathrm{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$. This set is comprised of the packets whose intended and completed delivery sets within the given timed trace overlap; that is, the packets for which there is at least one host that was and has remained a member of the reliable multicast group following the packet's transmission and, moreover, has either sent or received the packet.

Definition 3.4 (Active Packets) *Let β be any timed trace of* $\mathrm{RM}(\Delta) \times \mathrm{RMCLIENTS}$*, for any* $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$*. We define the set of* ***active packets within*** β*, denoted* $\mathit{active\text{-}pkts}(\beta)$*, to be the set of all packets* $p \in P_{\text{RM-CLIENT}}$ *such that* $\mathit{intended}(p, \beta) \cap \mathit{completed}(p, \beta) \neq \emptyset$*. If a packet* $p \in P_{\text{RM-CLIENT}}$ *is in the set* $\mathit{active\text{-}pkts}(\beta)$*, then we say that p is active within β.*

The following lemma relates the set of active packets defined in Definition 3.4 to the derived variable *active-pkts* of the RM automaton.

Lemma 3.10 *Let* $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$*,* $p \in P_{\text{RM-CLIENT}}$*, and α be any finite timed execution of* $\mathrm{RM}(\Delta) \times \mathrm{RMCLIENTS}$ *that contains the transmission of p. Letting* $s = \alpha.\mathit{lstate}$ *and* $\beta = \mathit{ttrace}(\alpha)$*, it is the case that* $s.\mathit{active\text{-}pkts} = \mathit{active\text{-}pkts}(\beta)$*.*

Proof: Follows directly from Lemmas 3.8 and 3.9, Definition 3.4, and the definition of the derived variable *active-pkts* of the RM automaton. ❒

Lemma 3.11 *Let* β, β' *be timed traces of* $\mathrm{RM}(\Delta) \times \mathrm{RMCLIENTS}$*, for any* $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$*, containing the transmission of a packet* $p \in P_{\text{RM-CLIENT}}$ *such that* $\beta' \leq \beta$*. Then, it is the case that if* $p \in \textit{active-pkts}(\beta)$ *then* $p \in \textit{active-pkts}(\beta')$*.*

Proof: We prove the above claim by contradiction. Suppose that it is the case that $p \notin \textit{active-pkts}(\beta')$ and $p \in \textit{active-pkts}(\beta)$. Thus, there must be some action π following β' such that $p \notin \textit{active-pkts}(\beta_\pi)$ and $p \in \textit{active-pkts}(\beta_\pi \cdot \pi)$, where β_π, β'_π are the trace fragments of β such that $\beta_\pi \cdot \pi \cdot \beta'_\pi = \beta$.

Let α be any timed execution of $\mathrm{RM}(\Delta) \times \mathrm{RMCLIENTS}$ such that $\beta = \textit{ttrace}(\alpha)$ and s_π and s'_π be the pre- and post-states of π within α. We proceed by considering the possibility of π being any of the actions of the $\mathrm{RM}_S(\Delta)$ automaton that affect the valuation of the derived variable *active-pkts*. Since $p \notin \textit{active-pkts}(\beta_\pi)$, Lemma 3.10 implies that $p \notin s_\pi.\textit{active-pkts}$. Thus, none of the $\texttt{rm-recv}_h(p)$, for $h \in H$, are enabled. Lemma 3.1 implies that none of the actions $\texttt{rm-send}_h(p)$, for $h \in H$, except for $h = \textit{source}(p)$ are enabled. Moreover, since p has already been sent within β_π, Lemma 3.2 implies that $\texttt{rm-send}_h(p)$, for $h = \textit{source}(p)$, is not enabled in s_π. The only other actions that affect the variable *active-pkts* are the $\texttt{crash}_h$ and $\texttt{rm-leave}_h$ actions, for $h \in H$. The effects of these actions are to remove the host h from both the $\textit{intended}(p)$ and $\textit{completed}(p)$ sets. Clearly, if $\textit{intended}(p) \cap \textit{completed}(p) = \emptyset$ in the state s_π, then the same holds for s'_π. Thus, it follows that $p \notin s'_\pi.\textit{active-pkts}$. Lemma 3.10 implies that $p \notin \textit{active-pkts}(\beta_\pi \cdot \pi)$, which contradicts our original supposition. ❒

Lemma 3.12 *Let* $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$*,* $h \in H$*,* $p \in P_{\text{RM-CLIENT}}$*, and* α *be any timed execution of* $\mathrm{RM}_S(\Delta)$ *that ends with the discrete transition* (s, π, s')*, for* $\pi = \textit{rm-send}_h(p)$*. Then, it is the case that* $p \in s'.\textit{sent-pkts}$*.*

Proof: From the precondition of $\textit{rm-send}_h(p)$, it follows that $s.\textit{status} = \texttt{member}$ and $\textit{source}(p) = h$. Thus, the effects of the $\textit{rm-send}_h(p)$ are to set the variable $\textit{trans-time}(p)$ to the value of $\textit{now}$. By the definition of the derived variable *sent-pkts* of $\mathrm{RM}(\Delta)$, it follows that $p \in s'.\textit{sent-pkts}$, as required. ❒

Lemma 3.13 *Let* $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$*,* $p \in P_{\text{RM-CLIENT}}$*,* $s \in \textit{states}(\mathrm{RM}(\Delta))$ *be any reachable state of* $\mathrm{RM}(\Delta)$ *such that* $p \in s.\textit{sent-pkts}$*, and* α *be any timed execution fragment of* $\mathrm{RM}(\Delta)$ *such that* $s = \alpha.\textit{fstate}$*. For any* $s' \in \textit{states}(\mathrm{RM}(\Delta))$ *in* α*, it is the case that* $p \in s'.\textit{sent-pkts}$*.*

Proof: Follows from a simple induction on the length of the prefix of α leading to s' and the fact that none of the actions of $\mathrm{RM}(\Delta)$ reset the variable $\textit{trans-time}(p)$ to $\perp$. ❒

Lemma 3.14 *Let* $h \in H$*,* $p \in P_{\text{RM-CLIENT}}$*,* $s \in \textit{states}(\mathrm{RM}(\Delta))$*, for* $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$*, and* α *be any timed execution fragment of* $\mathrm{RM}(\Delta)$*, such that* $s = \alpha.\textit{fstate}$*,* $h \in s.\textit{intended}(p)$ *(or, equivalently,* $\textit{id}(p) \in s.\textit{expected}(h, \textit{source}(p))$*), and* α *contains neither* $\texttt{crash}_h$ *nor* $\texttt{rm-leave}_h$ *actions. Then, for any state* $s' \in \textit{states}(\mathrm{RM}(\Delta))$ *in* α*, it is the case that* $h \in s'.\textit{intended}(p)$ *(or, equivalently,* $\textit{id}(p) \in s'.\textit{expected}(h, \textit{source}(p))$*).*

Proof: Follows from a simple induction on the length of the prefix of α leading to s' and the facts that: i) the variable $\mathit{expected}(h, \mathit{source}(p))$ may only be set to a non-empty set if it is empty, and ii) the variable $\mathit{expected}(h, \mathit{source}(p))$ is reset to the empty set only by the actions $\mathtt{crash}_h$ and $\mathtt{rm\text{-}leave}_h$. ❒

Invariant 3.1 *For $h \in H$ and any reachable state s of* $\mathrm{RM}(\Delta) \times \mathrm{RMCLIENTS}$*, for $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, it is the case that* $s[\text{RM-CLIENT}_h].\mathit{status} = s[\mathrm{RM}(\Delta)].\mathit{status}(h)$.

Proof: Follows by a simple induction on the length of any timed execution of $\mathrm{RM}_S(\Delta)$ leading to s. ❒

Invariant 3.2 *Let $h, h' \in H$ and s be any reachable state of* $\mathrm{RM}_S(\Delta)$*, for $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$. If* $s[\mathrm{RM}(\Delta)].\mathit{status}(h) \neq \mathtt{member}$*, then it is the case that* $s[\mathrm{RM}(\Delta)].\mathit{expected}(h, h') = \emptyset$ *and* $s[\mathrm{RM}(\Delta)].\mathit{delivered}(h, h') = \emptyset$.

Proof: Follows from a simple induction on the length of any execution of $\mathrm{RM}_S(\Delta)$ leading to s and the facts that: i) the actions that set the variable $\mathrm{RM}(\Delta).\mathit{expected}(h, h')$ are only enabled when $\mathrm{RM}(\Delta).\mathit{status}(h) = \mathtt{member}$, ii) the actions that add elements to the variable $\mathrm{RM}(\Delta).\mathit{delivered}(h, h')$ are only enabled when $\mathrm{RM}(\Delta).\mathit{status}(h) = \mathtt{member}$, and iii) the actions that reset the variables $\mathrm{RM}(\Delta).\mathit{expected}(h, h')$ and $\mathrm{RM}(\Delta).\mathit{delivered}(h, h')$ also set the variable $\mathrm{RM}(\Delta).\mathit{status}(h)$ to a value other than $\mathtt{member}$. ❒

Letting $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, the following invariant states that, for any active packet in any reachable state of $\mathrm{RM}(\Delta) \times \mathrm{RMCLIENTS}$, either Δ time units have yet to elapse past the packet's transmission time, or the packet has been delivered to all members that are aware of it. Thus, Δ bounds the delivery latency of any active packet.

Invariant 3.3 *Let s be any reachable state of the timed automaton* $\mathrm{RM}_S(\Delta)$*, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$. Then, for any active packet $p \in P_{\text{RM-CLIENT}}$ in s, i.e., $p \in s.\mathit{active\text{-}pkts}$, it is the case that either $s.\mathit{now} \leq s.\mathit{trans\text{-}time}(p) + \Delta$ or $s.\mathit{intended}(p) \subseteq s.\mathit{completed}(p)$.*

Proof: The proof is by induction of the number of steps $n \in \mathbb{N}$ of a timed execution α of $\mathrm{RM}_S(\Delta)$ leading to the state s. For the base case, consider a timed execution with no steps; that is, $n = 0$ and $\alpha = s$ for some $s \in \mathit{start}(\mathrm{RM}_S(\Delta))$. Since $s.\mathit{active\text{-}pkts} = \emptyset$, the invariant assertion is trivially satisfied.

For the inductive step, consider a timed execution α with $k+1$ steps. Let α' be the prefix of α containing the first k steps of α and s' be the last state of α'. The induction hypothesis is that for any active packet $p' \in P_{\text{RM-CLIENT}}$ in s', *i.e.*, $p' \in s'.\mathit{active\text{-}pkts}$, it is the case that either $s'.\mathit{now} \leq s'.\mathit{trans\text{-}time}(p') + \Delta$ or $s'.\mathit{intended}(p') \subseteq s'.\mathit{completed}(p')$. For the inductive step, we show that for any active packet $p \in P_{\text{RM-CLIENT}}$ in s, *i.e.*, $p \in s.\mathit{active\text{-}pkts}$, it is the case that either $s.\mathit{now} \leq s.\mathit{trans\text{-}time}(p) + \Delta$ or $s.\mathit{intended}(p) \subseteq s.\mathit{completed}(p)$.

Suppose that $p \in s.\mathit{active\text{-}pkts}$ and consider two cases depending on whether $p \in s'.\mathit{active\text{-}pkts}$. First, consider the case in which $p \notin s'.\mathit{active\text{-}pkts}$. Lemma 3.11 implies that the step from s' to s involves the action $\mathtt{rm\text{-}send}_h(p)$, for $h = \mathit{source}(p)$. Its effects are to set the variable $\mathit{trans\text{-}time}(p)$ to now. It follows that $s.\mathit{now} \leq s.\mathit{trans\text{-}time}(p) + \Delta$. Thus, the invariant assertion is satisfied in s.

Second, consider the case in which $p \in s'.\mathit{active\text{-}pkts}$. Then, the induction hypothesis implies that either $s'.\mathit{now} \leq s'.\mathit{trans\text{-}time}(p) + \Delta$ or $s'.\mathit{intended}(p) \subseteq s'.\mathit{completed}(p)$. We proceed by considering the effects of each of the actions that affect any of the variables present in the invariant assertion:

❒ $\mathtt{crash}_h$, for $h \in H$: the effects of this action are to remove the host h from the intended and completed delivery sets of p. Thus, the induction hypothesis implies that either $s.now \leq s.\mathit{trans\text{-}time}(p) + \Delta$ or $s.\mathit{intended}(p) \subseteq s.\mathit{completed}(p)$.

❒ $\mathtt{rm\text{-}leave}_h$, for $h \in H$: the reasoning for this action is similar to that of the $\mathtt{crash}_h$ action.

❒ $\mathtt{rm\text{-}send}_h(p)$, for $h = \mathit{source}(p)$: since $p \in s'.\mathit{active\text{-}pkts}$ it follows that p has been sent prior to state s' within α. Thus, Lemma 3.2 implies that the $\mathtt{rm\text{-}send}_h(p)$ action is not enabled in s'.

❒ $\mathtt{rm\text{-}recv}_h(p)$, for $h \in H$: we consider two cases depending on whether $s'.\mathit{expected}(h, \mathit{source}(p))$ is empty. First, if $s'.\mathit{expected}(h, \mathit{source}(p)) = \emptyset$, the precondition of $\mathtt{rm\text{-}recv}_h(p)$ implies that $s'.now \leq s'.\mathit{trans\text{-}time}(p) + \Delta$. Since the $\mathtt{rm\text{-}recv}_h(p)$ action affects neither the now nor the $\mathit{trans\text{-}time}(p)$ variables, it follows that $s.now \leq s.\mathit{trans\text{-}time}(p)+\Delta$. Thus, the invariant assertion is satisfied in s. Second, if $s'.\mathit{expected}(h, \mathit{source}(p)) \neq \emptyset$, the precondition of $\mathtt{rm\text{-}recv}_h(p)$ implies that $\mathit{id}(p) \in s'.\mathit{expected}(h, \mathit{source}(p))$. The effects of $\mathtt{rm\text{-}recv}_h(p)$ are to add the element $\mathit{id}(p)$ to the set $\mathit{delivered}(h, \mathit{source}(p))$. Thus, the induction hypothesis implies that either $s.now \leq s.\mathit{trans\text{-}time}(p) + \Delta$ or $s.\mathit{intended}(p) \subseteq s.\mathit{completed}(p)$.

❒ $\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$: the effects of the time-passage action are to allow t time units to elapse. However, the precondition of the action $\nu(t)$ implies that the invariant assertion is satisfied in s.

❒

3.3.2 Reliability Properties

The $\mathrm{RM}_S(\Delta)$ automaton, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, satisfies the *eventual delivery* and, equivalently, *pairwise eventual delivery*, properties. Eventual delivery is the property that if a host h is a member of the reliable multicast group, becomes aware of a packet p, remains a member of the group thereafter, and p remains active thereafter, then h delivers p since last joining the reliable multicast group. Its pairwise counterpart is the property that if two hosts are members of the reliable multicast group, become aware of the packet p, remain members of the group thereafter, and one of them delivers p since last joining the reliable multicast group, then so does the other. The eventual and pairwise eventual delivery properties are equivalent.

Theorem 3.15 (Eventual Delivery) *Let β be any fair admissible timed trace of $\mathrm{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, containing the transmission of a packet $p \in P_{\text{RM-CLIENT}}$. If $p \in \mathit{active\text{-}pkts}(\beta)$, then p is delivered by each host in the intended delivery set of p within β since each such host last joined the reliable multicast group; that is, $\mathit{intended}(p, \beta) \subseteq \mathit{completed}(p, \beta)$.*

Proof: Let α be any fair admissible timed execution of $\mathrm{RM}_S(\Delta)$, such that $\beta = \mathit{ttrace}(\alpha)$. Suppose that $p \in \mathit{active\text{-}pkts}(\beta)$ and let $h \in \mathit{intended}(p, \beta)$. It suffices to show that $h \in \mathit{completed}(p, \beta)$.

First, we consider the case where h is the source of p. Since $h \in \mathit{intended}(p, \beta)$, Definition 3.2 implies that the last $\mathtt{rm\text{-}join\text{-}ack}_h$ action in β is succeeded by a $\mathit{rm\text{-}send}_h(p')$ action, where $\mathit{source}(p') = \mathit{source}(p)$ and $\mathit{seqno}(p') \leq \mathit{seqno}(p)$. If $\mathit{seqno}(p') = \mathit{seqno}(p)$ and, consequently, $p' = p$, then it is the case that the last $\mathtt{rm\text{-}join\text{-}ack}_h$ action in β is succeeded by a $\mathit{rm\text{-}send}_h(p)$ action. By Definition 3.3, it follows that $h \in \mathit{completed}(p, \beta)$, as needed. If $\mathit{seqno}(p') < \mathit{seqno}(p)$, then Lemma 3.3 implies that the transmission of p in β succeeds the transmission of p' in β. Since the $\mathit{rm\text{-}send}_h(p')$ action succeeds the last $\mathtt{rm\text{-}join\text{-}ack}_h$ action in β, so does the $\mathit{rm\text{-}send}_h(p)$ action. By Definition 3.3, it follows that $h \in \mathit{completed}(p, \beta)$, as needed.

Second, consider the case where h is not the source of p. Since $h \in \mathit{intended}(p, \beta)$, Definition 3.2 implies that the last $\mathtt{rm\text{-}join\text{-}ack}_h$ action in β is succeeded by a $\mathit{rm\text{-}recv}_h(p')$ action, where $\mathit{source}(p') = \mathit{source}(p)$ and $\mathit{seqno}(p') \leq \mathit{seqno}(p)$. If $\mathit{seqno}(p') = \mathit{seqno}(p)$ and, consequently,

$p' = p$, then it is the case that the last $\texttt{rm-join-ack}_h$ action in β is succeeded by a $\mathit{rm\text{-}recv}_h(p)$ action. By Definition 3.3, it follows that $h \in \mathit{completed}(p, \beta)$, as needed.

Now, consider the case where $\mathit{seqno}(p') < \mathit{seqno}(p)$. Let (s'_-, π, s'_+) be the discrete transition in α corresponding to the particular occurrence of the $\mathit{rm\text{-}recv}_h(p')$ action in β and α' be the suffix of α that starts in the post-state s'_+ of (s'_-, π, s'_+). Moreover, let $s_{\alpha'}$ be any state in α'. Since $h \in \mathit{intended}(p, \beta)$, Lemma 3.7 implies that $h \in \mathit{members}(\beta)$. Since α' succeeds the last $\texttt{rm-join-ack}_h$ action in α, Lemma 3.6 implies that $h \in s_{\alpha'}.\mathit{members}$. Since $h \neq \mathit{source}(p)$, it follows that $h \in s_{\alpha'}.\mathit{members} \backslash \{\mathit{source}(p)\}$. The precondition and the effects of the $\mathit{rm\text{-}recv}_h(p')$ action imply that $\mathit{id}(p) \in s'_+.\mathit{expected}(h, \mathit{source}(p))$. Moreover, Lemma 3.14 implies that $\mathit{id}(p) \in s_{\alpha'}.\mathit{expected}(h, \mathit{source}(p))$.

Moreover, let (s''_-, π, s''_+) be the discrete transition in α corresponding to the occurrence of the $\mathit{rm\text{-}send}_{h'}(p)$ action in β, for $h' = \mathit{source}(p)$, and α'' be the suffix of α that starts in the post-state s''_+ of (s''_-, π, s''_+). Moreover, let $s_{\alpha''}$ be any state in α''. Lemma 3.12 implies that $p \in s''_+.\mathit{sent\text{-}pkts}$ and Lemma 3.13 implies that $p \in s_{\alpha''}.\mathit{sent\text{-}pkts}$.

Now, let α^* be any timed execution fragment that is a common suffix of α' and α'' and let s^* be any state in α^*. Since $h \in s_{\alpha'}.\mathit{members} \backslash \{\mathit{source}(p)\}$, $p \in s_{\alpha''}.\mathit{sent\text{-}pkts}$, and $\mathit{id}(p) \in s_{\alpha'}.\mathit{expected}(h, \mathit{source}(p))$, it is the case that $h \in s^*.\mathit{members} \backslash \{\mathit{source}(p)\}$, $p \in s^*.\mathit{sent\text{-}pkts}$, and $\mathit{id}(p) \in s^*.\mathit{expected}(h, \mathit{source}(p))$. Thus, the $\mathit{rm\text{-}recv}_h(p)$ action is enabled in s^*; that is, the $\mathit{rm\text{-}recv}_h(p)$ action is enabled in any state in α^*.

Since α^* is a suffix of α and α is an admissible timed execution of $\mathrm{RM}_S(\Delta)$, it is the case that α^* is infinite. Since the $\mathit{rm\text{-}recv}_h(p)$ action is enabled in any state of α^*, the $\mathit{rm\text{-}recv}_h(p)$ action is enabled infinitely often in α^*. Since α is fair, the $\texttt{rm-recv}_h(p)$ action occurs in α^*. Thus, the $\texttt{rm-recv}_h(p)$ action succeeds the last $\texttt{rm-join-ack}_h$ action in α. By Definition 3.3, it follows that $h \in \mathit{completed}(p, \beta)$, as needed. ❒

The following theorem defines the *pairwise eventual delivery* property of $\mathrm{RM}_S(\Delta)$. It states that if two hosts are members of the reliable multicast group, become aware of the packet p, remain members of the group thereafter, and one of them delivers p, then so does the other. The pairwise eventual delivery is equivalent to the eventual delivery property defined in Theorem 3.15.

Corollary 3.16 (Pairwise Eventual Delivery) *Let β be any fair admissible timed trace of the $\mathrm{RM}_S(\Delta)$ automaton, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, that contains the transmission of a packet $p \in P_{\text{RM-CLIENT}}$ and the hosts $h, h' \in H, h \neq h'$ be any two distinct hosts in the intended delivery set of p within β. Then, if h delivers p within β, then so does h'.*

Proof: Since h is in the intended delivery set of p within β and it delivers p within β, it follows that p is active within β; that is, $p \in \mathit{active\text{-}pkts}(\beta)$. Since h' is in the intended delivery set of p within β, Theorem 3.15 implies that h' delivers p within β. ❒

The following theorem defines the notion of *time-bounded delivery*; that is, the property that any packet that remains active for at least $\Delta \in \mathbb{R}^{\geq 0}$ time units past its transmission is delivered within these Δ time units to all hosts that become aware of it within these Δ time units.

Theorem 3.17 (Time-Bounded Delivery) *Let β be any admissible timed trace of $\mathrm{RM}(\Delta) \times$ RMCLIENTS, for any $\Delta \in \mathbb{R}^{\geq 0}$, that contains the transmission of a packet $p \in P_{\text{RM-CLIENT}}$. Let β' be the finite prefix of β ending with the transmission of p; that is, the last action contained in β' is the action $\texttt{rm-send}_h(p)$, for $h \in H, h = \mathit{source}(p)$. Let β'' be any finite prefix of β, such that $\beta' \leq \beta'' \leq \beta$ and $t' + \Delta < t''$, with $t', t'' \in \mathbb{R}^{\geq 0}$ being the time of occurrence of the last actions of β' and β'', respectively. Suppose that the host h' is in the intended delivery set of p within β''*

and that the packet p is active within β''. Then, the host h delivers the packet p within β''; that is, $h' \in completed(p, \beta'')$.

Proof: Let α be any admissible execution of $\mathrm{RM}(\Delta) \times$ RMCLIENTS such that $\beta = ttrace(\alpha)$. Moreover, let α' and α'' be finite prefixes of α such that $\alpha' \leq \alpha'' \leq \alpha$, $\beta' = ttrace(\alpha')$, $\beta'' = ttrace(\alpha'')$, and the last actions in α' and α'' are the last actions in β' and β'', respectively. Finally, let s' and s'' be the last states of α' and α'', respectively.

Since $t' + \Delta < t''$, it follows that $s''.trans\text{-}time(p) + \Delta < s''.now$. Since $p \in active\text{-}pkts(\beta'')$, Lemma 3.10 implies that $p \in s''.active\text{-}pkts$. Since $p \in s''.active\text{-}pkts$ and $s''.trans\text{-}time(p) + \Delta < s''.now$, Invariant 3.3 implies that $s''.intended(p) \subseteq s''.completed(p)$. Lemmas 3.8 and 3.9, imply that $intended(p, \beta'') \subseteq completed(p, \beta'')$. Finally, since $h' \in intended(p, \beta'')$, it follows that $h' \in completed(p, \beta'')$; that is, the host h' delivers the packet p within β''. ❒

Chapter 4

Scalable Reliable Multicast

In this chapter, we present a formal model of the Scalable Reliable Multicast (SRM) protocol of Floyd *et al.* [13]. Our model precisely specifies the behavior of the initial version of SRM presented in Ref. 13 — subsequent research on the SRM protocol resulted in versions of SRM involving adaptive and local recovery schemes [21, 22]. We begin the chapter by giving a brief description of SRM. We continue by specifying the behavior of SRM in terms of timed I/O automata. We then prove that, under certain assumptions, our formal specification of SRM is a correct implementation of the reliable multicast service specification of Chapter 3. We conclude by proving several performance claims about the protocol.

4.1 Overview of the SRM Protocol

In this section, we give a brief overview of the SRM protocol. SRM is an application layer reliable multicast protocol that was initially designed for a distributed white-board application. The protocol is implemented at the application layer, using the IP multicast service as a best-effort communication primitive. SRM uses multicast NACKs to alert the group of losses and suppression to reduce duplicate traffic. The SRM protocol consists of two distinct functional components: i) *packet loss recovery*, and ii) *session message exchange*. We proceed by describing each of these components. The physical system and the data naming scheme are as described in Sections 3.1.1 and 3.1.2.

4.1.1 Packet Loss Recovery

SRM's packet loss recovery scheme is receiver-based. Receivers detect packet losses by detecting sequence number gaps in the stream of packets received from each source. Subsequently, the recovery of packets proceeds in asynchronous rounds. A round involves the transmission of a retransmission request and the retransmission of a packet by either the source or any host that has successfully received the given packet. A recovery round may fail to recover a packet due to additional losses. Thus, several recovery rounds may be required to recover each packet. We proceed by describing in more detail SRM's recovery process.

Upon the detection of a packet loss, a receiver schedules a *repair request* — a retransmission request for the missing packet. This repair request is scheduled for some carefully selected point in time in the future using a request timeout timer. If a request for the packet is overheard prior to the expiration of the request timeout timer, the request timeout timer is reset by performing an exponential backoff. If the packet is received prior to the expiration of the request timeout timer, the scheduled request is canceled. Upon the expiration of the request timeout timer, a request for the particular packet is multicast to the reliable multicast group and a new request for the given packet is rescheduled by exponentially backing-off the request timeout timer; thus, SRM gives a chance for the prior request to result in the recovery of the packet. Thus, request is rescheduled due to either the reception of a request for the given packet or the request's transmission. Once a request is rescheduled, it observes a *back-off abstinence period*; a period during which the request is not backed-off upon the reception of other requests. The back-off abstinence period prevents requests from being backed-off multiple times by requests pertaining to the same recovery round.

Using the above scheme, repair requests are scheduled in rounds; that is, all hosts that detect a loss schedule and may eventually transmit a repair request. If duplicate repair requests are transmitted during each round of requests, then the scheduled requests at the hosts whose requests were suppressed, would exponentially back off their requests multiple times. SRM uses a heuristic to limit the number of times request timeout timers get backed off due to repair requests belonging to the same round. When it receives a request for a packet for which it has recently backed off the request timeout timer, it refrains from backing off the request timer again. Thus, presuming that requests belonging to the same round are received not too far apart in time, SRM backs off the request timeout timers only once per request round.

If a member of the reliable multicast group receives a request for a packet that it has previously either sent or received, it schedules a *repair reply* — a retransmission of the requested packet. This repair reply is scheduled for some carefully selected point in time in the future using a repair timeout timer. When the reply timeout timer expires, the requested packet is multicast to the reliable multicast group. If a repair reply for the packet is overheard prior to the expiration of the reply timeout timer, the repair reply is canceled.

Once a host either sends or receives a reply for a given packet, it observes a *reply abstinence period*; a period during which the host refrains from scheduling a reply for the same packet. Requests that are received during a packet's reply abstinence period are discarded. The reply abstinence period prevents multiple requests pertaining to a given recovery round from generating multiple replies for each packet.

SRM limits the number of packets sent to the multicast group while repairing a loss by suppressing duplicate repair requests and replies. In particular, SRM employs *deterministic* and *probabilistic suppression* techniques. In the case of repair requests, deterministic suppression is achieved by having hosts that are closer to the source of the missing packet schedule their requests sooner. A requestor of a missing packet sets its repair request timer proportionately to its distance estimate to the source of the missing packet. Thus, hosts that are closer to the source of the missing packet suppress their descendants in the underlying IP multicast tree.

Probabilistic suppression is achieved by spreading out the repair requests of the hosts that are equidistant to the source within an interval whose size is again proportional to the requestors' distance estimates to the source of the missing packet. In particular, a requestor of a missing packet sets its repair request timer to a point in time that is uniformly chosen within an interval. This interval's left endpoint is dictated by the deterministic suppression scheme and its right endpoint is, once again, proportional to the requestor's distance estimate to the source of the missing packet. This scheme probabilistically limits the number of requests multicast by equidistant requestors by allowing them to suppress each other.

For example, let h denote a host that has detected the loss of a packet p from the source s. h schedules its request for the packet p for a point in time in the future that is uniformly chosen within the interval $2^b[C_1\hat{d}_{hs}, (C_1 + C_2)\hat{d}_{hs}]$, where b is the request's backoff (initially equal to 0), C_1, C_2 are parameters of the deterministic and probabilistic suppression schemes pertaining to requests, and $\hat{d}_{hs}$ is h's distance estimate to the source s of the packet p.

Repair replies are scheduled in the same fashion as requests with the exception that the interval endpoints are set proportionately to the distance estimates between the replier and requestor hosts, rather than the distance estimates between the requestor and source hosts.

For example, let h' denote a host that is in the process of scheduling a reply to h's request for the packet p. h' schedules its reply to h's request for the packet p for a point in time in the future that is uniformly chosen within the interval $[D_1\hat{d}_{h'h}, (D_1 + D_2)\hat{d}_{h'h}]$, where D_1, D_2 are parameters of the deterministic and probabilistic suppression schemes pertaining to replies and $\hat{d}_{h'h}$ is h''s distance estimate to the requestor h of the packet p.

4.1.2 Session Messages

The reliable multicast group members periodically exchange session messages. These messages carry transmission state and timing information that allow the prompt detection of packet losses and the calculation of inter-host distance estimates; within SRM, inter-host distances are quantified by the one-way transmission latency between hosts. For simplicity, we assume that hosts transmit session messages with a fixed period. In practice however, so as to limit the overhead associated with the exchange of session messages, the frequency of session message transmission is reduced as the size of the reliable multicast group grows.

Receivers detect packet losses by detecting sequence number gaps in the stream of packets received from each source. However, this approach presumes either that later packets within the sequence of transmitted packets are received, or that receivers get informed of the transmission progress of each source through a separate service. Unfortunately, relying solely on the reception of later packets may result in long recovery latencies. This is evident when the total number of packets within a sequence is unknown *a priori* and either long transmission pauses, or long loss bursts are considered. Session messages mitigate this problem by allowing reliable multicast group members to exchange transmission progress state, in terms of ADU sequence numbers that they have observed with respect to each source. Discrepancies in the observed transmission progress for each source by each host reveal whether and which packets a particular host is missing.

In addition to contributing to packet loss detection, session messages are used to calculate inter-host distance estimates. Hosts estimate the one-way transmission latencies between them by exchanging timing information through their session messages. For the purposes of illustration, we demonstrate how a host h calculates its distance estimate to a host h'. This calculation is initiated when the host h transmits a session message, p. This session message includes a field containing its transmission time t_s. Let t'_r denote the time the host h' receives p. Upon receiving p, h' records the times at which p was transmitted and received, *i.e.*, it records a tuple of the form $\langle t_s, t'_r \rangle$. Subsequently, the host h' includes the tuple $\langle t_s, t'_d \rangle$ within its next session message, p', where t'_d corresponds to the time elapsed since the host h' received p and the time h' transmits p'. Finally, letting t_r denote the point in time that h receives p', h estimates its distance $\hat{d}_{hh'}$ to h' as $(t_r - t'_d - t_s)/2$ time units.

Although the above scheme for calculating inter-host transmission latencies is simple, it presumes that inter-host transmission latencies are symmetric — the one way inter-host transmission latency is estimated as half the *round-trip-time* (RTT) between hosts. Another drawback of this scheme is the dependence of its accuracy on the frequency of session message transmission. The frequency of calculating inter-host distance estimates is dictated by the frequency of session message transmission. Thus, if the frequency of session message transmission were adjusted based on the size of the reliable multicast group, then as the group would increase in size the accuracy of the inter-host distance estimates would drop.

Figure 4.1 SRM Parameters

$C_1, C_2, C_3 \in \mathbb{R}^{\geq 0}$	Request scheduling parameters.
$D_1, D_2, D_3 \in \mathbb{R}^{\geq 0}$	Reply scheduling parameters.
`DFLT-DIST` $\in \mathbb{R}^{\geq 0}$	Default inter-host distance estimate.
`SESS-PERIOD` $\in \mathbb{R}^{\geq 0}$	Period of session packet transmission.

4.2 Architecture of the SRM Protocol

In this section, we give an overview of our model of the SRM protocol and its environment. As in Chapter 3, the physical system is comprised of a set of hosts that communicate through an underlying network. We encapsulate the behavior of the underlying network by a single IP component. This component provides the best-effort IP multicast service which is the communication primitive of the SRM protocol. Resident on each host are two processes: a client and a reliable multicast process. The client process represents an application that uses the reliable multicast service. The reliable multicast processes at each host and the underlying IP multicast service collectively provide the reliable multicast service to the client processes.

Without loss of generality, we assume that there exists only a single reliable multicast address/group. Since we assume that there is a single client at each source and a single reliable multicast address/group, we do not distinguish among the host, reliable multicast process, and client process when considering membership to the reliable multicast group. In fact, for simplicity we usually associate the reliable multicast group membership with the host itself, rather that with the client or the reliable multicast processes.

We model the reliable multicast process running on each host as three interacting components, each with distinct functionalities. The first component, which we henceforth refer to as the *membership component*, manages the host's reliable multicast group membership. In particular, it handles the join and leave requests of the client process and issues join and leave requests to the underlying IP multicast service. The second component, which we henceforth refer to as the *IP buffer component*, buffers all packets received from and to be transmitted using the underlying IP multicast service. Finally, the third component, which we henceforth refer to as the *recovery component*, incorporates all the functionality pertaining to the transmission, recovery, and delivery of packets by the reliable multicast service. We proceed by briefly describing the functionality of each of these components. Figure 4.1 lists the parameters of the reliable multicast process. Each of these parameters is appropriately introduced within the upcoming descriptions of the components of the reliable multicast process.

4.2.1 Membership Component

The membership component of the reliable multicast process manages the membership of the host to the reliable multicast group. In particular, it handles the join and leave requests of the client and manages the membership of the host to the underlying IP multicast service.

The client initiates the process of joining the reliable multicast group by issuing a join request to the membership component. In turn, prior to acknowledging this request, the membership component issues a join request to the underlying IP multicast service. The membership component concludes that it has successfully joined the IP multicast group when its request to join the IP multicast group is acknowledged. Once the membership component has established its IP multicast group membership, it acknowledges the client's join request. The client is considered a member of the reliable multicast group from the point in time its join request is acknowledged by the membership component.

While the client is in the process of joining the reliable multicast group, the membership component discards additional client join requests; they are considered superfluous. Client join requests are also discarded while the client is either already a member of the reliable multicast group, or in the process of leaving the reliable multicast group.

The client initiates the process of leaving the reliable multicast group by issuing a leave request to the membership component. Upon issuing a leave request, the client relinquishes its right of further receiving packets from the reliable multicast service and ceases to be a member of the reliable multicast group. Subsequently, prior to acknowledging the client's leave request, the membership component issues a leave request to the underlying IP multicast service. Upon the reception of a leave acknowledgment from the IP multicast service, the membership component acknowledges the client's leave request. Once a host leaves and until it rejoins the reliable multicast group, the membership component simply discards any join and leave acknowledgments it receives from the IP multicast service.

While the client is in the process of leaving the reliable multicast group, the membership component discards additional client leave requests; they are considered as being superfluous. Client join requests are also discarded while the client is in the process of leaving the reliable multicast group. Client leave requests are also discarded while the client is idle with respect to the reliable multicast group. Finally, leave requests overrule join requests in the sense that if the membership component receives a leave request while in the process of joining, then it aborts the process of joining and initiates the process of leaving the reliable multicast group.

4.2.2 IP Buffer Component

The IP buffer component of the reliable multicast process serves as a buffer between the reliable multicast process and the underlying IP multicast service. In particular, the IP buffer component is responsible for: i) buffering the packets it receives from the underlying IP multicast service and handing them off to be processed by the recovery and/or reporting components, and ii) buffering and transmitting all the packets that are bound for transmission using the IP multicast service.

Packets received from the underlying IP multicast service are discarded if the host is not a member of the reliable multicast group. When a host is a member of the reliable multicast group, any packet received is buffered and, subsequently, handed off for processing to the recovery and the reporting components. The IP buffer component also buffers the packets generated by the recovery and reporting components. It subsequently multicasts each such packet using the underlying IP multicast service.

4.2.3 Recovery Component

The recovery component incorporates all the functionality pertaining to the transmission, recovery, and delivery of packets by the reliable multicast process. While the host is a member of the reliable multicast group, the recovery component processes all the packets either sent by the client or received by the IP buffer component from the underlying IP multicast service. The recovery component: i) tracks the transmission progress of each source by maintaining per-source transmission state that records this progress and archiving all ADUs that are either sent or received by the client, ii) , carries out the exchange of session packets among the members of the reliable multicast members by processing and periodically transmitting session packets, iii) schedules retransmission requests for missing packets, and iv) schedules retransmissions of requested packets.

We proceed by briefly describing these responsibilities. Recall that there are four different types of packets: original transmissions (DATA packets), repair requests (RQST packets), repair replies (REPL packets), and session messages (SESS packets).

Maintaining Transmission State

The recovery component tracks the host's reliable multicast group membership by observing the join acknowledgments sent by the membership component to the client and the client's leave requests. The recovery component operates only while the host is a member of the reliable multicast group; upon leaving the reliable multicast group, the transmission state pertaining to all sources is flushed.

The recovery component tracks the transmission of each source by maintaining per-source transmission state. For each source s, this state involves two sequence numbers corresponding to ADUs transmitted by s. The first such sequence number for s, henceforth denoted the *foremost* sequence number of s, is the sequence number of the ADU contained in the first DATA packet from s to be processed by the recovery component at the particular host since the host joined the reliable multicast group. The second sequence number for s, henceforth denoted the *hindmost* sequence number of s, is the maximum sequence number of an ADU of s to have been observed by the recovery component since the host joined the reliable multicast group and set its foremost sequence number of s. All packets pertaining to earlier ADUs from s than the foremost packet of s are considered *improper* and are discarded by the recovery component. All other packets pertaining to ADUs from s are considered *proper* and are processed by the recovery component. Thus, the reliability guarantees provided by the reliable multicast service with respect to a particular source apply only to proper packets.

Once a host becomes a member of the reliable multicast group, the recovery component begins processing the packets received either from the client or the underlying IP multicast group. Upon processing the first DATA packet from a source s, the recovery component initializes the foremost and hindmost sequence numbers of s to the sequence number of the ADU contained in this DATA packet. Thereafter, the recovery component updates the hindmost sequence number of s to reflect the observed transmission progress of s. Any DATA, RQST, and REPL packet pertaining to s and any session packet may advance the hindmost sequence number of s. The recovery component is responsible for updating the hindmost sequence number of s based on the transmission state information contained within any DATA, RQST, and REPL packets pertaining to s. The reporting component informs the recovery component of any progress in the transmission of the source s reported by any session packet.

Session Packet Exchange

The recovery component periodically generates session packets and passes them to the IP buffer component. The IP buffer component is responsible for buffering and subsequently transmitting these session packets using the underlying IP multicast service. The parameter `SESS-PERIOD` $\in \mathbb{R}^{\geq 0}$ specifies the period with which hosts generate and transmit session packets. In our treatment of SRM, we presume that the transmission period of session packets is constant.

We now describe the transmission state and timing information contained in a session packet p of a host h. First, for each source s that h is aware of, p reports the maximum sequence number observed by h to have been transmitted by s. If the session packet p reports a maximum sequence number for the source s, then we say that p is *state reporting* for s. Second, for each host h' from which h has received a session packet, p contains a tuple consisting of: i) the transmission time of the latest session packet of h' to have been received by h, and ii) the elapsed time between the reception of h''s session packet and the transmission of p by h. If p contains such a tuple for a host h', then we say that p is *distance reporting* for h'. Finally, p contains a field reporting its own transmission time.

Scheduling Requests

The recovery component maintains a set of *scheduled* and a set of *pending* requests. The set of scheduled requests identifies the packets for which a request has been scheduled and is awaiting transmission. The set of pending requests identifies the packets for which a request has recently been either sent or received and for which a retransmission is being awaited.

A host h schedules a request for the packet p by adding an element to its set of scheduled requests. This element identifies the packet p to be requested, specifies the request's transmission timeout (the time at which the request is scheduled for transmission), and records the number of times a request for the given packet has been scheduled. The transmission timeout of p's request is initialized to a point in time in the future that is uniformly chosen within the interval $now + [C_1\hat{d}_{hs}, (C_1 + C_2)\hat{d}_{hs}]$, where *now* refers to the then current point in time, C_1, C_2 are request scheduling parameters, and $\hat{d}_{hs}$ is h's distance estimate to the source s of p. If a request for the packet p is received while a scheduled request for p is awaiting transmission, then the request for p that is already scheduled is exponentially backed off. This is achieved by: i) resetting its transmission timeout to a point in time in the future that is uniformly chosen within the interval $now + 2^k[C_1\hat{d}_{hs}, (C_1 + C_2)\hat{d}_{hs}]$, where k is the number of requests that have already been scheduled for p, and ii) incrementing the number of requests, k, that have already been scheduled. Upon the expiration of the transmission timeout of a scheduled request for the packet p, the recovery component composes a request packet for the packet p and passes it on to the IP buffer component. The IP buffer component is responsible for buffering and subsequently transmitting this request packet using the underlying IP multicast service. Moreover, the recovery component schedules a new request for the packet as if a request had been received; that is, it sets the request timeout timer by exponentially backing off the previously set request timeout timer for p.

Once a request for a packet p is either sent or received, a request for p becomes pending. This pending request identifies the packet p and includes a *back-off abstinence* timeout. This timeout specifies the point in time in the future before which the recovery component refrains from backing-off its scheduled request for the packet p. All requests for p received prior to the expiration of the back-off abstinence timeout for p are considered to pertain to the prior request round and are discarded. The back-off abstinence timeout for p is set to a point in time that is $2^k C_3 \hat{d}_{hs}$ time units in the future, where k is the back-off used to schedule the next (current) request and $C_3 \in \mathbb{R}^{\geq 0}$ is the back-off abstinence parameter of our implementation. Back-off abstinence prevents requests from being backed off by requests pertaining to previous recovery rounds.

Our modeling of back-off abstinence departs slightly from the schemes proposed in the SRM protocol. In Ref. 12, 13, two schemes are proposed for ensuring that requests are backed off only one time per recovery round. The first scheme involves a back-off timeout as described above. However, the timeout is set to half the time to the next request. Our use of a parameter for specifying how long to abstain from backing off allows more tuning freedom. Moreover, setting the back-off timeout to half the time to the next request allows for the abstinence interval to overlap the request interval within which the next request was scheduled. This seems to go against the intention of the abstinence period. Requests received within the request interval, within which the next request was scheduled, should be considered to be requests of the next round and, thus, result in the next request being backed off. The second scheme annotates requests with their recovery round and backs off requests only upon receiving a request pertaining to the same round.

Scheduling Replies

The recovery component maintains a set of *scheduled* and a set of *pending* replies. The set of scheduled replies identifies the packets for which a reply has been scheduled and is awaiting transmission. The set of pending replies identifies the packets for which a reply has recently been either received or transmitted.

A host h schedules a reply to a request for a packet p by the host h' by adding an element to its set of scheduled replies. This element identifies the packet p and specifies the reply's transmission timeout (the time at which the reply is scheduled for transmission). The reply's transmission timeout is initialized to a point in time in the future that is uniformly chosen within the interval $now + [D_1 \hat{d}_{hh'}, (D_1 + D_2)\hat{d}_{hh'}]$, where *now* refers to the then current point in time, D_1, D_2 are reply scheduling parameters, and $\hat{d}_{hh'}$ is h's distance estimate to h'. Upon the expiration of the timeout of a scheduled reply for the packet p, a reply packet for p is generated and passed to the IP buffer component. The IP buffer component is responsible for buffering and subsequently transmitting this reply packet using the underlying IP multicast service. If a reply for p is received by the host h while a scheduled reply for p is awaiting transmission, then the scheduled reply at h is canceled.

Once a reply for a packet p is either generated or received by h, a reply for p becomes pending. This pending reply identifies the packet p and includes a *reply abstinence* timeout. This timeout specifies the point in time in the future before which the recovery component refrains from scheduling another reply for p. The timeout is set to $D_3 \hat{d}_{hh'}$ time units in the future, where $D_3 \in \mathbb{R}^{\geq 0}$ is the reply abstinence parameter of the SRM protocol. Replier abstinence prevents multiple requests pertaining to a given recovery round of a particular packet from generating multiple replies.

Processing Client and IP Multicast Packets

While the host is a member of the reliable multicast group, the recovery component processes all packets transmitted by the client and all DATA, RQST, and REPL packets received from the IP multicast service. The client packets are archived and handed off to the IP buffer component as DATA packets. The IP buffer component is responsible for buffering and subsequently transmitting each such DATA packet using the underlying IP multicast service. We proceed by briefly describing how the recovery component processes DATA, RQST, and REPL packets. Recall that the recovery component processes only proper packets. All improper packets are discarded.

A DATA packet p is processed as follows. Let s and i denote the source and the sequence number of the ADU contained in p. If p is the first DATA packet from s to be processed, then i is the foremost sequence number of s. In this case, the recovery component sets the foremost and hindmost sequence numbers of s to i. If p is a proper packet, then the ADU contained in p is archived, buffered, and subsequently delivered to the client. If i is larger than the hindmost sequence number of s, then the hindmost sequence number of s is set to i and any intervening ADUs are identified as missing. Finally, any scheduled requests and any scheduled replies for p are canceled.

A RQST packet p is processed as follows. Let s and i denote the source and the sequence number of the packet requested by p. If the request is for a proper packet that is archived by the recovery component, then the recovery component attempts to schedule a reply. A reply is scheduled only if a reply for the requested packet is neither already scheduled, nor pending. Finally, if i is larger than the hindmost sequence number of s, then the hindmost sequence number of s is set to i and any intervening ADUs are identified as missing. If the request is for a proper packet that is not archived by the recovery component and for which no request has already been scheduled, then the recovery component schedules a second round request as if it were backing off a prior request scheduled with a back-off of 0.

A REPL packet p is processed as follows. Let s and i denote the source and the sequence number of the ADU contained in p. If this ADU is proper, then a new pending reply is generated for the given ADU, the ADU is archived, buffered, and subsequently delivered to the client. If i is larger than the hindmost sequence number of s, then the hindmost sequence number of s is set to i and any intervening ADUs are identified as missing. Finally, any scheduled requests and replies for p are canceled.

A SESS packet p is processed as follows. If p corresponds to either the first or the most recently transmitted session packet from h' to be received by h, then h records both the transmission and the reception time of p; that is, the time that p was transmitted by h' and the time that it was received by h. Moreover, for each source s for which p is state reporting, the host h updates its transmission state. The transmission state for s is updated only when the reported sequence number for s is greater than that observed up to that point in time by h to have been transmitted by s. In such cases, the trailing packets are identified as missing.

Finally, if the packet p is distance reporting for h, the recovery component estimates the distance from h to h' as half the RTT from h to h'. Letting $\langle t_s, t'_d \rangle$ denote the distance report for h and t_r denote p's reception time by h, the recovery component estimates its distance from h to h' as $(t_r - t'_d - t_s)/2$ time units. Distance estimates are ordered based on the transmission time of the session packets that initiate their calculation; that is, distance estimates whose calculations are initiated by more recent session packets are considered more up-to-date. The recovery component of h updates its distance estimates only when more up-to-date distance estimates are calculated.

After processing a packet, the recovery component schedules a request for any packets that it has identified as missing.

In our treatment of SRM, the recovery component archives all packets either sent by or delivered to the client. Thus, we assume that the reliable multicast process has infinite memory. In future work, we intend to relax this assumption, in particular when the reliable multicast service implementation is capable of timely packet delivery; in this case, the recovery component need archive only the packets that have been sent no earlier than an amount of time in the past equal to the delivery bound guarantee.

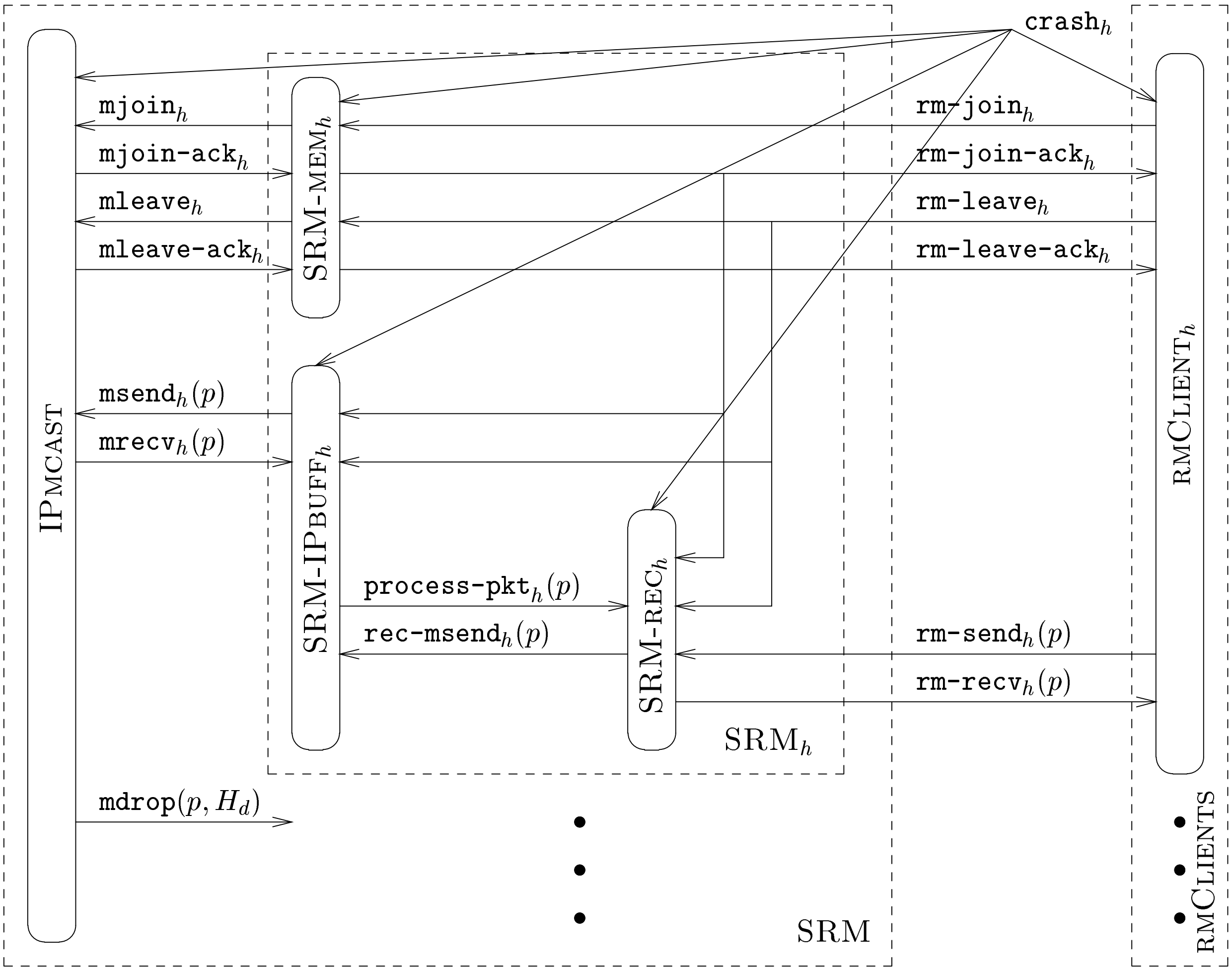
4.3 Formal Model of the SRM Protocol

Each of the components of the reliable multicast process at each host h is modeled by a timed I/O automaton. In particular, the membership component is modeled by SRM-MEM$_h$, the IP buffer component is modeled by SRM-IPBUFF$_h$, and the recovery component is modeled by SRM-REC$_h$. The reliable multicast process SRM$_h$, for each host h, is the composition SRM-MEM$_h$ $\times$ SRM-IPBUFF$_h$ $\times$ SRM-REC$_h$. Figure 4.2 depicts how the components of SRM interact among themselves and with their environment. The client at each host is modeled by the RM-CLIENT$_h$ timed I/O automaton of Chapter 3. The underlying best-effort IP multicast service is modeled by the IPMCAST timed I/O automaton. Prior to specifying each of the component automata, we present several definitions that are used in their specifications.

4.3.1 Preliminary Definitions

Figure 4.3 contains a list of set definitions that specify the format of the various types of packets used throughout the following sections. The set $P_{\text{RM-CLIENT}}$ represents the set of packets that may be transmitted by the client processes using the reliable multicast service. As defined in Chapter 3, for any packet $p \in P_{\text{RM-CLIENT}}$ the operations $source(p)$, $seqno(p)$, and $data(p)$ extract the source,

Figure 4.2 Interface of all components involved in the reliable multicast service.



sequence number, and data segment corresponding to the packet p. For shorthand, we use the operation $id(p)$ to extract the identifier of p; that is, its source and sequence number pair.

The set P_{SRM} is comprised of all packets whose format is that used by the reliable multicast process. The format of each packet $p \in P_{\mathrm{SRM}}$ depends on its type. The type of the packet p, $type(p)$, is one of the following: DATA, RQST, REPL, and SESS. The type of p denotes whether the packet is an original transmission, a repair request, a repair reply, or a session packet, respectively. Depending on its type, the packet p supports a different set of operations.

When the packet p is an original transmission, that is, when $type(p) = \texttt{DATA}$, p supports the operations $sender(p)$, $source(p)$, $seqno(p)$, $data(p)$, and $strip(p)$. These operations extract the sender, source, sequence number, data segment, and ADU corresponding to p. In the case of original transmissions, it is the case that $sender(p) = source(p)$. When p is a repair request, that is, when $type(p) = \texttt{RQST}$, p supports the operations $sender(p)$, $source(p)$, and $seqno(p)$. These operations extract the sender, source, and sequence number corresponding to the packet p. When p is a repair reply, that is, when $type(p) = \texttt{REPL}$, p supports the operations $sender(p)$, $requestor(p)$, $source(p)$, $seqno(p)$, $data(p)$, and $strip(p)$. These operations extract the sender, requestor, source, sequence number, data segment, and ADU packet corresponding to p. For DATA, RQST, and REPL packets, we also use the operation $id(p)$ to extract the identifier of p; that is, its source and sequence number pair.

When the packet p is a session packet, that is, when $type(p) = \texttt{SESS}$, p supports the operations

Figure 4.3 SRM Packet Definitions

$P_{\text{RM-CLIENT}}$ = Set of packets such that $\forall\, p \in P_{\text{RM-CLIENT}}$:
  $source(p) \in H$
  $seqno(p) \in \mathbb{N}$
  $data(p) \in \{0,1\}^*$
  $id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
  $suffix(p) = \{\langle s,i\rangle \in H \times \mathbb{N} \mid source(p) = s \wedge seqno(p) \leq i\}$

$P_{\text{RM-CLIENT}}[h] = \{p \in P_{\text{RM-CLIENT}} \mid source(p) = h\}$

P_{SRM} = Set of packets such that $\forall\, p \in P_{\text{SRM}}$:
  $type(p) \in \{\texttt{DATA}, \texttt{RQST}, \texttt{REPL}, \texttt{SESS}\}$
    `DATA` :
      $sender(p) \in H$
      $source(p) \in H$
      $seqno(p) \in \mathbb{N}$
      $data(p) \in \{0,1\}^*$
      $strip(p) \in P_{\text{RM-CLIENT}}$
      $id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
    `RQST` :
      $sender(p) \in H$
      $source(p) \in H$
      $seqno(p) \in \mathbb{N}$
      $id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
    `REPL` :
      $sender(p) \in H$
      $requestor(p) \in H$
      $source(p) \in H$
      $seqno(p) \in \mathbb{N}$
      $data(p) \in \{0,1\}^*$
      $strip(p) \in P_{\text{RM-CLIENT}}$
      $id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
    `SESS` :
      $sender(p) \in H$
      $time\text{-}sent(p) \in \mathbb{R}^{\geq 0}$
      $dist\text{-}rprt?(p) \subseteq H$
      $dist\text{-}rprt(p,h) \in \{\langle t,t'\rangle \mid t,t' \in \mathbb{R}^{\geq 0}\}$, for all $h \in H$
      $seqno\text{-}rprts(p) \subseteq \{\langle s,i\rangle \mid s \in H, i \in \mathbb{N}\}$

$P_{\text{IPMCAST-CLIENT}}$ = Set of packets such that $\forall\, p \in P_{\text{IPMCAST-CLIENT}}$:
  $source(p) \in H$
  $seqno(p) \in \mathbb{N}$
  $strip(p) \in \{0,1\}^*$

P_{IPMCAST} = Set of packets such that $\forall\, pkt \in P_{\text{IPMCAST}}$:
  $strip(pkt) \in P_{\text{IPMCAST-CLIENT}}$
  $intended(pkt) \subseteq H$
  $completed(pkt) \subseteq H$
  $dropped(pkt) \subseteq H$

$sender(p)$, $time\text{-}sent(p)$, $dist\text{-}rprt?(p)$, $dist\text{-}rprt(p,h)$, and $seqno\text{-}rprts(p)$. The operation $sender(p)$ extracts the sender of the session packet. The operation $time\text{-}sent(p)$ extracts the time the session packet p was sent. The operation $dist\text{-}rprt?(p)$ extracts the set of hosts for which the session packet is distance reporting. The operation $dist\text{-}rprt(p,h)$ extracts the distance report for h within p; that is, $dist\text{-}rprt(p,h)$ corresponds to a tuple comprised of two elements: the time the most recently observed session packet sent by h was received by the sender of p and the time that elapsed between the reception of h's session packet by the sender of p and the transmission of p. The operation $seqno\text{-}rprts(p)$ extracts the state reports included in p; that is, $seqno\text{-}rprts(p)$ corresponds to a set of tuples, each of which is comprised of two elements: the source and the maximum sequence number observed by the sender of p to have been transmitted by this source.

Figure 4.4 contains a list of set definitions used throughout the following sections.

Figure 4.4 SRM Set Definitions

$SRM\text{-}Status = \{\texttt{idle}, \texttt{member}, \texttt{crashed}\}$
$Joining = \{\texttt{join-rqst-pending}, \texttt{join-pending}, \texttt{join-ack-pending}\}$
$Leaving = \{\texttt{leave-rqst-pending}, \texttt{leave-pending}, \texttt{leave-ack-pending}\}$
$SRM\text{-}Mem\text{-}Status = SRM\text{-}Status \cup Joining \cup Leaving$
$Action\text{-}Pending = \{\texttt{join-rqst-pending}, \texttt{join-ack-pending}, \texttt{leave-rqst-pending}, \texttt{leave-ack-pending}\}$

$Pending\text{-}Rqsts = \{\langle s, i, t\rangle \mid s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}\}$
$Scheduled\text{-}Rqsts = \{\langle s, i, t, k\rangle \mid s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}\}$
$Pending\text{-}Repls = \{\langle s, i, t\rangle \mid s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}\}$
$Scheduled\text{-}Repls = \{\langle s, i, t, q\rangle \mid s, q \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}\}$

Figure 4.5 The SRM-MEM$_h$ Automaton — Signature

Parameters:

$h \in H$

Actions:

input
- $\texttt{crash}_h$
- $\texttt{rm-join}_h$
- $\texttt{rm-leave}_h$
- $\texttt{mjoin-ack}_h$
- $\texttt{mleave-ack}_h$

output
- $\texttt{mjoin}_h$
- $\texttt{mleave}_h$
- $\texttt{rm-join-ack}_h$
- $\texttt{rm-leave-ack}_h$

time-passage
- $\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

4.3.2 The Membership Component — SRM-MEM$_h$

The SRM-MEM$_h$ timed I/O automaton specifies the membership component of the reliable multicast process. Figures 4.5 and 4.6 present the signature, the variables, and the discrete transitions of SRM-MEM$_h$.

Variables

The variable *now* $\in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of SRM-MEM$_h$. The variable *status* captures the status of the host h. It evaluates to one of the following: `idle`, `join-rqst-pending`, `join-pending`, `join-ack-pending`, `leave-rqst-pending`, `leave-pending`, `leave-ack-pending`, `member`, and `crashed`.

The value `idle` indicates that the host h is *idle* with respect to the reliable multicast group; that is, it is neither a member, nor in the process of joining or leaving the reliable multicast group. The value `join-rqst-pending` indicates that SRM-MEM$_h$ has received a join request from the client but has yet to issue a join request to the underlying IP multicast service. The value `join-pending` indicates that SRM-MEM$_h$ has issued a join request to the underlying IP multicast service and is awaiting a join acknowledgment. The value `join-ack-pending` indicates that SRM-MEM$_h$ has successfully joined the underlying IP multicast service but has yet to issue a join acknowledgment to the client. The value `member` indicates that the host h is a member of the reliable multicast group. The value `leave-rqst-pending` indicates that SRM-MEM$_h$ has received a leave request from the client but has yet to issue a leave request to the underlying IP multicast service. The value `leave-pending` indicates that SRM-MEM$_h$ has issued a leave request to the underlying IP multicast service and is awaiting a leave acknowledgment. The value `leave-ack-pending` indicates that SRM-MEM$_h$ has successfully left the underlying IP multicast service but has yet to issue a leave acknowledgment to the client. The value `crashed` indicates that the host h has crashed. While the host h has not crashed, we say that it is *operational*. Once the host h crashes, none

Figure 4.6 The SRM-MEM$_h$ Automaton — Variables and Discrete Transitions

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status \in$ *SRM-Mem-Status*, initially $status =$ `idle`

Discrete Transitions:

input `crash`$_h$
eff $status :=$ `crashed`

input `rm-join`$_h$
eff **if** $status =$ `idle` **then**
$status :=$ `join-rqst-pending`

input `rm-leave`$_h$
eff **if** $status \in Joining \cup \{$`member`$\}$ **then**
$status :=$ `leave-rqst-pending`

input `mjoin-ack`$_h$
eff **if** $status \in Joining$ **then**
$status :=$ `join-ack-pending`

input `mleave-ack`$_h$
eff **if** $status \in Leaving$ **then**
$status :=$ `leave-ack-pending`

output `mjoin`$_h$
pre $status =$ `join-rqst-pending`
eff $status :=$ `join-pending`

output `mleave`$_h$
pre $status =$ `leave-rqst-pending`
eff $status :=$ `leave-pending`

output `rm-join-ack`$_h$
pre $status =$ `join-ack-pending`
eff $status :=$ `member`

output `rm-leave-ack`$_h$
pre $status =$ `leave-ack-pending`
eff $status :=$ `idle`

time-passage $\nu(t)$
pre $status \notin$ *Action-Pending*
eff $now := now + t$

of the input actions of SRM-MEM$_h$ affect the state of SRM-MEM$_h$ and none of the internal and output actions of SRM-MEM$_h$, except the time passage action, are enabled.

Input Actions

The input action **`crash`**$_h$ models the crashing of SRM-MEM$_h$. The effects of **`crash`**$_h$ are to set the variable *status* to **`crashed`** so as to record the fact that SRM-MEM$_h$ has crashed.

The input action **`rm-join`**$_h$ models the client's request to join the reliable multicast group. It is effective only when the host h is idle with respect to the reliable multicast group. If the client h is already either a member of, or in the process of joining, the reliable multicast group (that is, $status \in Joining \cup \{$**`member`**$\}$), then the scheduling of **`rm-join`**$_h$ is superfluous. If the client h is already in the process of leaving the reliable multicast group (that is, $status \in Leaving$), then **`rm-join`**$_h$ is ignored so as to allow the ongoing process of leaving the reliable multicast group to complete. When effective, **`rm-join`**$_h$ initiates the process of joining the reliable multicast group by setting the *status* variable to **`join-rqst-pending`**.

The input action **`rm-leave`**$_h$ models the client's request to leave the reliable multicast group. It is effective only when the host h is either a member of, or in the process of joining, the reliable multicast group. If the host h is either already in the process of leaving, or idle with respect to the reliable multicast group, then the **`rm-leave`**$_h$ action is superfluous. When effective, **`rm-leave`**$_h$ initiates the process of leaving the reliable multicast group by setting the *status* variable to **`leave-rqst-pending`**.

The input action **`mjoin-ack`**$_h$ acknowledges that the host h has successfully joined the underlying IP multicast group. It is effective only when the host h is in the process of joining the reliable multicast group; that is, when $status \in Joining$. When effective, **`mjoin-ack`**$_h$ enables the I/O component to acknowledge the client's join request by setting the *status* variable to **`join-ack-pending`**.

The input action **`mleave-ack`**$_h$ acknowledges that the host h has successfully left the underlying IP multicast group. It is effective only when the host h is in the process of leaving the reliable multicast group; that is, when $status \in Leaving$. When effective, **`mleave-ack`**$_h$ sets the *status* variable to `leave-ack-pending`. Thus, it enables the I/O component to acknowledge the client's leave request.

Output Actions

SRM-MEM$_h$ initiates the process of joining of the underlying IP multicast group by scheduling the output action $\mathtt{mjoin}_h$. This action is enabled whenever the client has effectively requested to join the reliable multicast group; that is, when *status* = `join-rqst-pending`. Its effects are to record the fact that SRM-MEM$_h$ has requested to join the IP multicast group; that is, it sets the *status* variable to `join-pending`. Joining the underlying IP multicast group is not always immediate. In order for the IP multicast service to forward packets to the host h, it may have to extend the IP multicast tree to include the host h. The time involved in extending the IP multicast tree to include the host h heavily depends on the location of the host h and the reach of the current IP multicast tree.

SRM-MEM$_h$ initiates the process of leaving of the underlying IP multicast group by scheduling the output action $\mathtt{mleave}_h$. This action is enabled whenever the client has effectively requested to leave the reliable multicast group; that is, *status* = `leave-rqst-pending`. Its effects are to record the fact that SRM-MEM$_h$ has requested to leave the IP multicast group; that is, it sets the *status* variable to `leave-pending`.

SRM-MEM$_h$ acknowledges the client's request to join the reliable multicast group by scheduling the $\mathtt{rm\text{-}join\text{-}ack}_h$ output action. This action is enabled whenever the join acknowledgment is pending; that is, *status* = `join-ack-pending`. Time is not allowed to elapse while a join acknowledgment is pending. Thus, a join acknowledgement is sent immediately after SRM-MEM$_h$ determines that it has successfully joined the IP multicast group.

SRM-MEM$_h$ acknowledges the client's request to leave the reliable multicast group by scheduling the $\mathtt{rm\text{-}leave\text{-}ack}_h$ output action. This action is enabled whenever the leave acknowledgment is pending; that is, *status* = `leave-ack-pending`. Time is not allowed to elapse while a leave acknowledgment is pending. Thus, a leave acknowledgement is sent immediately after SRM-MEM$_h$ determines that it has successfully left the IP multicast group.

Time Passage

The action $\nu(t)$ models the passage of t time units. Time is prevented from elapsing while there are pending actions — either pending requests to join or leave the underlying IP multicast group, or pending acknowledgments that the client has successfully joined or left the reliable multicast group. The effects of the $\nu(t)$ action are to increment the variable *now* by t time units.

4.3.3 The IP Buffer Component — SRM-IPBUFF$_h$

The SRM-IPBUFF$_h$ timed I/O automaton specifies the IP buffer component of the reliable multicast process. Figures 4.7 and 4.8 present the signature, the variables, and the discrete transitions of SRM-IPBUFF$_h$.

Variables

The variable $now \in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of SRM-IPBUFF$_h$. The variable *status* captures the status of the host h. It evaluates to one of the following: `idle`, `member`, and `crashed`. While the host h has not crashed, we say that it is

Figure 4.7 The SRM-IPBUFF$_h$ Automaton — Signature

Parameters:

$h \in H$

Actions:

input
`crash`$_h$
`rm-join-ack`$_h$
`rm-leave`$_h$
`mrecv`$_h(p)$, for $p \in P_{\text{IPMCAST-CLIENT}}$
`rec-msend`$_h(p)$, for $p \in P_{\text{SRM}}$

output
`process-pkt`$_h(p)$, for $p \in P_{\text{SRM}}$
`msend`$_h(p)$, for $p \in P_{\text{IPMCAST-CLIENT}}$

time-passage
$\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

Figure 4.8 The SRM-IPBUFF$_h$ Automaton — Variables and Discrete Transitions

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status \in \textit{SRM-Status}$, initially $status =$ `idle`
$seqno \in \mathbb{N}$, initially $seqno = 0$
$\textit{recv-buff} \subseteq P_{\text{SRM}}$, initially $\textit{recv-buff} = \emptyset$
$\textit{msend-buff} \subseteq P_{\text{IPMCAST-CLIENT}}$, initially $\textit{msend-buff} = \emptyset$

Discrete Transitions:

input `crash`$_h$
eff $status :=$ `crashed`

input `rm-join-ack`$_h$
eff **if** $status \neq$ `crashed` **then** $status :=$ `member`

input `rm-leave`$_h$
eff **if** $status \neq$ `crashed` **then**
Reinitialize all variables except *now* and *seqno*.

input `mrecv`$_h(p)$
eff **if** $status =$ `member` **then** $\textit{recv-buff} \cup= \{strip(p)\}$

input `rec-msend`$_h(p)$
eff **if** $status =$ `member` **then**
$\textit{msend-buff} \cup= \{\textit{comp-IPmcast-pkt}(h, seqno, p)\}$
$seqno := seqno + 1$

output `process-pkt`$_h(p)$
pre $status =$ `member` $\wedge\, p \in \textit{recv-buff}$
eff $\textit{recv-buff} \setminus= \{p\}$

output `msend`$_h(p)$
pre $status =$ `member` $\wedge\, p \in \textit{msend-buff}$
eff $\textit{msend-buff} \setminus= \{p\}$

time-passage $\nu(t)$
pre $status =$ `crashed`
$\vee(\textit{recv-buff} = \emptyset \wedge \textit{msend-buff} = \emptyset)$
eff $now := now + t$

operational. Once the host h has crashed, none of the input actions of SRM-IPBUFF$_h$ affect the state of SRM-IPBUFF$_h$ and none of the internal and output actions of SRM-IPBUFF$_h$, except the time passage action, are enabled. The variable $seqno \in \mathbb{N}$ is a counter of the number of packets transmitted by SRM-IPBUFF$_h$ using the underlying IP multicast service.

The set *recv-buff* is used to buffer all packets received from the underlying IP multicast service. The set *msend-buff* is used to buffer all packets to be multicast using the underlying IP multicast service.

Input Actions

The input action `crash`$_h$ models the crashing of SRM-IPBUFF$_h$. The effects of `crash`$_h$ are to set the *status* variable to `crashed`, denoting that the host h has crashed. After the host h has crashed, the SRM-IPBUFF$_h$ automaton does not restrict time from elapsing.

The input action `rm-join-ack`$_h$ informs the SRM-IPBUFF$_h$ automaton that the host h has joined the reliable multicast group. If the host h is operational, then the action `rm-join-ack`$_h$ records the fact that the host h has joined the reliable multicast group by setting the variable *status* to `member`.

The input action `rm-leave`$_h$ informs the SRM-IPBUFF$_h$ automaton that the host h has left the reliable multicast group. If the host h is operational, then the action $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-IPBUFF$_h$ except the variables *now* and *seqno*.

The input action $\texttt{mrecv}_h(p)$ models the reception of the packet p from the underlying IP multicast service. If the host h is a member of the reliable multicast group, then the $\texttt{mrecv}_h(p)$ action adds the packet p to the *recv-buff* buffer. Thus, the contents of the packet p may subsequently be processed by the reliable multicast service and, when appropriate, delivered to the client.

The input action $\texttt{rec-msend}_h(p)$ is performed by the recovery component so as to transmit the packet p using the underlying IP multicast service. If the host h is a member of the reliable multicast group, then SRM-IPBUFF$_h$ encapsulates h, *seqno*, and p into a packet *pkt*, buffers *pkt* in *msend-buff* for transmission using the underlying IP multicast service, and increments *seqno*. In effect, the encapsulation of p annotates it with the host h and the value of *seqno*. Since the variable *seqno* is persistent across host joins and leaves, packets transmitted by the SRM-IPBUFF$_h$ automata, for $h \in H$, are unique.

Output Actions

The output action $\texttt{process-pkt}_h(p)$ models the processing of the packet p by the reporting and recovery components. It is enabled when the host h is a member of the reliable multicast group and there is a packet *pkt* in the *recv-buff* buffer, such that $strip(pkt) = p$. Its effects are to remove the element *pkt* from the *recv-buff* buffer.

The output action $\texttt{msend}_h(p)$ models the transmission of the packet p using the underlying IP multicast service. It is enabled when the host h is a member of the group and the packet p is in the *msend-buff* buffer. Its effects are to remove the packet p from the *msend-buff* buffer.

Time Passage

The action $\nu(t)$ models the passage of t time units. Time is prevented from elapsing while the host h is operational and either of the buffers *recv-buff* and *msend-buff* is non-empty. The effects of the $\nu(t)$ action are to increment the variable *now* by t time units.

4.3.4 The Recovery Component — SRM-REC$_h$

The SRM-REC$_h$ timed I/O automaton specifies the recovery component of the reliable multicast service. Figure 4.9 presents the signature of SRM-REC$_h$, that is, its parameters, and actions. Figure 4.10 presents the variables of SRM-REC$_h$. Figures 4.11, 4.12, and 4.13 present the discrete transitions of SRM-REC$_h$. In order to provide the appropriate context, the description of each of the parameters of SRM-REC$_h$ is deferred to appropriate places within the description of its variables and actions.

Variables

The variable $now \in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of SRM-REC$_h$. The variable *status* captures the status of the host h. It evaluates to one of the following: `idle`, `member`, and `crashed`. While the host h has not crashed, we say that it is *operational*. The variable $rep\text{-}deadline \in \mathbb{R}^{\geq 0} \cup \perp$ denotes the point in time at which the next session packet is scheduled for transmission. When undefined, the variable *rep-deadline* is equal to $\perp$.

Figure 4.9 The SRM-$\textsc{rec}_h$ Automaton — Signature

Parameters:

$h \in H, C_1, C_2, C_3, D_1, D_2, D_3 \in \mathbb{R}^{\geq 0}$, $\texttt{DFLT-DIST} \in \mathbb{R}^{\geq 0}$, $\texttt{SESS-PERIOD} \in \mathbb{R}^{+}$

Actions:

input
- $\texttt{crash}_h$
- $\texttt{rm-join-ack}_h$
- $\texttt{rm-leave}_h$
- $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$
- $\texttt{process-pkt}_h(p)$, for $p \in P_{\text{SRM}}$

internal
- $\texttt{schdl-rqst}_h(s, i)$, for $s \in H, i \in \mathbb{N}$
- $\texttt{send-sess}_h$
- $\texttt{send-rqst}_h(s, i)$, for $s \in H, i \in \mathbb{N}$
- $\texttt{send-repl}_h(s, i)$, for $s \in H, i \in \mathbb{N}$

output
- $\texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$
- $\texttt{rec-msend}_h(p)$, for $p \in P_{\text{SRM}}$

time-passage
- $\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

The variable *dist-rprt*$(h') \in \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \cup \perp$, for each $h' \in H, h' \neq h$, records the transmission and the reception times of the most recent session packet of h' to be received by the host h. That is, for each $h' \in H$, the variable *dist-rprt*(h') is a tuple of the form $\langle t_{sent}, t_{rcvd} \rangle$, where t_{sent} is the transmission time of the most recent session packet of h' received by h and t_{rcvd} is the time at which h received this session packet. If the host h has not received a session packet from the host h' since joining the reliable multicast group, then the variable *dist-rprt*(h') is undefined; that is, *dist-rprt*$(h') = \perp$.

The variable *dist*$(h') \in \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$, for each $h' \in H, h' \neq h$, records the most up-to-date estimate of the distance from h to the host h'. Such distance estimates are ordered by the transmission time of the session packet of h that initiated their calculation; that is, a distance estimate calculated as a result of the transmission of a more recent session packet of h is considered more up-to-date. If two calculations are initiated by the same session packet of h, then the later calculation is considered more up-to-date. Thus, for each $h' \in H$, the variable *dist*(h') is a tuple of the form $\langle t_{rprt}, t_{dist} \rangle$, where t_{rprt} is the transmission time of the session packet of h that initiated the calculation of the particular distance estimate and t_{dist} is the distance estimate obtained as a result of the particular calculation. The variables *dist*(h'), for $h' \in H, h' \neq h$, are initialized to $\langle 0, \texttt{DFLT-DIST} \rangle$, where $\texttt{DFLT-DIST}$ is the default inter-host distance estimate parameter of SRM-$\textsc{rec}_h$.

Each of the *min-seqno*$(h') \in \mathbb{N}$ and *max-seqno*$(h') \in \mathbb{N}$ variables, for $h' \in H$, denotes the minimum and maximum ADU sequence numbers observed to have been transmitted by the host h'. The variable *archived-pkts* $\subseteq P_{\text{RM-CLIENT}} \times \mathbb{R}^{\geq 0}$ is comprised of pairs involving the ADUs that have either been sent by or buffered for delivery to the client at h and the first point in time at which each ADU has either been sent by or buffered for delivery to the client at h. The variable *to-be-requested?* $\subseteq H \times \mathbb{N}$ denotes the set of ADU packets that have been identified as missing and for which a request has yet to be scheduled. The elements of *to-be-requested?* are tuples of the form $\langle s, i \rangle$, with $s \in H$ and $i \in \mathbb{N}$ denoting the source s and the sequence number i of the missing ADU.

The set *pending-rqsts* $\subseteq$ *Pending-Rqsts* is comprised of tuples that correspond to packets for which a request is pending; that is, a request for the particular packet has recently either been sent or received and a reply is being awaited. The tuples of *pending-rqsts* are of the form $\langle s, i, t \rangle$, with $s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}$; s and i represent the source and sequence number of the packet whose request is pending and t represents the back-off abstinence deadline; that is, the time before which the request timeout timer for the given packet may not be backed off. A pending request *expires* when time elapses past its back-off abstinence timeout. Prior to its expiration, a pending request is said to be *active*.

The set *scheduled-rqsts* $\subseteq$ *Scheduled-Rqsts* is comprised of tuples that correspond to packets for which a request has been scheduled and is awaiting transmission. The tuples of *scheduled-rqsts* are of the form $\langle s, i, t, k\rangle$, with $s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}$; s and i correspond to the source and sequence number of the packet to be requested, t is the time for which the request is scheduled for transmission, and k is the number of times a request for the given packet has already been scheduled.

The set *pending-repls* $\subseteq$ *Pending-Repls* is comprised of tuples that correspond to packets for which a reply has recently been either sent or received. The tuples of *pending-repls* are of the form $\langle s, i, t\rangle$, with $s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}$; s and i correspond to the source and sequence number of the packet for which a reply has already been either sent or received and t is the abstinence timeout of the reply; that is, a deadline before which replies for the given packet may not be scheduled by the host h. A pending reply *expires* when time elapses past its abstinence timeout. Prior to its expiration, a pending reply is said to be *active*.

The set *scheduled-repls* $\subseteq$ *Scheduled-Repls* is comprised of tuples that correspond to packets for which a reply has been scheduled and is awaiting transmission. The tuples comprising the set *scheduled-repls* are of the form $\langle s, i, t, r\rangle$, with $s, r \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}$; s and i correspond to the source and sequence number of the packet to be retransmitted, t is the time for which the reply is scheduled for transmission, and r is the host whose request induced the scheduling of the particular reply.

The set *to-be-delivered* $\subseteq P_{\text{RM-CLIENT}}$ is used to buffer the packets that are to be subsequently delivered to the client. The set *msend-buff* $\subseteq P_{\text{SRM}}$ is used to buffer the packets that are to be subsequently multicast using the underlying IP multicast service; that is, it contains the data packets of the client and the requests and replies of the recovery component to be transmitted by the host h.

Derived Variables

The derived variable *dist?*$(h') \in \mathbb{R}^{\geq 0}$, for $h' \in H, h' \neq h$, is h's current distance estimate to the host h'.

The derived variable *dist-rprt* records the transmission and the reception times of the most recent session packet of all other hosts. It is a set of tuples of the form $\langle h', t_s, t_r\rangle$, for $h' \in H, h' \neq h$ and $t_s, t_r \in \mathbb{R}^{\geq 0}$, such that *dist-rprt*$(h') \neq \perp$ and $\langle t_s, t_r\rangle =$ *dist-rprt*(h'). In effect, *dist-rprt* summarizes the information recorded by the *dist-rprt*(h') variables, for all $h' \in H, h' \neq h$.

The derived variable *max-seqno* records the transmission progress of all other hosts. *max-seqno* is the set of tuples of the form $\langle h', \textit{max-seqno}(h')\rangle$, for $h' \in H, h' \neq h$, and *max-seqno*$(h') \neq \perp$. In effect, *max-seqno* summarizes the information recorded by the *max-seqno*(h') variables, for all $h' \in H, h' \neq h$.

The derived variable *proper?*(h'), for $h' \in H$, is the set comprised of the identifiers of the packets from h' whose sequence numbers are no less than *min-seqno*(h'). The derived variable *window?*(h'), for $h' \in H$, is the set comprised of the identifiers of the packets from h' whose sequence numbers are no less than *min-seqno*(h') and no greater than *max-seqno*(h').

The derived variable *archived-pkts?* $\subseteq H \times \mathbb{N}$ identifies all the packets for which there is a corresponding tuple in the set *archived-pkts*. The derived variable *archived-pkts?*$(h') \subseteq H \times \mathbb{N}$, for $h' \in H$, identifies all the packets from h' for which there is a corresponding tuple in the set *archived-pkts*.

The derived variable *to-be-requested?*$(h') \subseteq H \times \mathbb{N}$, for $h' \in H$, identifies all the packets from h' that are in the set *to-be-requested?*. The derived variable *to-be-delivered?* $\subseteq H \times \mathbb{N}$ identifies all the packets for which there is a corresponding tuple in the set *to-be-delivered*. The derived variable

Figure 4.10 The SRM-$\textsc{rec}_h$ Automaton — Variables

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status \in SRM\text{-}Status$, initially $status = \texttt{idle}$
$rep\text{-}deadline \in \mathbb{R}^{\geq 0} \cup \bot$, initially $rep\text{-}deadline = \bot$
$dist\text{-}rprt(h') \in \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \cup \bot$, for all $h' \in H, h' \neq h$, initially $dist\text{-}rprt(h') = \bot$
$dist(h') \in \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$, for all $h' \in H, h' \neq h$, initially $dist(h') = \langle 0, \texttt{DFLT-DIST} \rangle$
$min\text{-}seqno(h') \in \mathbb{N} \cup \bot$, for all $h' \in H$, initially $min\text{-}seqno(h') = \bot$, for all $h' \in H$
$max\text{-}seqno(h') \in \mathbb{N} \cup \bot$, for all $h' \in H$, initially $max\text{-}seqno(h') = \bot$, for all $h' \in H$
$archived\text{-}pkts \subseteq P_{\text{RM-Client}} \times \mathbb{R}^{\geq 0}$, initially $archived\text{-}pkts = \emptyset$
$to\text{-}be\text{-}requested? \subseteq H \times \mathbb{N}$, initially $to\text{-}be\text{-}requested? = \emptyset$
$pending\text{-}rqsts \subseteq Pending\text{-}Rqsts$, initially $pending\text{-}rqsts = \emptyset$
$scheduled\text{-}rqsts \subseteq Scheduled\text{-}Rqsts$, initially $scheduled\text{-}rqsts = \emptyset$
$pending\text{-}repls \subseteq Pending\text{-}Repls$, initially $pending\text{-}repls = \emptyset$
$scheduled\text{-}repls \subseteq Scheduled\text{-}Repls$, initially $scheduled\text{-}repls = \emptyset$
$to\text{-}be\text{-}delivered \subseteq P_{\text{RM-Client}}$, initially $to\text{-}be\text{-}delivered = \emptyset$
$msend\text{-}buff \subseteq P_{\text{SRM}}$, initially $msend\text{-}buff = \emptyset$

Derived Variables:

$dist?(h') = d$, for $d \in \mathbb{R}^{\geq 0}$, such that $dist(h') = \langle t, d \rangle$, for some $t \in \mathbb{R}^{\geq 0}$, for all $h' \in H$
$dist\text{-}rprt = \cup_{h' \in H, h' \neq h, dist\text{-}rprt(h') \neq \bot} \{\langle h', t_{sent}, t_{rcvd} \rangle \mid dist\text{-}rprt(h') = \langle t_{sent}, t_{rcvd} \rangle\}$
$max\text{-}seqno = \cup_{h' \in H, h' \neq h, max\text{-}seqno(h') \neq \bot} \{\langle h', max\text{-}seqno(h') \rangle\}$

$$\text{for all } h' \in H,\ proper?(h') = \begin{cases} \emptyset & \text{if } min\text{-}seqno(h') = \bot \\ \{\langle s,i \rangle \in H \times \mathbb{N} \mid s = h', min\text{-}seqno(h') \leq i\} & \text{otherwise} \end{cases}$$

$$\text{for all } h' \in H,\ window?(h') = \begin{cases} \emptyset & \text{if } min\text{-}seqno(h') = \bot \\ \{\langle s,i \rangle \in H \times \mathbb{N} \mid s = h', min\text{-}seqno(h') \leq i \leq max\text{-}seqno(h')\} & \text{otherwise} \end{cases}$$

$archived\text{-}pkts? = \{\langle s,i \rangle \in H \times \mathbb{N} \mid \exists\, p \in P_{\text{RM-Client}}, t \in \mathbb{R}^{\geq 0} : \langle p, t \rangle \in archived\text{-}pkts \wedge id(p) = \langle s,i \rangle\}$
$archived\text{-}pkts?(h') = \{\langle s,i \rangle \in archived\text{-}pkts? \mid s = h'\}$, for all $h' \in H$
$to\text{-}be\text{-}requested?(h') = \{\langle s,i \rangle \in to\text{-}be\text{-}requested? \mid s = h'\}$, for all $h' \in H$
$to\text{-}be\text{-}delivered? = \{\langle s,i \rangle \in H \times \mathbb{N} \mid \exists\, p \in to\text{-}be\text{-}delivered : \langle s,i \rangle = id(p)\}$
$to\text{-}be\text{-}delivered?(h') = \{\langle s,i \rangle \in to\text{-}be\text{-}delivered? \mid s = h'\}$, for all $h' \in H$
$scheduled\text{-}rqsts? = \{\langle s,i \rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N} : \langle s,i,t,k \rangle \in scheduled\text{-}rqsts\}$
$scheduled\text{-}rqsts?(h') = \{\langle s,i \rangle \in scheduled\text{-}rqsts? \mid s = h'\}$, for all $h' \in H$
$scheduled\text{-}repls? = \{\langle s,i \rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0}, q \in H : \langle s,i,t,q \rangle \in scheduled\text{-}repls\}$
$scheduled\text{-}repls?(h') = \{\langle s,i \rangle \in scheduled\text{-}repls? \mid s = h'\}$, for all $h' \in H$
$pending\text{-}rqsts? = \{\langle s,i \rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0} : now \leq t \wedge \langle s,i,t \rangle \in pending\text{-}rqsts\}$
$pending\text{-}repls? = \{\langle s,i \rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0} : now \leq t \wedge \langle s,i,t \rangle \in pending\text{-}repls\}$

$to\text{-}be\text{-}delivered?(h') \subseteq H \times \mathbb{N}$, for $h' \in H$, identifies all the packets from h' that are in the set $to\text{-}be\text{-}delivered?$.

The derived variable $scheduled\text{-}rqsts? \subseteq H \times \mathbb{N}$ identifies all the packets for which there is a corresponding scheduled request tuple in the set $scheduled\text{-}rqsts$. The derived variable $scheduled\text{-}rqsts?(h') \subseteq H \times \mathbb{N}$, for $h' \in H$, identifies all the packets from h' whose identifiers are in the set $scheduled\text{-}rqsts?$. The derived variable $scheduled\text{-}repls? \subseteq H \times \mathbb{N}$ identifies all the packets for which there is a corresponding scheduled reply tuple in the set $scheduled\text{-}repls$.

The derived variable $pending\text{-}rqsts? \subseteq H \times \mathbb{N}$ identifies all the packets for which there is an active pending request; that is, there is a corresponding tuple in the set $pending\text{-}rqsts$ whose back-off abstinence timeout has not yet expired. The derived variable $pending\text{-}repls? \subseteq H \times \mathbb{N}$ identifies all the packets for which there is an active pending reply; that is, there is a corresponding tuple in the set $pending\text{-}repls$ whose abstinence timeout has not yet expired.

Input Actions

The input action $\texttt{crash}_h$ models the crashing of the host h. The effects of $\texttt{crash}_h$ are to set the *status* variable to `crashed`. Once the host h has crashed, none of the input actions of SRM-$\textsc{rec}_h$ affect its state, none of the internal and output actions of SRM-$\textsc{rec}_h$ are enabled, and time is not restricted from elapsing.

The input action $\texttt{rm-join-ack}_h$ informs the SRM-REC_h automaton that the host h has joined the reliable multicast group. If the host h is operational, then the $\texttt{rm-join-ack}_h$ action records the fact that the host h has joined the reliable multicast group by setting the variable *status* to $\texttt{member}$. Subsequently, SRM-REC_h may transmit, process, and deliver packets. Moreover, the $\texttt{rm-join-ack}_h$ action schedules the transmission of a session packet no later than $\texttt{SESS-PERIOD}$ time units in the future by setting the *rep-deadline* variable to a value that is uniformly chosen within the interval $now + (0, \texttt{SESS-PERIOD}]$. The parameter $\texttt{SESS-PERIOD}$ specifies the period with which SRM-REC_h transmits session packets.

The input action $\texttt{rm-leave}_h$ informs the SRM-REC_h automaton that the host h has left the reliable multicast group. If the host h is operational, then the action $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-REC_h except the variable *now*. Subsequently, SRM-REC_h automaton ceases transmitting, processing, and delivering packets, exchanging session packets, and scheduling packet requests and replies.

The input action $\texttt{rm-send}_h(p)$ models the transmission of the packet p by the client at h using the reliable multicast service. $\texttt{rm-send}_h(p)$ is effective only when the host h is a member of the reliable multicast group and the host h is the source of the packet p. If p is the first packet to be transmitted by the client since it last joined the reliable multicast group, the $\texttt{rm-send}_h(p)$ action sets the *min-seqno*(h) variable to the sequence number of p. Otherwise, SRM-REC_h ensures that p corresponds to the next packet awaited; that is, the packet whose sequence number is one larger than the sequence number of the latest packet transmitted by h. If so, SRM-REC_h updates *max-seqno*(h), archives p, and generates a $\texttt{DATA}$ packet to subsequently be transmitted to the other members of the reliable multicast group through the underlying IP multicast service. The operation *comp-data-pkt*(p) composes a $\texttt{DATA}$ packet corresponding to the client packet p.

The input action $\texttt{process-pkt}_h(p)$ models the processing of the packet p by SRM-REC_h. The packet p is processed only when the host h is a member of the reliable multicast group. We proceed by describing the effects of $\texttt{process-pkt}_h(p)$ depending on the type of the packet p. When p is either a $\texttt{DATA}$, $\texttt{RQST}$, or $\texttt{REPL}$ packet, we let $s_p \in H$ and $i_p \in \mathbb{N}$ denote the source and the sequence number pertaining to the packet p.

First, consider the case where p is a $\texttt{DATA}$ packet. If h is not the source of p and p is the first packet from s_p to be received by h, then the variables *min-seqno*(s_p) and *max-seqno*(s_p) are set to i_p. Following this initial assignment of *min-seqno*(s_p) to i_p, all $\texttt{DATA}$, $\texttt{RQST}$, and $\texttt{REPL}$ packets pertaining to ADUs from s_p with sequence numbers less than i_p are considered *improper* and are discarded. Conversely, all $\texttt{DATA}$, $\texttt{RQST}$, and $\texttt{REPL}$ packets pertaining to ADUs from s_p with sequence numbers equal to or greater than i_p are considered *proper* and are processed.

The processing of packet p proceeds only while it is considered a proper packet. Unless either h is the source of p or p is already archived, p is archived by adding the tuple $\{\langle strip(p), now\rangle\}$ to *archived-pkts*. Unless h is the source of p, the ADU contained in p is buffered in *to-be-delivered* so that it may subsequently be delivered to the client. Thus, the reliable multicast process does not deliver packets sent by a client to itself. Moreover, the reliable multicast service may also deliver the same ADU to the client multiple times. The identifier of the ADU pertaining to p is removed from the *to-be-requested?* set and any scheduled requests and replies for the ADU pertaining to p are canceled. Finally, unless h is the source of p, SRM-REC_h adds any trailing missing packets to the set *to-be-requested?*, so that a request for each of them may subsequently be scheduled.

Second, consider the case where p is a $\texttt{RQST}$ packet. Once again, p is processed only if it pertains to a proper ADU. If p pertains to an ADU that has been archived and for which a reply is neither scheduled, nor pending, then SRM-REC_h schedules a retransmission of the requested ADU. This retransmission is scheduled for a point it time in the future that is chosen uniformly within the interval $now + [D_1 d_{repl}, (D_1 + D_2) d_{repl}]$, with $d_{repl} = dist?(sender(p))$. If p pertains to an ADU that

Figure 4.11 The SRM-REC_h Automaton — Discrete Transitions

input $\texttt{crash}_h$
eff $\mathit{status} := \texttt{crashed}$

input $\texttt{rm-join-ack}_h$
eff **if** $\mathit{status} \neq \texttt{crashed}$ **then**
$\quad \mathit{status} := \texttt{member}$
$\quad \mathit{rep\text{-}deadline} :\in \mathit{now} + (0, \texttt{SESS-PERIOD}]$

input $\texttt{rm-leave}_h$
eff **if** $\mathit{status} \neq \texttt{crashed}$ **then**
$\quad$ Reinitialize all variables except now.

input $\texttt{rm-send}_h(p)$
eff **if** $\mathit{status} = \texttt{member} \wedge h = \mathit{source}(p)$ **then**
$\quad \langle s_p, i_p \rangle = \mathit{id}(p)$
$\quad$ `\\ Record foremost DATA packet`
$\quad$ **if** $\mathit{min\text{-}seqno}(s_p) = \perp$ **then** $\mathit{min\text{-}seqno}(s_p) := i_p$
$\quad$ `\\ Only consider next packet`
$\quad$ **if** $\mathit{max\text{-}seqno}(s_p) = \perp$
$\quad\quad \vee i_p = \mathit{max\text{-}seqno}(s_p) + 1$
$\quad$ **then**
$\quad\quad \mathit{max\text{-}seqno}(s_p) := i_p$
$\quad\quad$ `\\ Archive packet`
$\quad\quad \mathit{archived\text{-}pkts} \cup= \{\langle p, \mathit{now} \rangle\}$
$\quad\quad$ `\\ Compose data packet`
$\quad\quad \mathit{msend\text{-}buff} \cup= \{\mathit{comp\text{-}data\text{-}pkt}(p)\}$

output $\texttt{rm-recv}_h(p)$
pre $\mathit{status} = \texttt{member} \wedge p \in \mathit{to\text{-}be\text{-}delivered}$
$\quad \wedge(\nexists\, p' \in \mathit{to\text{-}be\text{-}delivered}:$
$\quad\quad \mathit{source}(p') = \mathit{source}(p) \wedge \mathit{seqno}(p') < \mathit{seqno}(p))$
eff $\mathit{to\text{-}be\text{-}delivered} \setminus= \{p\}$

output $\texttt{rec-msend}_h(p)$
pre $\mathit{status} = \texttt{member} \wedge p \in \mathit{msend\text{-}buff}$
eff $\mathit{msend\text{-}buff} \setminus= \{p\}$

internal $\texttt{schdl-rqst}_h(s, i)$
pre $\mathit{status} = \texttt{member} \wedge \langle s, i \rangle \in \mathit{to\text{-}be\text{-}requested?}$
eff `\\ Schedule new request`
$\quad k_r := 1;\ d_r := \mathit{dist?}(s)$
$\quad t_r :\in \mathit{now} + 2^{k_r - 1}[C_1 d_r, (C_1 + C_2) d_r]$
$\quad \mathit{scheduled\text{-}rqsts} \cup= \{\langle s, i, t_r, k_r \rangle\}$
$\quad$ `\\ Pkt request has been scheduled`
$\quad \mathit{to\text{-}be\text{-}requested?} \setminus= \{\langle s, i \rangle\}$

internal $\texttt{send-sess}_h$
pre $\mathit{status} = \texttt{member} \wedge \mathit{now} = \mathit{rep\text{-}deadline}$
eff `\\ Compose session packet`
$\quad \mathit{msend\text{-}buff} \cup=$
$\quad\quad \{\mathit{comp\text{-}sess\text{-}pkt}(h, \mathit{now}, \mathit{dist\text{-}rprt}, \mathit{max\text{-}seqno})\}$
$\quad$ `\\ Reset session packet deadline`
$\quad \mathit{rep\text{-}deadline} := \mathit{now} + \texttt{SESS-PERIOD}$

internal $\texttt{send-rqst}_h(s, i)$
choose $t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}$
pre $\mathit{status} = \texttt{member}$
$\quad \wedge t = \mathit{now} \wedge \langle s, i, t, k \rangle \in \mathit{scheduled\text{-}rqsts}$
eff `\\ Compose request packet`
$\quad \mathit{msend\text{-}buff} \cup= \{\mathit{comp\text{-}rqst\text{-}pkt}(s, i, h)\}$
$\quad$ `\\ Back-off scheduled request`
$\quad \mathit{scheduled\text{-}rqsts} \setminus= \{\langle s, i, t, k \rangle\}$
$\quad k_r := k + 1;\ d_r := \mathit{dist?}(s)$
$\quad t_r :\in \mathit{now} + 2^{k_r - 1}[C_1 d_r, (C_1 + C_2) d_r]$
$\quad \mathit{scheduled\text{-}rqsts} \cup= \{\langle s, i, t_r, k_r \rangle\}$
$\quad$ `\\ A request becomes pending`
$\quad \mathit{pending\text{-}rqsts} \setminus= \{\langle s, i, t_* \rangle \mid t_* \in \mathbb{R}^{\geq 0}\}$
$\quad t_r := \mathit{now} + 2^{k_r - 1} C_3 d_r$
$\quad \mathit{pending\text{-}rqsts} \cup= \{\langle s, i, t_r \rangle\}$

internal $\texttt{send-repl}_h(s, i)$
choose $t \in \mathbb{R}^{\geq 0}, q \in H$
pre $\mathit{status} = \texttt{member}$
$\quad \wedge t = \mathit{now} \wedge \langle s, i, t, q \rangle \in \mathit{scheduled\text{-}repls}$
eff `\\ Compose reply packet`
$\quad$ **choose** $p \in P_{\text{RM-CLIENT}}, t \in \mathbb{R}^{\geq 0}$
$\quad\quad$ **where** $\langle p, t \rangle \in \mathit{archived\text{-}pkts} \wedge \mathit{id}(p) = \langle s, i \rangle$
$\quad \mathit{msend\text{-}buff} \cup= \{\mathit{comp\text{-}repl\text{-}pkt}(p, q, h)\}$
$\quad$ `\\ A reply becomes pending`
$\quad \mathit{pending\text{-}repls} \setminus= \{\langle s, i, t_* \rangle \mid t_* \in \mathbb{R}^{\geq 0}\}$
$\quad t_{repl} := \mathit{now} + D_3\, \mathit{dist?}(r)$
$\quad \mathit{pending\text{-}repls} \cup= \{\langle s, i, t_{repl} \rangle\}$
$\quad$ `\\ Cancel scheduled reply`
$\quad \mathit{scheduled\text{-}repls} \setminus= \{\langle s, i, t, q \rangle\}$

time-passage $\nu(t)$
pre $\mathit{status} = \texttt{crashed}$
$\quad \vee(\mathit{to\text{-}be\text{-}requested?} = \emptyset \wedge \mathit{to\text{-}be\text{-}delivered} = \emptyset$
$\quad\quad \wedge \mathit{msend\text{-}buff} = \emptyset$
$\quad\quad \wedge(\mathit{rep\text{-}deadline} = \perp \vee \mathit{now} + t \leq \mathit{rep\text{-}deadline})$
$\quad\quad \wedge$ no requests scheduled earlier than $\mathit{now} + t$
$\quad\quad \wedge$ no replies scheduled earlier than $\mathit{now} + t$)
eff $\mathit{now} := \mathit{now} + t$

has not been archived, then the effects of $\texttt{process-pkt}_h(p)$ depend on whether there is a request for the given ADU already scheduled. If h is not the source of p and there is no request for the ADU of p already scheduled, then a request for the given ADU is scheduled. This request is scheduled for a point it time in the future that is chosen uniformly within the interval $\mathit{now} + 2[C_1 d_r, (C_1 + C_2) d_r]$, with $d_r = \mathit{dist?}(s_p)$; that is, the request is scheduled as if a first round request is being backed off. If h is not the source of p, there is a request for the ADU of p already scheduled and there, are there are no pending requests for the ADU of p still active, then the request for the ADU of p that is already scheduled is exponentially backed off. When either a new request is scheduled or an existing request is backed-off, a request for the given ADU becomes pending with a back-off abstinence timeout equal to $\mathit{now} + 2^{k-1} C_3 d_r$, where k is the round of the rescheduled request and $d_r = \mathit{dist?}(s_p)$. Finally, unless h is the source of p, SRM-REC_h adds any trailing missing packets to the set $\mathit{to\text{-}be\text{-}requested?}$, so that a request for each of them may subsequently be scheduled.

Third, consider the case where p is a `REPL` packet. The processing of a a `REPL` packet is similar to

Figure 4.12 The SRM-REC_h Automaton — Discrete Transitions (Cnt'd)

```
input process-pkt_h(p)
where type(p) = DATA
eff  if status = member then
       ⟨s_p, i_p⟩ = id(p)
       \\ Record foremost DATA packet
       if h ≠ s_p ∧ min-seqno(s_p) =⊥ then
         min-seqno(s_p) := i_p; max-seqno(s_p) := i_p
       \\ Only consider proper packets
       if min-seqno(s_p) ≠⊥ ∧min-seqno(s_p) ≤ i_p then
         \\ Archive and deliver packet
         if h ≠ s_p ∧ ⟨s_p, i_p⟩ ∉ archived-pkts? then
           archived-pkts ∪= {⟨strip(p), now⟩}
         if h ≠ s_p then to-be-delivered ∪= {strip(p)}
         \\ Pkt need not be requested
         to-be-requested? \= {⟨s_p, i_p⟩}
         \\ Cancel any scheduled requests and replies
         scheduled-rqsts \= {⟨s_p, i_p, t, k⟩ | t ∈ ℝ^≥0, k ∈ ℕ}
         scheduled-repls \={⟨s_p, i_p, t, q⟩ | t ∈ ℝ^≥0, q ∈ H}
         \\ Cancel any pending requests
         pending-rqsts \= {⟨s_p, i_p, t⟩ | t ∈ ℝ^≥0}
         \\ Discover any trailing missing packets
         if h ≠ s_p ∧ max-seqno(s_p) < i_p then
           to-be-requested? ∪=
             {⟨s_p, i⟩ | i ∈ ℕ, max-seqno(s_p) < i < i_p}
           max-seqno(s_p) := i_p
```

```
input process-pkt_h(p)
where type(p) = SESS
eff  if status = member then
       s_p := sender(p)
       if dist-rprt(s_p) =⊥ then
         dist-rprt(s_p) := ⟨time-sent(p), now⟩
       else
         ⟨t_sent, t_rcvd⟩ := dist-rprt(s_p)
         if t_sent ≤ time-sent(p) then
           dist-rprt(s_p) := ⟨time-sent(p), now⟩
       if h ∈ dist-rprt?(p) then
         ⟨t_sent, t_delayed⟩ := dist-rprt(p, h)
         ⟨t_rprt, t_dist⟩ := dist(s_p)
         if t_rprt ≤ t_sent then
           t'_dist := (now − t_delayed − t_sent)/2
           dist(s_p) := ⟨t_sent, t'_dist⟩
       foreach ⟨h'', i''⟩ ∈ seqno-rprts(p) do:
         if min-seqno(h'') ≠⊥ then
           \\ Discover any trailing missing packets
           if h ≠ h'' ∧ max-seqno(h'') < i'' then
             to-be-requested? ∪=
               {⟨h'', i⟩ | i ∈ ℕ, max-seqno(h'') < i < i''}
             max-seqno(h'') := i''
```

that of a `DATA` packet. The differences are that p is processed only if it pertains to a proper ADU and that in addition to the effects of processing a `DATA` packet, a reply for the given ADU becomes pending. While this pending reply is active, SRM-REC_h does not schedule replies for the ADU pertaining to p.

Finally, consider the case where p is a `SESS` packet. Let s_p denote the sender of p. If p is either the first or the most recent session packet of s_p to be received by h, then SRM-REC_h sets the variable $\textit{dist-rprt}(s_p)$ to $\langle \textit{time-sent}(p), \textit{now}\rangle$. Thus, SRM-$\text{REC}_h$ records the reception of a more recent session packet from the host s_p. Moreover, if p is distance reporting for h and the session packet that initiated this report is at least as recent as the session packet that initiated the calculation of the current distance estimate to s_p, then a new distance estimate to s_p is calculated. If the calculation of the current distance estimate was initiated by the same session packet as the new calculation, then the new distance estimate is considered more recent since the latency observed from s_p to h is more recent. SRM-REC_h records the new distance estimate to s_p by appropriately setting the tuple $\textit{dist}(s_p)$.

SRM-REC_h goes through the transmission state reports contained in p to determine whether s_p has observed further progress in the transmission of any of the sources; that is, whether s_p has observed the transmission of later ADU packets by any of the sources. For each state report indicating further transmission progress, SRM-REC_h adds the trailing missing packets to the set *to-be-requested?*, so that a request for each of them may subsequently be scheduled, and updates the corresponding *max-seqno* variable.

Output Actions

Each output action $\texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, models the delivery of the packet p to the client. It is enabled when the host h is a member of the reliable multicast group and the packet p is the packet in the *to-be-delivered* buffer with the smallest sequence number. This ordering constraint ensures that the foremost packet from each source is delivered to the client prior to

Figure 4.13 The SRM-REC_h Automaton — Discrete Transitions (Cnt'd)

input $\texttt{process-pkt}_h(p)$
where $type(p) = \texttt{RQST}$
eff **if** $status = \texttt{member}$ **then**
$\langle s_p, i_p \rangle = id(p)$
$\backslash\backslash$ `Only consider proper packets`
if $min\text{-}seqno(s_p) \neq \perp \wedge min\text{-}seqno(s_p) \leq i_p$ **then**
if $\langle s_p, i_p \rangle \in archived\text{-}pkts?$ **then**
if $\langle s_p, i_p \rangle \notin scheduled\text{-}repls?$
$\wedge \langle s_p, i_p \rangle \notin pending\text{-}repls?$
then
$\backslash\backslash$ `Schedule a new reply`
$q := sender(p)$
$d_{repl} := dist?(q)$
$t_{repl} :\in now + [D_1 d_{repl}, (D_1 + D_2) d_{repl}]$
$scheduled\text{-}repls \cup= \{\langle s_p, i_p, t_{repl}, q \rangle\}$
else
if $h \neq s_p$ **then**
if $\langle s_p, i_p \rangle \notin scheduled\text{-}rqsts?$ **then**
$\backslash\backslash$ `Schedule a backed-off request`
$k_r := 2;\ d_r := dist?(s_p)$
$t_r :\in now + 2^{k_r - 1}[C_1 d_r, (C_1 + C_2) d_r]$
$scheduled\text{-}rqsts \cup= \{\langle s_p, i_p, t_r, k_r \rangle\}$
$\backslash\backslash$ `Pkt request has been scheduled`
$to\text{-}be\text{-}requested? \setminus= \{\langle s_p, i_p \rangle\}$
$\backslash\backslash$ `A request becomes pending`
$pending\text{-}rqsts \setminus= \{\langle s_p, i_p, t_* \rangle \mid t_* \in \mathbb{R}^{\geq 0}\}$
$t_r := now + 2^{k_r - 1} C_3 d_r$
$pending\text{-}rqsts \cup= \{\langle s_p, i_p, t_r \rangle\}$
else
if $\langle s_p, i_p \rangle \notin pending\text{-}rqsts?$ **then**
$\backslash\backslash$ `Backoff scheduled request`
choose $t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}$
where $\langle s_p, i_p, t, k \rangle \in scheduled\text{-}rqsts$
$scheduled\text{-}rqsts \setminus= \{\langle s_p, i_p, t, k \rangle\}$
$k_r := k + 1;\ d_r := dist?(s_p)$
$t_r :\in now + 2^{k_r - 1}[C_1 d_r, (C_1 + C_2) d_r]$
$scheduled\text{-}rqsts \cup= \{\langle s_p, i_p, t_r, k_r \rangle\}$
$\backslash\backslash$ `A request becomes pending`
$pending\text{-}rqsts \setminus= \{\langle s_p, i_p, t_* \rangle \mid t_* \in \mathbb{R}^{\geq 0}\}$
$t_r := now + 2^{k_r - 1} C_3 d_r$
$pending\text{-}rqsts \cup= \{\langle s_p, i_p, t_r \rangle\}$
$\backslash\backslash$ `Discover any trailing missing packets`
if $h \neq s_p \wedge max\text{-}seqno(s_p) < i_p$ **then**
$to\text{-}be\text{-}requested? \cup=$
$\{\langle s_p, i \rangle \mid i \in \mathbb{N}, max\text{-}seqno(s_p) < i < i_p\}$
$max\text{-}seqno(s_p) := i_p$

input $\texttt{process-pkt}_h(p)$
where $type(p) = \texttt{REPL}$
eff **if** $status = \texttt{member}$ **then**
$\langle s_p, i_p \rangle = id(p)$
$\backslash\backslash$ `Only consider proper packets`
if $min\text{-}seqno(s_p) \neq \perp \wedge min\text{-}seqno(s_p) \leq i_p$ **then**
$\backslash\backslash$ `A reply becomes pending`
$pending\text{-}repls \setminus= \{\langle s_p, i_p, t_* \rangle \mid t_* \in \mathbb{R}^{\geq 0}\}$
$t_{repl} := now + D_3\, dist?(requestor(p))$
$pending\text{-}repls \cup= \{\langle s_p, i_p, t_{repl} \rangle\}$
$\backslash\backslash$ `Archive and deliver packet`
if $h \neq s_p \wedge \langle s_p, i_p \rangle \notin archived\text{-}pkts?$ **then**
$archived\text{-}pkts \cup= \{\langle strip(p), now \rangle\}$
if $h \neq s_p$ **then** $to\text{-}be\text{-}delivered \cup= \{strip(p)\}$
$\backslash\backslash$ `Pkt need not be requested`
$to\text{-}be\text{-}requested? \setminus= \{\langle s_p, i_p \rangle\}$
$\backslash\backslash$ `Cancel any scheduled requests and replies`
$scheduled\text{-}rqsts \setminus= \{\langle s_p, i_p, t, k \rangle \mid t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}\}$
$scheduled\text{-}repls \setminus= \{\langle s_p, i_p, t, q \rangle \mid t \in \mathbb{R}^{\geq 0}, q \in H\}$
$\backslash\backslash$ `Cancel any pending requests`
$pending\text{-}rqsts \setminus= \{\langle s_p, i_p, t \rangle \mid t \in \mathbb{R}^{\geq 0}\}$
$\backslash\backslash$ `Discover any trailing missing packets`
if $h \neq s_p \wedge max\text{-}seqno(s_p) < i_p$ **then**
$to\text{-}be\text{-}requested? \cup=$
$\{\langle s_p, i \rangle \mid i \in \mathbb{N}, max\text{-}seqno(s_p) < i < i_p\}$
$max\text{-}seqno(s_p) := i_p$

any other packets from the particular source. Its effects are to remove the packet p from the *rm-recv-buff* buffer.

Each output action $\texttt{rec-msend}_h(p)$, for $p \in P_{\text{SRM}}$, hands off the packet p from SRM-REC_h to SRM-IPBUFF_h so that it may subsequently be multicast by SRM-IPBUFF_h using the underlying IP multicast service. The precondition of the $\texttt{rec-msend}_h(p)$ action is that the host h is a member of the reliable multicast group and p is in the *msend-buff* buffer. Its effects are to remove p from the *msend-buff* buffer.

Internal Actions

Each internal action $\texttt{schdl-rqst}_h(s, i)$, for $s \in H, s \neq h, i \in \mathbb{N}$, schedules a request for the packet $\langle s, i \rangle$. The precondition of the $\texttt{schdl-rqst}_h(s, i)$ action is that the host h is a member of the reliable multicast group and the tuple $\langle s, i \rangle$ is in the set *to-be-requested?*. The effects of the

$\texttt{schdl-rqst}_h(s,i)$ action are to schedule a new request for a point in time in the future that is chosen uniformly within the interval $now + [C_1 d_r, (C_1 + C_2)d_r]$, with $d_r = dist?(s)$, and to remove the tuple $\langle s, i\rangle$ from the set *to-be-requested?*.

The internal action $\texttt{send-sess}_h$ models the expiration of the session packet transmission timeout. The precondition of $\texttt{send-sess}_h$ is that the host h is a member of the reliable multicast group and that the transmission time of the next session packet has arrived; that is, $status = \texttt{member}$ and $now = rep\text{-}deadline$. $\texttt{send-sess}_h$ composes a session packet and adds it to the buffer *msend-buff*. The operation $comp\text{-}sess\text{-}pkt(h, now, dist\text{-}rprt, max\text{-}seqno)$ composes a SESS packet from h. Moreover, $\texttt{send-sess}_h$ schedules the transmission of the next session packet for a point in time that is SESS-PERIOD time units in the future by reseting the variable *rep-deadline* to the value $now + \texttt{SESS-PERIOD}$.

Each internal action $\texttt{send-rqst}_h(s,i)$, for $s \in H, i \in \mathbb{N}$, models the expiration of the transmission timeout of a scheduled request for the packet $\langle s, i\rangle$. The precondition of $\texttt{send-rqst}_h(s,i)$ is that the host h is a member of the reliable multicast group and a previously scheduled request for the packet $\langle s, i\rangle$ has expired; that is, there is a tuple $\langle s, i, t, k\rangle$ in *scheduled-rqsts* such that $t = now$. Let the tuple $\langle s, i, t, k\rangle$ be the element of *scheduled-rqsts* corresponding to the packet $\langle s, i\rangle$. $\texttt{send-rqst}_h(s,i)$ composes a request packet and adds it to the buffer *msend-buff*. The operation $comp\text{-}rqst\text{-}pkt(h, \langle s, i\rangle)$ composes a RQST packet from h for the packet $\langle s, i\rangle$.

Moreover, the request $\langle s, i, t, k\rangle$ is backed off and a request for the given ADU becomes pending. The timeout timer of the rescheduled request is set to a point it time in the future that is chosen uniformly within the interval $now + 2^{k_r - 1}[C_1 d_r, (C_1 + C_2)d_r]$ and the back-off abstinence timeout of the pending request is set to $now + 2^{k_r - 1}C_3 d_r$, with $k_r = k + 1$ and $d_r = dist?(s)$.

Each internal action $\texttt{send-repl}_h(s,i)$, for $s \in H, i \in \mathbb{N}$, models the expiration of the transmission timeout of a scheduled reply for the packet $\langle s, i\rangle$. The precondition of $\texttt{send-repl}_h(s,i)$ is that the host h is a member of the reliable multicast group and a previously scheduled reply for the packet $\langle s, i\rangle$ has expired; that is, there is a tuple $\langle s, i, t, r\rangle$ in *scheduled-repls* such that $t = now$. Let the tuple $\langle s, i, t, r\rangle$ be the element of *scheduled-repls* corresponding to the packet $\langle s, i\rangle$. $\texttt{send-repl}_h(s,i)$ composes a reply packet and adds it to the buffer *msend-buff*. The operation $comp\text{-}repl\text{-}pkt(h, r, p)$ composes a REPL packet from h for the packet p. This reply is annotated with the host r that induced the particular reply for p.

Moreover, the tuple corresponding to $\langle s, i\rangle$ is removed from the set *scheduled-repls* and a tuple corresponding to $\langle s, i\rangle$ is added to the set *pending-repls*. The reply abstinence timeout of this pending reply is set to $now + D_3\, dist?(r)$. This pending reply prevents the scheduling of replies for the given ADU for $D_3\, dist?(r)$ time units.

Time Passage

The action $\nu(t)$ models the passage of t time units. If the host h has crashed, then time is allowed to elapse. Otherwise, time is prevented from elapsing while either there are packets in the delivery and IP multicast transmission buffers or there are packets which have been declared missing but for which a request has yet to be scheduled; that is, while either of the buffers *to-be-delivered*, *msend-buff*, or *to-be-requested?* is non-empty. Furthermore, time is prevented from elapsing past the transmission deadline of any scheduled session, request, or reply packets.

4.3.5 The IP Multicast Component — IPmcast

In this section, we give an abstract specification of the IP multicast service; the IP primitive that provides best-effort point to multi-point communication. In order to simplify the presentation, we

Figure 4.14 The IPmcast Automaton — Signature

Actions:

input	**output**
`crash`$_h$, for $h \in H$	`mjoin-ack`$_h$, for $h \in H$
`mjoin`$_h$, for $h \in H$	`mleave-ack`$_h$, for $h \in H$
`mleave`$_h$, for $h \in H$	`mrecv`$_h(p)$, for $h \in H, p \in P_{\text{IPmcast-Client}}$
`msend`$_h(p)$, for $h \in H, p \in P_{\text{IPmcast-Client}}$	`mdrop`(p, H_d), for $p \in P_{\text{IPmcast-Client}}, H_d \subseteq H$
internal	**time-passage**
`mgrbg-coll`(pkt), for $pkt \in P_{\text{IPmcast}}$	$\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

assume that only a single multicast group exists. Furthermore, we abstract away the specifics of the underlying protocols that collectively provide the IP multicast service. In our model, hosts join, leave, and send data packets to the IP multicast group by issuing join and leave requests and by multicasting data packets, respectively. Following the initial service model of IP multicast, a host need not be a member of the IP multicast group to send messages addressed to the group. However, a host must join the IP multicast group in order to receive packets addressed to the IP multicast group. The IP multicast service guarantees that only hosts who are members of the IP multicast group actually receive IP multicast packets.

Figures 4.14 and 4.15 present the signature, variables, and discrete transitions of the the IPmcast timed I/O automaton; an abstract specification of the IP multicast service.

Variables

The variable *now* $\in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of IPmcast. Each variable $status(h) \in$ *IPmcast-Status*, for $h \in H$, denotes the IP multicast membership status of the host h. The value `idle` indicates that h is *idle* with respect to the IP multicast group; that is, it is neither a member, nor in the process of joining or leaving the IP multicast group. The value `joining` indicates that h is in the process of joining the IP multicast group; that is, the client has issued a request to join the IP multicast group and is awaiting an acknowledgment of this join request from the IP multicast service. The value `leaving` indicates that h is in the process of leaving the IP multicast group; that is, the client has issued a request to leave the IP multicast group and is awaiting an acknowledgment of this leave request from the IP multicast service. The value `member` indicates that h is a member of the IP multicast group. The value `crashed` indicates that h has crashed. When the host h has crashed, none of the input actions pertaining to h affect the state of IPmcast and none of the locally controlled actions pertaining to h are enabled. While the host h has not crashed, we say that it is *operational*.

The set $mpkts \subseteq P_{\text{IPmcast}}$ is comprised of the tuples that track the transmission progress of the packets transmitted during the particular execution of IPmcast. Of course, the size of the intended delivery set of each transmission progress tuple decreases monotonically as the hosts it consists of may leave the IP multicast group or crash.

Derived Variables

The derived variable $up \subseteq H$ is the set of hosts that are operational; that is, the set of hosts that have not yet crashed. The derived variable $idle \subseteq H$ is a set of hosts that are idle with respect to the IP multicast group. The derived variable $joining \subseteq H$ is a set of hosts that are in the process of joining the IP multicast group. The derived variable $leaving \subseteq H$ is a set of hosts that are in the process of leaving the IP multicast group. The derived variable $members \subseteq H$ is a set of hosts that are members of the IP multicast group.

Figure 4.15 The IPMCAST automaton — Variables and Discrete Transitions

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status(h) \in \mathit{IPmcast\text{-}Status}$, for all $h \in H$,
initially $status(h) = \texttt{idle}$, for all $h \in H$
$mpkts \subseteq P_{\text{IPMCAST}}$, initially $mpkts = \emptyset$

Derived Variables:

$up = \{h \in H | status(h) \neq \texttt{crashed}\}$
$idle = \{h \in H | status(h) = \texttt{idle}\}$
$joining = \{h \in H | status(h) = \texttt{joining}\}$
$leaving = \{h \in H | status(h) = \texttt{leaving}\}$
$members = \{h \in H | status(h) = \texttt{member}\}$

Discrete Transitions:

input $\texttt{crash}_h$
eff **if** $h \in up$ **then**
$status(h) := \texttt{crashed}$
foreach $pkt \in mpkts$ **do:**
$intended(pkt) \setminus= \{h\}$

input $\texttt{mjoin}_h$
eff **if** $h \in idle$ **then**
$status(h) := \texttt{joining}$

input $\texttt{mleave}_h$
eff **if** $h \in joining \cup members$ **then**
$status(h) := \texttt{leaving}$
foreach $pkt \in mpkts$ **do:**
$intended(pkt) \setminus= \{h\}$

input $\texttt{msend}_h(p)$
eff **if** $h \in up$ **then**
$mpkts \cup= \{\langle p, members, \{h\}, \emptyset\rangle\}$

internal $\texttt{mgrbg-coll}(p)$
choose $pkt \in P_{\text{IPMCAST}}$
pre $pkt \in mpkts \wedge p = strip(pkt)$
$\wedge intended(pkt) \subseteq (completed(pkt) \cup dropped(pkt))$
eff $mpkts \setminus= \{pkt\}$

output $\texttt{mjoin-ack}_h$
pre $h \in joining$
eff $status(h) := \texttt{member}$

output $\texttt{mleave-ack}_h$
pre $h \in leaving$
eff $status(h) := \texttt{idle}$

output $\texttt{mrecv}_h(p)$
choose $pkt \in P_{\text{IPMCAST}}$
pre $pkt \in mpkts \wedge p = strip(pkt)$
$\wedge h \neq source(p) \wedge h \in members \setminus dropped(pkt)$
eff $completed(pkt) \cup= \{h\}$

output $\texttt{mdrop}(p, H_d)$
choose $pkt \in P_{\text{IPMCAST}}$
pre $pkt \in mpkts \wedge p = strip(pkt)$
$\wedge H_d \subseteq members \setminus (completed(pkt) \cup dropped(pkt))$
eff $dropped(pkt) \cup= H_d$

time-passage $\nu(t)$
pre None
eff $now := now + t$

Input Actions

Each input action $\texttt{crash}_h$, for $h \in H$, models the crashing of the host h. The $\texttt{crash}_h$ action records the fact that h has crashed by setting the $status(h)$ variable to $\texttt{crashed}$. Moreover, the $\texttt{crash}_h$ action removes the host h from the intended delivery set of any packet in the set of pending packets $mpkts$.

The input action $\texttt{mjoin}_h$ models the request of the client at h to join the IP multicast group. The $\texttt{mjoin}_h$ action is effective only while the host is idle with respect to the IP multicast group. When effective, the $\texttt{mjoin}_h$ action sets the $status(h)$ variable to $\texttt{joining}$ so as to record that the host h has initiated the process of joining the IP multicast group. If the client is either a member of or in the process of joining the IP multicast group, then the $\texttt{mjoin}_h$ action is superfluous. If the client is already in the process of leaving the group, then the $\texttt{mjoin}_h$ action is discarded so as to allow the process of leaving the IP multicast group to complete.

The input action $\texttt{mleave}_h$ models the request of the client at h to leave the IP multicast group. The $\texttt{mleave}_h$ action is effective only while the host is either a member of or in the process of joining the IP multicast group. When effective, the $\texttt{mleave}_h$ action sets the $status(h)$ variable to $\texttt{leaving}$ so as to record that the host h has initiated the process of leaving the IP multicast group. Moreover, the $\texttt{mleave}_h$ action removes the host h from the intended delivery set of any packet in the set of pending packets $mpkts$. Leave requests overrule join requests; that is, when a $\texttt{mleave}_h$ action is performed while the host h is in the process of joining the IP multicast group, its effects are to abort the process of joining and to initiate the process of leaving the IP multicast group. If the client is either idle with respect to or already in the process of leaving the IP multicast group, then the $\texttt{mleave}_h$ action is superfluous.

The input action $\texttt{msend}_h(p)$ models the transmission by the client at h of the packet p using the IP multicast service. The $\texttt{msend}_h(p)$ action is effective only if the client is operational; recall that a client need not be a member of the IP multicast group to multicast packets using the IP multicast service. The effects of the $\texttt{msend}_h(p)$ action are to add a tuple corresponding to the transmission of the packet p to *mpkts*. This tuple is initialized as follows: its intended delivery set is initialized to the current members of the IP multicast group, its completed delivery set is initialized to the host h as if the packet p has already been delivered to the client at the host h, and its dropped set is initialized to the empty set.

Output Actions

The output action $\texttt{mjoin-ack}_h$ acknowledges the join request of the client at h. The $\texttt{mjoin-ack}_h$ action is enabled only when the host is in the process of joining the IP multicast group. Its effects are to set the *status*(h) variable to `member` so as to indicate that the client at h has become a member of the IP multicast group.

The output action $\texttt{mleave-ack}_h$ acknowledges the leave request of the client at h. The action $\texttt{mleave-ack}_h$ is enabled when the host is in the process of leaving the IP multicast group. Its effects are to set the *status*(h) variable to `idle` so as to indicate that the client at h has become idle with respect to the IP multicast group.

The output action $\texttt{mrecv}_h(p)$ models the delivery of the packet p to the client at h. The $\texttt{mrecv}_h(p)$ action is enabled when there is a transmission tuple *pkt* in *mpkts* pertaining to p, the host h is not the source of p, and h is both a member of the IP multicast group and absent from the dropped set of *pkt*. We thus presume that the IP multicast communication service does not deliver packets to their respective sources and that packets may be delivered in duplicate to members of the IP multicast group. The effects of the $\texttt{mrecv}_h(p)$ action are to add the host h to the completed delivery set of p's transmission progress tuple *pkt*.

The output action $\texttt{mdrop}(p, H_d)$, for any $p \in P_{\text{IPMCAST-CLIENT}}$ and $H_d \subseteq H$, models the drop of the packet p on a link of the underlying IP multicast tree whose descendants are the hosts in the set H_d. The $\texttt{mdrop}(p, H_d)$ action is enabled when p is a pending packet and H_d is comprised of members of the IP multicast group for which the delivery of the packet p has neither completed, nor failed due to prior packet drops. The $\texttt{mdrop}(p, H_d)$ action adds the hosts comprising H_d to the dropped set of the transmission progress tuple *pkt* in *mpkts* pertaining to p.

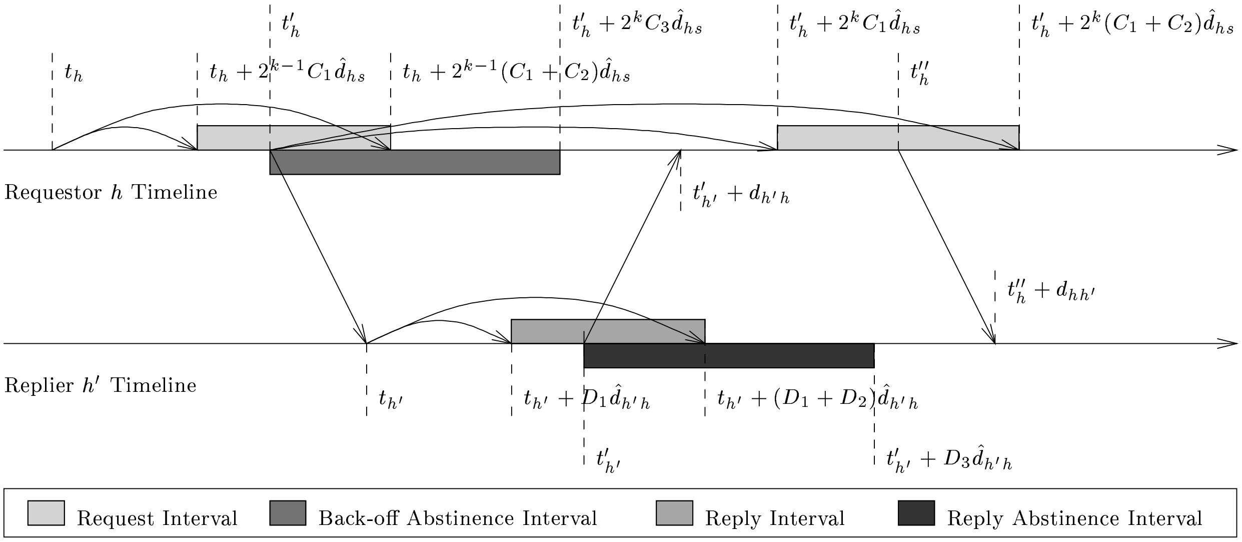
Internal Actions

The internal action $\texttt{mgrbg-coll}(p)$ models the garbage collection of the packet p. A packet p may only be garbage collected after all the hosts comprising its intended delivery set either receive the packet or suffer a loss that prevents the packet from being forwarded to them. The effects of the $\texttt{mgrbg-coll}(p)$ action are to remove the transmission progress tuple *pkt* pertaining to p from the set *mpkts*.

Time Passage

The time-passage action $\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$, models the passage of t time units. The action $\nu(t)$ is enabled at any point in time and increments the variable *now* by t time units.

Figure 4.16 Timing Diagram of SRM's Loss Recovery Scheme



4.4 Analysis of SRM

In this section, we show that our model of the SRM protocol actually implements our reliable multicast service specification of Chapter 3. According to our model's architecture, the SRM model involves the SRM processes at each host and the underlying IP multicast service; that is, the automaton $\prod_{h \in H} \mathrm{SRM}_h \times \mathrm{IP}\textsc{mcast}$. We define the automaton SRM to be the composition $\prod_{h \in H} \mathrm{SRM}_h \times \mathrm{IP}\textsc{mcast}$ after hiding all output actions that are not output actions of the specification $\mathrm{RM}(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$; that is, $\mathrm{SRM} = \mathit{hide}_\Phi(\prod_{h \in H} \mathrm{SRM}_h \times \mathrm{IP}\textsc{mcast})$, with $\Phi = \mathit{out}(\prod_{h \in H} \mathrm{SRM}_h \times \mathrm{IP}\textsc{mcast}) \backslash \mathit{out}(\mathrm{RM}(\Delta))$. SRM is parameterized by the parameters listed in Figure 4.1.

We let SRM_I and $\mathrm{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, denote the implementation and the specification of the reliable multicast service each composed with all the client automata; that is, $\mathrm{SRM}_I = \mathrm{SRM} \times \textsc{rmClients}$ and $\mathrm{RM}_S(\Delta) = \mathrm{RM}(\Delta) \times \textsc{rmClients}$.

We proceed by presenting several constraints regarding the request and reply scheduling parameters of SRM. Then, we define some history variables that facilitate the proof that our model of SRM implements the abstract reliable multicast service specification. We define a relation between the states of the SRM protocol and the reliable multicast service and prove that this relation is indeed a timed forward simulation relation. We conclude by showing that the SRM protocol, under certain constraints, guarantees the eventual and time-bounded delivery guarantees defined in Chapter 3.

4.4.1 Request and Reply Scheduling Parameter Constraints

Figure 4.16 illustrates the behavior of SRM's packet loss recovery scheme. In particular, for any $k \in \mathbb{N}^+$, it depicts the transmission of a k-th round request by h, the scheduling of a $k+1$-st round request by h, and the scheduling of a reply to h's k-th round request by a host h'. t_h is the point in time at which h schedules its k-th round request, t'_h is the point in time for which h schedules its k-th round request, $t_{h'}$ is the point in time h' receives h's k-th round request, and $t'_{h'}$ is the point in time for which h' schedules its reply to h's k-th round request. $\hat{d}_{hs}$ is half of h's RTT estimate to the source s of the packet being recovered, $d_{hh'}$ and $d_{h'h}$ are the actual transmission latencies between h and h', and $\hat{d}_{h'h}$ is half of the RTT estimate of h' to h.

SRM must ensure that the back-off abstinence intervals do not overlap with request intervals. From Figure 4.16, this requirement is enforced by imposing the parameter constraint $C_3 < C_1$. Moreover, SRM must ensure that requestors schedule their retransmission requests such that they succeed the reception of replies pertaining to prior recovery rounds. Prematurely transmitting requests would result in wasteful recovery traffic. From Figure 4.16, this requirement corresponds to the satisfaction of the inequalities $d_{hh'} + (D_1 + D_2)\hat{d}_{h'h} + d_{h'h} < 2^k C_1 \hat{d}_{hs}$, for $k \in \mathbb{N}^+$. Presuming that inter-host transmission latencies are fixed and symmetric and that SRM's inter-host RTT estimates are accurate, these inequalities are satisfied if $D_1 + D_2 + 2 < 2C_1$. Finally, SRM must also ensure that a particular round's requests are not discarded by potential repliers because they are received during the repliers' abstinence periods pertaining to the prior recovery round. From Figure 4.16, this requirement corresponds to the satisfaction of the inequalities $d_{hh'} + (D_1 + D_2)\hat{d}_{h'h} + D_3\hat{d}_{h'h} < 2^k C_1 \hat{d}_{hs} + d_{hh'}$, for $k \in \mathbb{N}^+$. Presuming that inter-host transmission latencies are fixed and symmetric and that SRM's inter-host RTT estimates are accurate, these inequalities are satisfied if $D_1 + D_2 + D_3 < 2C_1$.

The following assumption summarizes the constraints on SRM's parameters.

Assumption 4.1 *SRM_I's parameters C_1, C_2, C_3, D_1, D_2, and D_3 satisfy the following constraints: $C_3 < C_1$, $D_1 + D_2 + 2 < 2C_1$, and $D_1 + D_2 + D_3 < 2C_1$.*

To our knowledge, these constraints on SRM's request/reply scheduling parameters, or even similar ones, have not been expressed to date. In fact, most analyses and simulations presume that no recovery packets are lost; that is, they presume that the initial recovery round is always successful. Our timing analysis illustrates that if the parameters are chosen arbitrarily it is possible to cause either superfluous requests and replies or the failure of a recovery round due to replier abstinence. Although in practice, due to inaccurate inter-host RTT estimates and varying and non-symmetric inter-host transmission latencies, superfluous traffic and/or recovery round failure may indeed be unavoidable, it is still important to realize their tie to SRM's parameters.

4.4.2 History Variables

Figure 4.17 introduces history and derived history variables for the automata SRM-REC_h and SRM, respectively.

The history variables of the SRM-REC_h automata, for $h \in H$, are the variables *trans-time*(p), for all $p \in P_{\text{RM-CLIENT}}[h]$, *expected*$(h') \subseteq H \times \mathbb{N}$, for $h' \in H$, and *delivered*$(h') \subseteq H \times \mathbb{N}$, for $h' \in H$. Each *trans-time*(p) variable, for $p \in P_{\text{RM-CLIENT}}[h]$, records the transmission time of the packet p by the host h. Each *expected*(h') variable , for $h' \in H$, is comprised of the identifiers of the packets from h' that the host h expects to deliver since it last joined the reliable multicast group. Each *delivered*(h') variable, for $h' \in H$, is comprised of the identifiers of the packets from h' that the host h has already delivered since it last joined the reliable multicast group. Figure 4.18 specifies how the actions of SRM-REC_h affect these history variables.

The derived history variables of SRM are the set of identifiers of all packets sent since the beginning of the execution, *sent-pkts*, the intended delivery set of p, *intended*(p), for all $p \in P_{\text{RM-CLIENT}}$, the completed delivery set of p, *completed*(p), for all $p \in P_{\text{RM-CLIENT}}$, and the set of active packets, *active-pkts*.

4.4.3 Correctness Analysis Preliminaries

In this section, we present several preliminary invariants and lemmas that are later used in the analysis of the SRM_I automaton.

Figure 4.17 History and Derived History Variables

History Variables of SRM-REC$_h$**:**

$\mathit{trans\text{-}time}(p) \in \mathbb{R}^{\geq 0} \cup \bot$, for all $p \in P_{\text{RM-CLIENT}}[h]$, initially $\mathit{trans\text{-}time}(p) = \bot$, for all $p \in P_{\text{RM-CLIENT}}[h]$
$\mathit{expected}(h') \subseteq H \times \mathbb{N}$, for all $h' \in H$, initially $\mathit{expected}(h') = \emptyset$, for all $h' \in H$
$\mathit{delivered}(h') \subseteq H \times \mathbb{N}$, for all $h' \in H$, initially $\mathit{delivered}(h') = \emptyset$, for all $h' \in H$

Derived History Variables of SRM**:**

$\mathit{sent\text{-}pkts} = \{p \in P_{\text{RM-CLIENT}} \mid \mathit{trans\text{-}time}(p) \neq \bot\}$
$\mathit{sent\text{-}pkts?} = \{\langle s,i \rangle \in H \times \mathbb{N} \mid \exists\, p \in \mathit{sent\text{-}pkts} : id(p) = \langle s,i \rangle\}$
$\mathit{sent\text{-}pkts?}(h) = \{\langle s,i \rangle \in H \times \mathbb{N} \mid s = h \wedge \exists\, p \in \mathit{sent\text{-}pkts} : id(p) = \langle s,i \rangle\}$, for all $h \in H$
$\mathit{intended}(p) = \{h \in H \mid id(p) \in \text{SRM-REC}_h.\mathit{expected}(\mathit{source}(p))\}$, for all $p \in P_{\text{RM-CLIENT}}$
$\mathit{completed}(p) = \{h \in H \mid id(p) \in \text{SRM-REC}_h.\mathit{delivered}(\mathit{source}(p))\}$, for all $p \in P_{\text{RM-CLIENT}}$
$\mathit{active\text{-}pkts} = \{p \in P_{\text{RM-CLIENT}} \mid p \in \mathit{sent\text{-}pkts} \wedge \mathit{intended}(p) \cap \mathit{completed}(p) \neq \emptyset\}$

Figure 4.18 SRM-REC$_h$ History Variable Assignments

```
input crash_h
eff ...
    foreach h' ∈ H do:
      expected(h') := ∅
      delivered(h') := ∅

input rm-leave_h
eff if status ≠ crashed then
      Reinitialize all variables except now.
      foreach h' ∈ H do:
        expected(h') := ∅
        delivered(h') := ∅

input rm-send_h(p)
eff ...
      \\ Record foremost DATA packet
      if min-seqno(s_p) =⊥ then
        ...
        expected(h) := suffix(p)
      ...
      if max-seqno(s_p) =⊥
        ∨i_p = max-seqno(s_p) + 1
      then
        ...
        trans-time(p) := now
        delivered(h) ∪= {id(p)}

output rm-recv_h(p)
pre ...
eff ...
    ⟨s_p, i_p⟩ := id(p)
    if expected(s_p) = ∅ then
      expected(s_p) := suffix(p)
    delivered(s_p) ∪= {id(p)}
```

Lemma 4.1 (IP Multicast Transmission Integrity) *For any timed trace β of* IPMCAST*, it is the case that any* $\texttt{mrecv}_h(p)$ *action, for $h \in H$ and $p \in P_{\text{IPMCAST-CLIENT}}$, in β is preceded in β by a* $\texttt{msend}_{h'}(p)$ *action, for some $h' \in H$.*

Proof: Let α be any timed execution of IPMCAST such that $\beta = \mathit{ttrace}(\alpha)$. Consider a particular occurrence of an action $\texttt{mrecv}_h(p)$ in α, for $h \in H$ and $p \in P_{\text{IPMCAST-CLIENT}}$. Let $(u, \texttt{mrecv}_h(p), u') \in \mathit{trans}(\text{IPMCAST})$ be the discrete transition in α corresponding to the particular occurrence of the action $\texttt{mrecv}_h(p)$ in α. From the precondition of $\texttt{mrecv}_h(p)$, it is the case that there is a packet $\mathit{pkt} \in u.\mathit{mpkts}$, such that $p = \mathit{strip}(\mathit{pkt})$. However, such a packet may be added to mpkts only by the occurrence of an action $\texttt{msend}_{h'}(p)$, for some $h \in H$. It follows that the occurrence of any action $\texttt{mrecv}_h(p)$ in α is preceded by the occurrence of an action $\texttt{msend}_{h'}(p)$, for some $h' \in H$. ❐

Invariant 4.1 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case that* $u.\mathit{window?}(h') \subseteq u.\mathit{proper?}(h')$.

Proof: Follows directly from the definitions of the derived variables SRM-REC$_h.\mathit{window?}(h')$ and SRM-REC$_h.\mathit{proper?}(h')$. ❐

Invariant 4.2 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, if $u.\mathit{status} \neq$* $\texttt{member}$*, then* $u.\mathit{expected}(h') = \emptyset$ *and* $u.\mathit{delivered}(h') = \emptyset$.

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.status = \texttt{idle}$, $u.expected(h') = \emptyset$, and $u.delivered(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables $status$, $expected(h')$, and $delivered(h')$.

❒ $\texttt{crash}_h$: the action $\texttt{crash}_h$ sets the variable $status$ to $\texttt{crashed}$ and the variables $expected(h')$ and $delivered(h')$ to $\emptyset$. Thus, the invariant assertion is satisfied in u.

❒ $\texttt{rm-join-ack}_h$: if $u_k.status \neq \texttt{crashed}$, then the action $\texttt{rm-join-ack}_h$ sets the variable $status$ to $\texttt{member}$. Thus, the invariant assertion is satisfied in u. Otherwise, if $u_k.status = \texttt{crashed}$, then the action $\texttt{rm-join-ack}_h$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

❒ $\texttt{rm-leave}_h$: if $u_k.status \neq \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ sets the variable $status$ to $\texttt{idle}$ and the $expected(h')$ and $delivered(h')$ variables to $\emptyset$. Thus, the invariant assertion is satisfied in u. Otherwise, if $u_k.status = \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$: first, consider the case where $\neg(u_k.status = \texttt{member} \wedge h = source(p))$. In this case, $\texttt{rm-send}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

  Second, consider the case where $u_k.status = \texttt{member}$ and $h = source(p)$. Since $u_k.status = \texttt{member}$ and the $\texttt{rm-send}_h(p)$ does not affect the $status$ variable, it follows that $u.status = \texttt{member}$. Thus, the invariant assertion is satisfied in u.

❒ $\texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$: the precondition of the action $\texttt{rm-recv}_h(p)$ implies that $u_k.status = \texttt{member}$. Since the $\texttt{rm-recv}_h(p)$ does not affect the $status$ variable, it follows that $u.status = \texttt{member}$. Thus, the invariant assertion is satisfied in u.

❒

Invariant 4.3 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case that:*

1. $u.min\text{-}seqno(h') \neq \perp \Leftrightarrow u.max\text{-}seqno(h') \neq \perp$ *and*
2. $u.min\text{-}seqno(h') \neq \perp \ \Rightarrow u.min\text{-}seqno(h') \leq u.max\text{-}seqno(h')$.

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.min\text{-}seqno(h) = \perp$ and $u.max\text{-}seqno(h) = \perp$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables $min\text{-}seqno(h')$ and $max\text{-}seqno(h')$.

❒ $\texttt{rm-leave}_h$: if $u_k.status \neq \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ sets the variables $min\text{-}seqno(h')$ and $max\text{-}seqno(h')$ to $\perp$. Thus, the induction assertion is satisfied in u. Otherwise, if $u_k.status = \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of $\texttt{rm-send}_h(p)$ by cases. First, consider the case where $\neg(u_k.status = \texttt{member} \wedge h =$

s_p). In this case, $\texttt{rm-send}_h(p)$ does not affect the variables *min-seqno*(h') and *max-seqno*(h'). Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

Second, consider the case where $u_k.status = \texttt{member}$ and $h = s_p$. Since $s_p = h'$, it follows that $h = h' = s_p$. If p is the foremost packet from s_p, that is, $u_k.min\text{-}seqno(s_p) =\perp$, then the $\texttt{rm-send}_h(p)$ action sets both *min-seqno*(h') and *max-seqno*(h') to i_p. It follows that $u.min\text{-}seqno(h') \neq\perp$, $u.max\text{-}seqno(h') \neq\perp$, and $u.min\text{-}seqno(h') \leq u.max\text{-}seqno(h')$. Thus, the invariant assertion is satisfied in u.

If p is the next packet from s_p, that is, $u_k.max\text{-}seqno(h') \neq\perp$ and $i_p = u_k.max\text{-}seqno(h') + 1$, then the action $\texttt{rm-send}_h(p)$ does not affect *min-seqno*(h') and sets *max-seqno*(h') to i_p; that is, $u.min\text{-}seqno(h') = u_k.min\text{-}seqno(h')$ and $u.max\text{-}seqno(h') = u_k.max\text{-}seqno(h') + 1$. Since $u_k.max\text{-}seqno(h') \neq\perp$, the induction hypothesis implies that $u_k.min\text{-}seqno(h') \neq\perp$ and $u_k.min\text{-}seqno(h') \leq u_k.max\text{-}seqno(h')$.

Since $i_p = u_k.max\text{-}seqno(h') + 1$, it follows that $u_k.max\text{-}seqno(h') < u.max\text{-}seqno(h')$. Since $u_k.min\text{-}seqno(h') \leq u_k.max\text{-}seqno(h')$, it follows that $u.min\text{-}seqno(h') \neq\perp$, $u.max\text{-}seqno(h') \neq\perp$, and $u.min\text{-}seqno(h') \leq u.max\text{-}seqno(h')$.

If p is neither the foremost nor the next packet from s_p, then the action $\texttt{rm-send}_h(p)$ does not affect the variables *min-seqno*(h') and *max-seqno*(h'). Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-pkt}_h(p)$, for $p \in P_{\mathrm{SRM}}$, such that $type(p) = \texttt{SESS}$: First, if $u_k.status \neq \texttt{member}$, then $\texttt{process-pkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.status = \texttt{member}$. In this case, if $h = h'$ or there does not exist $\langle h', i'\rangle \in seqno\text{-}rprts(p)$, for $i' \in \mathbb{N}$, such that $u_k.max\text{-}seqno(h') < i'$, then $\texttt{process-pkt}_h(p)$ affects neither *min-seqno*(h') nor *min-seqno*(h'). Thus, the induction hypothesis implies that the invariant assertion holds in u.

If $h \neq h'$ and there exists $\langle h', i'\rangle \in seqno\text{-}rprts(p)$, for $i' \in \mathbb{N}$, such that $u_k.min\text{-}seqno(h') \neq\perp$ and $u_k.max\text{-}seqno(h') < i'$, then $\texttt{process-pkt}_h(p)$ does not affect *min-seqno*(h') and sets *max-seqno*(h') to i'; that is, $u.min\text{-}seqno(h') = u_k.min\text{-}seqno(h')$ and $u_k.max\text{-}seqno(h') < i' = u.max\text{-}seqno(h')$.

Since $u_k.max\text{-}seqno(h') < i'$ and $u.max\text{-}seqno(h') = i'$, it follows that $u_k.max\text{-}seqno(h') < u.max\text{-}seqno(h')$. From the induction hypothesis, it is the case that $u_k.min\text{-}seqno(h') \leq u_k.max\text{-}seqno(h')$. Thus, it follows that $u.min\text{-}seqno(h') \neq\perp$, $u.max\text{-}seqno(h') \neq\perp$, and $u.min\text{-}seqno(h') \leq u.max\text{-}seqno(h')$, as needed.

❒ $\texttt{process-pkt}_h(p)$, for $p \in P_{\mathrm{SRM}}$, such that $source(p) = h'$ and $type(p) \neq \texttt{SESS}$: letting $\langle s_p, i_p\rangle = id(p)$, we analyze the effects of $\texttt{process-pkt}_h(p)$ by cases. First, if $u_k.status \neq \texttt{member}$, then $\texttt{process-pkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.status = \texttt{member}$. If p is the foremost packet from s_p, that is, $type(p) = \texttt{DATA}$, $h \neq s_p$, and $u_k.min\text{-}seqno(s_p) =\perp$, then the action $\texttt{process-pkt}_h(p)$ sets both *min-seqno*(h') and *max-seqno*(h') to i_p. It follows that $u.min\text{-}seqno(h') \neq\perp$, $u.max\text{-}seqno(h') \neq\perp$, and $u.min\text{-}seqno(h') \leq u.max\text{-}seqno(h')$, as needed.

If p is not the foremost packet from s_p but is proper, that is, $u_k.min\text{-}seqno(s_p) \neq\perp$ and $u_k.min\text{-}seqno(s_p) \leq i_p$, then the action $\texttt{process-pkt}_h(p)$ does not affect *min-seqno*(h') and may increase the value of *max-seqno*(h'). It follows that $u.min\text{-}seqno(h') = u_k.min\text{-}seqno(h')$ and $u_k.max\text{-}seqno(h') \leq u.max\text{-}seqno(h')$. Since $u_k.min\text{-}seqno(s_p) \neq\perp$, the induction hypothesis implies that $u_k.min\text{-}seqno(h') \neq\perp$, $u_k.max\text{-}seqno(h') \neq\perp$, and $u_k.min\text{-}seqno(h') \leq$

$u_k.max\text{-}seqno(h')$. Thus, it follows that $u.min\text{-}seqno(h') \neq \perp$, $u.max\text{-}seqno(h') \neq \perp$, and $u.min\text{-}seqno(h') \leq u.max\text{-}seqno(h')$.

Otherwise, if p is neither the foremost nor a proper packet from s_p, then $\texttt{process-pkt}_h(p)$ does not affect the variables $min\text{-}seqno(h')$ and $max\text{-}seqno(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒

Invariant 4.4 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case that:*

1. $u.delivered(h') \cup u.to\text{-}be\text{-}delivered?(h') \subseteq u.archived\text{-}pkts?(h')$ *and*
2. $u.status = \texttt{member} \Rightarrow u.delivered(h') \cup u.to\text{-}be\text{-}delivered?(h') = u.archived\text{-}pkts?(h')$.

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.status = \texttt{idle}$. Thus, the invariant assertion holds in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u, we consider only the actions that affect the variables $archived\text{-}pkts?(h')$, $delivered(h')$, and $to\text{-}be\text{-}delivered?(h')$.

❒ $\texttt{crash}_h$: the action $\texttt{crash}_h$ sets the variable $status$ to $\texttt{crashed}$, initializes the variables $u.delivered(h')$, for all $h' \in H$, and does not affect the variables $to\text{-}be\text{-}delivered?(h')$ and $archived\text{-}pkts?(h')$, for $h' \in H$; that is, it is the case that $u.delivered(h') \subseteq u_k.delivered(h')$, $u.to\text{-}be\text{-}delivered?(h') = u_k.to\text{-}be\text{-}delivered?(h')$, and $u.archived\text{-}pkts?(h') = u_k.archived\text{-}pkts?(h')$. Thus, the induction hypothesis implies that $u.delivered(h') \cup u.to\text{-}be\text{-}delivered?(h') \subseteq u.archived\text{-}pkts?(h')$.

❒ $\texttt{rm-leave}_h$: if $u_k.status \neq \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ sets the variable $status$ to $\texttt{idle}$. Thus, the invariant assertion holds in u.

Otherwise, if $u_k.status = \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ does not affect the state of SRM-REC$_h$. It follows that $u.status = \texttt{crashed}$. Thus, the invariant assertion holds in u.

❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of $\texttt{rm-send}_h(p)$ by cases. First, if $\neg(u_k.status = \texttt{member} \wedge h = s_p)$, then $\texttt{rm-send}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.status = \texttt{member} \wedge h = s_p$. If p is either the foremost or the next packet from h, then $\texttt{rm-send}_h(p)$ archives p and records it as having been delivered. Thus, the induction hypothesis and the fact that the packet p is both archived and recorded as having been delivered imply that the invariant assertion holds in u.

Otherwise, if p is neither the foremost nor the next packet from h, then the action $\texttt{rm-send}_h(p)$ does not affect the variables $archived\text{-}pkts?(h')$, $delivered(h')$, and $to\text{-}be\text{-}delivered?(h')$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

❒ $\texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $source(p) = h'$: $\texttt{rm-send}_h(p)$ removes $id(p)$ from $to\text{-}be\text{-}delivered?(h')$ and adds it to $delivered(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-pkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of $\texttt{process-pkt}_h(p)$ by cases. First, if $u_k.status \neq \texttt{member}$, then $\texttt{rm-send}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.status = \texttt{member}$. We begin by considering the case where $type(p) \in \{\texttt{DATA}, \texttt{REPL}\}$. In this case, consider the case where p is either the foremost or a proper packet from s_p and $h \neq s_p$. In this case, if p has not already been archived, then $\texttt{process-pkt}_h(p)$ adds $id(p)$ to both $archived\text{-}pkts?(h')$ and $to\text{-}be\text{-}delivered?(h')$. This fact and the induction hypothesis imply that the invariant assertion is satisfied in u. Otherwise, if p has already been archived, then $\texttt{process-pkt}_h(p)$ adds $id(p)$ to $to\text{-}be\text{-}delivered?(h')$ only. Since $id(p) \in u_k.archived\text{-}pkts?(h')$ and $\texttt{process-pkt}_h(p)$ does not affect $archived\text{-}pkts$, it follows that $u.archived\text{-}pkts?(h') = u_k.archived\text{-}pkts?(h')$ and, thus, $id(p) \in u.archived\text{-}pkts?(h')$. Moreover, since $\texttt{process-pkt}_h(p)$ adds $id(p)$ to $to\text{-}be\text{-}delivered?(h')$, it follows that $u.to\text{-}be\text{-}delivered?(h') = u_k.to\text{-}be\text{-}delivered?(h') \cup \{id(p)\}$. From the induction hypothesis, it is the case that $u_k.archived\text{-}pkts?(h') = u_k.delivered(h') \cup u_k.to\text{-}be\text{-}delivered?(h')$. Since $\texttt{process-pkt}_h(p)$ does not affect $delivered(h')$, it follows that the invariant assertion holds in u.

Otherwise, if either p is neither the foremost nor a proper packet from s_p or $h = s_p$, $\texttt{process-pkt}_h(p)$ does not affect $archived\text{-}pkts?(h')$, $delivered(h')$, and $to\text{-}be\text{-}delivered?(h')$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

If $type(p) \in \{\texttt{RQST}, \texttt{SESS}\}$, then the action $\texttt{process-pkt}_h(p)$ does not affect the variables $archived\text{-}pkts?(h')$, $delivered(h')$, and $to\text{-}be\text{-}delivered?(h')$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

❐

Invariant 4.5 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$.*

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it is the case that $u.min\text{-}seqno(h') = \perp$ and $u.archived\text{-}pkts?(h') = \emptyset$. Since $u.min\text{-}seqno(h') = \perp$, it is the case that $u.window?(h') = \emptyset$. Thus, it follows that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$, as needed. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables $min\text{-}seqno(h')$, $max\text{-}seqno(h')$, and $archived\text{-}pkts?(h')$.

❐ $\texttt{rm-leave}_h$: if $u_k.status \neq \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-REC$_h$ except the variable now. Thus, it is the case that $u.min\text{-}seqno(h') = \perp$ and $u.archived\text{-}pkts?(h') = \emptyset$. Since $u.min\text{-}seqno(h') = \perp$, it is the case that $u.window?(h') = \emptyset$. Thus, it follows that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$, as needed.

Otherwise, if $u_k.status = \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❐ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of $\texttt{rm-send}_h(p)$ by cases. First, consider the case where $\neg(u_k.status = \texttt{member} \wedge h = s_p)$. In this case, $\texttt{rm-send}_h(p)$ does not affect the variables $min\text{-}seqno(h')$ and $max\text{-}seqno(h')$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

Second, consider the case where $u_k.status = \texttt{member}$ and $h = s_p$. Since $s_p = h'$, it follows that $h = h' = s_p$. If p is the foremost packet from s_p, that is, $u_k.min\text{-}seqno(s_p) = \perp$, then the $\texttt{rm-send}_h(p)$ action sets both $min\text{-}seqno(s_p)$ and $max\text{-}seqno(s_p)$ to i_p and adds the element $\langle p, now \rangle$ to $archived\text{-}pkts$. Since $u_k.min\text{-}seqno(s_p) = \perp$, it is the case that $u_k.window?(h') = \emptyset$. Thus, the induction hypothesis implies that $u_k.archived\text{-}pkts?(h') = \emptyset$. It follows that $u.archived\text{-}pkts?(h') = \{id(p)\}$. Moreover, since $u.min\text{-}seqno(h') = u.max\text{-}seqno(h') = i_p$, it follows that $u_k.window?(h') = \{id(p)\}$. Thus, if follows that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$, as needed.

If p is the next packet from s_p, that is, $u_k.min\text{-}seqno(s_p) \neq \perp$ and $i_p = u_k.max\text{-}seqno(s_p) + 1$, then $\texttt{rm-send}_h(p)$ sets $max\text{-}seqno(s_p)$ to i_p and adds the element $\langle p, now\rangle$ to *archived-pkts*. It follows that $u.archived\text{-}pkts?(h') = u_k.archived\text{-}pkts?(h') \cup \{id(p)\}$ and $u.window?(h') = u_k.window?(h') \cup \{id(p)\}$. From the induction hypothesis, it is the case that $u_k.archived\text{-}pkts?(h') \subseteq u_k.window?(h')$. Thus, it follows that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$, as needed.

❐ $\texttt{process-pkt}_h(p)$, for $p \in P_{\mathrm{SRM}}$, such that $type(p) \in \{\texttt{DATA}, \texttt{REPL}\}$ and $source(p) = h'$: letting $\langle s_p, i_p\rangle = id(p)$, we analyze the effects of $\texttt{process-pkt}_h(p)$ by cases.

First, consider the case where p is the foremost packet from s_p; that is, $type(p) = \texttt{DATA}$, $h \neq s_p$, and $u_k.min\text{-}seqno(s_p) = \perp$. Since $u_k.min\text{-}seqno(s_p) = \perp$, it is the case that $u_k.window?(s_p) = \emptyset$. Thus, the induction hypothesis implies that $u_k.archived\text{-}pkts?(s_p) = \emptyset$. Since $\texttt{process-pkt}_h(p)$ sets both variables $min\text{-}seqno(h')$ and $max\text{-}seqno(h')$ to i_p and adds $\langle strip(p), now\rangle$ to *archived-pkts*, it follows that $u.archived\text{-}pkts?(h') = u.window?(s_p) = \{id(p)\}$. Thus, it follows that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$.

Second, consider the case where p is not the foremost packet from s_p but is proper; that is, $u_k.min\text{-}seqno(s_p) \neq \perp$ and $u_k.min\text{-}seqno(s_p) \leq i_p$. In this case, the $\texttt{process-pkt}_h(p)$ action: i) adds the element $\langle strip(p), now\rangle$ to *archived-pkts*, if $h \neq s_p \wedge \langle s_p, i_p\rangle \notin u_k.archived\text{-}pkts?$, and ii) sets $max\text{-}seqno(s_p)$ to i_p, if $u_k.max\text{-}seqno(s_p) < i_p$. It follows that $u.archived\text{-}pkts?(s_p) \subseteq u_k.archived\text{-}pkts?(s_p) \cup \{id(p)\}$ and $u_k.window?(s_p) \cup \{id(p)\} \subseteq u.window?(s_p)$. Moreover, from the induction hypothesis, it is the case that $u_k.archived\text{-}pkts?(h') \subseteq u_k.window?(h')$. Thus, it follows that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$, as needed.

❐ $\texttt{process-pkt}_h(p)$, for $p \in P_{\mathrm{SRM}}$, such that $type(p) \in \{\texttt{RQST}, \texttt{SESS}\}$. In this case, the $\texttt{process-pkt}_h(p)$ does not affect the variable *archived-pkts* and may add elements to the variable $window?(h')$ by increasing the value of $max\text{-}seqno(h')$. This fact and the induction hypothesis imply that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$, as needed.

❐

Invariant 4.6 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case thatu.to-be-delivered?$(h') \subseteq u.window?(h')$.*

Proof: Follows directly from Invariants 4.4 and 4.5. ❐

Invariant 4.7 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case thatu.delivered$(h') \subseteq u.window?(h')$.*

Proof: Follows directly from Invariants 4.4 and 4.5. ❐

Invariant 4.8 *For $h \in H$, $p \in P_{\text{RM-CLIENT}}$, and any reachable state u of* SRM-REC$_h$*, if $p \in u.to\text{-}be\text{-}delivered$, then $u.min\text{-}seqno(source(p)) \neq \perp$ and $u.min\text{-}seqno(source(p)) \leq seqno(p)$.*

Proof: From the effects of the $process\text{-}pkt_h(p)$ action, for $h \in H$ and $p \in P_{\mathrm{SRM}}$, such that $id(p) = \langle s_p, i_p\rangle$, it follows that a packet p may be added to *to-be-delivered* only if h is not the source of p and p is a proper packet; that is, $h \neq s_p$, $min\text{-}seqno(s_p) \neq \perp$, and $min\text{-}seqno(s_p) \leq i_p$. ❐

Invariant 4.9 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case that:*

1. $u.\mathit{min\text{-}seqno}(h') = \bot \Rightarrow u.\mathit{expected}(h') = \emptyset$,
2. $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$,
3. $h = h' \wedge u.\mathit{status} \neq \texttt{crashed} \Rightarrow u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, *and*
4. $u.\mathit{expected}(h') \neq \emptyset \Rightarrow u.\mathit{expected}(h') = u.\mathit{proper?}(h')$

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it is the case that $u.\mathit{min\text{-}seqno}(h') = \bot$, $u.\mathit{delivered}(h') = \emptyset$, $u.\mathit{expected}(h') = \emptyset$, and $u.\mathit{proper?}(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.\mathit{lstate}$. For the step from u_k to u we consider only the actions that affect the variables $\mathit{min\text{-}seqno}(h')$, $\mathit{delivered}(h')$, $\mathit{expected}(h')$, and $\mathit{proper?}(h')$.

❒ $\texttt{crash}_h$: the $\texttt{crash}_h$ action sets $\mathit{delivered}(h')$ and $\mathit{expected}(h')$ to $\emptyset$. Thus, the invariant assertion is satisfied in u.

❒ $\texttt{rm-leave}_h$: if $u_k.\mathit{status} \neq \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-REC$_h$ except the variable now and sets the variables $\mathit{delivered}(h')$ and $\mathit{expected}(h')$ to $\emptyset$. It follows that $u.\mathit{min\text{-}seqno}(h') = \bot$, $u.\mathit{delivered}(h') = \emptyset$, $u.\mathit{expected}(h') = \emptyset$, and $u.\mathit{proper?}(h') = \emptyset$. Thus, the invariant assertion is satisfied in u.

Otherwise, if $u_k.\mathit{status} = \texttt{crashed}$, then the action $\texttt{rm-leave}_h$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $\mathit{source}(p) = h'$: letting $\langle s_p, i_p \rangle = \mathit{id}(p)$, we analyze the effects of $\texttt{rm-recv}_h(p)$ by cases. First, if $\neg(u_k.\mathit{status} = \texttt{member} \wedge h = s_p)$, then $\texttt{rm-send}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.\mathit{status} = \texttt{member} \wedge h = s_p$. If p is the foremost packet to be transmitted by s_p; that is, $u_k.\mathit{min\text{-}seqno}(s_p) = \bot$, then $\texttt{rm-send}_h(p)$ sets $\mathit{min\text{-}seqno}(h')$ to i_p, sets $\mathit{expected}(h')$ to $\mathit{suffix}(p)$, and adds $\mathit{id}(p)$ to $\mathit{delivered}(h')$. The induction hypothesis and the fact that $u_k.\mathit{min\text{-}seqno}(s_p) = \bot$ imply that $u_k.\mathit{expected}(s_p) = \emptyset$. Moreover, from the induction hypothesis it is the case that $u_k.\mathit{delivered}(s_p) \subseteq u_k.\mathit{expected}(s_p)$. Since $u_k.\mathit{expected}(s_p) = \emptyset$, it follows that $u_k.\mathit{delivered}(s_p) = \emptyset$. Thus, from the effects of $\texttt{rm-send}_h(p)$, it follows that $u.\mathit{expected}(s_p) = \mathit{suffix}(p)$ and $u.\mathit{delivered}(s_p) = \{\mathit{id}(p)\}$. Since $\mathit{id}(p) \in \mathit{suffix}(p)$, it follows that $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$. Moreover, since $u.\mathit{proper?}(h') = \mathit{suffix}(p)$, it follows that $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$. Since $u.\mathit{min\text{-}seqno}(s_p) = i_p$, $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$, and $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, it follows that the invariant assertion is satisfied in u.

If p is the next packet from s_p, that is, $u_k.\mathit{min\text{-}seqno}(s_p) \neq \bot$ and $i_p = u_k.\mathit{max\text{-}seqno}(s_p) + 1$, then $\texttt{rm-send}_h(p)$ does not affect $\mathit{min\text{-}seqno}(h')$, sets $\mathit{max\text{-}seqno}(h')$ to i_p, and adds $\mathit{id}(p)$ to $\mathit{delivered}(h')$; that is, $u.\mathit{min\text{-}seqno}(s_p) = u_k.\mathit{min\text{-}seqno}(s_p)$, $u.\mathit{max\text{-}seqno}(s_p) = i_p$, and $u.\mathit{delivered}(s_p) = u_k.\mathit{delivered}(s_p) \cup \{\mathit{id}(p)\}$.

Since $h = h' \wedge u_k.\mathit{status} \neq \texttt{crashed}$, the induction hypothesis implies that $u_k.\mathit{expected}(h') = u_k.\mathit{proper?}(h')$. Since $\texttt{rm-send}_h(p)$ affects neither $\mathit{min\text{-}seqno}(h')$ nor $\mathit{expected}(h')$, it follows that $u.\mathit{proper?}(h') = u_k.\mathit{proper?}(h')$ and $u.\mathit{expected}(h') = u_k.\mathit{expected}(h')$. Thus, it follows that $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, as needed.

From the induction hypothesis, it is the case that $u_k.\mathit{delivered}(h') \subseteq u_k.\mathit{expected}(h')$. Since $i_p = u_k.\mathit{max\text{-}seqno}(s_p) + 1$ and $u.\mathit{max\text{-}seqno}(s_p) = i_p$, it is the case that $u_k.\mathit{max\text{-}seqno}(s_p) < u.\mathit{max\text{-}seqno}(s_p)$. Thus, Invariant 4.3 implies that $u_k.\mathit{min\text{-}seqno}(s_p) < i_p$. Since $u_k.\mathit{min\text{-}seqno}(s_p) < i_p$, it follows that $\mathit{id}(p) \in u_k.\mathit{proper?}(h')$. Since $u_k.\mathit{expected}(h') = u_k.\mathit{proper?}(h')$, it follows that $\mathit{id}(p) \in u_k.\mathit{expected}(h')$. Since $u.\mathit{delivered}(s_p) = u_k.\mathit{delivered}(s_p) \cup \{\mathit{id}(p)\}$, $u_k.\mathit{delivered}(h') \subseteq u_k.\mathit{expected}(h')$, $\mathit{id}(p) \in u_k.\mathit{expected}(h')$, and $u.\mathit{expected}(h') = u_k.\mathit{expected}(h')$, it follows that $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$. Since $u.\mathit{min\text{-}seqno}(s_p) \neq \perp$, $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$, and $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, it follows that the invariant assertion is satisfied in u.

❒ $\texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $\mathit{source}(p) = h'$: letting $\langle s_p, i_p \rangle = \mathit{id}(p)$, we analyze the effects of $\texttt{rm-recv}_h(p)$ by cases. First, consider the case where $u_k.\mathit{expected}(h') = \emptyset$. From the induction hypothesis, it is the case that $u_k.\mathit{delivered}(h') \subseteq u_k.\mathit{expected}(h')$. Thus, it follows that $u_k.\mathit{delivered}(h') = \emptyset$. Since $u_k.\mathit{expected}(h') = \emptyset$, $\texttt{rm-recv}_h(p)$ sets $\mathit{expected}(h')$ to $\mathit{suffix}(p)$ and adds $\mathit{id}(p)$ to $\mathit{delivered}(h')$; that is, $u.\mathit{expected}(s_p) = \mathit{suffix}(p)$ and $u.\mathit{delivered}(s_p) = \{\mathit{id}(p)\}$. Since $\mathit{id}(p) \in \mathit{suffix}(p)$, it follows that $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$, as needed.

Since $u_k.\mathit{delivered}(h') = \emptyset$, Invariant 4.4 implies that $u_k.\mathit{archived\text{-}pkts?}(h') = u_k.\mathit{to\text{-}be\text{-}delivered?}(h')$. From the precondition of $\texttt{rm-recv}_h(p)$, it follows that p is h's foremost packet from h'; that is, $i_p = u_k.\mathit{min\text{-}seqno}(h')$. Since $\mathit{suffix}(p) = \{\langle s, i \rangle \in H \times \mathbb{N} \mid s_p = s \wedge i_p \leq i\}$, it follows that $u.\mathit{proper?}(h') = \mathit{suffix}(p)$. Thus, it follows that $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, as needed.

Finally, since $p \in u_k.\mathit{to\text{-}be\text{-}delivered}$, Invariant 4.8 implies that $u_k.\mathit{min\text{-}seqno}(s_p) \neq \perp$. Since $\texttt{rm-recv}_h(p)$ does not affect $\mathit{min\text{-}seqno}(s_p)$, it follows that $u.\mathit{min\text{-}seqno}(s_p) \neq \perp$. Since $u.\mathit{min\text{-}seqno}(s_p) \neq \perp$, $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$, and $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, it follows that the invariant assertion is satisfied in u.

Second, consider the case where $u_k.\mathit{expected}(h') \neq \emptyset$. In this case, $\texttt{rm-recv}_h(p)$ does not affect $\mathit{min\text{-}seqno}(s_p)$, does not affect $\mathit{expected}(h')$, and adds $\mathit{id}(p)$ to $\mathit{delivered}(h')$; that is, $u.\mathit{proper?}(h') = u_k.\mathit{proper?}(h')$, $u.\mathit{expected}(s_p) = u_k.\mathit{expected}(s_p)$, and $u.\mathit{delivered}(s_p) = u_k.\mathit{delivered}(s_p) \cup \{\mathit{id}(p)\}$. Since $u_k.\mathit{expected}(h') \neq \emptyset$, the induction hypothesis implies that $u_k.\mathit{expected}(h') = u_k.\mathit{proper?}(h')$. Since $u.\mathit{proper?}(h') = u_k.\mathit{proper?}(h')$, $u.\mathit{expected}(s_p) = u_k.\mathit{expected}(s_p)$, it follows that $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, as needed.

Since $p \in u_k.\mathit{to\text{-}be\text{-}delivered}$, Invariant 4.4 implies that $\mathit{id}(p) \in u_k.\mathit{archived\text{-}pkts?}(h')$. Thus, Invariant 4.5 implies that $\mathit{id}(p) \in u_k.\mathit{window?}(h')$. By definition it follows that $\mathit{window?}(h') \subseteq \mathit{proper?}(h')$. Thus, it is the case that $\mathit{id}(p) \in u_k.\mathit{proper?}(h')$ and, since $u.\mathit{proper?}(h') = u_k.\mathit{proper?}(h')$, $\mathit{id}(p) \in u.\mathit{proper?}(h')$. Thus, it follows that $u.\mathit{delivered}(s_p) \subseteq u.\mathit{expected}(s_p)$, as needed.

Finally, since $p \in u_k.\mathit{to\text{-}be\text{-}delivered}$, Invariant 4.8 implies that $u_k.\mathit{min\text{-}seqno}(s_p) \neq \perp$. Since $\texttt{rm-recv}_h(p)$ does not affect $\mathit{min\text{-}seqno}(s_p)$, it follows that $u.\mathit{min\text{-}seqno}(s_p) \neq \perp$. Since it is the case that $u.\mathit{min\text{-}seqno}(s_p) \neq \perp$, $u.\mathit{delivered}(h') \subseteq u.\mathit{expected}(h')$, and $u.\mathit{expected}(h') = u.\mathit{proper?}(h')$, it follows that the invariant assertion is satisfied in u.

❒ $\texttt{process-pkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $\mathit{source}(p) = h'$: letting $\langle s_p, i_p \rangle = \mathit{id}(p)$, we analyze the effects of $\texttt{process-pkt}_h(p)$ by cases.

First, if $\mathit{type}(p) = \texttt{DATA}$, $u_k.\mathit{status} = \texttt{member}$, $h \neq s_p$, and $u_k.\mathit{min\text{-}seqno}(h') = \perp$, then the action $\texttt{process-pkt}_h(p)$ sets $\mathit{min\text{-}seqno}(h')$ to i_p and affects neither $\mathit{delivered}(h')$ nor $\mathit{expected}(h')$. Since $u_k.\mathit{min\text{-}seqno}(h') = \perp$, the induction hypothesis implies that $u_k.\mathit{expected}(h') = \emptyset$. Moreover, from the induction hypothesis, it is the case that $u_k.\mathit{delivered}(h') \subseteq u_k.\mathit{expected}(h')$. Thus, since $u_k.\mathit{expected}(h') = \emptyset$, it follows that $u_k.\mathit{delivered}(h') = \emptyset$. Since $\texttt{process-pkt}_h(p)$ affects neither $\mathit{expected}(h')$ nor $\mathit{delivered}(h')$, it follows that $u.\mathit{expected}(h') = \emptyset$ and $u.\mathit{delivered}(h') = \emptyset$.

Thus, it follows that $u.delivered(h') \subseteq u.expected(h')$, as needed. Since $h \neq s_p$ and $s_p = h'$, it follows that $h \neq h'$. Thus, since $u.min\text{-}seqno(h') \neq \perp$, $u.delivered(h') \subseteq u.expected(h')$, $h \neq h'$, $u.expected(h') = \emptyset$, it follows that the invariant assertion is satisfied in u.

Otherwise, $\texttt{process-pkt}_h(p)$ does not affect the variables *min-seqno*(h'), *expected*(h') and *delivered*(h'). Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒

Invariant 4.10 *Let $h \in H$ and u be any reachable state u of* SRM-REC$_h$. *For any $p \in P_{\text{SRM}}$, such that type$(p) \in \{\texttt{DATA}, \texttt{REPL}\}$ and $p \in u.msend\text{-}buff$, it is the case that $id(p) \in u.archived\text{-}pkts?$.*

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it is the case that $u.msend\text{-}buff = \emptyset$. Thus, the invariant assertion is trivially satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables *msend-buff* and *archived-pkts*.

❒ $\texttt{rm-leave}_h$: the action $\texttt{rm-leave}_h$ initializes the variables *msend-buff* and *archived-pkts*. Thus, the invariant assertion holds in u.

❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$: the action $\texttt{rm-send}_h(p)$ adds the packet *comp-data-pkt*(p) to *msend-buff* if and only if it adds the element $\langle p, now\rangle$ to the variable *archived-pkts*. This fact and the induction hypothesis imply that the invariant assertion holds in u.

❒ $\texttt{send-repl}_h(s,i)$, for $s \in H$ and $i \in \mathbb{N}$: the action $\texttt{send-repl}_h(s,i)$ adds the packet $pkt = comp\text{-}repl\text{-}pkt(h,p)$, for $p \in P_{\text{RM-CLIENT}}, t \in \mathbb{R}^{\geq 0}$, such that $\langle p,t\rangle \in archived\text{-}pkts$ and $id(p) = \langle s,i\rangle$, to *msend-buff*. Since $id(pkt) \in u_k.archived\text{-}pkts?$ and the $\texttt{send-repl}_h(s,i)$ action does not affect the variable *archived-pkts*, it follows that $id(pkt) \in u.archived\text{-}pkts?$. The induction hypothesis and the facts that $pkt \in u.msend\text{-}buff$ and $id(pkt) \in u.archived\text{-}pkts?$ imply that the invariant assertion is satisfied in u.

❒ $\texttt{process-pkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $source(p) = h'$: $\texttt{process-pkt}_h(p)$ does not affect *msend-buff* and may only add the element $id(p)$ to *archived-pkts?*. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒

Invariant 4.11 *For $h \in H$, $p \in P_{\text{RM-CLIENT}}$, and any reachable state u of* SRM-REC$_h$, *if $p \in u.to\text{-}be\text{-}delivered$, then $source(p) \neq h$.*

Proof: From the effects of the *process-pkt*$_h(p)$ action, for $h \in H$ and $p \in P_{\text{SRM}}$, it follows that a packet p may be added to *to-be-delivered* only if $source(p) \neq h$. ❒

Invariant 4.12 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$, *if $u.expected(h') \neq \emptyset$, then $u.to\text{-}be\text{-}delivered?(h') \subseteq u.expected(h')$.*

Proof: Suppose that $u.expected(h') \neq \emptyset$. Invariant 4.2 implies that $u.status = \texttt{member}$. Moreover, Invariant 4.9 implies that $u.expected(h') = u.proper?(h')$. From Invariant 4.5, it is the case that $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$. Moreover, since $u.status = \texttt{member}$, Invariant 4.4 implies that $u.to\text{-}be\text{-}delivered?(h') \subseteq u.window?(h')$. Since by definition $u.window?(h') \subseteq u.proper?(h')$, it follows that $u.\textit{to-be-delivered?}(h') \subseteq u.\textit{proper?}(h')$. Finally, since $u.\textit{expected}(h') = u.\textit{proper?}(h')$, it follows that $u.\textit{to-be-delivered?}(h') \subseteq u.\textit{expected}(h')$. ❒

Invariant 4.13 *For $h, h' \in H$ and any reachable state u of SRM-REC$_h$, it is the case that $u.\textit{to-be-requested?}(h') \subseteq u.\textit{window?}(h')$.*

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.\textit{min-seqno}(h') = \perp$ and $u.\textit{to-be-requested?}(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.\textit{lstate}$. For the step from u_k to u we consider only the actions that affect the variables $\textit{min-seqno}(h')$, $\textit{max-seqno}(h')$, and $\textit{to-be-requested?}(h')$.

❒ `rm-leave`$_h$: if $u_k.\textit{status} = \texttt{crashed}$, then `rm-leave`$_h$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u. Otherwise, if $u_k.\textit{status} \neq \texttt{crashed}$, then `rm-leave`$_h$ reinitializes all the variables of SRM-REC$_h$ except the variable *now*. It follows that $u.\textit{min-seqno}(h') = \perp$ and $u.\textit{to-be-requested?}(h') = \emptyset$. Thus, the invariant assertion holds in u.

❒ `rm-send`$_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $\textit{source}(p) = h'$: letting $\langle s_p, i_p \rangle = \textit{id}(p)$, we analyze the effects of `rm-send`$_h(p)$ by cases. First, if $\neg(u_k.\textit{status} = \texttt{member} \wedge h = s_p)$, then `rm-send`$_h(p)$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

  Second, consider the case where $u_k.\textit{status} = \texttt{member} \wedge h = s_p$. If $u_k.\textit{min-seqno}(h') = \perp$, then `rm-send`$_h(p)$ sets $\textit{min-seqno}(h')$ and $\textit{max-seqno}(h')$ to i_p. Since $u_k.\textit{min-seqno}(h') = \perp$, it follows that $u_k.\textit{window?}(h') = \emptyset$. Thus, the induction hypothesis implies that $u_k.\textit{to-be-requested?}(h') = \emptyset$. Since `rm-send`$_h(p)$ does not affect the variable *to-be-requested?*, it follows that $u.\textit{to-be-requested?}(h') = \emptyset$. Thus, the invariant assertion holds in u.

  Otherwise, if $u_k.\textit{min-seqno}(h') \neq \perp$, then `rm-send`$_h(p)$ may only increase the value of the variable $\textit{max-seqno}(h')$ and does not affect the variable *to-be-requested?*; that is, $u_k.\textit{window?}(h') \subseteq u.\textit{window?}(h')$ and $u.\textit{to-be-requested?}(h') = u_k.\textit{to-be-requested?}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ `schdl-rqst`$_h(s,i)$, for $s \in H$ and $i \in \mathbb{N}$, such that $s = h'$: the action `schdl-rqst`$_h(s,i)$ removes the element $\langle s, i \rangle$ from the set $u_k.\textit{to-be-requested?}$ and does not affect $\textit{min-seqno}(h')$ and $\textit{max-seqno}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ `process-pkt`$_h(p)$, for $p \in P_{\text{SRM}}$, such that $\textit{type}(p) = \texttt{DATA}$ and $\textit{source}(p) = h'$: letting $\langle s_p, i_p \rangle = \textit{id}(p)$, we analyze the effects of the `process-pkt`$_h(p)$ action by cases. First, if $u_k.\textit{status} \neq \texttt{member}$, then `process-pkt`$_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

  Second, consider the case where $u_k.\textit{status} = \texttt{member}$. If $h \neq s_p$ and $u_k.\textit{min-seqno}(s_p) = \perp$, then `process-pkt`$_h(p)$ sets the variables $\textit{min-seqno}(h')$ and $\textit{max-seqno}(h')$ to i_p and does not affect the variable *to-be-requested?*. Since $u_k.\textit{min-seqno}(h') = \perp$, it follows that $u_k.\textit{window?}(h') = \emptyset$. Thus, the induction hypothesis implies that $u_k.\textit{to-be-requested?}(h') = \emptyset$. Since `process-pkt`$_h(p)$ does not affect the variable *to-be-requested?*, it follows that $u.\textit{to-be-requested?}(h') = \emptyset$. Thus, the invariant assertion holds in u.

  If $u_k.\textit{min-seqno}(s_p) \neq \perp$, $u_k.\textit{min-seqno}(s_p) \leq i_p$, $h \neq s_p$, and $u_k.\textit{max-seqno}(s_p) < i_p$, then the action `process-pkt`$_h(p)$ adds $\{\langle s_p, i \rangle \mid i \in \mathbb{N}, u_k.\textit{max-seqno}(s_p) < i < i_p\}$ to *to-be-requested?*

and sets $max\text{-}seqno(h')$ to i_p. Since $u_k.min\text{-}seqno(h') \leq i_p$ and $\texttt{process-pkt}_h(p)$ does not affect the variable $min\text{-}seqno(h')$, it follows that $u.min\text{-}seqno(h') \leq i_p$. Since $u.min\text{-}seqno(h') \leq i$ and $u.max\text{-}seqno(h') = i$, it follows that $\{\langle s_p, i\rangle \mid i \in \mathbb{N}, u_k.max\text{-}seqno(s_p) < i < i_p\} \subseteq u.window(h')$. This fact and the induction hypothesis imply that $u.to\text{-}be\text{-}requested?(h') \subseteq u.window?(h')$.

Otherwise, $\texttt{process-pkt}_h(p)$ does not affect the variables $min\text{-}seqno(h')$, $max\text{-}seqno(h')$, and $to\text{-}be\text{-}requested?(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❐ $\texttt{process-pkt}_h(p)$, for $p \in P_{\mathrm{SRM}}$, such that $type(p) \in \{\texttt{SESS}\}$: we analyze the effects of the $\texttt{process-pkt}_h(p)$ action by cases. First, if $u_k.status \neq \texttt{member}$, then $\texttt{process-pkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.status = \texttt{member}$. In this case, if $h \neq h'$, $u_k.status = \texttt{member}$ and there exists $\langle h', i'\rangle \in seqno\text{-}rprts(p)$, for $i' \in \mathbb{N}$, such that $u_k.max\text{-}seqno(h') < i'$, then $\texttt{process-pkt}_h(p)$ adds $\{\langle h', i''\rangle \mid i'' \in \mathbb{N}, u_k.max\text{-}seqno(h') < i'' \leq i'\}$ to $to\text{-}be\text{-}requested?$ and sets $max\text{-}seqno(h')$ to i'. Invariant 4.3 and the fact that $u_k.max\text{-}seqno(h') < i'$ imply that $u_k.min\text{-}seqno(h') < i'$. Since $\texttt{process-pkt}_h(p)$ does not affect the variable $min\text{-}seqno(s)$, it follows that $u.min\text{-}seqno(s) < i$. Thus, since $u.min\text{-}seqno(s) < i$ and $u.max\text{-}seqno(s) = i$, it follows that $\{\langle h', i''\rangle \mid i'' \in \mathbb{N}, u_k.max\text{-}seqno(h') < i'' \leq i'\} \subseteq u.window(h')$. This fact and the induction hypothesis imply that $u.to\text{-}be\text{-}requested?(h') \subseteq u.window?(h')$.

Otherwise, if either $h = h'$, $u_k.status \neq \texttt{member}$, or there does not exist $\langle h', i'\rangle \in seqno\text{-}rprts(p)$, for $i' \in \mathbb{N}$, such that $u_k.max\text{-}seqno(h') < i'$, then $\texttt{process-pkt}_h(p)$ does not affect the variables $min\text{-}seqno(h')$, $max\text{-}seqno(h')$, and $to\text{-}be\text{-}requested?(h')$. Thus, induction hypothesis implies that the invariant assertion holds in u.

❐ $\texttt{process-pkt}_h(p)$, for $p \in P_{\mathrm{SRM}}$, such that $type(p) \in \{\texttt{REPL}, \texttt{RQST}\}$ and $source(p) = h'$: letting $\langle s_p, i_p\rangle = id(p)$, we analyze the effects of the $\texttt{process-pkt}_h(p)$ action by cases. First, if $u_k.status \neq \texttt{member}$, then $\texttt{process-pkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.status = \texttt{member}$. If it is the case that $u_k.min\text{-}seqno(s_p) \neq \perp$, $u_k.min\text{-}seqno(s_p) \leq i_p$, $h \neq s_p$, and $u_k.max\text{-}seqno(s_p) < i_p$, then the action $\texttt{process-pkt}_h(p)$ adds $\{\langle s_p, i\rangle \mid i \in \mathbb{N}, u_k.max\text{-}seqno(s_p) < i < i_p\}$ to $to\text{-}be\text{-}requested?$ and sets $max\text{-}seqno(h')$ to i_p. Since $u_k.min\text{-}seqno(h') \leq i_p$ and $\texttt{process-pkt}_h(p)$ does not affect the variable $min\text{-}seqno(h')$, it follows that $u.min\text{-}seqno(h') \leq i_p$. Thus, since $u.min\text{-}seqno(h') \leq i$ and $u.max\text{-}seqno(h') = i$, it follows that $\{\langle s_p, i\rangle \mid i \in \mathbb{N}, u_k.max\text{-}seqno(s_p) < i < i_p\} \subseteq u.window(h')$. This fact and the induction hypothesis imply that $u.to\text{-}be\text{-}requested?(h') \subseteq u.window?(h')$.

Otherwise, $\texttt{process-pkt}_h(p)$ does not affect the variables $min\text{-}seqno(h')$, $max\text{-}seqno(h')$, and $to\text{-}be\text{-}requested?(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❐

Invariant 4.14 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case that $u.scheduled\text{-}repls?(h') \subseteq u.archived\text{-}pkts?(h')$.*

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.scheduled\text{-}repls?(h') = \emptyset$ and $u.archived\text{-}pkts?(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables *scheduled-repls?*(h') and *archived-pkts?*(h').

❒ $\texttt{rm-leave}_h$: if $u_k.status = \texttt{crashed}$, then $\texttt{rm-leave}_h$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u. Otherwise, if $u_k.status \neq \texttt{crashed}$, then $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-REC$_h$ except the variable *now*. It follows that $u.scheduled\text{-}repls?(h') = \emptyset$ and $u.archived\text{-}pkts?(h') = \emptyset$. Thus, the invariant assertion holds in u.

❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $source(p) = h'$, such that $\langle s_p, i_p \rangle = id(p)$: the $\texttt{rm-send}_h(p)$ action does not affect *scheduled-repls?*(s_p) and may add elements to *archived-pkts?*(s_p); that is, $u.scheduled\text{-}repls?(h') = u_k.scheduled\text{-}repls?(h')$ and $u_k.archived\text{-}pkts?(h') \subseteq u.archived\text{-}pkts?(h')$. Moreover, $\texttt{process-pkt}_h(p)$ affects none of the variables *scheduled-repls?*(h') and *archived-pkts?*(h'), for $h' \in H, h' \neq s_p$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{send-repl}_h(s, i)$, for $s \in H$ and $i \in \mathbb{N}$: the $\texttt{send-repl}_h(s, i)$ may remove an element from *scheduled-repls?*(s) and does not affect *archived-pkts?*(s). Moreover, $\texttt{process-pkt}_h(p)$ affects none of the variables *scheduled-repls?*(h') and *archived-pkts?*(h'), for $h' \in H, h' \neq s$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-pkt}_h(p)$, for $h \in H$ and $p \in P_{\text{SRM}}$, such that $\langle s_p, i_p \rangle = id(p)$: if $type(p) \in \{\texttt{DATA}, \texttt{REPL}\}$, then the $\texttt{process-pkt}_h(p)$ action may remove the element $\langle s_p, i_p \rangle$ from *scheduled-repls?*(s_p) and may add the element $\langle s_p, i_p \rangle$ to *archived-pkts?*(s_p). Moreover, $\texttt{process-pkt}_h(p)$ affects none of the variables *scheduled-repls?*(h') and *archived-pkts?*(h'), for $h' \in H, h' \neq s_p$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

If $type(p) = \texttt{RQST}$, the $\texttt{process-pkt}_h(p)$ action may add the element $\langle s_p, i_p \rangle$ to *scheduled-repls?*(s_p) only if $\langle s_p, i_p \rangle \in u.archived\text{-}pkts?(s_p)$. Moreover, $\texttt{process-pkt}_h(p)$ affects none of the variables *scheduled-repls?*(h') and *archived-pkts?*(h'), for $h' \in H, h' \neq s_p$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Otherwise, if $type(p) = \texttt{SESS}$, the $\texttt{process-pkt}_h(p)$ action affects none of the variables *scheduled-repls?*(h') and *archived-pkts?*(h'), for $h' \in H$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒

Invariant 4.15 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case that* $u.scheduled\text{-}rqsts?(h') \subseteq u.window?(h')$.

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.min\text{-}seqno(h') = \perp$ and $u.scheduled\text{-}rqsts?(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k + 1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables *min-seqno*(h'), *max-seqno*(h'), and *scheduled-rqsts?*(h').

❒ $\texttt{rm-leave}_h$: if $u_k.status = \texttt{crashed}$, then $\texttt{rm-leave}_h$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u. Otherwise, if $u_k.status \neq \texttt{crashed}$, then $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-REC$_h$ except the variable *now*. It follows that $u.min\text{-}seqno(h') = \perp$ and $u.scheduled\text{-}rqsts?(h') = \emptyset$. Thus, the invariant assertion is satisfied in u.

❒ $\mathtt{rm\text{-}send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of $\mathtt{rm\text{-}send}_h(p)$ by cases. First, if $\neg(u_k.status = \mathtt{member} \wedge h = s_p)$, then $\mathtt{rm\text{-}send}_h(p)$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

Second, consider the case where $u_k.status = \mathtt{member} \wedge h = s_p$. If $u_k.min\text{-}seqno(h') = \perp$, then $\mathtt{rm\text{-}send}_h(p)$ sets $min\text{-}seqno(h')$ and $max\text{-}seqno(h')$ to i_p. Since $u_k.min\text{-}seqno(h') = \perp$, it follows that $u_k.window?(h') = \emptyset$. Thus, the induction hypothesis implies that $u_k.scheduled\text{-}rqsts?(h') = \emptyset$. Since $\mathtt{rm\text{-}send}_h(p)$ does not affect the variable *scheduled-rqsts*, it follows that $u.scheduled\text{-}rqsts?(h') = \emptyset$. Thus, the invariant assertion holds in u.

Otherwise, if $u_k.min\text{-}seqno(h') \neq \perp$, then $\mathtt{rm\text{-}send}_h(p)$ may only increase the value of the variable $max\text{-}seqno(h')$ and does not affect the variable *scheduled-rqsts*; that is, $u_k.window?(h') \subseteq u.window?(h')$ and $u.scheduled\text{-}rqsts(h') = u_k.scheduled\text{-}rqsts(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\mathtt{schdl\text{-}rqst}_h(s,i)$, for $s \in H$ and $i \in \mathbb{N}$, such that $s = h'$: $\mathtt{schdl\text{-}rqst}_h(s,i)$ adds the tuple $\langle s,i \rangle$ to $scheduled\text{-}rqsts?(h')$. From the precondition of $\mathtt{schdl\text{-}rqst}_h(s,i)$, it follows that $\langle s,i \rangle \in u_k.to\text{-}be\text{-}requested?(h')$. Thus, Invariant 4.13 implies that $\langle s,i \rangle \in u_k.window?(h')$. Since $\mathtt{schdl\text{-}rqst}_h(s,i)$ does not affect the variables $min\text{-}seqno(h')$ and $max\text{-}seqno(h')$, it follows that $u.window?(h') = u_k.window?(h')$. From the induction hypothesis, it is the case that $u_k.scheduled\text{-}rqsts?(h') \subseteq u_k.window?(h')$. Since $u.window?(h') = u_k.window?(h')$ and $u.scheduled\text{-}rqsts?(h') = u_k.scheduled\text{-}rqsts?(h') \cup \langle s,i \rangle$, it follows that the invariant assertion hold in u.

❒ $\mathtt{send\text{-}rqst}_h(s,i)$, for $s \in H$ and $i \in \mathbb{N}$, such that $s = h'$: from the precondition of the action $\mathtt{send\text{-}rqst}_h(s,i)$, it is the case that $\langle s,i \rangle \in u_k.scheduled\text{-}rqsts?(h')$. Since $\mathtt{send\text{-}rqst}_h(s,i)$ simply backs-off the request scheduled for $\langle s,i \rangle$, it does not affect $min\text{-}seqno(h')$, $max\text{-}seqno(h')$, and $scheduled\text{-}rqsts?(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\mathtt{process\text{-}pkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $type(p) = \mathtt{DATA}$ and $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of the $\mathtt{process\text{-}pkt}_h(p)$ action by cases. First, if $u_k.status \neq \mathtt{member}$, then $\mathtt{process\text{-}pkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

Second, consider the case where $u_k.status = \mathtt{member}$. If p is neither the foremost nor a proper packet from s_p, then $\mathtt{process\text{-}pkt}_h(p)$ affects neither of the variables $min\text{-}seqno(h')$, $max\text{-}seqno(h')$, and $scheduled\text{-}rqsts?(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

If p is the foremost packet from s_p, then $\mathtt{process\text{-}pkt}_h(p)$ sets the variables $min\text{-}seqno(h')$ and $max\text{-}seqno(h')$ to i_p. From the induction hypothesis, it follows that $u_k.scheduled\text{-}rqsts?(h') = \emptyset$. Since $\mathtt{process\text{-}pkt}_h(p)$ may only remove elements from $scheduled\text{-}rqsts?(h')$, it follows that $u.scheduled\text{-}rqsts?(h') = \emptyset$. Thus, the invariant assertion holds in u.

Finally, if $u_k.min\text{-}seqno(s_p) \neq \perp$, then $\mathtt{process\text{-}pkt}_h(p)$ may only remove elements from the set $scheduled\text{-}rqsts?(h')$ and increase the value of $max\text{-}seqno(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\mathtt{process\text{-}pkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $type(p) = \mathtt{RQST}$ and $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of the $\mathtt{process\text{-}pkt}_h(p)$ action by cases. First, if $u_k.status \neq \mathtt{member}$, then $\mathtt{process\text{-}pkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

Second, consider the case where $u_k.status = \mathtt{member}$. If p does not pertain to a proper packet, then the action $\texttt{process-pkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, in this case, the induction hypothesis implies that the invariant assertion holds in u.

If p pertains to a proper packet and h is not the source of p, then $\texttt{process-pkt}_h(p)$ may add the tuple $id(p)$ to $scheduled\text{-}rqsts?(h')$ and ensures that $i_p \leq u.max\text{-}seqno(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-pkt}_h(p)$, for $p \in P_{\mathrm{SRM}}$, such that $type(p) = \texttt{REPL}$ and $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of the $\texttt{process-pkt}_h(p)$ action by cases. First, if $u_k.status \neq \texttt{member}$, then $\texttt{process-pkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

Second, consider the case where $u_k.status = \texttt{member}$. If p is not a proper packet, then the action $\texttt{process-pkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

If p is a proper packet, then $\texttt{process-pkt}_h(p)$ may only remove elements from the variable $scheduled\text{-}rqsts?(h')$ and increase the value of $max\text{-}seqno(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-pkt}_h(p)$, for $p \in P_{\mathrm{SRM}}$, such that $type(p) \in \{\texttt{SESS}\}$: the $\texttt{process-pkt}_h(p)$ action does not affect the variable $scheduled\text{-}rqsts?(h')$, does not affect the variable $min\text{-}seqno(h')$, and may only increase the value of $max\text{-}seqno(h')$. Thus, it follows that $u_k.scheduled\text{-}rqsts?(h') = u.scheduled\text{-}rqsts?(h')$ and $u_k.window?(h') \subseteq u.window?(h')$. Moreover, from the induction hypothesis, it is the case that $u_k.scheduled\text{-}rqsts?(h') \subseteq u_k.window?(h')$. Thus, it follows that $u.scheduled\text{-}rqsts?(h') \subseteq u.window?(h')$.

❒

Invariant 4.16 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case that $u.to\text{-}be\text{-}requested?(h') \cap u.archived\text{-}pkts?(h') = \emptyset$.*

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.to\text{-}be\text{-}requested?(h') = \emptyset$ and $u.archived\text{-}pkts?(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables $to\text{-}be\text{-}requested?(h')$ and $archived\text{-}pkts?(h')$.

❒ $\texttt{rm-leave}_h$: if $u_k.status = \texttt{crashed}$, then $\texttt{rm-leave}_h$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u. Otherwise, if $u_k.status \neq \texttt{crashed}$, then $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-REC$_h$ except the variable now. It follows that $u.to\text{-}be\text{-}requested?(h') = \emptyset$ and $u.archived\text{-}pkts?(h') = \emptyset$. Thus, the invariant assertion holds in u.

❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $source(p) = h'$, such that $\langle s_p, i_p \rangle = id(p)$: we analyze the effects of $\texttt{rm-send}_h(p)$ by cases. First, if $\neg(u_k.status = \texttt{member} \wedge h = s_p)$, then $\texttt{rm-send}_h(p)$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

Second, consider the case where $u_k.status = \texttt{member} \wedge h = s_p$. If p is the foremost packet to be transmitted by h', that is, $u_k.min\text{-}seqno(h') = \perp$, then it follows that $u_k.window?(h') = \emptyset$. Thus, Invariants 4.5 and 4.13 imply that $u_k.archived\text{-}pkts?(h') = \emptyset$ and $u_k.to\text{-}be\text{-}requested?(h') = \emptyset$. If p is the next packet from h', that is, $u_k.min\text{-}seqno(h') \neq \perp$ and $i_p = u_k.max\text{-}seqno(h') + 1$, then it is the case that $id(p) \notin u_k.window?(h')$. Thus, Invariants 4.5 and 4.13 imply that $id(p) \notin u_k.archived\text{-}pkts?(h')$ and $id(p) \notin u_k.to\text{-}be\text{-}requested?(h')$.

In either case the $\texttt{rm-send}_h(p)$ adds $id(p)$ to the variable *archived-pkts?*(h') and does not affect *to-be-requested?*(h'). It follows that $u.to\text{-}be\text{-}requested?(h') = u_k.to\text{-}be\text{-}requested?(h')$ and $u.archived\text{-}pkts?(h') = u_k.archived\text{-}pkts?(h') \cup id(p)$. From the induction hypothesis, it is the case that $u_k.to\text{-}be\text{-}requested?(h') \cap u_k.archived\text{-}pkts?(h') = \emptyset$. Since it is the case that $id(p) \notin u_k.to\text{-}be\text{-}requested?(h')$, it follows that $u.to\text{-}be\text{-}requested?(h') \cap u.archived\text{-}pkts?(h') = \emptyset$.

❒ $\texttt{schdl-rqst}_h(s,i)$, for $s \in H, i \in \mathbb{N}$, such that $s = h'$: the $\texttt{schdl-rqst}_h(s,i)$ action removes the element $\langle s,i \rangle$ from *to-be-requested?*(h') and does not affect *archived-pkts?*(h'). From the induction hypothesis, it is the case that $u_k.to\text{-}be\text{-}requested?(h') \cap u.archived\text{-}pkts?(h') = \emptyset$. Thus, it follows that $u.to\text{-}be\text{-}requested?(h') \cap u.archived\text{-}pkts?(h') = \emptyset$, as needed.

❒ $\texttt{process-pkt}_h(p)$, for $p \in P_{\mathrm{SRM}}$, such that $type(p) \in \{\texttt{DATA}, \texttt{RQST}, \texttt{REPL}\}$, $\langle s_p, i_p \rangle = id(p)$, and $s_p = h'$: the action $\texttt{process-pkt}_h(p)$ adds $\{\langle s_p, i' \rangle \mid i' \in \mathbb{N}, u_k.max\text{-}seqno(s_p) < i' < i\}$ to *to-be-requested?*(h') only if $h \neq s_p$ and $u_k.max\text{-}seqno(s_p) < i$. Moreover, the action $\texttt{process-pkt}_h(p)$ removes $\langle s_p, i_p \rangle$ from *to-be-requested?*(h') whenever it adds it to *archived-pkts?*(h').

Invariant 4.5 implies that $u_k.archived\text{-}pkts? \cap \{\langle s_p, i' \rangle \mid i' \in \mathbb{N}, u_k.max\text{-}seqno(s_p) < i' < i\} = \emptyset$. From the induction hypothesis, it is the case that $u_k.to\text{-}be\text{-}requested?(h') \cap u.archived\text{-}pkts?(h') = \emptyset$. Thus, it follows that the invariant assertion holds in u.

❒ $\texttt{process-pkt}_h(p)$, for $p \in P_{\mathrm{SRM}}$, such that $type(p) = \texttt{SESS}$: we analyze the effects of the $\texttt{process-pkt}_h(p)$ action by cases. First, if $u_k.status \neq \texttt{member}$, then $\texttt{process-pkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.status = \texttt{member}$. In this case, if $h \neq h'$, $u_k.status = \texttt{member}$ and there exists $\langle h', i' \rangle \in seqno\text{-}rprts(p)$, for $i' \in \mathbb{N}$, such that $u_k.max\text{-}seqno(h') < i'$, then $\texttt{process-pkt}_h(p)$ adds $\{\langle h', i'' \rangle \mid i'' \in \mathbb{N}, u_k.max\text{-}seqno(h') < i'' \leq i'\}$ to *to-be-requested?* and sets $max\text{-}seqno(h')$ to i'. Moreover, from Invariant 4.3 it is the case that $u_k.min\text{-}seqno(h') \leq u_k.max\text{-}seqno(h')$. Thus, by the definition of $window(h')$, it follows that $u_k.window(h') \cap \{\langle h', i'' \rangle \mid i'' \in \mathbb{N}, u_k.max\text{-}seqno(h') < i'' \leq i'\} = \emptyset$. From the induction hypothesis it is the case that $u_k.to\text{-}be\text{-}requested?(h') \cap u_k.archived\text{-}pkts?(h') = \emptyset$. Moreover, since the $\texttt{process-pkt}_h(p)$ action does not affect the variable *archived-pkts?*(h'), it is the case that $u.archived\text{-}pkts?(h') = u_k.archived\text{-}pkts?(h')$. Thus, it follows that $u.to\text{-}be\text{-}requested?(h') \cap u.archived\text{-}pkts?(h') = \emptyset$.

Otherwise, if either $h = h'$, $u_k.status \neq \texttt{member}$, or there does not exist $\langle h', i' \rangle \in seqno\text{-}rprts(p)$, for $i' \in \mathbb{N}$, such that $u_k.max\text{-}seqno(h') < i'$, then $\texttt{process-pkt}_h(p)$ does not affect the variables $min\text{-}seqno(h')$, $max\text{-}seqno(h')$, and *to-be-requested?*(h'). Thus, induction hypothesis implies that the invariant assertion holds in u.

❒

Invariant 4.17 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case that* $u.scheduled\text{-}rqsts?(h') \cap u.archived\text{-}pkts?(h') = \emptyset$.

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.scheduled\text{-}rqsts?(h') = \emptyset$ and $u.archived\text{-}pkts?(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables $scheduled\text{-}rqsts?(h')$ and $archived\text{-}pkts?(h')$.

❒ $\texttt{rm-leave}_h$: if $u_k.status = \texttt{crashed}$, then $\texttt{rm-leave}_h$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u. Otherwise, if $u_k.status \neq \texttt{crashed}$, then $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-REC$_h$ except the variable now. It follows that $u.scheduled\text{-}rqsts?(h') = \emptyset$ and $u.archived\text{-}pkts?(h') = \emptyset$. Thus, the invariant assertion holds in u.

❒ $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$, such that $source(p) = h'$: letting $\langle s_p, i_p \rangle = id(p)$, we analyze the effects of $\texttt{rm-send}_h(p)$ by cases. First, if $\neg(u_k.status = \texttt{member} \wedge h = s_p)$, then $\texttt{rm-send}_h(p)$ does not affect the state of RM-CLIENT$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

Second, consider the case where $u_k.status = \texttt{member} \wedge h = s_p$. If p is the foremost packet to be transmitted by h', that is, $u_k.min\text{-}seqno(h') = \perp$, then it follows that $u_k.window?(h') = \emptyset$. Thus, Invariants 4.5 and 4.15 imply that $u_k.scheduled\text{-}rqsts?(h') = \emptyset$ and $u_k.archived\text{-}pkts?(h') = \emptyset$. If p is the next packet from h', that is, $u_k.min\text{-}seqno(s_p) \neq \perp$ and $i_p = u_k.max\text{-}seqno(s_p) + 1$, then it is the case that $id(p) \notin u_k.window?(h')$. Thus, Invariants 4.5 and 4.15 imply that $id(p) \notin u_k.scheduled\text{-}rqsts?(h')$ and $id(p) \notin u_k.archived\text{-}pkts?(h')$.

In either case the $\texttt{rm-send}_h(p)$ adds $id(p)$ to the variable $archived\text{-}pkts?(h')$ and does not affect $scheduled\text{-}rqsts?(h')$. It follows that $u.scheduled\text{-}rqsts?(h') = u_k.scheduled\text{-}rqsts?(h')$ and $u.archived\text{-}pkts?(h') = u_k.archived\text{-}pkts?(h') \cup id(p)$. From the induction hypothesis, it is the case that $u_k.scheduled\text{-}rqsts?(h') \cap u_k.archived\text{-}pkts?(h') = \emptyset$. Since it is the case that $id(p) \notin u_k.scheduled\text{-}rqsts?(h')$, it follows that $u.scheduled\text{-}rqsts?(h') \cap u.archived\text{-}pkts?(h') = \emptyset$, as needed.

❒ $\texttt{schdl-rqst}_h(s,i)$, for $s \in H, i \in \mathbb{N}$, such that $s = h'$: the $\texttt{schdl-rqst}_h(s,i)$ action schedules a request for $\langle s,i \rangle$ and does not affect $archived\text{-}pkts?(h')$; that is, $u.scheduled\text{-}rqsts?(h') = u_k.scheduled\text{-}rqsts?(h') \cup \langle s,i \rangle$ and $u.archived\text{-}pkts?(h') = u_k.archived\text{-}pkts?(h')$.

From the precondition of $\texttt{schdl-rqst}_h(s,i)$, it follows that $\langle s,i \rangle \in u_k.to\text{-}be\text{-}requested?(h')$. From Invariant 4.16, it follows that $\langle s,i \rangle \notin u_k.archived\text{-}pkts?(h')$. Since it is the case that $u.archived\text{-}pkts? = u_k.archived\text{-}pkts?$, it follows that $\langle s,i \rangle \notin u.archived\text{-}pkts?(h')$. From the induction hypothesis, it is the case that $u_k.scheduled\text{-}rqsts?(h') \cap u_k.archived\text{-}pkts?(h') = \emptyset$. Thus, it follows that $u.scheduled\text{-}rqsts?(h') \cap u.archived\text{-}pkts?(h') = \emptyset$, as needed.

❒ $\texttt{send-rqst}_h(s,i)$, for $s \in H, i \in \mathbb{N}$, such that $s = h'$: from the precondition of $\texttt{send-rqst}_h(s,i)$, it is the case that $\langle s,i \rangle \in u_k.scheduled\text{-}rqsts?(h')$. Since $\texttt{send-rqst}_h(s,i)$ simply backs-off the request scheduled for $\langle s,i \rangle$, it follows that $u.scheduled\text{-}rqsts?(h') = u_k.scheduled\text{-}rqsts?(h')$. Moreover, $\texttt{send-rqst}_h(s,i)$ does not affect the variable $archived\text{-}pkts?(h')$. Thus, it follows that $u.archived\text{-}pkts?(h') = u_k.archived\text{-}pkts?(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-pkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $type(p) \in \{\texttt{DATA}, \texttt{REPL}\}$ and $source(p) = h'$: in this case, if the $\texttt{process-pkt}_h(p)$ action archives the packet $strip(p)$, then it also cancels any requests scheduled for $id(p)$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-pkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $type(p) = \texttt{RQST}$ and $source(p) = h'$: in this case, the $\texttt{process-pkt}_h(p)$ action schedules a request for $id(p)$ only if $h \neq s_p$ and $id(p) \notin u_k.archived\text{-}pkts?(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒

Invariant 4.18 *For $h, h' \in H$ and any reachable state u of* SRM-REC$_h$*, it is the case that* $u.\textit{to-be-requested?}(h') \cap u.\textit{scheduled-rqsts?}(h') = \emptyset$.

Proof: Let α be any finite timed execution of SRM-REC$_h$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM-REC$_h$, it follows that $u.\textit{to-be-requested?}(h') = \emptyset$ and $u.\textit{scheduled-rqsts?}(h') = \emptyset$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.\textit{lstate}$. For the step from u_k to u we consider only the actions that affect the variables $\textit{to-be-requested?}(h')$ and $\textit{scheduled-rqsts?}(h')$.

❐ $\texttt{rm-leave}_h$: if $u_k.\textit{status} = \texttt{crashed}$, then $\texttt{rm-leave}_h$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u. Otherwise, if $u_k.\textit{status} \neq \texttt{crashed}$, then $\texttt{rm-leave}_h$ reinitializes all the variables of SRM-REC$_h$ except the variable *now*. It follows that $u.\textit{to-be-requested?}(h') = \emptyset$ and $u.\textit{scheduled-rqsts?}(h') = \emptyset$. Thus, the invariant assertion holds in u.

❐ $\texttt{schdl-rqst}_h(s,i)$, for $s \in H$ and $i \in \mathbb{N}$, such that $s = h'$: from the precondition of the action $\texttt{schdl-rqst}_h(s,i)$, it follows that $\langle s,i\rangle \in u_k.\textit{to-be-requested?}(h')$. From the induction hypothesis, it follows that $\langle s,i\rangle \notin u_k.\textit{scheduled-rqsts?}(h')$. The action $\texttt{schdl-rqst}_h(s,i)$ adds the element $\langle s,i\rangle$ to $\textit{scheduled-rqsts?}(h')$ and removes it from $\textit{to-be-requested?}(h')$; that is, $u.\textit{to-be-requested?}(h') = u_k.\textit{to-be-requested?}(h') \setminus \{\langle s,i\rangle\}$ and $u.\textit{scheduled-rqsts?}(h') = u_k.\textit{scheduled-rqsts?}(h') \cup \{\langle s,i\rangle\}$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❐ $\texttt{send-rqst}_h(s,i)$, for $s \in H, i \in \mathbb{N}$, such that $s = h'$: from the precondition of $\texttt{send-rqst}_h(s,i)$, it is the case that $\langle s,i\rangle \in u_k.\textit{scheduled-rqsts?}(h')$. Thus, the induction hypothesis implies that $\langle s,i\rangle \notin u_k.\textit{to-be-requested?}(h')$. The $\texttt{send-rqst}_h(s,i)$ action does not affect the variable $\textit{to-be-requested?}(h')$. Thus, it follows that $u.\textit{to-be-requested?}(h') = u_k.\textit{to-be-requested?}(h')$. Moreover, since $\texttt{send-rqst}_h(s,i)$ simply backs-off the request scheduled for $\langle s,i\rangle$, it follows that $u.\textit{scheduled-rqsts?}(h') = u_k.\textit{scheduled-rqsts?}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❐ $\texttt{process-pkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $\textit{type}(p) \in \{\texttt{DATA}, \texttt{REPL}\}$, $\langle s_p, i_p\rangle = \textit{id}(p)$, and $s_p = h'$: we analyze the effects of $\texttt{process-pkt}_h(p)$ by cases. First, if $u_k.\textit{status} \neq \texttt{member}$, then $\texttt{rm-send}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.\textit{status} = \texttt{member}$. If p is either the foremost or a proper packet from s_p, then $\texttt{process-pkt}_h(p)$ removes $\langle s_p, i_p\rangle$ from $\textit{to-be-requested?}(h')$ and $\textit{scheduled-rqsts?}(h')$ and adds $\{\langle s_p, i'\rangle \mid i' \in \mathbb{N}, u_k.\textit{max-seqno}(s_p) < i' < i_p\}$ to $\textit{to-be-requested?}(h')$ only if $h \neq s_p$ and $u_k.\textit{max-seqno}(s_p) < i_p$. Invariant 4.15 implies that $u_k.\textit{scheduled-rqsts?}(h') \cap \{\langle s_p, i'\rangle \mid i' \in \mathbb{N}, u_k.\textit{max-seqno}(s_p) < i' < i_p\} = \emptyset$. Thus, the induction hypothesis implies that $u.\textit{to-be-requested?}(h') \cap u.\textit{scheduled-rqsts?}(h') = \emptyset$.

Otherwise, if p is neither the foremost nor a proper packet from s_p, $\texttt{process-pkt}_h(p)$ does not affect the variables $\textit{to-be-requested?}(h')$ and $\textit{scheduled-rqsts?}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

❐ $\texttt{process-pkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $\textit{type}(p) = \texttt{RQST}$, $\langle s_p, i_p\rangle = \textit{id}(p)$, and $s_p = h'$: we analyze the effects of $\texttt{process-pkt}_h(p)$ by cases. First, if $u_k.\textit{status} \neq \texttt{member}$, then $\texttt{rm-send}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.\textit{status} = \texttt{member}$, p is a proper packet from s_p, and $h \neq s_p$. In this case, if $\langle s_p, i_p\rangle \notin u_k.\textit{scheduled-rqsts?}(h')$, then the $\texttt{process-pkt}_h(p)$ action add $\langle s_p, i_p \rangle$ to *scheduled-rqsts?*(h') by scheduling a request for $\langle s_p, i_p \rangle$ and removes $\langle s_p, i_p \rangle$ from *to-be-requested?*(h'). Moreover, if $u_k.\mathit{max\text{-}seqno}(s_p) < i_p$, then $\texttt{process-pkt}_h(p)$ adds $\{\langle s_p, i' \rangle \mid i' \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(s_p) < i' < i_p\}$ to *to-be-requested?*(h'). Invariant 4.15 implies that $u_k.\mathit{scheduled\text{-}rqsts?}(h') \cap \{\langle s_p, i' \rangle \mid i' \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(s_p) < i' < i_p\} = \emptyset$. Thus, the induction hypothesis implies that $u.\mathit{to\text{-}be\text{-}requested?}(h') \cap u.\mathit{scheduled\text{-}rqsts?}(h') = \emptyset$.

Otherwise, the $\texttt{process-pkt}_h(p)$ action does not affect the variables *to-be-requested?*(h') and *scheduled-rqsts?*(h'). Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒ $\texttt{process-pkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $\mathit{type}(p) = \texttt{SESS}$: we analyze the effects of the $\texttt{process-pkt}_h(p)$ action by cases. First, if $u_k.\mathit{status} \neq \texttt{member}$, then $\texttt{process-pkt}_h(p)$ does not affect the state of SRM-REC$_h$. Thus, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k.\mathit{status} = \texttt{member}$. In this case, if $h \neq h'$, $u_k.\mathit{status} = \texttt{member}$ and there exists $\langle h', i' \rangle \in \mathit{seqno\text{-}rprts}(p)$, for $i' \in \mathbb{N}$, such that $u_k.\mathit{max\text{-}seqno}(h') < i'$, then $\texttt{process-pkt}_h(p)$ adds $\{\langle h', i'' \rangle \mid i'' \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(h') < i'' \leq i'\}$ to *to-be-requested?*. Invariant 4.15 implies that $u_k.\mathit{scheduled\text{-}rqsts?}(h') \cap \{\langle h', i'' \rangle \mid i'' \in \mathbb{N}, u_k.\mathit{max\text{-}seqno}(h') < i'' \leq i'\} = \emptyset$. Moreover, since the $\texttt{process-pkt}_h(p)$ action does not affect the variable *scheduled-rqsts?*(h'), it is the case that $u.\mathit{scheduled\text{-}rqsts?}(h') = u_k.\mathit{scheduled\text{-}rqsts?}(h')$. Thus, it follows that $u.\mathit{to\text{-}be\text{-}requested?}(h') \cap u.\mathit{scheduled\text{-}rqsts?}(h') = \emptyset$.

Otherwise, if either $h = h'$, $u_k.\mathit{status} \neq \texttt{member}$, or there does not exist $\langle h', i' \rangle \in \mathit{seqno\text{-}rprts}(p)$, for $i' \in \mathbb{N}$, such that $u_k.\mathit{max\text{-}seqno}(h') < i'$, then $\texttt{process-pkt}_h(p)$ does not affect the variables *to-be-requested?*(h') and *scheduled-rqsts?*(h'). Thus, the induction hypothesis implies that the invariant assertion holds in u.

❒

Invariant 4.19 *Let u be any reachable state of* SRM-REC$_h$. *For $s \in H$, $i \in \mathbb{N}$, $t, t' \in \mathbb{R}^{\geq 0}$, and $k \in \mathbb{N}^+$, if $\langle s, i, t \rangle \in$ pending-rqsts and $\langle s, i, t', k \rangle \in$ scheduled-rqsts, then $t < t'$.*

Proof: From Assumption 4.1, it is the case that $C_3 < C_1$. Thus, the expiration time of the back-off abstinence period precedes the transmission time of the respective request. ❒

Invariant 4.20 *Let u be any reachable state of* SRM-REC$_h$. *For $h, s \in H$ and $i \in \mathbb{N}$, if the action $\texttt{send-rqst}_h(s,i)$ is enabled in u, i.e., $u.\mathit{Pre}(\texttt{send-rqst}_h(s,i)) = \texttt{True}$, then $\langle s, i \rangle \notin u.\mathit{pending\text{-}rqsts?}$.*

Proof: Suppose that $u.\mathit{Pre}(\texttt{send-rqst}_h(s,i)) = \texttt{True}$. From the precondition of the action $\texttt{send-rqst}_h(s,i)$, it follows that there exists $k \in \mathbb{N}^+$ such that $\langle s, i, t', k \rangle \in \mathit{scheduled\text{-}rqsts}$, for $t' = u.\mathit{now}$. Invariant 4.19 implies that there does not exist $t \in \mathbb{R}^{\geq 0}$ such that $\langle s, i, t \rangle \in \mathit{pending\text{-}rqsts}$ and $t' \leq t$. Since $t' = u.\mathit{now}$, it follows that $\langle s, i \rangle \notin u.\mathit{pending\text{-}rqsts?}$. ❒

Lemma 4.2 *Let $u, u' \in \mathit{states}(\mathrm{SRM}_I)$ be any reachable states of SRM_I, α be any timed execution fragment of SRM_I, such that $u = \alpha.\mathit{fstate}$ and $u' = \alpha.\mathit{lstate}$. It is the case that $u[\mathrm{SRM}].\mathit{sent\text{-}pkts} \subseteq u'[\mathrm{SRM}].\mathit{sent\text{-}pkts}$.*

Proof: Follows from a simple induction on the length of the timed execution fragment $\alpha_{uu'}$ of SRM_I leading from u to u'. The key to this proof is that, for any $p \in P_{\text{RM-CLIENT}}$, the variable *trans-time*(p) is affected only by the action *rm-send*$_h(p)$, for $h = \mathit{source}(p)$, and this action may set it only to a value other than $\perp$. ❒

Invariant 4.21 *Let $u \in states(\mathrm{SRM}_I)$ be any reachable state of SRM_I. For any $s \in H$ and $i, i' \in \mathbb{N}, i \leq i'$, if $\langle s, i \rangle \in u[\mathrm{SRM}].sent\text{-}pkts?(s)$ and $\langle s, i' \rangle \in u[\mathrm{SRM}].sent\text{-}pkts?(s)$, then it is the case that $\langle s, i'' \rangle \in u[\mathrm{SRM}].sent\text{-}pkts?(s)$, for any $i'' \in \mathbb{N}, i \leq i'' \leq i'$.*

Proof: Follows directly from Lemma 3.3 and the fact that, for any $p \in P_{\text{RM-CLIENT}}$, the variable *trans-time*(p) may only be set by SRM_I to a value other than $\perp$ by the *rm-send*$_h(p)$, for $h = source(p)$. ❒

Lemma 4.3 *Let $s, h \in H$, $i \in \mathbb{N}$, and $u \in states(\mathrm{SRM}_I)$ be any reachable state of SRM_I, such that $\langle s, i \rangle \in u[\mathrm{SRM\text{-}REC}_h].archived\text{-}pkts?$. Moreover, let α be any timed execution fragment of SRM_I that starts in u, does not contain a* `rm-leave`$_h$ *action, and ends in some $u' \in states(\mathrm{SRM}_I)$. Then, it is the case that $\langle s, i \rangle \in u'[\mathrm{SRM\text{-}REC}_h].archived\text{-}pkts?$.*

Proof: Follows from a simple induction on the length of α. The key point of the induction is that none of the actions of SRM-REC$_h$, except the action `rm-leave`$_h$ which is absent in α, either remove elements from or initialize the set SRM-REC$_h$.*archived-pkts?*. ❒

Lemma 4.4 *Let $h \in H$, $i \in \mathbb{N}$, and $u \in states(\mathrm{SRM}_I)$ be any reachable state of SRM_I, such that $u[\mathrm{SRM\text{-}MEM}_h].status =$* `crashed`. *Moreover, let α be any timed execution fragment of SRM_I that starts in u and ends in some $u' \in states(\mathrm{SRM}_I)$. Then, it is the case that $u'[\mathrm{SRM\text{-}MEM}_h].status =$* `crashed`.

Proof: Follows from a simple induction on the length of α. The key point of the induction is that, once the host h has crashed, i) none of the input actions of SRM-MEM$_h$, except the action `crash`$_h$ which sets the SRM-MEM$_h$.*status* variable to the value `crashed`, affect the state of the SRM-MEM$_h$ automaton, and ii) none of the locally controlled actions, except the time passage action which does not affect the SRM-MEM$_h$.*status* variable, are enabled. ❒

Lemma 4.5 *Let $s, h \in H$, $i \in \mathbb{N}$, and $u \in states(\mathrm{SRM}_I)$ be any reachable state of SRM_I, such that $\langle s, i \rangle \in u[\mathrm{SRM\text{-}REC}_h].scheduled\text{-}rqsts?$. Moreover, let α be any timed execution fragment of SRM_I that starts in u, does not contain a* `rm-leave`$_h$ *action, and ends in some $u' \in states(\mathrm{SRM}_I)$. Then, either $\langle s, i \rangle \in u'[\mathrm{SRM\text{-}REC}_h].scheduled\text{-}rqsts?$ or $\langle s, i \rangle \in u'[\mathrm{SRM\text{-}REC}_h].archived\text{-}pkts?$.*

Proof: Follows from a simple induction on the length of α. The key points of the induction are that: i) whenever the elements of SRM-REC$_h$.*scheduled-rqsts* pertaining to $\langle s, i \rangle$ are removed from SRM-REC$_h$.*scheduled-rqsts* then an element pertaining to $\langle s, i \rangle$ is added to either SRM-REC$_h$.*scheduled-rqsts* or $\langle s, i \rangle \in$ SRM-REC$_h$.*archived-pkts?*, and ii) from Lemma 4.3, none of the actions of SRM-REC$_h$, except the action `rm-leave`$_h$ which is absent in α, remove elements from the set SRM-REC$_h$.*archived-pkts?*. ❒

Lemma 4.6 *Let $s, h \in H$, $i \in \mathbb{N}$, $t \in \mathbb{R}^{\geq 0}$, $k \in \mathbb{N}^+$, and $u \in states(\mathrm{SRM}_I)$ be any reachable state of SRM_I, such that $u[\mathrm{SRM\text{-}REC}_h].status =$* `member` *and $\langle s, i, t, k \rangle \in u[\mathrm{SRM\text{-}REC}_h].scheduled\text{-}rqsts$. Moreover, let α be any timed execution fragment of SRM_I that starts in u, contains neither* `crash`$_h$ *nor* `rm-leave`$_h$ *actions, and ends in some $u' \in states(\mathrm{SRM}_I)$, such that $t < u'.now$ and $\langle s, i, t', k' \rangle \in u'[\mathrm{SRM\text{-}REC}_h].scheduled\text{-}rqsts$, for $t' \in \mathbb{R}^{\geq 0}$ and $k' \in \mathbb{N}^+$. Then, it is the case that $k < k'$.*

Proof: Invariant 4.17 and Lemmas 4.3 and 4.5 imply that in any state u'' in α it is the case that $\langle s,i\rangle \in u''[\text{SRM-REC}_h].\mathit{scheduled\text{-}rqsts?}$. However, since $\langle s,i,t,k\rangle \in u[\text{SRM-REC}_h].\mathit{scheduled\text{-}rqsts}$, $t < u'.\mathit{now}$ and time is not allowed to progress past the scheduled transmission time of any request, it follows that the request for $\langle s,i\rangle$ is rescheduled for transmission in α for a point in time no earlier than $u'.\mathit{now}$. The only actions that may reschedule the request for $\langle s,i\rangle$ are the actions $\mathit{send\text{-}rqst}_h(s,i)$ and $\mathit{process\text{-}pkt}_h(p)$, for $p \in P_{\text{SRM}}$, such that $id(p) = \langle s,i\rangle$ and $type(p) = \texttt{RQST}$. Whenever either of these actions reschedule the request for $\langle s,i\rangle$, they increment the element of the tuple corresponding to the round count. ❐

Lemma 4.7 *The occurrence of either a* $\texttt{send-rqst}_h(s,i)$*, or* $\texttt{send-repl}_h(s,i)$ *action, for* $h,s \in H$*, and* $i \in \mathbb{N}$*, in any admissible timed execution* α *of* SRM_I *is instantaneously succeeded in* α *by the occurrence of either a* $\texttt{crash}_h$*,* $\texttt{rm-leave}_h$*, or* $\texttt{rec-msend}_h(p)$ *action, for* $p \in P_{\text{SRM}}$*,* $id(p) = \langle s,i\rangle$*, and* $type(p)$ *equal to either* RQST*, or* REPL*, respectively.*

Proof: We consider the case of a $\texttt{send-rqst}_h(s,i)$ action; the case of a $\texttt{send-repl}_h(s,i)$ action is analogous. The $\texttt{send-rqst}_h(s,i)$ action adds a RQST packet for $\langle s,i\rangle$ to the variable $\text{SRM-REC}_h.\mathit{msend\text{-}buff}$. Moreover, SRM-REC_h prevents time from elapsing while h is operational and the buffer $\text{SRM-REC}_h.\mathit{msend\text{-}buff}$ is non-empty; that is, while $\text{SRM-REC}_h.\mathit{status} \neq \texttt{crashed} \wedge \text{SRM-REC}_h.\mathit{msend\text{-}buff} \neq \emptyset$. ❐

Lemma 4.8 *The occurrence of an action* $\texttt{rec-msend}_h(p)$*, for* $h \in H$ *and* $p \in P_{\text{SRM}}$*, in any admissible timed execution* α *of* SRM_I *is instantaneously succeeded in* α *by the occurrence of either a* $\texttt{crash}_h$*,* $\texttt{rm-leave}_h$*, or* $\texttt{msend}_h(pkt)$ *action, for* $pkt \in P_{\text{IPMCAST-CLIENT}}$*, such that* $strip(pkt) = p$*.*

Proof: The $\texttt{rec-msend}_h(p)$ action adds an element to the variable $\text{SRM-IPBUFF}_h.\mathit{msend\text{-}buff}$. Moreover, SRM-IPBUFF_h prevents time from elapsing while $\text{SRM-IPBUFF}_h.\mathit{status} \neq \texttt{crashed} \wedge \text{SRM-IPBUFF}_h.\mathit{msend\text{-}buff} \neq \emptyset$. ❐

Lemma 4.9 *The occurrence of an action* $\texttt{mrecv}_h(pkt)$*, for* $h \in H$ *and* $pkt \in P_{\text{IPMCAST-CLIENT}}$*, in a state* $u \in states(\text{SRM}_I)$ *in any admissible timed execution* α *of* SRM_I*, such that* $u[\text{SRM-IPBUFF}_h].\mathit{status} = \texttt{member}$*, is instantaneously succeeded in* α *by the occurrence of either a* $\texttt{crash}_h$*,* $\texttt{rm-leave}_h$*, or* $\texttt{process-pkt}_h(p)$ *action, for* $p \in P_{\text{SRM}}$*, such that* $p = strip(pkt)$*.*

Proof: Since $u[\text{SRM-IPBUFF}_h].\mathit{status} = \texttt{member}$, the particular occurrence of the $\texttt{mrecv}_h(pkt)$ action adds the element $strip(pkt)$ to the variable $\text{SRM-IPBUFF}_h.\mathit{recv\text{-}buff}$. Moreover, SRM-IPBUFF_h prevents time from elapsing while $\text{SRM-IPBUFF}_h.\mathit{status} \neq \texttt{crashed} \wedge \text{SRM-IPBUFF}_h.\mathit{recv\text{-}buff} \neq \emptyset$. ❐

Lemma 4.10 *Let* α *be any admissible execution of* SRM_I *containing the discrete transition* (u,π,u')*, for* $u,u' \in states(\text{SRM}_I)$*,* $h \in H$*,* $p \in P_{\text{RM-CLIENT}}$*,* $\langle s_p, i_p\rangle = id(p)$*, and* $\pi = \texttt{rm-send}_h(p)$*. If either* $u[\text{SRM-REC}_h].\mathit{min\text{-}seqno}(s_p) =\perp$ *or* $u[\text{SRM-REC}_h].\mathit{min\text{-}seqno}(s_p) \neq\perp \wedge i_p = u[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(s_p) + 1$*, then the discrete transition* (u,π,u') *is instantaneously succeeded in* α *by the occurrence of either a* $\texttt{crash}_h$*,* $\texttt{rm-leave}_h$*, or* $\texttt{rec-msend}_h(pkt)$ *action, for* $pkt \in P_{\text{SRM}}$*, such that* $pkt = \mathit{comp\text{-}data\text{-}pkt}(p)$*.*

Proof: Suppose that either $u[\text{SRM-REC}_h].\mathit{min\text{-}seqno}(s_p) =\perp$ or $u[\text{SRM-REC}_h].\mathit{min\text{-}seqno}(s_p) \neq\perp$ and $i_p = u[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(s_p) + 1$. Then, the discrete transition (u,π,u') adds the element *pkt* to SRM-REC$_h$.*msend-buff*. Moreover, SRM-REC$_h$ prevents time from elapsing while SRM-REC$_h$.*status* $\neq$ `crashed` $\wedge$ SRM-REC$_h$.*msend-buff* $\neq \emptyset$. ❐

We now present some invariants pertaining to the SRM$_I$ automaton.

Invariant 4.22 *For $h \in H$ and any reachable state u of* SRM$_I$*, it is the case that:*

1. u[RM-CLIENT$_h$].*status* = `idle` $\Leftrightarrow$ u[SRM-MEM$_h$].*status* = `idle`,
2. u[RM-CLIENT$_h$].*status* = `member` $\Leftrightarrow$ u[SRM-MEM$_h$].*status* = `member`,
3. u[RM-CLIENT$_h$].*status* = `crashed` $\Leftrightarrow$ u[SRM-MEM$_h$].*status* = `crashed`,
4. u[RM-CLIENT$_h$].*status* = `joining` $\Leftrightarrow$ u[SRM-MEM$_h$].*status* $\in$ *Joining, and*
5. u[RM-CLIENT$_h$].*status* = `leaving` $\Leftrightarrow$ u[SRM-MEM$_h$].*status* $\in$ *Leaving.*

Proof: Let α be any finite timed execution of SRM$_I$ leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM$_I$, it is the case that u[RM-CLIENT$_h$].*status* = `idle` and u[SRM-MEM$_h$].*status* = `idle`. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions that affect the variables RM-CLIENT$_h$.*status* and SRM-MEM$_h$.*status*.

❐ `crash`$_h$: the action `crash`$_h$ sets both variables RM-CLIENT$_h$.*status* and SRM-MEM$_h$.*status* to the value `crashed`. Thus, the invariant assertion holds in u.

❐ `rm-join`$_h$: from the precondition of the `rm-join`$_h$ action, it follows that u_k[RM-CLIENT$_h$].*status* = `idle`. From the induction hypothesis it follows that u_k[SRM-MEM$_h$].*status* = `idle`. Thus, the action `rm-join`$_h$ sets RM-CLIENT$_h$.*status* to `joining` and SRM-MEM$_h$.*status* to `join-rqst-pending`; that is, u[RM-CLIENT$_h$].*status* = `joining` and u[SRM-MEM$_h$].*status* $\in$ *Joining*. It follows that the invariant assertion holds in u.

❐ `mjoin`$_h$: from the precondition of the `mjoin`$_h$ action, it follows that u_k[SRM-MEM$_h$].*status* $\in$ *Joining*. From the induction hypothesis it follows that u_k[RM-CLIENT$_h$].*status* = `joining`. The action `mjoin`$_h$ sets the variable SRM-MEM$_h$.*status* to `join-pending` and does not affect the variable RM-CLIENT$_h$.*status*. Thus, it is the case that u[SRM-MEM$_h$].*status* $\in$ *Joining* and u[RM-CLIENT$_h$].*status* = `joining`. It follows that the invariant assertion holds in u.

❐ `mjoin-ack`$_h$: we first consider the case where u_k[SRM-MEM$_h$].*status* $\notin$ *Joining*. In this case, `mjoin-ack`$_h$ affects neither RM-CLIENT$_h$.*status* nor SRM-MEM$_h$.*status*. Thus, the induction hypothesis implies the invariant assertion in u.

   Second, we consider the case where u_k[SRM-MEM$_h$].*status* $\in$ *Joining*. In this case, `mjoin-ack`$_h$ sets the variable SRM-MEM$_h$.*status* to `join-ack-pending` and does not affect RM-CLIENT$_h$.*status*. Since u_k[SRM-MEM$_h$].*status* $\in$ *Joining*, the induction hypothesis implies that u_k[RM-CLIENT$_h$].*status* = `joining`. Moreover, since `mjoin-ack`$_h$ does not affect RM-CLIENT$_h$.*status*, it follows that u[RM-CLIENT$_h$].*status* = `joining`. Thus, the invariant assertion holds in u.

❐ `rm-join-ack`$_h$: from the precondition of `rm-join-ack`$_h$, it follows that u_k[SRM-MEM$_h$].*status* $\in$ *Joining*. From the induction hypothesis it follows that u_k[RM-CLIENT$_h$].*status* = `joining`. Thus, the `rm-join-ack`$_h$ action sets both SRM-MEM$_h$.*status* and RM-CLIENT$_h$.*status* to `member`. It follows that the invariant assertion holds in u.

❐ `rm-leave`$_h$: the reasoning for this action is analogous to that of `rm-join`$_h$.

❐ `mleave`$_h$: the reasoning for this action is analogous to that of `mjoin`$_h$.

❒ $\texttt{mleave-ack}_h$: the reasoning for this action is analogous to that of $\texttt{mjoin-ack}_h$.

❒ $\texttt{rm-leave-ack}_h$: the reasoning for this action is analogous to that of $\texttt{rm-join-ack}_h$.

❒

Invariant 4.23 *For $h \in H$ and any reachable state u of* SRM_I*, it is the case that* $u[\text{RM-CLIENT}_h].\mathit{seqno} = u[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(h)$.

Proof: Let α be any finite timed execution of SRM_I leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM_I, it follows that $u[\text{RM-CLIENT}_h].\mathit{seqno} = \perp$ and $u[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(h) = \perp$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.\mathit{lstate}$. For the step from u_k to u, we consider only the $\texttt{rm-send}_h(p)$ action, since this is the only action that affects the variables $\text{RM-CLIENT}_h.\mathit{seqno}$ and $\text{SRM-REC}_h.\mathit{max\text{-}seqno}(h)$.

From the precondition of $\texttt{rm-send}_h(p)$, it is the case that $u_k[\text{RM-CLIENT}_h].\mathit{status} = \texttt{member}$, $\mathit{source}(p) = h$, and either $u_k[\text{RM-CLIENT}_h].\mathit{seqno} = \perp$ or $\mathit{seqno}(p) = u_k[\text{RM-CLIENT}_h].\mathit{seqno} + 1$. The effects of $\texttt{rm-send}_h(p)$ are to set $\text{RM-CLIENT}_h.\mathit{seqno}$ to $\mathit{seqno}(p)$.

Since $u_k[\text{RM-CLIENT}_h].\mathit{status} = \texttt{member}$, Invariant 4.22 implies that $u_k[\text{SRM-REC}_h].\mathit{status} = \texttt{member}$. From the induction hypothesis, it is the case that $u_k[\text{RM-CLIENT}_h].\mathit{seqno} = u_k[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(h)$. Thus, it is the case that either $u_k[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(h) = \perp$ or $\mathit{seqno}(p) = u_k[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(h) + 1$. In either case, the $\texttt{rm-send}_h(p)$ sets $\text{SRM-REC}_h.\mathit{max\text{-}seqno}(h)$ to $\mathit{seqno}(p)$. Thus, it follows that $u[\text{RM-CLIENT}_h].\mathit{seqno} = u[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(h)$. ❒

Invariant 4.24 *For $h \in H$ and any reachable state u of* SRM_I*, it is the case that:*

1. $u[\text{SRM-MEM}_h].\mathit{status} = \texttt{crashed} \Leftrightarrow u[\text{SRM-IPBUFF}_h].\mathit{status} = \texttt{crashed}$
   $\wedge u[\text{SRM-MEM}_h].\mathit{status} = \texttt{member} \Leftrightarrow u[\text{SRM-IPBUFF}_h].\mathit{status} = \texttt{member}$ *and*
2. $u[\text{SRM-MEM}_h].\mathit{status} = \texttt{crashed} \Leftrightarrow u[\text{SRM-REC}_h].\mathit{status} = \texttt{crashed}$
   $\wedge u[\text{SRM-MEM}_h].\mathit{status} = \texttt{member} \Leftrightarrow u[\text{SRM-REC}_h].\mathit{status} = \texttt{member}$.

Proof: We prove that $u[\text{SRM-MEM}_h].\mathit{status} = \texttt{crashed} \Leftrightarrow u[\text{SRM-IPBUFF}_h].\mathit{status} = \texttt{crashed} \wedge u[\text{SRM-MEM}_h].\mathit{status} = \texttt{member} \Leftrightarrow u[\text{SRM-IPBUFF}_h].\mathit{status} = \texttt{member}$; the proof of the second claim is analogous.

Let α be any finite timed execution of SRM_I leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM_I, it follows that $u[\text{SRM-MEM}_h].\mathit{status} = \texttt{idle}$ and $u[\text{SRM-IPBUFF}_h].\mathit{status} = \texttt{idle}$. Thus, the invariant assertion is satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.\mathit{lstate}$. For the step from u_k to u, we consider only the actions that affect the variables $\text{SRM-MEM}_h.\mathit{status}$ and $\text{SRM-IPBUFF}_h.\mathit{status}$.

❒ $\texttt{crash}_h$: the action $\texttt{crash}_h$ sets both variables $\text{SRM-MEM}_h.\mathit{status}$ and $\text{SRM-IPBUFF}_h.\mathit{status}$ to the value $\texttt{crashed}$. Thus, the invariant assertion holds in u.

❒ $\texttt{rm-join}_h$: from the precondition of $\texttt{rm-join}_h$, it follows that $u_k[\text{RM-CLIENT}_h].\mathit{status} = \texttt{idle}$. Thus, Invariant 4.22 implies that $u_k[\text{SRM-MEM}_h].\mathit{status} = \texttt{idle}$. Since

$u_k[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$, the induction hypothesis implies that $u_k[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$.

Since `rm-join`$_h$ sets SRM-MEM$_h$.*status* to `join-rqst-pending`, it follows that $u[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Since `rm-join`$_h$ does not affect the variable SRM-IPBUFF$_h$.*status*, it follows that $u[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Thus, it follows that the invariant assertion holds in u.

❒ `mjoin`$_h$: from the precondition of `mjoin`$_h$, it follows that $u_k[\text{SRM-MEM}_h].status \in Joining$; that is, $u_k[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Thus, the induction hypothesis implies that $u_k[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$.

Since the action `mjoin`$_h$ sets the variable SRM-MEM$_h$.*status* to `join-pending`, it follows that $u[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Moreover, since `mjoin`$_h$ does not affect the variable SRM-IPBUFF$_h$.*status*, it follows that $u[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Thus, it follows that the invariant assertion holds in u.

❒ `mjoin-ack`$_h$: first, consider the case where $u_k[\text{SRM-MEM}_h].status \notin Joining$. Since in this case `mjoin-ack`$_h$ affects neither SRM-MEM$_h$.*status* nor SRM-IPBUFF$_h$.*status*, the induction hypothesis implies that the invariant assertion holds in u.

Second, consider the case where $u_k[\text{SRM-MEM}_h].status \in Joining$. Since $u_k[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$, the induction hypothesis implies that $u_k[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Since $u_k[\text{SRM-MEM}_h].status \in Joining$, the action `mjoin-ack`$_h$ sets SRM-MEM$_h$.*status* to `join-ack-pending`; that is, $u[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Since `mjoin`$_h$ does not affect the variable SRM-IPBUFF$_h$.*status*, it follows that $u[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Thus, it follows that the invariant assertion holds in u.

❒ `rm-join-ack`$_h$: from the precondition of `rm-join-ack`$_h$, it is the case that $u_k[\text{SRM-MEM}_h].status \in Joining$. Since $u_k[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$, the induction hypothesis implies that $u_k[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$.

The action `rm-join-ack`$_h$ sets SRM-MEM$_h$.*status* to `member`. Since $u_k[\text{SRM-IPBUFF}_h].status \neq$ `crashed`, it also sets SRM-IPBUFF$_h$.*status* to `member`. It follows that the invariant assertion holds in u.

❒ `rm-leave`$_h$: from the precondition of `rm-leave`$_h$, it is the case that $u_k[\text{RM-CLIENT}_h].status =$ `member`. Thus, Invariant 4.22 implies that $u_k[\text{SRM-MEM}_h].status =$ `member`. Moreover, the induction hypothesis implies that $u_k[\text{SRM-IPBUFF}_h].status =$ `member`.

Since $u_k[\text{SRM-MEM}_h].status =$ `member`, the `rm-leave`$_h$ action sets SRM-MEM$_h$.*status* to `leave-rqst-pending` and SRM-IPBUFF$_h$.*status* to `idle`. Thus, it is the case that $u[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$ and $u[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Thus, it follows that the invariant assertion holds in u.

❒ `mleave`$_h$: the reasoning for this action is analogous to that of `mjoin`$_h$.

❒ `mleave-ack`$_h$: the reasoning for this action is analogous to that of `mjoin-ack`$_h$.

❒ `rm-leave-ack`$_h$: from the precondition of the action `rm-leave-ack`$_h$, it is the case that $u_k[\text{SRM-MEM}_h].status =$ `leave-ack-pending`. Since $u_k[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$, the induction hypothesis implies that $u_k[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$.

The action `rm-leave-ack`$_h$ sets SRM-MEM$_h$.*status* to `idle` and does not affect the variable SRM-IPBUFF$_h$.*status*. Thus, it follows that $u[\text{SRM-MEM}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$ and $u[\text{SRM-IPBUFF}_h].status \notin \{\texttt{crashed}, \texttt{member}\}$. Thus, it follows that the invariant assertion holds in u.

❒

Invariant 4.25 *For $h \in H$ and any reachable state u of* SRM_I*, it is the case that, for any packet $p \in u[\text{SRM-REC}_h].\mathit{msend\text{-}buff}$:*

1. *$\mathit{type}(p) = \texttt{SESS} \Rightarrow \forall \langle h', i' \rangle \in \mathit{seqno\text{-}rprts}(p), \langle h', i' \rangle \in u[\text{SRM-REC}_h].\mathit{window?}(h')$, and*
2. *$\mathit{type}(p) \neq \texttt{SESS} \Rightarrow \mathit{id}(p) \in u[\text{SRM-REC}_h].\mathit{window?}(\mathit{source}(p))$.*

Proof: Let α be any finite timed execution of SRM_I leading to u. The proof is by induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM_I, it is the case that $u[\text{SRM-REC}_h].\mathit{msend\text{-}buff} = \emptyset$. Thus, the invariant assertion is trivially satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.\mathit{lstate}$. For the step from u_k to u we consider only the actions of SRM_I that affect the variables $\text{SRM-REC}_h.\mathit{msend\text{-}buff}$, $\text{SRM-REC}_h.\mathit{min\text{-}seqno}(h')$, and $\text{SRM-REC}_h.\mathit{max\text{-}seqno}(h')$, for all $h' \in H$.

❒ $\texttt{rm-leave}_h$, for $h \in H$: the action $\texttt{rm-leave}_h$ reinitializes the variables $\text{SRM-REC}_h.\mathit{msend\text{-}buff}$, $\text{SRM-REC}_h.\mathit{min\text{-}seqno}(h')$, and $\text{SRM-REC}_h.\mathit{max\text{-}seqno}(h')$, for all $h' \in H$. Thus, the invariant assertion is trivially satisfied in u.

❒ $\texttt{rm-send}_h(p)$, for $h \in H$ and $p \in P_{\text{RM-CLIENT}}$, such that $\langle s_p, i_p \rangle = \mathit{id}(p)$: from the precondition of $\texttt{rm-send}_h(p)$, it is the case that $u_k[\text{RM-CLIENT}_h].\mathit{status} = \texttt{member}$ and either $u_k[\text{RM-CLIENT}_h].\mathit{seqno} = \perp$ or $i_p = u_k[\text{RM-CLIENT}_h].\mathit{seqno} + 1$. Since $u_k[\text{RM-CLIENT}_h].\mathit{status} = \texttt{member}$, Invariants 4.22 and 4.24 imply that $u_k[\text{SRM-REC}_h].\mathit{status} = \texttt{member}$. Moreover, Invariant 4.23 implies that either $u_k[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(s_p) = \perp$ or $i_p = u_k[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(s_p) + 1$.

Thus, the $\texttt{rm-send}_h(p)$ action adds $\mathit{pkt} = \mathit{comp\text{-}data\text{-}pkt}(p)$ to $\text{SRM-REC}_h.\mathit{msend\text{-}buff}$ and $\mathit{id}(p)$ to $\text{SRM-REC}_h.\mathit{window?}(s_p)$. It follows that $u_k[\text{SRM-REC}_h].\mathit{window?}(s_p) \subseteq u[\text{SRM-REC}_h].\mathit{window?}(s_p)$. Moreover, the $\texttt{rm-send}_h(p)$ action does not affect the $\text{SRM-REC}_h.\mathit{window?}(h')$ variables, for $h' \in H, h' \neq s_p$.

The induction hypothesis and that facts $\mathit{type}(\mathit{pkt}) = \texttt{DATA}$, $\mathit{id}(\mathit{pkt}) = \mathit{id}(p)$, and $\mathit{id}(p) \in u[\text{SRM-REC}_h].\mathit{window?}(s_p)$, imply that the invariant assertion is satisfied in u.

❒ $\texttt{rec-msend}_h(p)$, for $h \in H$ and $p \in P_{\text{RM-CLIENT}}$: the $\texttt{rec-msend}_h(p)$ removes p from $\text{SRM-REC}_h.\mathit{msend\text{-}buff}$ and does not affect the $\text{SRM-REC}_h.\mathit{window?}(h')$ variables, for $h' \in H$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

❒ $\texttt{send-rqst}_h(s,i)$, for $s \in H$ and $i \in \mathbb{N}$: from the precondition of $\texttt{send-rqst}_h(s,i)$, it follows that $\langle s, i \rangle \in u_k[\text{SRM-REC}_h].\mathit{scheduled\text{-}rqsts?}(s)$. Thus, Invariant 4.15 implies that $\langle s, i \rangle \in u_k[\text{SRM-REC}_h].\mathit{window?}(s)$. Since the $\texttt{send-rqst}_h(s,i)$ action does not affect the variables $\text{SRM-REC}_h.\mathit{min\text{-}seqno}(s)$ and $\text{SRM-REC}_h.\mathit{max\text{-}seqno}(s)$, it does not affect the variable $\text{SRM-REC}_h.\mathit{window?}(s)$. Since $\langle s, i \rangle \in u_k[\text{SRM-REC}_h].\mathit{window?}(s)$, it follows that $\langle s, i \rangle \in u[\text{SRM-REC}_h].\mathit{window?}(s)$.

The $\texttt{send-rqst}_h(s,i)$ action adds a packet $\mathit{pkt} \in P_{SRM}$, such that $\mathit{type}(\mathit{pkt}) = \texttt{RQST}$ and $\mathit{id}(\mathit{pkt}) = \langle s, i \rangle$, to $\text{SRM-REC}_h.\mathit{msend\text{-}buff}$. Moreover, the $\texttt{send-repl}_h(s,i)$ action does not affect the $\text{SRM-REC}_h.\mathit{window?}(h')$ variables, for $h' \in H, h' \neq s$. The induction hypothesis and the fact that $\langle s, i \rangle \in u[\text{SRM-REC}_h].\mathit{window?}(s)$ imply that the invariant assertion is satisfied in u.

❒ $\texttt{send-repl}_h(s,i)$, for $s \in H$ and $i \in \mathbb{N}$: from the precondition of $\texttt{send-repl}_h(s,i)$, it follows that $\langle s, i \rangle \in u_k[\text{SRM-REC}_h].\mathit{scheduled\text{-}repls?}(s)$. Thus, Invariant 4.14 implies that $\langle s,i\rangle \in u_k[\text{SRM-REC}_h].archived?(s)$. Moreover, Invariant 4.5 implies that $\langle s,i\rangle \in u_k[\text{SRM-REC}_h].window?(s)$. Since the $\texttt{send-rqst}_h(s,i)$ action does not affect the variables $\text{SRM-REC}_h.min\text{-}seqno(s)$ and $\text{SRM-REC}_h.max\text{-}seqno(s)$, it does not affect the variable $\text{SRM-REC}_h.window?(s)$. Since $\langle s,i\rangle \in u_k[\text{SRM-REC}_h].window?(s)$, it follows that $\langle s,i\rangle \in u[\text{SRM-REC}_h].window?(s)$.

The $\texttt{send-repl}_h(s,i)$ action adds a packet $pkt \in P_{SRM}$, such that $type(pkt) = \texttt{REPL}$ and $id(pkt) = \langle s,i\rangle$, to $\text{SRM-REC}_h.msend\text{-}buff$. Moreover, the $\texttt{send-repl}_h(s,i)$ action does not affect the $\text{SRM-REC}_h.window?(h')$ variables, for $h' \in H, h' \neq s$. The induction hypothesis and the fact that $\langle s,i\rangle \in u[\text{SRM-REC}_h].window?(s)$ imply that the invariant assertion is satisfied in u.

❒ $\texttt{send-sess}_h$: the $\texttt{send-sess}_h$ action adds a packet $pkt \in P_{SRM}$ to $\text{SRM-REC}_h.msend\text{-}buff$, such that $type(pkt) = \texttt{SESS}$ and $seqno\text{-}rprts(pkt) = u_k[\text{SRM-REC}_h].max\text{-}seqno$. Since $seqno\text{-}rprts(pkt) = u_k[\text{SRM-REC}_h].max\text{-}seqno$, it follows that, for any $\langle s,i\rangle \in seqno\text{-}rprts(pkt)$, it is the case that $i = u_k[\text{SRM-REC}_h].max\text{-}seqno(s)$. Thus, Invariant 4.3 implies that $u_k[\text{SRM-REC}_h].min\text{-}seqno(s) \leq i$. It follows that $\langle s,i\rangle \in u_k[\text{SRM-REC}_h].window?(s)$. Since $\texttt{send-sess}_h$ affects neither $\text{SRM-REC}_h.min\text{-}seqno(s)$ nor $\text{SRM-REC}_h.max\text{-}seqno(s)$, it follows that $\langle s,i\rangle \in u[\text{SRM-REC}_h].window?(s)$.

The induction hypothesis and the fact that, for any $\langle s,i\rangle \in seqno\text{-}rprts(pkt)$, it is the case that $\langle s,i\rangle \in u[\text{SRM-REC}_h].window?(s)$, imply that the invariant assertion is satisfied in u.

❒ $\texttt{process-pkt}_h(p)$, for $h \in H$ and $p \in P_{\text{SRM}}$: the $\texttt{process-pkt}_h(p)$ action does not affect the variable $\text{SRM-REC}_h.msend\text{-}buff$ and may only add elements to the variables $u[\text{SRM-REC}_h].window?(h')$, for $h' \in H$. Thus, it follows that $u[\text{SRM-REC}_h].msend\text{-}buff = u_k[\text{SRM-REC}_h].msend\text{-}buff$ and $u_k[\text{SRM-REC}_h].window?(h') \subseteq u[\text{SRM-REC}_h].window?(h')$, for $h' \in H$. Thus, the induction hypothesis implies that the invariant assertion is satisfied in u.

❒

Invariant 4.26 *For any reachable state u of SRM_I, it is the case that, for all $h,h' \in H$, $u[\text{SRM-REC}_h].window?(h') \subseteq u[\text{SRM}].sent\text{-}pkts?(h')$.*

Proof: Let α be any finite timed execution of SRM_I leading to u. The proof is by strong induction on the length $n \in \mathbb{N}$ of α. For the base case, consider the finite timed execution α of length 0; that is, $\alpha = u$. Since u is a start state of SRM_I, it is the case that $u[\text{SRM-REC}_h].window?(h') = \emptyset$ and $u[\text{SRM}].sent\text{-}pkts?(h') = \emptyset$, for all $h,h' \in H$. Thus, the invariant assertion is trivially satisfied in u. For the inductive step, consider a timed execution α of length $k+1$, for $k \in \mathbb{N}$. Let α_k be the prefix of α containing the first k steps of α and $u_k = \alpha_k.lstate$. For the step from u_k to u we consider only the actions of SRM_I that affect the variables $\text{SRM-REC}_h.min\text{-}seqno(h')$, $\text{SRM-REC}_h.max\text{-}seqno(h')$, and $\text{SRM}.sent\text{-}pkts?(h')$, for all $h,h' \in H$.

❒ $\texttt{rm-leave}_m$, for $m \in H$: the action $\texttt{rm-leave}_m$ reinitializes the variables $\text{SRM-REC}_m.min\text{-}seqno(m')$ and $\text{SRM-REC}_m.max\text{-}seqno(m')$, for all $m' \in H$, and does not affect the variables $\text{SRM}.sent\text{-}pkts?(m')$, for all $m' \in H$. Thus, it follows that $u[\text{SRM-REC}_m].window?(m') \subseteq u[\text{SRM}].sent\text{-}pkts?(m')$, for all $m' \in H$. Moreover, the action $\texttt{rm-leave}_m$ does not affect any of the variables $\text{SRM-REC}_n.min\text{-}seqno(m')$, $\text{SRM-REC}_n.max\text{-}seqno(m')$, and $\text{SRM}.sent\text{-}pkts?(m')$, for $n \in H, n \neq m$ and $m' \in H$. Thus, from the induction hypothesis it is the case that $u[\text{SRM-REC}_n].window?(m') \subseteq u[\text{SRM}].sent\text{-}pkts?(m')$, for $n \in H, n \neq m$ and $m' \in H$. It follows that $u[\text{SRM-REC}_h].window?(h') \subseteq u[\text{SRM}].sent\text{-}pkts?(h')$, for all $h,h' \in H$.

❒ $\texttt{rm-send}_m(p)$, for $m \in H$ and $p \in P_{\text{RM-CLIENT}}$: from the precondition of $\texttt{rm-send}_m(p)$, it is the case that $u_k[\text{RM-CLIENT}_m].status = \texttt{member}$ and either $u_k[\text{RM-CLIENT}_m].seqno = \perp$ or

$seqno(p) = u_k[\text{RM-Client}_m].seqno + 1$. Since $u_k[\text{RM-Client}_m].status = \texttt{member}$, Invariants 4.22 and 4.24 imply that $u_k[\text{SRM-rec}_m].status = \texttt{member}$. Moreover, Invariant 4.23 implies that either $u_k[\text{SRM-rec}_m].max\text{-}seqno =\perp$ or $seqno(p) = u_k[\text{SRM-rec}_m].max\text{-}seqno + 1$. Thus, the $\texttt{rm-send}_m(p)$ action adds the element $id(p)$ to $\text{SRM-rec}_m.window?(m)$ and $\text{SRM}.sent\text{-}pkts?(m)$; that is, $u[\text{SRM-rec}_m].window?(m) = u_k[\text{SRM-rec}_m].window?(m) \cup \{id(p)\}$ and $u[\text{SRM}].sent\text{-}pkts?(m) = u_k[\text{SRM}].sent\text{-}pkts?(m) \cup \{id(p)\}$. From the induction hypothesis, it is the case that $u_k[\text{SRM-rec}_m].window?(m) \subseteq u_k[\text{SRM}].sent\text{-}pkts?(m)$. Thus, it follows that $u[\text{SRM-rec}_m].window?(m) \subseteq u[\text{SRM}].sent\text{-}pkts?(m)$.

The $\texttt{rm-send}_m(p)$ action does not affect the variables $\text{SRM-rec}_m.window?(m')$, $\text{SRM-rec}_{m'}.window?(m'')$ $\text{SRM}.sent\text{-}pkts?(m')$, for all $m' \in H, m' \neq m$ and $m'' \in H$. Thus, the induction hypothesis implies that $u[\text{SRM-rec}_m].window?(m') \subseteq u[\text{SRM}].sent\text{-}pkts?(m')$ and $u[\text{SRM-rec}_{m'}].window?(m'') \subseteq u[\text{SRM}].sent\text{-}pkts?(m'')$, for all $m' \in H, m' \neq m$ and $m'' \in H$.

It follows that $u[\text{SRM-rec}_h].window?(h') \subseteq u[\text{SRM}].sent\text{-}pkts?(h')$, for all $h, h' \in H$.

❒ $\texttt{process-pkt}_m(p)$, for $m \in H$ and $p \in P_{\text{SRM}}$, such that $type(p) \in \{\texttt{DATA}\}$: let $s_p \in H$ and $i_p \in \mathbb{N}$ be the source and the sequence number, respectively, of the packet p. From the precondition of $\texttt{process-pkt}_m(p)$, it follows that $p \in u_k[\text{SRM-IPbuff}_m].recv\text{-}buff$. Since the only action that may add p to the variable $\text{SRM-IPbuff}_m.recv\text{-}buff$ is $\texttt{mrecv}_m(pkt)$, for $pkt \in P_{\text{IPmcast-Client}}$, such that $strip(pkt) = p$, it follows that the action $\texttt{process-pkt}_m(p)$ is preceded in α by an action $\texttt{mrecv}_m(pkt)$. Let $(u_2, \texttt{mrecv}_m(pkt), u_1)$ be the discrete transition in α_k corresponding to the particular occurrence of the action $\texttt{mrecv}_m(pkt)$. Lemma 4.1 implies that the action $\texttt{mrecv}_m(pkt)$ is preceded in α_k by an action $\texttt{msend}_{m'}(pkt)$, for some $m' \in H$. Let $(u_4, \texttt{msend}_{m'}(pkt), u_3)$ be the discrete transition in α_k corresponding to the particular occurrence of the action $\texttt{msend}_{m'}(pkt)$. From the precondition of the action $\texttt{msend}_{m'}(pkt)$, it follows that $pkt \in u_4[\text{SRM-IPbuff}_{m'}].msend\text{-}buff$. The only action that may add pkt to the variable $\text{SRM-IPbuff}_{m'}.msend\text{-}buff$ is the action $\texttt{rec-msend}_{m'}(p)$. Thus, an action $\texttt{rec-msend}_{m'}(p)$ precedes u_4 in α_k. Let $(u_6, \texttt{rec-msend}_{m'}(p), u_5)$ be the discrete transition in α_k corresponding to the particular occurrence of the action $\texttt{rec-msend}_{m'}(p)$. From the precondition of the action $\texttt{rec-msend}_{m'}(p)$, it follows that $p \in u_6[\text{SRM-rec}_{m'}].msend\text{-}buff$.

Invariant 4.25 implies that $id(p) \in u_6[\text{SRM-rec}_{m'}].window?(s_p)$. From the induction hypothesis, it is the case that $u_6[\text{SRM-rec}_{m'}].window?(s_p) \subseteq u_6[\text{SRM}].sent\text{-}pkts?(s_p)$. Since $id(p) \in u_6[\text{SRM-rec}_{m'}].window?(s_p)$ and $u_6[\text{SRM-rec}_{m'}].window?(s_p) \subseteq u_6[\text{SRM}].sent\text{-}pkts?(s_p)$, it follows that $id(p) \in u_6[\text{SRM}].sent\text{-}pkts?(s_p)$. Thus, Lemma 4.2 implies that $id(p) \in u_k[\text{SRM}].sent\text{-}pkts?(s_p)$. Since $\texttt{process-pkt}_m(p)$ does not affect the variable $\text{SRM}.sent\text{-}pkts?(s_p)$, it follows that $u[\text{SRM}].sent\text{-}pkts?(s_p) = u_k[\text{SRM}].sent\text{-}pkts?(s_p)$. Thus, it is the case that $id(p) \in u[\text{SRM}].sent\text{-}pkts?(s_p)$.

We now consider the effects of the $\texttt{process-pkt}_m(p)$ action. If $m \neq s_p$ and $u_k[\text{SRM-rec}_m].min\text{-}seqno(s_p) =\perp$, then the $\texttt{process-pkt}_m(p)$ action sets both $\text{SRM-rec}_m.min\text{-}seqno(s_p)$ and $\text{SRM-rec}_m.max\text{-}seqno(s_p)$ variables to i_p. Thus, it follows that $u[\text{SRM-rec}_m].window?(s_p) = \{\langle s_p, i_p\rangle\}$. Since $u[\text{SRM-rec}_m].window?(s_p) = \{\langle s_p, i_p\rangle\}$ and $id(p) \in u[\text{SRM}].sent\text{-}pkts?(s_p)$, it follows that $u[\text{SRM-rec}_m].window?(s_p) \subseteq u[\text{SRM}].sent\text{-}pkts?(s_p)$.

If $m \neq s_p$, $u_k[\text{SRM-rec}_m].min\text{-}seqno(s_p) \neq\perp$, and $u_k[\text{SRM-rec}_m].max\text{-}seqno(s_p) < i_p$, then the $\texttt{process-pkt}_m(p)$ action does not affect the variable $\text{SRM-rec}_m.min\text{-}seqno(s_p)$ and sets the variable $\text{SRM-rec}_m.max\text{-}seqno(s_p)$ to i_p. From the induction hypothesis, it is the case that $u_k[\text{SRM-rec}_m].window?(s_p) \subseteq u_k[\text{SRM}].sent\text{-}pkts?(s_p)$. It follows that $\langle s_p, u_k[\text{SRM-rec}_m].min\text{-}seqno(s_p)\rangle \in u_k[\text{SRM}].sent\text{-}pkts?(s_p)$. Since $\texttt{process-pkt}_m(p)$ does not affect the variable $\text{SRM-rec}_m.min\text{-}seqno(s_p)$, it is the case that $u[\text{SRM-REC}_m].\mathit{min\text{-}seqno}(s_p) = u_k[\text{SRM-REC}_m].\mathit{min\text{-}seqno}(s_p)$. Thus, it follows that $\langle s_p, u[\text{SRM-REC}_m].\mathit{min\text{-}seqno}(s_p)\rangle \in u[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$. Since $\langle s_p, u[\text{SRM-REC}_m].\mathit{min\text{-}seqno}(s_p)\rangle \in u[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$ and $\langle s_p, u[\text{SRM-REC}_m].\mathit{max\text{-}seqno}(s_p)\rangle \in u[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$, Invariant 4.21 implies that $u[\text{SRM-REC}_m].\mathit{window?}(s_p) \subseteq u[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$.

Otherwise, $\texttt{process-pkt}_m(p)$ does not affect the variables $\text{SRM-REC}_m.\mathit{min\text{-}seqno}(s_p)$, $\text{SRM-REC}_m.\mathit{max\text{-}seqno}(s_p)$, and $\text{SRM}.\mathit{sent\text{-}pkts?}(s_p)$. Thus, the induction hypothesis implies that $u[\text{SRM-REC}_m].\mathit{window?}(s_p) \subseteq u[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$.

Moreover, the $\texttt{process-pkt}_m(p)$ action does not affect the variables $\text{SRM-REC}_m.\mathit{window?}(m')$, $\text{SRM-REC}_n.\mathit{window?}(n')$, and $\text{SRM}.\mathit{sent\text{-}pkts?}(n')$, for all $m' \in H, m' \neq s_p$, $n \in H, n \neq m$, and $n' \in H$. Thus, the induction hypothesis implies that $u[\text{SRM-REC}_m].\mathit{window?}(m') \subseteq u[\text{SRM}].\mathit{sent\text{-}pkts?}(m')$ and $u[\text{SRM-REC}_n].\mathit{window?}(n') \subseteq u[\text{SRM}].\mathit{sent\text{-}pkts?}(n')$, for all $m' \in H, m' \neq s_p$, $n \in H, n \neq m$, and $n' \in H$.

Thus, it follows that $u[\text{SRM-REC}_h].\mathit{window?}(h') \subseteq u[\text{SRM}].\mathit{sent\text{-}pkts?}(h')$, for all $h, h' \in H$.

❒ $\texttt{process-pkt}_m(p)$, for $m \in H$ and $p \in P_{\text{SRM}}$, such that $\mathit{type}(p) \in \{\texttt{RQST}, \texttt{REPL}\}$: let $s_p \in H$ and $i_p \in \mathbb{N}$ be the source and the sequence number, respectively, of the packet p. From the precondition of $\texttt{process-pkt}_m(p)$, it follows that $p \in u_k[\text{SRM-IPBUFF}_m].\mathit{recv\text{-}buff}$. Since the only action that may add p to the variable $\text{SRM-IPBUFF}_m.\mathit{recv\text{-}buff}$ is $\texttt{mrecv}_m(\mathit{pkt})$, for $\mathit{pkt} \in P_{\text{IPMCAST-CLIENT}}$, such that $\mathit{strip}(\mathit{pkt}) = p$, it follows that the action $\texttt{process-pkt}_m(p)$ is preceded in α by an action $\texttt{mrecv}_m(\mathit{pkt})$. Let $(u_2, \texttt{mrecv}_m(\mathit{pkt}), u_1)$ be the discrete transition in α_k corresponding to the particular occurrence of the action $\texttt{mrecv}_m(\mathit{pkt})$. Lemma 4.1 implies that the action $\texttt{mrecv}_m(\mathit{pkt})$ is preceded in α_k by an action $\texttt{msend}_{m'}(\mathit{pkt})$, for some $m' \in H$. Let $(u_4, \texttt{msend}_{m'}(\mathit{pkt}), u_3)$ be the discrete transition in α_k corresponding to the particular occurrence of the action $\texttt{msend}_{m'}(\mathit{pkt})$. From the precondition of the action $\texttt{msend}_{m'}(\mathit{pkt})$, it follows that $\mathit{pkt} \in u_4[\text{SRM-IPBUFF}_{m'}].\mathit{msend\text{-}buff}$. The only action that may add pkt to the variable $\text{SRM-IPBUFF}_{m'}.\mathit{msend\text{-}buff}$ is the action $\texttt{rec-msend}_{m'}(p)$. Thus, an action $\texttt{rec-msend}_{m'}(p)$ precedes u_4 in α_k. Let $(u_6, \texttt{rec-msend}_{m'}(p), u_5)$ be the discrete transition in α_k corresponding to the particular occurrence of the action $\texttt{rec-msend}_{m'}(p)$. From the precondition of the action $\texttt{rec-msend}_{m'}(p)$, it follows that $p \in u_6[\text{SRM-REC}_{m'}].\mathit{msend\text{-}buff}$.

Invariant 4.25 implies that $\mathit{id}(p) \in u_6[\text{SRM-REC}_{m'}].\mathit{window?}(s_p)$. From the induction hypothesis, it is the case that $u_6[\text{SRM-REC}_{m'}].\mathit{window?}(s_p) \subseteq u_6[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$. Since $\mathit{id}(p) \in u_6[\text{SRM-REC}_{m'}].\mathit{window?}(s_p)$ and $u_6[\text{SRM-REC}_{m'}].\mathit{window?}(s_p) \subseteq u_6[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$, it follows that $\mathit{id}(p) \in u_6[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$. Thus, Lemma 4.2 implies that $\mathit{id}(p) \in u_k[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$. Since $\texttt{process-pkt}_m(p)$ does not affect the variable $\text{SRM}.\mathit{sent\text{-}pkts?}(s_p)$, it follows that $u[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p) = u_k[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$. Thus, it is the case that $\mathit{id}(p) \in u[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$.

We now consider the effects of the $\texttt{process-pkt}_m(p)$ action. If $m \neq s_p$, $u_k[\text{SRM-REC}_m].\mathit{min\text{-}seqno}(s_p) \neq \perp$, and $u_k[\text{SRM-REC}_m].\mathit{max\text{-}seqno}(s_p) < i_p$, then the $\texttt{process-pkt}_m(p)$ action does not affect the variable $\text{SRM-REC}_m.\mathit{min\text{-}seqno}(s_p)$ and sets the variable $\text{SRM-REC}_m.\mathit{max\text{-}seqno}(s_p)$ to i_p. From the induction hypothesis, it is the case that $u_k[\text{SRM-REC}_m].\mathit{window?}(s_p) \subseteq u_k[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$. It follows that $\langle s_p, u_k[\text{SRM-REC}_m].\mathit{min\text{-}seqno}(s_p)\rangle \in u_k[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$. Since $\texttt{process-pkt}_m(p)$ does not affect the variable $\text{SRM-REC}_m.\mathit{min\text{-}seqno}(s_p)$, it is the case that $u[\text{SRM-REC}_m].\mathit{min\text{-}seqno}(s_p) = u_k[\text{SRM-REC}_m].\mathit{min\text{-}seqno}(s_p)$. Thus, it follows that $\langle s_p, u[\text{SRM-REC}_m].\mathit{min\text{-}seqno}(s_p)\rangle \in u[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$. Since $\langle s_p, u[\text{SRM-REC}_m].\mathit{min\text{-}seqno}(s_p)\rangle \in u[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$ and $\langle s_p, u[\text{SRM-REC}_m].\mathit{max\text{-}seqno}(s_p)\rangle \in u[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$, Invariant 4.21 implies that $u[\text{SRM-REC}_m].\mathit{window?}(s_p) \subseteq u[\text{SRM}].\mathit{sent\text{-}pkts?}(s_p)$.

Otherwise, $\texttt{process-pkt}_m(p)$ does not affect the variables SRM-REC$_m$.*min-seqno*(s_p), SRM-REC$_m$.*max-seqno*(s_p), and SRM.*sent-pkts?*(s_p). Thus, the induction hypothesis implies that $u[\text{SRM-REC}_m].\textit{window?}(s_p) \subseteq u[\text{SRM}].\textit{sent-pkts?}(s_p)$.

Finally, the $\texttt{process-pkt}_m(p)$ action does not affect the variables SRM-REC$_m$.*window?*(m'), SRM-REC$_n$.*window?*(n'), and SRM.*sent-pkts?*(n'), for all $m' \in H, m' \neq s_p$, $n \in H, n \neq m$, and $n' \in H$. Thus, the induction hypothesis implies that $u[\text{SRM-REC}_m].\textit{window?}(m') \subseteq u[\text{SRM}].\textit{sent-pkts?}(m')$ and $u[\text{SRM-REC}_n].\textit{window?}(n') \subseteq u[\text{SRM}].\textit{sent-pkts?}(n')$, for all $m' \in H, m' \neq s_p$, $n \in H, n \neq m$, and $n' \in H$.

Thus, it follows that $u[\text{SRM-REC}_h].\textit{window?}(h') \subseteq u[\text{SRM}].\textit{sent-pkts?}(h')$, for all $h, h' \in H$.

❒ $\texttt{process-pkt}_m(p)$, for $h \in H$ and $p \in P_{\text{SRM}}$, such that *type*$(p) \in \{\texttt{SESS}\}$: From the precondition of $\texttt{process-pkt}_m(p)$, it follows that $p \in u_k[\text{SRM-IPBUFF}_m].\textit{recv-buff}$. Since the only action that may add p to the variable SRM-IPBUFF$_m$.*recv-buff* is $\texttt{mrecv}_m(pkt)$, for $pkt \in P_{\text{IPMCAST-CLIENT}}$, such that $strip(pkt) = p$, it follows that the action $\texttt{process-pkt}_m(p)$ is preceded in α by an action $\texttt{mrecv}_m(pkt)$. Let $(u_2, \texttt{mrecv}_m(pkt), u_1)$ be the discrete transition in α_k corresponding to the particular occurrence of the action $\texttt{mrecv}_m(pkt)$. Lemma 4.1 implies that the action $\texttt{mrecv}_m(pkt)$ is preceded in α_k by an action $\texttt{msend}_{m'}(pkt)$, for some $m' \in H$. Let $(u_4, \texttt{msend}_{m'}(pkt), u_3)$ be the discrete transition in α_k corresponding to the particular occurrence of the action $\texttt{msend}_{m'}(pkt)$. From the precondition of the action $\texttt{msend}_{m'}(pkt)$, it follows that $pkt \in u_4[\text{SRM-IPBUFF}_{m'}].\textit{msend-buff}$. The only action that may add pkt to the variable SRM-IPBUFF$_{m'}$.*msend-buff* is the action $\texttt{rec-msend}_{m'}(p)$. Thus, an action $\texttt{rec-msend}_{m'}(p)$ precedes u_4 in α_k. Let $(u_6, \texttt{rec-msend}_{m'}(p), u_5)$ be the discrete transition in α_k corresponding to the particular occurrence of the action $\texttt{rec-msend}_{m'}(p)$. From the precondition of the action $\texttt{rec-msend}_{m'}(p)$, it follows that $p \in u_6[\text{SRM-REC}_{m'}].\textit{msend-buff}$.

For any $\langle h'', i'' \rangle \in \textit{seqno-rprts}(p)$, such that $h'' \neq m$, $u_k[\text{SRM-REC}_m].\textit{min-seqno}(h'') \neq \perp$, and $u_k[\text{SRM-REC}_m].\textit{max-seqno}(h'') < i''$, the $\texttt{process-pkt}_m(p)$ action does not affect the variable SRM-REC$_m$.*min-seqno*(h'') and sets the variable SRM-REC$_m$.*max-seqno*(h'') to i''. From the induction hypothesis, it is the case that $u_k[\text{SRM-REC}_m].\textit{window?}(h'') \subseteq u_k[\text{SRM}].\textit{sent-pkts?}(h'')$. It follows that $\langle h'', u_k[\text{SRM-REC}_m].\textit{min-seqno}(h'') \rangle \in u_k[\text{SRM}].\textit{sent-pkts?}(h'')$. Since $\texttt{process-pkt}_m(p)$ does not affect the variable SRM-REC$_m$.*min-seqno*(h''), it is the case that $u[\text{SRM-REC}_m].\textit{min-seqno}(h'') = u_k[\text{SRM-REC}_m].\textit{min-seqno}(h'')$. Thus, it follows that $\langle h'', u[\text{SRM-REC}_m].\textit{min-seqno}(h'') \rangle \in u[\text{SRM}].\textit{sent-pkts?}(h'')$.

Invariant 4.25 implies that $\langle h'', i'' \rangle \in u_6[\text{SRM-REC}_{m'}].\textit{window?}(h'')$. From the induction hypothesis, it is the case that $u_6[\text{SRM-REC}_{m'}].\textit{window?}(h'') \subseteq u_6[\text{SRM}].\textit{sent-pkts?}(h'')$. Since $\langle h'', i'' \rangle \in u_6[\text{SRM-REC}_{m'}].\textit{window?}(h'')$ and $u_6[\text{SRM-REC}_{m'}].\textit{window?}(h'') \subseteq u_6[\text{SRM}].\textit{sent-pkts?}(h'')$, it follows that $\langle h'', i'' \rangle \in u_6[\text{SRM}].\textit{sent-pkts?}(h'')$. Thus, Lemma 4.2 implies that $\langle h'', i'' \rangle \in u_k[\text{SRM}].\textit{sent-pkts?}(h'')$. Since $\texttt{process-pkt}_m(p)$ does not affect the variable SRM.*sent-pkts?*(h''), it follows that $u[\text{SRM}].\textit{sent-pkts?}(h'') = u_k[\text{SRM}].\textit{sent-pkts?}(h'')$. Thus, it is the case that $\langle h'', i'' \rangle \in u[\text{SRM}].\textit{sent-pkts?}(h'')$.

Since $\langle h'', u[\text{SRM-REC}_m].\textit{min-seqno}(h'') \rangle \in u[\text{SRM}].\textit{sent-pkts?}(h'')$ and $\langle h'', u[\text{SRM-REC}_m].\textit{max-seqno}(h'') \rangle \in u[\text{SRM}].\textit{sent-pkts?}(h'')$, Invariant 4.21 implies that $u[\text{SRM-REC}_m].\textit{window?}(h'') \subseteq u[\text{SRM}].\textit{sent-pkts?}(h'')$.

Otherwise, For any $\langle h'', i'' \rangle \in \textit{seqno-rprts}(p)$, such that it is not the case that $h'' \neq m$, $u_k[\text{SRM-REC}_m].\textit{min-seqno}(h'') \neq \perp$, and $u_k[\text{SRM-REC}_m].\textit{max-seqno}(h'') < i''$, the $\texttt{process-pkt}_m(p)$ action does not affect the variables SRM-REC$_m$.*min-seqno*(h''), SRM-REC$_m$.*max-seqno*(h''), and SRM.*sent-pkts?*(h''). Thus, the induction hypothesis implies that $u[\text{SRM-REC}_m].\textit{window?}(h'') \subseteq u[\text{SRM}].\textit{sent-pkts?}(h'')$.

Finally, the $\texttt{process-pkt}_m(p)$ action does not affect the variables SRM-REC$_n$.*window?*(n') and

SRM.*sent-pkts?*(n'), for all $n \in H, n \neq m$, and $n' \in H$. Thus, the induction hypothesis implies that $u[\text{SRM-REC}_n].\mathit{window?}(n') \subseteq u[\text{SRM}].\mathit{sent\text{-}pkts?}(n')$, for all $n \in H, n \neq m$ and $n' \in H$.

Thus, it follows that $u[\text{SRM-REC}_h].\mathit{window?}(h') \subseteq u[\text{SRM}].\mathit{sent\text{-}pkts?}(h')$, for all $h, h' \in H$.

❐

Invariant 4.27 *For any reachable state u of SRM_I, it is the case that, for all $h, h' \in H$, $u[\text{SRM-REC}_h].\mathit{archived\text{-}pkts?}(h') \subseteq u[\text{SRM}].\mathit{sent\text{-}pkts?}(h')$.*

Proof: Follows directly from Invariants 4.5 and 4.26. ❐

Invariant 4.28 *For $h, h' \in H$ and any reachable state u of SRM_I, it is the case that $u[\text{SRM-REC}_h].\mathit{to\text{-}be\text{-}delivered?}(h') \subseteq u[\text{SRM}].\mathit{sent\text{-}pkts?}(h')$.*

Proof: Follows directly from Invariants 4.4 and 4.27. ❐

4.4.4 Correctness Analysis

In this section, we show that our reliable multicast implementation SRM_I indeed implements the reliable multicast service specification $\text{RM}_S(\infty)$.

We begin by defining a relation R from SRM_I to $\text{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$.

Definition 4.1 *Let R be the relation between states of SRM_I and $\text{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, such that for any states u and s of SRM_I and $\text{RM}_S(\Delta)$, respectively, $(u, s) \in R$ provided that, for all $h, h' \in H$ and $p \in P_{\text{RM-CLIENT}}$, such that $\langle s_p, i_p \rangle = id(p)$, it is the case that:*

$$
\begin{aligned}
s.\mathit{now} &= u.\mathit{now} \\
s[\text{RM-CLIENT}_h].\mathit{status} &= u[\text{RM-CLIENT}_h].\mathit{status} \\
s[\text{RM-CLIENT}_h].\mathit{seqno} &= u[\text{RM-CLIENT}_h].\mathit{seqno} \\
s[\text{RM}(\Delta)].\mathit{status}(h) &= \begin{cases} \texttt{idle} & \textit{if } u[\text{SRM-MEM}_h].\mathit{status} = \texttt{idle} \\ \texttt{joining} & \textit{if } u[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Joining} \\ \texttt{leaving} & \textit{if } u[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Leaving} \\ \texttt{member} & \textit{if } u[\text{SRM-MEM}_h].\mathit{status} = \texttt{member} \\ \texttt{crashed} & \textit{if } u[\text{SRM-MEM}_h].\mathit{status} = \texttt{crashed} \end{cases} \\
s[\text{RM}(\Delta)].\mathit{trans\text{-}time}(p) &= u[\text{SRM-REC}_{s_p}].\mathit{trans\text{-}time}(p) \\
s[\text{RM}(\Delta)].\mathit{expected}(h, h') &= u[\text{SRM-REC}_h].\mathit{expected}(h') \\
s[\text{RM}(\Delta)].\mathit{delivered}(h, h') &= u[\text{SRM-REC}_h].\mathit{delivered}(h')
\end{aligned}
$$

The following lemma states that the relation R of Definition 4.1 is a timed forward simulation relation from SRM_I to $\text{RM}_S(\infty)$.

Lemma 4.11 *R is a timed forward simulation relation from SRM_I to $\text{RM}_S(\infty)$.*

Proof: We must show that: i) if $u \in \mathit{start}(\text{SRM}_I)$, then there is some $s \in \mathit{start}(\text{RM}_S(\infty))$ such that $(u, s) \in R$, and ii) if u is a reachable state of SRM_I, s is a reachable state of $\text{RM}_S(\infty)$ such that $(u, s) \in R$, and $(u, \pi, u') \in trans(\mathrm{SRM}_I)$, then there exists a timed execution fragment α of $\mathrm{RM}_S(\infty)$ such that: $\alpha.fstate = s$, $ttrace(\alpha) = ttrace(u\pi u')$, the total amount of time-passage in α is the same as the total amount of time-passage in $u\pi u'$, and $(u', s') \in R$, for $s' = \alpha.lstate$.

The satisfaction of the start condition is straightforward. For the step, we consider only the actions in $acts(\mathrm{SRM}_I)$ that affect the variables of SRM_I that are used in R to obtain the corresponding state in $\mathrm{RM}_S(\infty)$. Moreover, since the client automata RM-CLIENT$_h$, for all $h \in H$, are identical in both SRM_I and $\mathrm{RM}_S(\infty)$, we do not consider the effect of the actions of SRM_I on the state of the client automata. Thus, we consider only the actions of the SRM component of SRM_I that affect the variables of SRM that are present in R.

- `crash`$_h$, for any $h \in H$: the corresponding execution fragment of $\mathrm{RM}_S(\infty)$ is comprised solely of the `crash`$_h$ action. The `crash`$_h$ action of SRM_I simply sets the variable $u[\text{SRM-MEM}_h].status$ to `crashed` and resets $u[\text{SRM-REC}_h].expected(h')$ and $u[\text{SRM-REC}_h].completed(h')$, for all $h' \in H$. It is straightforward to see that the `crash`$_h$ action of $\mathrm{RM}_S(\infty)$ mirrors these effects. Thus, it follows that $(u', s') \in R$.

- `rm-join`$_h$, for any $h \in H$: the corresponding execution fragment of $\mathrm{RM}_S(\infty)$ is comprised solely of the `rm-join`$_h$ action. It is straightforward to see that the effects of the `rm-join`$_h$ action in the specification correspond to those in the implementation.

- `mjoin`$_h$, for any $h \in H$: the corresponding execution fragment of $\mathrm{RM}_S(\infty)$ is the empty timed execution fragment. Since the `mjoin`$_h$ action is enabled in state u, it follows that $u[\text{SRM-MEM}_h].status \in Joining$. Thus, R implies that $s[\mathrm{RM}(\infty)].status(h) =$ `joining`. The effects of the `mjoin`$_h$ action are to set the $status$ variable to `join-pending`. It follows that $u'[\text{SRM-MEM}_h].status \in Joining$. Since the corresponding execution fragment of $\mathrm{RM}_S(\infty)$ is the empty timed execution fragment it is the case that $s' = s$ and $s'[\mathrm{RM}(\infty)].status(h) =$ `joining`. Thus, it follows that $(u', s') \in R$.

- `mjoin-ack`$_h$, for any $h \in H$: the corresponding execution fragment of $\mathrm{RM}_S(\infty)$ is the empty timed execution fragment. The `mjoin-ack`$_h$ action affects the state of the SRM-MEM$_h$ automaton only when the host h is in the process of joining the reliable multicast group; that is, $u[\text{SRM-MEM}_h].status \in Joining$. Thus, R implies that $s[\mathrm{RM}(\infty)].status(h) =$ `joining`. The effects of the `mjoin-ack`$_h$ action are to set the $status$ variable to `join-ack-pending`. It follows that $u'[\text{SRM-MEM}_h].status \in Joining$. Since the corresponding execution fragment of $\mathrm{RM}_S(\infty)$ is the empty timed execution fragment it is the case that $s' = s$ and $s'[\mathrm{RM}(\infty)].status(h) =$ `joining`. Thus, it follows that $(u', s') \in R$.

- `rm-leave`$_h$, for any $h \in H$: the corresponding execution fragment of $\mathrm{RM}_S(\infty)$ is comprised solely of the `rm-leave`$_h$ action. From the precondition of the `rm-leave`$_h$ action in the RM-CLIENT$_h$ automaton, it follows that $u[\text{RM-CLIENT}_h].status =$ `member`. Thus, Invariant 4.22 implies that $u[\text{SRM-MEM}_h].status =$ `member` and, since $(u, s) \in R$, it is the case that $s[\mathrm{RM}(\infty)].status(h) =$ `member`.

  Since $u[\text{SRM-MEM}_h].status =$ `member`, the `rm-leave`$_h$ action of SRM_I sets the $status$ variable of SRM-MEM$_h$ to `leave-rqst-pending`. The `rm-leave`$_h$ action of $\mathrm{RM}_S(\infty)$ sets the $status(h)$ variable of $\mathrm{RM}(\infty)$ to `leaving`. Thus, it follows that $u'[\text{SRM-MEM}_h].status \in Leaving$ and $s'[\mathrm{RM}(\infty)].status(h) =$ `leaving`, as required by R.

  Moreover, the `rm-leave`$_h$ action of SRM_I resets the expected and delivered packet sets of SRM-REC$_h$; that is, $u'[\text{SRM-REC}_h].expected(h') = \emptyset$ and $u'[\text{SRM-REC}_h].delivered(h') = \emptyset$, for all $h' \in H$. Similarly, the `rm-leave`$_h$ action of $\mathrm{RM}_S(\infty)$ also resets the variables $expected(h, h')$ and $delivered(h, h')$, for $h' \in H$; that is, $s'[\mathrm{RM}(\infty)].expected(h, h') = \emptyset$ and $s'[\mathrm{RM}(\infty)].delivered(h, h') = \emptyset$, for $h' \in H$. Thus, it follows that $(u', s') \in R$.

- `mleave`$_h$, for any $h \in H$: the corresponding execution fragment of $\mathrm{RM}_S(\infty)$ is the empty timed execution fragment. Since the $\texttt{mleave}_h$ action is enabled in state u, it follows that $u[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Leaving}$. Thus, R implies that $s[\text{RM}(\infty)].\mathit{status}(h) = \texttt{leaving}$. The effects of the $\texttt{mleave}_h$ action of SRM_I are to set the *status* variable of SRM-MEM_h to $\texttt{leave-pending}$. It follows that $u'[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Leaving}$. Since the corresponding execution fragment of $\text{RM}_S(\infty)$ is the empty timed execution fragment it is the case that $s' = s$ and $s'[\text{RM}(\infty)].\mathit{status}(h) = \texttt{leaving}$. Thus, it follows that $(u', s') \in R$.

❒ $\texttt{mleave-ack}_h$, for any $h \in H$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is the empty timed execution fragment. The $\texttt{mleave-ack}_h$ action affects the state of the SRM-MEM_h automaton only when the host h is in the process of leaving the reliable multicast group; that is, $u[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Leaving}$. In this case, R implies that $s[\text{RM}(\infty)].\mathit{status}(h) = \texttt{leaving}$. The effects of the $\texttt{mleave-ack}_h$ action of SRM_I are to set the *status* variable of SRM-MEM_h to $\texttt{leave-ack-pending}$. It follows that $u'[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Leaving}$. Since the corresponding execution fragment of $\text{RM}_S(\infty)$ is the empty timed execution fragment it is the case that $s' = s$ and $s'[\text{RM}(\infty)].\mathit{status}(h) = \texttt{leaving}$. Thus, it follows that $(u', s') \in R$.

❒ $\texttt{rm-join-ack}_h$, for any $h \in H$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is comprised solely of the $\texttt{rm-join-ack}_h$ action. We begin by showing that the $\texttt{rm-join-ack}_h$ action of $\text{RM}_S(\infty)$ is enabled in s. The precondition of the $\texttt{rm-join-ack}_h$ action of SRM_I implies that $u[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Joining}$. Since $(u, s) \in R$, it follows that $s[\text{RM}(\infty)].\mathit{status}(h) = \texttt{joining}$. Thus, it follows that the $\texttt{rm-join-ack}_h$ action of $\text{RM}_S(\infty)$ is enabled in s.

The $\texttt{rm-join-ack}_h$ action of SRM_I sets the *status* variable of SRM-MEM_h to $\texttt{member}$. Similarly, the $\texttt{rm-join-ack}_h$ action of $\text{RM}_S(\infty)$ sets the $\mathit{status}(h)$ variable of $\text{RM}(\infty)$ to $\texttt{member}$. Thus, it follows that $(u', s') \in R$.

❒ $\texttt{rm-leave-ack}_h$, for any $h \in H$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is comprised solely of the $\texttt{rm-leave-ack}_h$ action. We begin by showing that the $\texttt{rm-leave-ack}_h$ action of $\text{RM}_S(\infty)$ is enabled in s. The precondition of the $\texttt{rm-leave-ack}_h$ action of SRM_I implies that $u[\text{SRM-MEM}_h].\mathit{status} \in \mathit{Leaving}$. Since $(u, s) \in R$, it follows that $s[\text{RM}(\infty)].\mathit{status}(h) = \texttt{leaving}$. Thus, it follows that the $\texttt{rm-leave-ack}_h$ action of $\text{RM}_S(\infty)$ is enabled in s.

The $\texttt{rm-leave-ack}_h$ action of SRM_I sets the *status* variable of SRM-MEM_h to $\texttt{idle}$. Similarly, the $\texttt{rm-leave-ack}_h$ action of $\text{RM}_S(\infty)$ sets the $\mathit{status}(h)$ variable of $\text{RM}(\infty)$ to $\texttt{idle}$. Thus, it follows that $(u', s') \in R$.

❒ $\texttt{rm-send}_h(p)$, for any $h \in H$ and $p \in P_{\text{RM-CLIENT}}$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is comprised solely of the $\texttt{rm-send}_h(p)$ action. Let s_p and i_p denote the source and sequence number of p, respectively.

From the precondition of the $\texttt{rm-send}_h(p)$ action of SRM_I, it follows that $u[\text{RM-CLIENT}_h].\mathit{status} = \texttt{member}$ and $h = s_p$. Invariant 4.22 implies that $u[\text{SRM-MEM}_h].\mathit{status} = \texttt{member}$ and, since $(u, s) \in R$, it is the case that $s[\text{RM}(\infty)].\mathit{status}(h) = \texttt{member}$.

We consider the effects of $\texttt{rm-send}_h(p)$ according to whether p is the foremost packet from h. First, consider the case where p is the foremost packet from h; that is, $u[\text{SRM-REC}_h].\mathit{min\text{-}seqno}(s_p) = \perp$. In this case, the $\texttt{rm-send}_h(p)$ action of SRM_I sets the expected set from h to the set $\mathit{suffix}(p)$, adds $\mathit{id}(p)$ to the set of delivered packets from h, and records the transmission time of p.

Since it is the case that $u[\text{SRM-REC}_h].\mathit{min\text{-}seqno}(s_p) = \perp$, Invariant 4.9 implies that $u[\text{SRM-REC}_h].\mathit{expected}(h) = \emptyset$. Since $(u, s) \in R$, it follows that $s[\text{RM}(\infty)].\mathit{expected}(h, h) = \emptyset$. Thus, the $\texttt{rm-send}_h(p)$ action of $\text{RM}_S(\infty)$ matches the effects of the $\texttt{rm-send}_h(p)$ action of SRM_I. It follows that $(u', s') \in R$.

Second, consider the case where p is not the foremost packet from h; that is, $u[\text{SRM-REC}_h].\mathit{min\text{-}seqno}(s_p) \neq \perp$. In this case, Invariant 4.23 and the precondition of $\texttt{rm-send}_h(p)$ imply that $i_p = u[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(s_p) + 1$. Thus, the $\texttt{rm-send}_h(p)$ action of SRM_I records the transmission time of p and adds $id(p)$ to the set of delivered packets from h.

Since it is the case that $i_p = u[\text{SRM-REC}_h].\mathit{max\text{-}seqno}(s_p) + 1$, Invariant 4.3 implies that $u[\text{SRM-REC}_h].\mathit{min\text{-}seqno}(s_p) < i_p$. Thus, it follows that $id(p) \in u[\text{SRM-REC}_h].\mathit{proper?}(h)$. Since $u[\text{SRM-MEM}_h].\mathit{status} = \texttt{member}$, Invariant 4.9 implies that $u[\text{SRM-REC}_h].\mathit{expected}(h) = u[\text{SRM-REC}_h].\mathit{proper?}(h)$. Thus, it follows that $id(p) \in u[\text{SRM-REC}_h].\mathit{expected}(h)$. Since $(u, s) \in R$, it is the case that $s[\text{RM}(\infty)].\mathit{expected}(h, h) = u[\text{SRM-REC}_h].\mathit{expected}(h)$. Thus, it follows that $id(p) \in s[\text{RM}(\infty)].\mathit{expected}(h, h)$. Thus, the $\texttt{rm-send}_h(p)$ action of $\text{RM}_S(\infty)$ also records the transmission time of p and adds p to the set of delivered packets from h. Thus, it follows that $(u', s') \in R$.

❒ $\texttt{rm-recv}_h(p)$, for any $h \in H$ and $p \in P_{\text{RM-CLIENT}}$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is comprised solely of the $\texttt{rm-recv}_h(p)$ action. Let s_p and i_p denote the source and sequence number of p, respectively.

We first show that the $\texttt{rm-recv}_h(p)$ action of $\text{RM}_S(\infty)$ is enabled in s. From the precondition of the $\texttt{rm-recv}_h(p)$ action of SRM_I, it follows that $u[\text{SRM-REC}_h].\mathit{status} = \texttt{member}$ and $p \in u[\text{SRM-REC}_h].\mathit{to\text{-}be\text{-}delivered}$. Invariant 4.24 implies that $u[\text{SRM-MEM}_h].\mathit{status} = \texttt{member}$ and, since $(u, s) \in R$, it follows $s[\text{RM}(\infty)].\mathit{status}(h) = \texttt{member}$. Since $p \in u[\text{SRM-REC}_h].\mathit{to\text{-}be\text{-}delivered}$, Invariant 4.11 implies that $h \neq \mathit{source}(p)$. Moreover, Invariant 4.28 implies that $p \in u[\text{SRM}].\mathit{sent\text{-}pkts}$. Since $(u, s) \in R$, it follows that $p \in s[\text{RM}(\infty)].\mathit{sent\text{-}pkts}$.

We proceed by showing that s satisfies the last two terms in the precondition of the $\texttt{rm-recv}_h(p)$ action of $\text{RM}_S(\infty)$. Since the delivery delay parameter Δ is equal to ∞ for the $\text{RM}_S(\infty)$ automaton, $s[\text{RM}(\infty)]$ trivially satisfies the term $\mathit{expected}(h, s_p) = \emptyset \Rightarrow \mathit{now} \leq \mathit{trans\text{-}time}(p) + \Delta$.

Finally, we show that $s[\text{RM}(\infty)]$ satisfies the term $\mathit{expected}(h, s_p) \neq \emptyset \Rightarrow id(p) \in \mathit{expected}(h, s_p)$. Suppose that it is the case that $s[\text{RM}(\infty)].\mathit{expected}(h, s_p) \neq \emptyset$. Since $(u, s) \in R$, it follows that $u[\text{SRM-REC}_h].\mathit{expected}(s_p) \neq \emptyset$. Thus, since $p \in u[\text{SRM-REC}_h].\mathit{to\text{-}be\text{-}delivered}$, Invariant 4.12 implies that $id(p) \in u[\text{SRM-REC}_h].\mathit{expected}(s_p)$. Finally, since $(u, s) \in R$, it follows that $id(p) \in s[\text{RM}(\infty)].\mathit{expected}(h, s_p)$, as needed.

The $\texttt{rm-recv}_h(p)$ action of SRM_I sets the expected set of packets from s_p to the set $\mathit{suffix}(p)$, unless already non-empty, and adds p to the set of delivered packets from s_p. The $\texttt{rm-recv}_h(p)$ action of $\text{RM}(\infty)$ matches these effects. Thus, it follows that $(u', s') \in R$.

❒ $\nu(t)$, for any $t \in \mathbb{R}^{\geq 0}$: the corresponding execution fragment of $\text{RM}_S(\infty)$ is comprised solely of the $\nu(t)$ action. Since the effects of the $\nu(t)$ actions of the SRM_I and the $\text{RM}_S(\infty)$ automata are identical, it suffices to show that the $\nu(t)$ action is enabled in s. Since the delivery delay parameter Δ is equal to ∞ for the $\text{RM}_S(\infty)$ automaton, the term $\mathit{now} + t \leq \mathit{trans\text{-}time}(p) + \Delta$ of the precondition of the $\nu(t)$ action of $\text{RM}_S(\infty)$ is satisfied for all $p \in P_{\text{RM-CLIENT}}$. Thus, it follows that the $\nu(t)$ action of $\text{RM}_S(\infty)$ is enabled in s.

❒

Theorem 4.12 $\text{SRM}_I \leq \text{RM}_S(\infty)$

Proof: Follows directly from Lemma 4.11. ❒

4.4.5 Timeliness Analysis Preliminaries

Preliminary Definitions

Suppose $p \in P_{\text{RM-CLIENT}}$, $pkt \in P_{\text{SRM}}$, and α is an admissible timed execution of SRM_I that contains the transmission of p; that is, α contains the action $\texttt{rm-send}_{s_p}(p)$, for $s_p = \mathit{source}(p)$. For $pkt \in P_{\text{SRM}}$, we say that *pkt pertains to p* if $\mathit{type}(pkt) \in \{\texttt{DATA}, \texttt{RQST}, \texttt{REPL}\}$ and $\mathit{id}(pkt) = \mathit{id}(p)$. We let $P_{\text{SRM}}[p]$ denote the elements of P_{SRM} that pertain to p. We let the number of packet drops in α pertaining to p, denoted $\alpha.\mathit{drops}(p)$, be the number of packet drops suffered by packets pertaining to p; that is, $\alpha.\mathit{drops}(p)$ is the number of occurrences of an action $\texttt{mdrop}(pkt', H_d)$ in α, for $pkt' \in P_{\text{IPMCAST-CLIENT}}$ and $H_d \subseteq H$, such that $\mathit{strip}(pkt') \in P_{\text{SRM}}[p]$.

We let $\mathit{aexecs}_k(\text{SRM}_I)$, for $k \in \mathbb{N}^+$, be the set of admissible timed executions of SRM_I in which the number of drops suffered by IP packets pertaining to the transmission and, potentially, the recovery of any packet $p \in P_{\text{RM-CLIENT}}$ is at most k. That is, $\alpha \in \mathit{aexecs}_k(\text{SRM}_I)$ if and only if $\alpha.\mathit{drops}(p') \leq k$, for any packet $p' \in P_{\text{RM-CLIENT}}$ transmitted in α. Finally, we let $\mathit{attraces}_k(\text{SRM}_I)$ be the traces of all executions of SRM_I in $\mathit{aexecs}_k(\text{SRM}_I)$.

We let the transmission time of p in α, denoted $\alpha.\mathit{trans\text{-}time}(p)$, be the point in time in α at which p is transmitted; that is, the time of occurrence of $\texttt{rm-send}_{s_p}(p)$ in α. Since packets are transmitted by the clients of the reliable multicast service at most once (Lemma 3.2), it follows that the transmission time of any packet transmitted in any admissible timed execution of SRM_I is well-defined and unique.

Execution Definitions

We proceed by defining several constraints on admissible executions of SRM_I. These constraints facilitate the statement of conditional claims regarding the timely transmission of packets for SRM_I.

Let $\underline{d}, \overline{d} \in \mathbb{R}^{\geq 0}$, such that $\underline{d} > 0$, $\overline{d} > 0$, and $\underline{d} \leq \overline{d}$. The following constraint specifies the set of executions of SRM_I in which the transmission latency between any two hosts $h, h' \in H, h \neq h'$ is bounded from below and above by $\underline{d}$ and $\overline{d}$, respectively.

Constraint 4.1 (Bounded Multicast Transmission Latency) *Let α be any admissible timed execution of SRM_I and h, h' be any two distinct hosts in H. The transmission latency incurred by any packet multicast using the IP multicast service by h and received by h' in α lies in the interval $[\underline{d}, \overline{d}]$; that is, if $p \in P_{\text{IPMCAST-CLIENT}}$ is a packet multicast by h in α, then the time elapsing from the time of occurrence of the action $\texttt{msend}_h(p)$ to that of any action $\texttt{mrecv}_{h'}(p)$ lies in the interval $[\underline{d}, \overline{d}]$.*

The following constraint specifies the set of executions of SRM_I in which the fate of any packet transmitted using the IP multicast service is resolved within $\underline{d}$ time units.

Constraint 4.2 (Bounded Multicast Transmission Resolution) *Let α be any admissible execution of SRM_I containing the discrete transition (u, π, u'), for $u, u' \in \mathit{states}(\text{SRM}_I)$, $p \in P_{\text{IPMCAST-CLIENT}}$, $s_p = \mathit{source}(p)$, and $\pi = \texttt{msend}_{s_p}(p)$. Then, for all $h \in u[\text{IPMCAST}].\mathit{members}, h \neq s_p$, either a $\texttt{crash}_h$, $\texttt{mleave}_h$, $\texttt{mrecv}_h(p)$, or $\texttt{mdrop}(p, H_d)$, for $H_d \subseteq H$, $h \in H_d$, action occurs no later than $\overline{d}$ time units after the particular occurrence of the discrete transition (u, π, u') in α.*

The following constraint specifies the set of executions of SRM_I in which the inter-host distance estimates of any host always lie in the interval $[\underline{d}, \overline{d}]$. The satisfaction of this constraint requires that $\texttt{DFLT-DIST} \in [\underline{d}, \overline{d}]$.

Constraint 4.3 (Bounded Inter-host Distance Estimates) *Let α be any admissible timed execution of* SRM_I. *For any state u of* SRM_I *in α, the inter-host distance estimates of the recovery component of each reliable multicast process of* SRM_I *lie in the interval $[\underline{d}, \overline{d}]$; that is, $u[\mathrm{SRM\text{-}REC}_h].\mathit{dist?}(h') \in [\underline{d}, \overline{d}]$, for all $h, h' \in H, h \neq h'$.*

Letting $\texttt{DET-BOUND} \in \mathbb{R}^{\geq 0}$, such that $\overline{d} \leq \texttt{DET-BOUND}$, the following constraint specifies the set of executions of SRM_I in which the delay in detecting packet losses is bounded by $\texttt{DET-BOUND}$.

Constraint 4.4 (Bounded Detection Latency) *Let α be any admissible timed execution of* SRM_I. *Let $p \in P_{\mathrm{RM\text{-}CLIENT}}$ be any packet transmitted in α, $id(p) = \langle s_p, i_p \rangle$, and $h \in H, h \neq s_p$. Moreover, let u be any state of* SRM_I *in α such that $\alpha.\mathit{trans\text{-}time}(p) + \texttt{DET-BOUND} < u.\mathit{now}$. Then, if $id(p) \in u[\mathrm{SRM\text{-}REC}_h].\mathit{expected}(s_p)$, then either $id(p) \in u[\mathrm{SRM\text{-}REC}_h].\mathit{archived\text{-}pkts?}(s_p)$ or $id(p) \in u[\mathrm{SRM\text{-}REC}_h].\mathit{scheduled\text{-}rqsts?}$.*

Let *timely-aexecs*(SRM_I), for $\Delta \in \mathbb{R}^{\geq 0}$, be the set of all admissible timed executions of SRM_I in *aexecs*(SRM_I) that satisfy Constraints 4.1, 4.2, 4.3, and 4.4. Let *timely-attraces*(SRM_I) be the traces of all the executions of SRM_I in *timely-aexecs*(SRM_I). Let *timely-aexecs*$_k(\mathrm{SRM}_I)$, for $k \in \mathbb{N}^+$, be the subset of *aexecs*$_k(\mathrm{SRM}_I)$ comprised of all admissible timed executions of SRM_I that satisfy Constraints 4.1, 4.2, 4.3, and 4.4; that is, for $k \in \mathbb{N}^+$, *timely-aexecs*$_k(\mathrm{SRM}_I) =$ *aexecs*$_k(\mathrm{SRM}_I) \cap$ *timely-aexecs*(SRM_I). Moreover, let *timely-attraces*$_k(\mathrm{SRM}_I)$ be the traces of all executions of SRM_I in *timely-aexecs*$_k(\mathrm{SRM}_I)$.

The following two constraints specify the set of executions of SRM_I in which none of the hosts either crash or leave the reliable multicast group, respectively.

Constraint 4.5 (No Crashes) *Let α be any admissible timed execution of* SRM_I. *None of the hosts crash in α; that is, for any $h \in H$, no $\texttt{crash}_h$ actions occur in α.*

Constraint 4.6 (No Leaves) *Let α be any admissible timed execution of* SRM_I. *None of the hosts leave the reliable multicast group in α; that is, for any $h \in H$, no $\texttt{rm-leave}_h$ actions occur in α.*

Let *recoverable-aexecs*(SRM_I) be the set of all admissible timed executions of SRM_I in *aexecs*(SRM_I) that satisfy Constraints 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6. Let *recoverable-attraces*(SRM_I) be the traces of all the executions of SRM_I in *recoverable-aexecs*(SRM_I). Let *recoverable-aexecs*$_k(\mathrm{SRM}_I)$, for $k \in \mathbb{N}^+$, be the subset of *aexecs*$_k(\mathrm{SRM}_I)$ comprised of all admissible timed executions of SRM_I that satisfy Constraints 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6; that is, for $k \in \mathbb{N}^+$, *recoverable-aexecs*$_k(\mathrm{SRM}_I) =$ *aexecs*$_k(\mathrm{SRM}_I) \cap$ *recoverable-aexecs*(SRM_I). Moreover, let *recoverable-attraces*$_k(\mathrm{SRM}_I)$ be the traces of all executions of SRM_I in *recoverable-aexecs*$_k(\mathrm{SRM}_I)$.

The following constraint specifies the set of executions of SRM_I in which the source of each packet transmitted does not crash and remains a member of the reliable multicast group for at least $\Delta_L \in \mathbb{R}^{\geq 0}$ time units past the transmission of the given packet. Thus, each source is capable of replying to retransmission requests for at least Δ_L time units past each packet's transmission time. The parameter Δ_L is presumed to correspond to the upper bound on the transmission latency of any packet transmitted within an admissible timed execution of SRM_I that satisfies this constraint.

Constraint 4.7 (Δ_L-Source Recoverable) *Let $\Delta_L \in \mathbb{R}^{\geq 0}$ and α be any admissible timed execution of* SRM_I. *For any packet $p \in P_{\mathrm{RM\text{-}CLIENT}}$ transmitted in α, the source $h \in H$ of p*

neither crashes nor leaves the reliable multicast group for at least Δ_L time units past the point in time p is transmitted; that is, for any discrete transition (u, π, u'), for $u, u' \in states(\mathrm{SRM}_I)$, $h \in H$, $p \in P_{\textrm{RM-CLIENT}}$, and $\pi = \texttt{rm-send}_h(p)$, and the earliest state $u'' \in states(\mathrm{SRM}_I)$ in α, such that $u \leq_\alpha u''$ and $u.now + \Delta_L < u''.now$, it is the case that the timed execution fragment $\alpha_{uu''}$ of α leading from u to u'' contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions.

Let Δ_L-*src-recoverable-aexecs*(SRM_I), for some $\Delta_L \in \mathbb{R}^{\geq 0}$, be the set of all admissible timed executions of SRM_I in *aexecs*(SRM_I) that satisfy Constraints 4.1, 4.2, 4.3, 4.4, and 4.7, for some $\Delta_L \in \mathbb{R}^{\geq 0}$. Let Δ_L-*src-recoverable-attraces*(SRM_I) be the traces of all the executions of SRM_I in Δ_L-*src-recoverable-aexecs*(SRM_I). Let Δ_L-*src-recoverable-aexecs*$_k(\mathrm{SRM}_I)$, for $k \in \mathbb{N}^+$, be the subset of *aexecs*$_k(\mathrm{SRM}_I)$ comprised of all admissible timed executions of SRM_I that satisfy Constraints 4.1, 4.2, 4.3, 4.4, and 4.7, for some $\Delta_L \in \mathbb{R}^{\geq 0}$; that is, for $k \in \mathbb{N}^+$, Δ_L-*src-recoverable-aexecs*$_k(\mathrm{SRM}_I)$ = *aexecs*$_k(\mathrm{SRM}_I) \cap \Delta_L$-*src-recoverable-aexecs*(SRM_I). Moreover, let Δ_L-*src-recoverable-attraces*$_k(\mathrm{SRM}_I)$ be the traces of all executions of SRM_I in Δ_L-*src-recoverable-aexecs*$_k(\mathrm{SRM}_I)$.

Lemma 4.13 *For any $\Delta_L \in \mathbb{R}^{\geq 0}$ and $k \in \mathbb{N}$, it is the case that:*

1. *recoverable-aexecs*$(\mathrm{SRM}_I) \subseteq \Delta_L$-*src-recoverable-aexecs*(SRM_I) *and*
2. *recoverable-aexecs*$_k(\mathrm{SRM}_I) \subseteq \Delta_L$-*src-recoverable-aexecs*$_k(\mathrm{SRM}_I)$.

Proof: Follows directly from the definitions of the admissible timed execution sets of SRM_I *recoverable-aexecs*(SRM_I), Δ_L-*src-recoverable-aexecs*(SRM_I), *recoverable-aexecs*$_k(\mathrm{SRM}_I)$, and Δ_L-*src-recoverable-aexecs*$_k(\mathrm{SRM}_I)$, for $\Delta_L \in \mathbb{R}^{\geq 0}$ and $k \in \mathbb{N}$. ❒

Lemma 4.14 *For any $\Delta_L \in \mathbb{R}^{\geq 0}$ and $k \in \mathbb{N}$, it is the case that:*

1. *recoverable-attraces*$(\mathrm{SRM}_I) \subseteq \Delta_L$-*src-recoverable-attraces*(SRM_I) *and*
2. *recoverable-attraces*$_k(\mathrm{SRM}_I) \subseteq \Delta_L$-*src-recoverable-attraces*$_k(\mathrm{SRM}_I)$.

Proof: Follows directly from Lemma 4.13. ❒

The following constraint specifies the executions of SRM_I in which, when any packet p is detected as missing by any host h, there exists another host h' that has either sent or received the packet and is capable of retransmitting it for at least $\Delta_R \in \mathbb{R}^{\geq 0}$ time units past the point in time h detects the loss of p; that is, h' remains a member of the reliable multicast group for Δ_R time units past the point in time h detects the loss of p. The parameter Δ_R is presumed to correspond to the delay in recovering each packet; that is, the time elapsing from the point in time the loss of a packet is detected and a request for the given packet is scheduled, to the point in time the given packet is received and delivered to the client.

Constraint 4.8 (Δ_R-Recoverable) *Let $\Delta_R \in \mathbb{R}^{\geq 0}$ and α be any admissible timed execution of SRM_I. Let $h, s \in H, h \neq s$, $i \in \mathbb{N}$, and (u, π, u') be any discrete transition of SRM_I in α, for $u, u' \in states(\mathrm{SRM}_I)$ and $\pi \in acts(\mathrm{SRM}_I)$, such that $\langle s, i\rangle \notin u[\textrm{SRM-REC}_h].scheduled\text{-}rqsts?(s)$ and $\langle s, i\rangle \in u'[\textrm{SRM-REC}_h].scheduled\text{-}rqsts?(s)$. Moreover, let $u'' \in states(\mathrm{SRM}_I)$ be the earliest state in α such that $u'.now + \Delta_R < u''.now$ and $\alpha_{u'u''}$ be the timed execution fragment of α leading from u' to u''. Then, there exists $h' \in H, h' \neq h$ such that $\langle s, i\rangle \in u'[\textrm{SRM-REC}_{h'}].delivered(s)$ and the timed execution fragment $\alpha_{u'u''}$ contains neither $\texttt{crash}_{h'}$ nor $\texttt{rm-leave}_{h'}$ actions.*

Let Δ_R-*recoverable-aexecs*(SRM_I), for some $\Delta_R \in \mathbb{R}^{\geq 0}$, be the set of all admissible timed executions of SRM_I in *aexecs*(SRM_I) that satisfy Constraints 4.1, 4.2, 4.3, 4.4, and 4.8, for some $\Delta_R \in \mathbb{R}^{\geq 0}$. Let Δ_R-*recoverable-attraces*(SRM_I) be the traces of all the executions of SRM_I in Δ_R-*recoverable-aexecs*(SRM_I). Let Δ_R-*recoverable-aexecs*$_k(\mathrm{SRM}_I)$, for $k \in \mathbb{N}^+$, be the subset of *aexecs*$_k(\mathrm{SRM}_I)$ comprised of all admissible timed executions of SRM_I that satisfy Constraints 4.1, 4.2, 4.3, 4.4, and 4.8, for some $\Delta_R \in \mathbb{R}^{\geq 0}$; that is, for $k \in \mathbb{N}^+$, Δ_R-*recoverable-aexecs*$_k(\mathrm{SRM}_I)$ = *aexecs*$_k(\mathrm{SRM}_I) \cap \Delta_R$-*recoverable-aexecs*$(\mathrm{SRM}_I)$. Moreover, let Δ_R-*recoverable-attraces*$_k(\mathrm{SRM}_I)$ be the traces of all executions of SRM_I in Δ_R-*recoverable-aexecs*$_k(\mathrm{SRM}_I)$.

Lemma 4.15 *For any $\Delta_R \in \mathbb{R}^{\geq 0}$ and $k \in \mathbb{N}$, it is the case that:*

1. *recoverable-aexecs*$(\mathrm{SRM}_I) \subseteq \Delta_R$-*recoverable-aexecs*$(\mathrm{SRM}_I)$ *and*
2. *recoverable-aexecs*$_k(\mathrm{SRM}_I) \subseteq \Delta_R$-*recoverable-aexecs*$_k(\mathrm{SRM}_I)$.

Proof: Follows directly from the definitions of the admissible timed execution sets of SRM_I *recoverable-aexecs*(SRM_I), Δ_R-*recoverable-aexecs*(SRM_I), *recoverable-aexecs*$_k(\mathrm{SRM}_I)$, and Δ_R-*recoverable-aexecs*$_k(\mathrm{SRM}_I)$, for $\Delta_R \in \mathbb{R}^{\geq 0}$ and $k \in \mathbb{N}$. ❒

Lemma 4.16 *For any $\Delta_R \in \mathbb{R}^{\geq 0}$ and $k \in \mathbb{N}$, it is the case that:*

1. *recoverable-attraces*$(\mathrm{SRM}_I) \subseteq \Delta_R$-*recoverable-attraces*$(\mathrm{SRM}_I)$ *and*
2. *recoverable-attraces*$_k(\mathrm{SRM}_I) \subseteq \Delta_R$-*recoverable-attraces*$_k(\mathrm{SRM}_I)$.

Proof: Follows directly from Lemma 4.15. ❒

Let α' be any timed execution fragment of SRM_I that contains the transmission of a packet $p \in P_{\text{RM-CLIENT}}$ and in which some host $h \in H$ neither crashes nor leaves the reliable multicast group; that is, α' contains the action $\texttt{rm-send}_{s_p}(p)$, for $s_p = source(p)$, and α' contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, for some $h \in H$. We say that *the host h detects the loss of p in α'* if it schedules a request for $p \in P_{\text{RM-CLIENT}}$ in α'. If the host h detects the loss of p in α', then we let $\alpha'.det\text{-}time_h(p)$ denote the point in time in α' at which h detects the loss of p. We let $\alpha'.det\text{-}latency_h(p)$ denote *the loss detection latency of p for h* in α'; that is, the time elapsing from the time p is transmitted to the time the host h detects the loss of p in α'. Supposing that h receives p is α' following the point in time at which h detects the loss of p, we let $\alpha'.rec\text{-}latency_h(p)$ denote the *loss recovery latency of p for h in α'*; that is, the time elapsing from the time the host h detects the loss of p to the time it receives p in α'.

When a host $h \in H$ schedules a request for $p \in P_{\text{RM-CLIENT}}$ with a back-off of $k-1$, for any $k \in \mathbb{N}^+$, we say that it initiates a k-th recovery round for p. Each recovery round (except the first) also initiates a back-off abstinence period. Any request for p received during this back-off abstinence period is discarded. If the packet p is received while a scheduled request for p by h is awaiting transmission, then the scheduled request is canceled. Once the back-off abstinence period expires, either the reception of a request for p or the transmission of the scheduled request for p by h initiates the $k+1$-st recovery round of h for p. In this case, we let the *k-th round request* of h for p be the request for p upon whose reception or transmission the host h initiates the $k+1$-st recovery round for p. Moreover, we define the *completion time* of the k-th recovery round for p of h to be the point in time at which h either receives p or initiates its $k+1$-st recovery round for p.

Suppose that a host $h' \in H$ receives the k-th round request of h for p while it is a member of the reliable multicast group and after archiving the packet p. When h' receives this request, either i) a reply for p is already scheduled, ii) a reply for p is already pending, or iii) a reply for p is neither scheduled, nor pending. When a reply for p is already scheduled, h's request for p is discarded. In this case, the reply that is already scheduled at h' is considered to be the reply pertaining to the k-th round request of h for p. When a reply for p is already pending, h's request for p is discarded. In this case, the reply that is pending at h' is considered to be the reply pertaining to the k-th round request of h for p. When a reply for p is neither scheduled, nor pending, h' schedules a reply for p. In this case, the reply that is either received or transmitted by h' and that results in the cancellation of the reply scheduled by h' for p is considered to be h''s reply to the k-th round request of h for p.

Preliminary Lemmas

Lemma 4.17 *Let α be any admissible timed execution of* SRM_I *that satisfies Constraint 4.1 and contains the occurrence of a discrete transition* (u, π, u')*, for* $u, u' \in \mathit{states}(\mathrm{SRM}_I)$*,* $h \in H$*,* $p \in P_{\text{IPMCAST-CLIENT}}$*, and* $\pi = \mathtt{mrecv}_h(p)$*. Then, any* $\mathtt{mrecv}_{h'}(p)$ *action, for* $h' \in H$*, in α occurs no earlier and no later than* $\overline{d} - \underline{d}$ *time units from the discrete transition* (u, π, u') *in* α*.*

Proof: Let (w, π, w'), for $w, w' \in \mathit{states}(\mathrm{SRM}_I)$, $p \in P_{\text{IPMCAST-CLIENT}}$, $s_p = \mathit{source}(p)$, and $\pi = \mathtt{msend}_{s_p}(p)$, be the discrete transition in α involving the transmission of p. Constraint 4.1 implies that the time elapsing from the time of occurrence of the action $\mathtt{msend}_{s_p}(p)$ to that of any action $\mathtt{mrecv}_{h''}(p)$, for $h'' \in H, h'' \neq s_p$, lies in the interval $[\underline{d}, \overline{d}]$. Thus, any two such actions that occur in α are separated in time by at most $\overline{d} - \underline{d}$ time units. ❒

Lemma 4.18 *Let α be any admissible timed execution of* SRM_I *that contains the transmission of a packet* $p \in P_{\text{RM-CLIENT}}$*. For any state* $u \in \mathit{states}(\mathrm{SRM}_I)$ *in* α*, if* $u.\mathit{trans\text{-}time}(p) \neq \perp$*, then* $u.\mathit{trans\text{-}time}(p) = \alpha.\mathit{trans\text{-}time}(p)$*.*

Proof: The only action that sets the variable $\mathit{trans\text{-}time}(p)$ is the action $\mathtt{rm\text{-}send}_{s_p}(p)$, for $s_p = \mathit{source}(p)$. By Lemma 3.2, the action $\mathtt{rm\text{-}send}_{s_p}(p)$ occurs only once in α. Let (w, π, w'), for $w, w' \in \mathit{states}(\mathrm{SRM}_I)$, $p \in P_{\text{IPMCAST-CLIENT}}$, $s_p = \mathit{source}(p)$, and $\pi = \mathtt{msend}_{s_p}(p)$, be the discrete transition in α involving the transmission of p. By the definition of $\alpha.\mathit{trans\text{-}time}(p)$, it follows that $\alpha.\mathit{trans\text{-}time}(p) = w.\mathit{now}$. The action $\mathtt{rm\text{-}send}_{s_p}(p)$ sets the variable $\mathit{trans\text{-}time}(p)$ to the value of now. It follows that $w'.\mathit{trans\text{-}time}(p) = \alpha.\mathit{trans\text{-}time}(p)$.

Since the action $\mathtt{rm\text{-}send}_{s_p}(p)$ occurs in α only once, it follows that, for any $w_-, w'_+ \in \alpha$, such that $w_- \leq_\alpha w$ and $w' \leq_\alpha w'_+$, it is the case that $w_-.\mathit{trans\text{-}time}(p) = \perp$ and $w'_+.\mathit{trans\text{-}time}(p) = w'.\mathit{trans\text{-}time}(p)$. Since $w'.\mathit{trans\text{-}time}(p) = \alpha.\mathit{trans\text{-}time}(p)$, it follows that $w'_+.\mathit{trans\text{-}time}(p) = \alpha.\mathit{trans\text{-}time}(p)$. ❒

Lemma 4.19 *Let* $h, h' \in H$*,* $\alpha \in \mathit{aexecs}(\mathrm{SRM}_I)$*,* $u, u' \in \mathit{states}(\mathrm{SRM}_I)$ *be any states in* α*, such that* $u \leq_\alpha u'$*, and* $\alpha_{uu'}$ *be the finite execution fragment of* α *leading from* u *to* u'*. If* $u[\text{SRM-REC}_h].\mathit{expected}(h') \neq \emptyset$ *and* $\alpha_{uu'}$ *contains neither* $\mathtt{crash}_h$ *nor* $\mathtt{rm\text{-}leave}_h$ *actions, then it is the case that* $u[\text{SRM-REC}_h].\mathit{expected}(h') = u'[\text{SRM-REC}_h].\mathit{expected}(h')$*.*

Proof: Suppose that $u[\text{SRM-REC}_h].\mathit{expected}(h') \neq \emptyset$ and $\alpha_{uu'}$ contains neither $\mathtt{crash}_h$ nor $\mathtt{rm\text{-}leave}_h$ actions. The proof is by induction on the length $n \in \mathbb{N}$ of $\alpha_{uu'}$. For the base case, consider a finite execution fragment $\alpha_{uu'}$ of length $n = 0$. Since $u = u'$, it trivially follows that $u[\text{SRM-REC}_h].\mathit{expected}(h') = u'[\text{SRM-REC}_h].\mathit{expected}(h')$.

For the inductive step, consider an execution fragment $\alpha_{uu'}$ of length $n = k+1$. Let α_k be the prefix of $\alpha_{uu'}$ involving the first k steps and $u_k = \alpha_k.\mathit{lstate}$. Suppose that $u[\text{SRM-REC}_h].\mathit{expected}(h') \neq \emptyset$

and $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. The induction hypothesis implies that $u[\text{SRM-REC}_h].\mathit{expected}(h') = u_k[\text{SRM-REC}_h].\mathit{expected}(h')$.

Now, consider the step from u_k to u'. The only actions of SRM-REC_h that may affect the variable $\text{SRM-REC}_h.\mathit{expected}(h')$ are the actions $\texttt{crash}_h$, $\texttt{rm-leave}_h$, $\texttt{rm-send}_h(p)$, and $\texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$. Since $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, the step from u_k to u' is neither a $\texttt{crash}_h$ nor a $\texttt{rm-leave}_h$ action. The action $\texttt{rm-send}_h(p)$ affects the variable $\text{SRM-REC}_h.\mathit{expected}(h')$ only when $h' = h = \mathit{source}(p)$ and $\text{SRM-REC}_h.\mathit{expected}(h') = \emptyset$. The action $\texttt{rm-recv}_h(p)$ affects the variable $\text{SRM-REC}_h.\mathit{expected}(h')$ only when $h' = \mathit{source}(p)$ and $\text{SRM-REC}_h.\mathit{expected}(h') = \emptyset$. Since $u[\text{SRM-REC}_h].\mathit{expected}(h') \neq \emptyset$, the step from u_k to u' does not affect the variable $\text{SRM-REC}_h.\mathit{expected}(h')$. Thus, it is the case that $u_k[\text{SRM-REC}_h].\mathit{expected}(h') = u'[\text{SRM-REC}_h].\mathit{expected}(h')$. Since $u[\text{SRM-REC}_h].\mathit{expected}(h') = u_k[\text{SRM-REC}_h].\mathit{expected}(h')$ and $u_k[\text{SRM-REC}_h].\mathit{expected}(h') = u'[\text{SRM-REC}_h].\mathit{expected}(h')$, it follows that $u[\text{SRM-REC}_h].\mathit{expected}(h') = u'[\text{SRM-REC}_h].\mathit{expected}(h')$. ❒

Lemma 4.20 *Let $h, h' \in H$, $\alpha \in \mathit{aexecs}(\text{SRM}_I)$, $u, u' \in \mathit{states}(\text{SRM}_I)$ be any states in α, such that $u \leq_\alpha u'$, and $\alpha_{uu'}$ be the execution fragment of α leading from u to u'. If $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, then it is the case that $u[\text{SRM-REC}_h].\mathit{expected}(h') \subseteq u'[\text{SRM-REC}_h].\mathit{expected}(h')$.*

Proof: Suppose that $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. If it is the case that $u[\text{SRM-REC}_h].\mathit{expected}(h') = \emptyset$, then it trivially follows that $u[\text{SRM-REC}_h].\mathit{expected}(h') \subseteq u'[\text{SRM-REC}_h].\mathit{expected}(h')$. Otherwise, if $u[\text{SRM-REC}_h].\mathit{expected}(h') \neq \emptyset$, then Lemma 4.19 implies that $u[\text{SRM-REC}_h].\mathit{expected}(h') = u'[\text{SRM-REC}_h].\mathit{expected}(h')$. It follows that $u[\text{SRM-REC}_h].\mathit{expected}(h') \subseteq u'[\text{SRM-REC}_h].\mathit{expected}(h')$. ❒

Lemma 4.21 *Let $h, h' \in H$, $\alpha \in \mathit{aexecs}(\text{SRM}_I)$, $u, u' \in \mathit{states}(\text{SRM}_I)$ be any states in α, such that $u \leq_\alpha u'$, and $\alpha_{uu'}$ be the finite execution fragment of α leading from u to u'. If $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, then it is the case that $u[\text{SRM-REC}_h].\mathit{delivered}(h') \subseteq u'[\text{SRM-REC}_h].\mathit{delivered}(h')$.*

Proof: Suppose that the finite execution fragment $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. The fact that $u[\text{SRM-REC}_h].\mathit{delivered}(h') \subseteq u'[\text{SRM-REC}_h].\mathit{delivered}(h')$ follows by induction on the length $n \in \mathbb{N}$ of $\alpha_{uu'}$ after recognizing that all the actions, except the actions $\texttt{crash}_h$ and $\texttt{rm-leave}_h$, may only add elements to the variable $\text{SRM-REC}_h.\mathit{delivered}(h')$. ❒

Lemma 4.22 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of SRM_I in $\mathit{timely\text{-}aexecs}(\text{SRM}_I)$ that contains the transmission of p. Let (w, π, w'), for $w, w' \in \mathit{states}(\text{SRM}_I)$, $s_p = \mathit{source}(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of SRM_I involving the transmission of p using the reliable multicast service. For any states u, u' of SRM_I in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. For any $k \in \mathbb{N}^+$ and $h \in H, h \neq s_p$, suppose that $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions and h schedules k-th and $k+1$-st round requests for p in $\alpha_{uu'}$. Let $t_k, t_{k+1} \in \mathbb{R}^{\geq 0}$ be the points in time in $\alpha_{uu'}$ at which the host h schedules its k-th and $k+1$-st round requests for p, respectively. Then, it is the case that $t_{k+1} \leq t_k + 2^{k-1}(C_1 + C_2)\overline{d}$.*

Proof: This follows from the fact that time in the SRM-REC_h automaton is not allowed to elapse past the transmission time of any scheduled request. Constraint 4.3 implies that the k-th round request is scheduled for transmission no later than $t_k + 2^{k-1}(C_1 + C_2)\overline{d}$. Thus, if no request is received by h prior to the time at which its k-th round request for p is scheduled for transmission, then h transmits its k-th round request. Thus, h either sends or receives its k-th round request for p no later than $t_k + 2^{k-1}(C_1 + C_2)\overline{d}$. ❒

Corollary 4.23 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of SRM_I in timely-aexecs(SRM_I) that contains the transmission of p. Let (w, π, w'), for $w, w' \in$ states(SRM_I), $p \in P_{\text{RM-CLIENT}}$, $s_p =$ source(p), and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of SRM_I involving the transmission of p using the reliable multicast service. For any states u, u' of SRM_I in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. For any $k \in \mathbb{N}^+$ and $h \in H, h \neq s_p$, suppose that $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions and contains the discrete transition in which h detects the loss of p. Moreover, suppose that, following the detection of p in $\alpha_{uu'}$, h schedules a $k+1$-st round request for p in $\alpha_{uu'}$. Let $t_{k+1} \in \mathbb{R}^{\geq 0}$ be the point in time in $\alpha_{uu'}$ at which the host h schedules its $k+1$-st round request for p. Then, it is the case that $t_{k+1} \leq \alpha_{uu'}.\text{det-time}_h(p) + (2^k - 1)(C_1 + C_2)\overline{d}$.*

Proof: Follows from Lemma 4.22 and the fact that h detects the loss of p at the point in time when it first schedules a request for p. According to the SRM-REC$_h$ automaton, the first request scheduled for a packet is either a 1-st or 2-nd round request for the given packet. ❒

Lemma 4.24 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of SRM_I in timely-aexecs(SRM_I) that contains the transmission of p. Let (w, π, w'), for $w, w' \in$ states(SRM_I), $s_p =$ source(p), and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of SRM_I involving the transmission of p using the reliable multicast service. For any states u, u' of SRM_I in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. For any $k \in \mathbb{N}^+$ and $h \in H, h \neq s_p$, suppose that $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions and h schedules k-th and $k+1$-st round requests for p in $\alpha_{uu'}$. Let $t_k, t_{k+1} \in \mathbb{R}^{\geq 0}$ be the points in time in $\alpha_{uu'}$ at which the host h schedules its k-th and $k+1$-st round requests for p, respectively. Then, it is the case that $t_k + 2^{k-1}C_3\underline{d} < t_{k+1}$.*

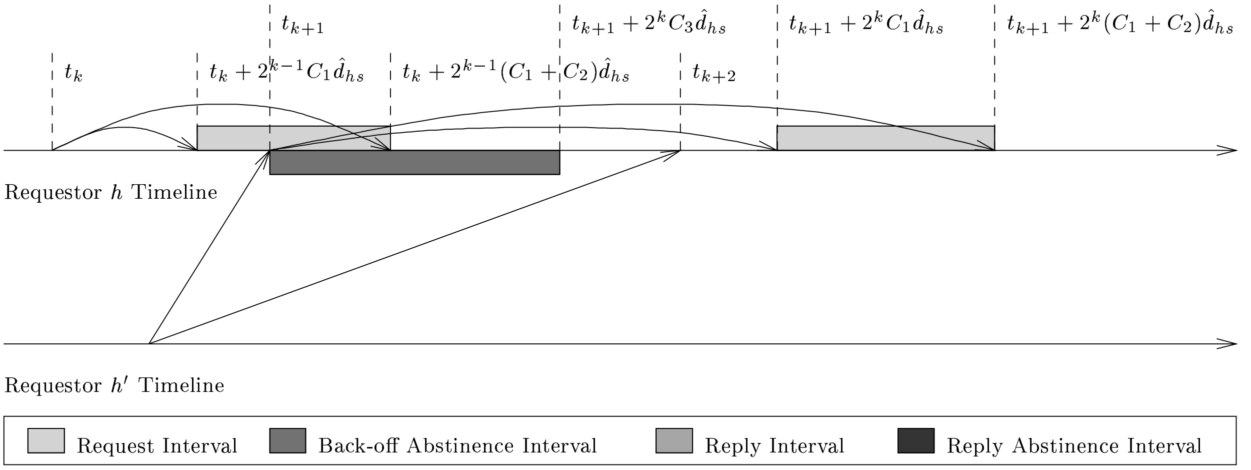
Proof: Constraint 4.3 implies that the k-th round back-off abstinence period expires no earlier than $2^{k-1}C_3\underline{d}$ time units past t_k; that is, no earlier than $t_k + 2^{k-1}C_3\underline{d}$ in α. The k-th round request of h for p is scheduled for transmission for a point in time no earlier than $t_k + 2^{k-1}C_1\underline{d}$. Thus, Assumption 4.1 implies that the k-th round request is scheduled for transmission at a point in time that succeeds $t_k + 2^{k-1}C_3\underline{d}$ in α.

The host h schedules its $k+1$-st round request for p when it either sends or receives its k-th round request for p; that is, upon the occurrence of either a $\texttt{send-rqst}_h(s, i)$ action, such that $\langle s, i \rangle = id(p)$, or a $\texttt{process-pkt}_h(pkt)$ action, for $pkt \in P_{\text{SRM}}$, such that $id(pkt) = id(p)$ and $type(pkt) = \texttt{RQST}$. In the case of a $\texttt{send-rqst}_h(s, i)$ action, Invariant 4.20 implies that if the $\texttt{send-rqst}_h(s, i)$ action is enabled, then a request for p is not pending. In the case of a $\texttt{process-pkt}_h(pkt)$ action, the effects of the action $\texttt{process-pkt}_h(pkt)$ imply that the k-th round request for p is backed-off only while a request for p is not pending.

It follows that the point in time at which the host h either sends or receives its k-th round request for p succeeds the expiration time of the back-off abstinence period of the k-th round request of h for p; that is, $t_k + 2^{k-1}C_3\underline{d} < t_{k+1}$. ❒

Let $k^*_{rqst} = \lceil \log_2(\overline{d} - \underline{d}) - \log_2(C_3\underline{d}) \rceil$. The following lemma states that, under Constraints 4.1, 4.2, 4.3, and 4.4, k^*_{rqst} is the number of requests that must be scheduled before the back-off abstinence periods become large enough to ensure that the request pertaining to one round is distinct from the

Figure 4.19 Timing Diagram Demonstrating Non-distinct Consecutive Round Requests



request pertaining to the next (and, consequently, any following) round. Consider for instance the timing diagram depicted in Figure 4.19. The figure portrays the scenario in which a request from h' is received in duplicate by h. Thus, the requests pertaining to the k-th and $k+1$-st recovery rounds are in fact a single request that is received by h in duplicate. Lemma 4.17 implies that duplicate requests may be received within at most $\overline{d} - \underline{d}$ time units. Since requests received during abstinence periods are discarded, k^*_{rqst} is the number of requests that must be scheduled before the back-off abstinence periods become large enough to ensure that any duplicates of a round's request are received prior to the expiration time of the particular round's back-off abstinence period.

Lemma 4.25 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of SRM_I in timely-aexecs(SRM_I) that contains the transmission of p. Let (w, π, w'), for $w, w' \in$ states(SRM_I), $s_p =$ source(p), and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of SRM_I in α involving the transmission of p using the reliable multicast service. For any states u, u' of SRM_I in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. Moreover, let $h \in H$ be any member of the reliable multicast group in u, such that id$(p) \in u[\text{SRM-REC}_h]$.expected$(s_p)$ and id$(p) \notin u[\text{SRM-REC}_h]$.scheduled-rqsts?.*

For $k \in \mathbb{N}^+, k \geq k^*_{rqst}$, suppose that $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, h schedules k-th and $k+1$-st round requests for p in $\alpha_{uu'}$, and h either sends or receives its k-th and $k+1$-st round requests for p at the points in time $t_{k+1}, t_{k+2} \in \mathbb{R}^{\geq 0}$ in $\alpha_{uu'}$.

Then, the k-th and $k+1$-st round requests of h for p are distinct.

Proof: It suffices to show that the back-off abstinence period pertaining to the k-th round request of h for p expires no earlier than the latest point in time h may receive any duplicate of the request pertaining to its k-th round request for p.

Let $t_k, t_{k+1} \in \mathbb{R}^{\geq 0}$ be the points in time in α at which h schedules its k-th and $k+1$-st round requests for p. From Lemma 4.24, the back-off abstinence period pertaining to k-th round request of h for p expires no earlier than $t_{k+1} + 2^k C_3 \underline{d}$. From Lemma 4.17, h may receive a duplicate of its k-th round request for p no later than $t_{k+1} + (\overline{d} - \underline{d})$.

Since $k^*_{rqst} = \lceil \log_2(\overline{d} - \underline{d}) - \log_2(C_3 \underline{d}) \rceil$ and $k \geq k^*_{rqst}$, it follows that $t_{k+1} + (\overline{d} - \underline{d}) \leq t_{k+1} + 2^k C_3 \underline{d}$.

Since h either sends or receives its $k+1$-st round request for p after the point in time $t_{k+1} + 2^k C_3 \underline{d}$ and h may receive a duplicate of its k-th round request for p no later than $t_{k+1} + (\overline{d} - \underline{d})$, it follows that the requests pertaining to the k-th and $k+1$-st round requests of h for p are distinct. ❒

Lemma 4.26 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of SRM_I in timely-aexecs(SRM_I) that contains the transmission of p. Let (w, π, w'), for $w, w' \in$ states(SRM_I), $s_p =$ source(p), and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of SRM_I involving the transmission of p using the reliable multicast service. For any states u, u' of SRM_I in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. For any $k \in \mathbb{N}^+$ and $h, h' \in H, h \neq h'$, suppose that $u[\text{SRM-MEM}_h].status = \texttt{member}$, $u[\text{SRM-MEM}_{h'}].status = \texttt{member}$, $\alpha_{uu'}$ contains neither $\texttt{crash}_h$, $\texttt{rm-leave}_h$, $\texttt{crash}_{h'}$, nor $\texttt{rm-leave}_{h'}$ actions, h schedules k-th and $k+1$-st round requests for the packet p in $\alpha_{uu'}$, h either sends or receives its k-th round request for p and schedules its $k+1$-st round request for p at the point in time $t_{k+1} \in \mathbb{R}^{\geq 0}$ in $\alpha_{uu'}$, and $t_{k+1} + \overline{d} < u'.now$. Then, h' may receive the k-th round request of h for p no later than $t_{k+1} + \overline{d}$ in α.*

Proof: The host h either sends or receives its k-th round request for p and schedules its $k+1$-st round request for p upon the occurrence of either a $\texttt{send-rqst}_h(s,i)$ or a $\texttt{process-pkt}_h(pkt)$ action, where $id(pkt) = id(p)$ and $type(pkt) = \texttt{RQST}$. We consider there two cases separately.

First, in the case of a $\texttt{send-rqst}_h(s,i)$ action, Constraints 4.5 and 4.6 and Lemmas 4.7 and 4.8 imply that the $\texttt{send-rqst}_h(s,i)$ action is instantaneously followed by a $\texttt{msend}_h(pkt')$ action, for $pkt' \in P_{\text{IPMCAST-CLIENT}}$, such that $id(strip(pkt')) = id(p)$ and $type(strip(pkt')) = \texttt{RQST}$. Furthermore, Constraint 4.1 implies that h' receives this request within at most $\overline{d}$ time units.

Second, in the case of a $\texttt{process-pkt}_h(pkt)$ action, a $\texttt{mrecv}_h(pkt')$ action, for $pkt' \in P_{\text{IPMCAST-CLIENT}}$, such that $pkt = strip(pkt')$, instantaneously precedes $\texttt{process-pkt}_h(pkt)$. Lemma 4.17 implies that h' may only receive this request within at most $\overline{d} - \underline{d}$ time units from the occurrence of the $\texttt{mrecv}_h(pkt')$ action. ❐

Lemma 4.27 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of SRM_I in timely-aexecs(SRM_I) that contains the transmission of p. Let (w, π, w'), for $w, w' \in$ states(SRM_I), $s_p =$ source(p), and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of SRM_I involving the transmission of p using the reliable multicast service. For any states u, u' of SRM_I in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. For any $k \in \mathbb{N}^+$ and $h, h' \in H, h \neq h'$, suppose that $u[\text{SRM-MEM}_h].status = \texttt{member}$, $u[\text{SRM-MEM}_{h'}].status = \texttt{member}$, $id(p) \in u[\text{SRM-REC}_{h'}].archived\text{-}pkts?$, $\alpha_{uu'}$ contains neither $\texttt{crash}_h$, $\texttt{rm-leave}_h$, $\texttt{crash}_{h'}$, nor $\texttt{rm-leave}_{h'}$ actions, h' receives a request for p from h at time $t' \in \mathbb{R}^{\geq 0}$ in $\alpha_{uu'}$, and $t' + (D_1 + D_2)\overline{d} < u'.now$. Then, the reply of h' pertaining to this particular request of h for p is either sent or received by h' no later than $t' + (D_1 + D_2)\overline{d}$ in α.*

Proof: Constraint 4.3 implies a reply is scheduled for transmission no later than $(D_1 + D_2)\overline{d}$ time units past its scheduling time. When h' receives the request of h for p, a reply for p is either already scheduled, already pending, or neither scheduled nor pending. We consider each of these scenarios separately.

First, if a reply for p is already scheduled, its transmission time is no later than $t' + (D_1 + D_2)\overline{d}$ in α. Thus, if either an original transmission or a reply for p is not received by h' by the scheduled transmission time of this reply, then the host h' transmits this reply. It follows that the reply of h' pertaining to the particular request of h for p is either sent or received by h' no later than the point in time $t' + (D_1 + D_2)\overline{d}$ in α.

Second, if a reply for p is already pending, then the reply of h' pertaining to the particular request of h for p has already been either sent or received; that is, the reply of h' pertaining to the particular request of h for p has been either sent or received by h' no later than t'.

Finally, if a reply for p is neither scheduled nor pending, then the reply of h' pertaining to the particular request of h for p is scheduled for no later than $t' + (D_1 + D_2)\overline{d}$. In either scenario, the reply of h' pertaining to the particular request of h for p is either sent or received by h' no later than $t' + (D_1 + D_2)\overline{d}$ in α. ❒

Lemma 4.28 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of* SRM_I *in timely-aexecs*(SRM_I) *that contains the transmission of p. Let (w, π, w'), for $w, w' \in$ states(SRM_I), $s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of SRM_I involving the transmission of p using the reliable multicast service. For any states u, u' of SRM_I in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. For any $k \in \mathbb{N}^+$ and $h, h' \in H, h \neq h'$, suppose that $u[\text{SRM-MEM}_h].status = \texttt{member}$, $u[\text{SRM-MEM}_{h'}].status = \texttt{member}$, $id(p) \in u[\text{SRM-REC}_{h'}].archived\text{-}pkts?$, $\alpha_{uu'}$ contains neither $\texttt{crash}_h$, $\texttt{rm-leave}_h$, $\texttt{crash}_{h'}$, nor $\texttt{rm-leave}_{h'}$ actions, h' receives a request for p from h at time $t' \in \mathbb{R}^{\geq 0}$ in $\alpha_{uu'}$, and $t' + (D_1 + D_2)\overline{d} + \overline{d} - \underline{d} + D_3\overline{d} < u'.now$. Then, the reply abstinence period of the reply of h' pertaining to this particular request of h for p expires no later than $t' + (D_1 + D_2)\overline{d} + \overline{d} - \underline{d} + D_3\overline{d}$ in α.*

Proof: Lemma 4.27 implies that the reply period pertaining to the particular request of h for p expires no later than the point in time $t' + (D_1 + D_2)\overline{d}$. Thus, the reply of h' pertaining to the particular request of h for p may be either sent or received no later than $t' + (D_1 + D_2)\overline{d}$.

First, consider the case in which h' sends a reply pertaining to the particular request of h for p. Such a reply is sent no later than $t' + (D_1 + D_2)\overline{d}$. Constraint 4.3 implies that the reply abstinence period corresponding to such a reply expires no later than $t' + (D_1 + D_2 + D_3)\overline{d}$.

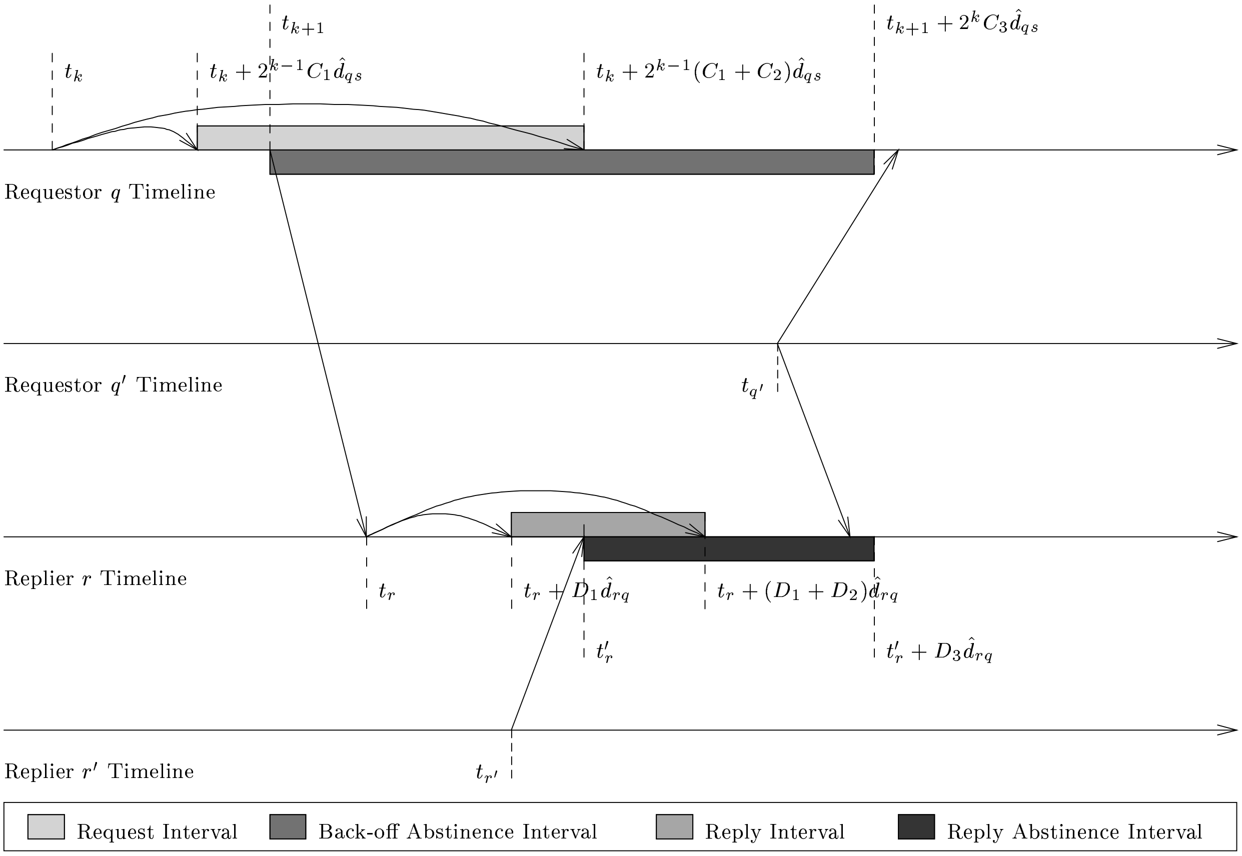
Second, consider the case in which h' receives a reply pertaining to the particular request of h for p prior to transmitting its own reply. Such a reply is received by h' no later than $t' + (D_1 + D_2)\overline{d}$. Lemma 4.17 implies that any duplicates of this reply may be received within at most $\overline{d} - \underline{d}$ time units. Thus, such duplicates are received by h' no later than $t' + (D_1 + D_2)\overline{d} + \overline{d} - \underline{d}$. Thus, the reply abstinence period pertaining to any such duplicate expires no later than $t' + (D_1 + D_2)\overline{d} + \overline{d} - \underline{d} + D_3\overline{d}$. ❒

Let $k^*_{repl} = \lceil \log_2[(D_1 + D_2 + D_3 + 3)\overline{d} - 2\underline{d}] - \log_2(C_3\underline{d}) \rceil$. The following lemma states that, under Constraints 4.1, 4.2, 4.3, and 4.4, k^*_{repl} is the number of requests that must be scheduled before the back-off abstinence periods become large enough to ensure that the reply pertaining to a particular round is distinct from that pertaining to the next (and, consequently, any following) round.

Lemma 4.29 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of* SRM_I *in timely-aexecs*(SRM_I) *that contains the transmission of p. Let (w, π, w'), for $w, w' \in$ states(SRM_I), $s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of SRM_I in α involving the transmission of p using the reliable multicast service. For any states u, u' of SRM_I in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. Moreover, let $q, r \in H, q \neq r$ be any members of the reliable multicast group in u, such that $id(p) \in u[\text{SRM-REC}_q].expected(s_p)$, $id(p) \notin u[\text{SRM-REC}_q].scheduled\text{-}rqsts?$, and $id(p) \in u[\text{SRM-REC}_r].delivered(s_p)$.*

For $k \in \mathbb{N}^+, k \geq k^*_{repl}$, suppose that $\alpha_{uu'}$ contains neither $\texttt{crash}_q$, $\texttt{rm-leave}_q$, $\texttt{crash}_r$, nor $\texttt{rm-leave}_r$ actions, q schedules k-th, $k+1$-st, and $k+2$nd round requests for the packet p in $\alpha_{uu'}$, q either sends or receives its k-th and $k+1$-st round requests for p at the points in time $t_{k+1}, t_{k+2} \in \mathbb{R}^{\geq 0}$ in $\alpha_{uu'}$, r receives the k-th and $k+1$-st round requests of q for p in $\alpha_{uu'}$, and r either sends or receives the replies pertaining to the k-th and $k+1$-st round requests of q for p in $\alpha_{uu'}$.

Figure 4.20 Timing Diagram Demonstrating Non-distinct Consecutive Round Replies



Then, the replies of r pertaining to the k-th and $k+1$-st round requests of q for p are distinct.

Proof: It suffices to show that the reply abstinence period pertaining to r's reply to the k-th round request of q for p expires prior to the time at which r receives the $k+1$-st round request of q for p.

Let $t_k, t_{k+1} \in \mathbb{R}^{\geq 0}$ be the points in time in α at which q schedules its k-th and $k+1$-st round requests for p. From Lemma 4.26, r receives the k-th round request of q for p no later than $t_{k+1} + \overline{d}$. From Lemma 4.28, the abstinence period of the reply of q to the k-th round request of q for p expires no later than $t_{k+1} + \overline{d} + (D_1 + D_2)\overline{d} + (\overline{d} - \underline{d}) + D_3\overline{d}$.

From Lemma 4.24, q either sends or receives its $k+1$-st round request after the point in time $t_{k+1} + 2^k C_3 \underline{d}$. From Lemma 4.17, r receives this request after the point in time $t_{k+1} + 2^k C_3 \underline{d} - (\overline{d} - \underline{d})$. Since $k^*_{repl} = \lceil \log_2[(D_1 + D_2 + D_3 + 3)\overline{d} - 2\underline{d}] - \log_2(C_3\underline{d}) \rceil$ and $k \geq k^*_{repl}$, it follows that $t_{k+1} + \overline{d} + (D_1 + D_2)\overline{d} + (\overline{d} - \underline{d}) + D_3\overline{d} \leq t_{k+1} + 2^k C_3 \underline{d} - (\overline{d} - \underline{d})$.

Since r receives the $k+1$-st round request of q for p after the point in time $t_{k+1} + 2^k C_3 \underline{d} - \overline{d} + \underline{d}$ and $t_{k+1} + \overline{d} + (D_1 + D_2)\overline{d} + \overline{d} - \underline{d} + D_3\overline{d} \leq t_{k+1} + 2^k C_3 \underline{d} - \overline{d} + \underline{d}$, it follows that r receives the $k+1$-st round request of q for p after the expiration of the abstinence period of the reply of r to the k-th round request of q for p. It follows that the replies of r to the k-th and $k+1$-st round requests of q for p are distinct. ❒

Let $k^* = \max(k^*_{rqst}, k^*_{repl}) = \lceil \log_2[(D_1 + D_2 + D_3 + 3)\overline{d} - 2\underline{d}] - \log_2(C_3\underline{d}) \rceil$ and $\texttt{REC-BOUND}(k) = [(2^k - 1)(C_1 + C_2) + D_1 + D_2 + 2]\overline{d}$, for $k \in \mathbb{N}^+$. The following lemma states that, for $k \in \mathbb{N}^+$, the recovery of any packet in an admissible execution $\alpha \in \mathit{timely\text{-}aexecs}_k(\mathrm{SRM}_I)$ involves at most

$k^* + k$ recovery rounds. Following the k^*-th recovery round, the requests and replies of any host's recovery rounds are distinct. Thus, the k^*-th and each subsequent recovery round may fail only due to at least one packet drop; that is, the drop of either the particular round's request or the particular round's reply. Since the number of packet drops pertaining to the recovery of any packet in α is at most k, it follows that at most $k^* + k$ recovery rounds are needed to recover any packet in α.

Lemma 4.30 *Let $k \in \mathbb{N}^+$, $p \in P_{\text{RM-CLIENT}}$, and α be any admissible timed execution of SRM_I in $\mathit{timely\text{-}aexecs}_k(\text{SRM}_I)$ that contains the transmission of p. Let (w, π, w'), for $w, w' \in \mathit{states}(\text{SRM}_I)$, $s_p = \mathit{source}(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of SRM_I in α involving the transmission of p using the reliable multicast service. Suppose that the host $h \in H$ schedules a request for p following the transmission of p in α. Let $u \in \mathit{states}(\text{SRM}_I)$ be the first state in α such that $\mathit{id}(p) \in u[\text{SRM-REC}_h].\mathit{scheduled\text{-}rqsts?}(s_p)$, $u' \in \mathit{states}(\text{SRM}_I)$ be any state in α such that $u.\mathit{now} + \texttt{REC-BOUND}(k^* + k) < u'.\mathit{now}$, and $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. Suppose that $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, there exists a host $h' \in H, h' \neq h$, such that $\mathit{id}(p) \in u[\text{SRM-REC}_{h'}].\mathit{delivered}(s_p)$, and $\alpha_{uu'}$ contains neither $\texttt{crash}_{h'}$, nor $\texttt{rm-leave}_{h'}$ actions. Then, it is the case that $\mathit{id}(p) \in u'[\text{SRM-REC}_h].\mathit{delivered}(s_p)$.*

Proof: Since $\alpha \in \mathit{timely\text{-}aexecs}_k(\text{SRM}_I)$, it contains at most k packet drops pertaining to the transmission and recovery of p. Thus, it follows that at most k packet drops may occur during the recovery of p by h. Lemmas 4.5 and 4.6 imply that following the state u in α, the host h continues initiating recovery rounds for p until it is recovered. We proceed by showing that the host h recovers p by the completion time of its $k^* + k$-st recovery round for p.

Consider the interaction of h and h' pertaining to h's recovery of p. From Lemma 4.29, the replies of h' to the k^*-th and any subsequent round requests of h for p are distinct. Thus, the k^*-th and all subsequent recovery rounds of h for p may fail due to the loss of either the round's request or the round's reply; that is, the k^*-th and each subsequent recovery rounds of h for p account for at least one packet drop. It follows that at most $k^* + k$ recovery rounds are required for h to successfully recover p.

Corollary 4.23, Lemma 4.27, and Constraint 4.1 imply that h completes its $k^* + k$-st recovery round no later than $\texttt{REC-BOUND}(k^* + k)$ time units past the point in time at which it schedules its first request for p. Since u is the first state in α such that $\mathit{id}(p) \in \text{SRM-REC}_h.\mathit{scheduled\text{-}rqsts?}(s_p)$ and $u.\mathit{now} + \texttt{REC-BOUND}(k^* + k) < u'.\mathit{now}$, it follows that h receives p prior to u' in α. Lemma 4.21 implies that $\mathit{id}(p) \in u'[\text{SRM-REC}_h].\mathit{delivered}(s_p)$. ❒

Lemma 4.31 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of SRM_I in $\mathit{timely\text{-}aexecs}(\text{SRM}_I)$ that contains the transmission of p. Let (w, π, w'), for $w, w' \in \mathit{states}(\text{SRM}_I)$, $s_p = \mathit{source}(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of SRM_I involving the transmission of p using the reliable multicast service. Let $h \in H$, u, u' be any states of SRM_I in α, such that $w'.\mathit{now} + \overline{d} < u.\mathit{now}$ and $u \leq_\alpha u'$, and $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. If $\mathit{id}(p) \in u'[\text{SRM-REC}_h].\mathit{expected}(s_p)$, then $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions.*

Proof: Suppose that $\mathit{id}(p) \in u'[\text{SRM-REC}_h].\mathit{expected}(s_p)$. Let $\langle s_p, i' \rangle \in u'[\text{SRM-REC}_h].\mathit{expected}(s_p)$ be the earliest packet of s_p expected by h in u'; that is, for any $i'' \in \mathbb{N}$, such that $\langle s_p, i'' \rangle \in u'[\text{SRM-REC}_h].\mathit{expected}(s_p)$, it is the case that $i' \leq i''$. Let $p' \in P_{\text{RM-CLIENT}}$ be the packet transmitted in α such that $\mathit{id}(p') = \langle s_p, i' \rangle$.

Let (v, π, v'), for $v, v' \in \mathit{states}(\text{SRM}_I)$, and $\pi = \mathit{acts}(\text{SRM}_I)$, be the latest discrete transition of SRM_I in α prior to state u' in which the variable $\text{SRM-REC}_h.\mathit{expected}(s_p)$ is set from $\emptyset$ to the value $u'[\text{SRM-REC}_h].expected(s_p)$; that is, $v[\text{SRM-REC}_h].expected(s_p) = \emptyset$ and $v'[\text{SRM-REC}_h].expected(s_p) = u'[\text{SRM-REC}_h].expected(s_p)$ and there is no discrete transition following (v, π, v') and preceding u' in α that sets the variable $\text{SRM-REC}_h.expected(s_p)$ to a set other than $\emptyset$.

Since the actions of SRM_I may either reinitialize the variable $\text{SRM-REC}_h.expected(s_p)$, or set its value from the value $\emptyset$ to a set other than $\emptyset$, it follows that the timed execution fragment $\alpha_{v'u'}$ of α leading from v' to u' contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions.

Moreover, since the time passage action does not affect the variable $\text{SRM-REC}_h.expected(s_p)$, it follows that the action π leading from v to v' is not a time passage action. Thus, it is the case that $v.now = v'.now$. Since only either the original transmission of p' or the reception of the original transmission of p' may result in setting $\text{SRM-REC}_h.expected(s_p)$ to $u[\text{SRM-REC}_h].expected(s_p)$, Constraint 4.1 implies that $v.now = v'.now \leq \alpha.trans\text{-}time(p') + \overline{d}$. Moreover, Lemma 3.3 and Theorem 4.12 imply that $\alpha.trans\text{-}time(p') \leq \alpha.trans\text{-}time(p)$. Since $v.now = v'.now \leq \alpha.trans\text{-}time(p') + \overline{d}$, $\alpha.trans\text{-}time(p') \leq \alpha.trans\text{-}time(p)$, and $\alpha.trans\text{-}time(p) + \overline{d} < u.now$, it follows that $v'.now < u.now$. Thus, it follows that $v' <_\alpha u$.

Since $v' <_\alpha u$, $u \leq_\alpha u'$, and $\alpha_{v'u'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, it follows that $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. ❒

Lemma 4.32 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of SRM_I in $timely\text{-}aexecs(\text{SRM}_I)$ that contains the transmission of p. Let (w, π, w'), for $w, w' \in states(\text{SRM}_I)$, $s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of SRM_I in α involving the transmission of p using the reliable multicast service. Let $h \in H$, u, u' be any states of SRM_I in α, such that $w'.now + \overline{d} < u.now$ and $u \leq_\alpha u'$. If $id(p) \in u'[\text{SRM-REC}_h].expected(s_p)$, then it is the case that $id(p) \in u[\text{SRM-REC}_h].expected(s_p)$.*

Proof: Suppose that $id(p) \in u'[\text{SRM-REC}_h].expected(s_p)$. Lemma 4.31 implies that the timed execution fragment $\alpha_{uu'}$ of α leading from u to u' contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. Thus, Lemma 4.19 implies that $u[\text{SRM-REC}_h].expected(s_p) = u'[\text{SRM-REC}_h].expected(s_p)$. Since $id(p) \in u'[\text{SRM-REC}_h].expected(s_p)$, it follows that $id(p) \in u[\text{SRM-REC}_h].expected(s_p)$. ❒

Lemma 4.33 *Let $k \in \mathbb{N}^+$, $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$, $p \in P_{\text{RM-CLIENT}}$, and α be any admissible timed execution of SRM_I in $\Delta_L\text{-}src\text{-}recoverable\text{-}aexecs_k(\text{SRM}_I)$ that contains the transmission of p. Let (w, π, w'), for $w, w' \in states(\text{SRM}_I)$, $s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of SRM_I in α involving the transmission of p using the reliable multicast service. For any state w'' of SRM_I in α, such that $w'.now + \Delta_L < w''.now$, let $\alpha_{w'w''}$ be the timed execution fragment of α leading from w' to w''. If $h \in w''.intended(p)$, then it is the case that $h \in w''.completed(p)$.*

Proof: Suppose that $h \in w''.intended(p)$. Since $h \in w''.intended(p)$, it follows that $id(p) \in w''[\text{SRM-REC}_h].expected(s_p)$.

Let $u \in states(\text{SRM}_I)$ be the earliest state in α, such that $w'.now + \overline{d} < u.now$. Let $(u_-, \nu(t), u)$, for $u_- \in states(\text{SRM}_I)$ and $t \in \mathbb{R}^{\geq 0}, t > 0$, be the discrete transition in α leading to the particular occurrence of u in α. Since u is the earliest state in α, such that $w'.now + \overline{d} < u.now$, it follows that $u_-.now \leq w'.now + \overline{d}$. Let $u' \in states(\text{SRM}_I)$ be the earliest state in α, such that $w'.now + \texttt{DET-BOUND} < u'.now$. Let $(u'_-, \nu(t'), u')$, for $u'_- \in states(\text{SRM}_I)$ and $t' \in \mathbb{R}^{\geq 0}, t' > 0$, be the discrete transition in α leading to the particular occurrence of u' in α. Since u' is the earliest state in α, such that $w'.now + \texttt{DET-BOUND} < u'.now$, it follows that $u'_-.now \leq w'.now + \texttt{DET-BOUND}$. Since $\overline{d} \leq \texttt{DET-BOUND}$, it follows that $u \leq_\alpha u'$.

Since $\overline{d} \leq$ DET-BOUND and $id(p) \in w''[\text{SRM-REC}_h].\mathit{expected}(s_p)$, Lemma 4.32 implies that $id(p) \in u'[\text{SRM-REC}_h].\mathit{expected}(s_p)$. Thus, Constraint 4.4 implies that either $id(p) \in u'[\text{SRM-REC}_h].\mathit{archived\text{-}pkts?}(s_p)$ or $id(p) \in u'[\text{SRM-REC}_h].\mathit{scheduled\text{-}rqsts?}$.

First, consider the case where $id(p) \in u'[\text{SRM-REC}_h].\mathit{archived\text{-}pkts?}(s_p)$. Since $id(p) \in u'[\text{SRM-REC}_h].\mathit{expected}(s_p)$, $id(p) \in u'[\text{SRM-REC}_h].\mathit{archived\text{-}pkts?}(s_p)$, and the $\nu(t')$ action affects neither $\text{SRM-REC}_h.\mathit{expected}(s_p)$ nor $\text{SRM-REC}_h.\mathit{archived\text{-}pkts?}(s_p)$, it follows that $id(p) \in u'_{-}[\text{SRM-REC}_h].\mathit{expected}(s_p)$ and $id(p) \in u'_{-}[\text{SRM-REC}_h].\mathit{archived\text{-}pkts?}(s_p)$. Since $id(p) \in u'_{-}[\text{SRM-REC}_h].\mathit{expected}(s_p)$ and, consequently, $u'_{-}[\text{SRM-REC}_h].\mathit{expected}(s_p) \neq \emptyset$, Invariant 4.2 implies that $u'_{-}[\text{SRM-REC}_h].\mathit{status} = \texttt{member}$. Thus, Invariant 4.4 implies that $u'_{-}[\text{SRM-REC}_h].\mathit{delivered}(s_p) \cup u'_{-}[\text{SRM-REC}_h].\mathit{to\text{-}be\text{-}delivered?}(s_p) = u'_{-}[\text{SRM-REC}_h].\mathit{archived\text{-}pkts?}(s_p)$. However, from the precondition of $\nu(t')$, it follows that $u'_{-}[\text{SRM-REC}_h].\mathit{to\text{-}be\text{-}delivered?}(s_p) = \emptyset$. Since $id(p) \in u'_{-}[\text{SRM-REC}_h].\mathit{archived\text{-}pkts?}(s_p)$, $u'_{-}[\text{SRM-REC}_h].\mathit{delivered}(s_p) \cup u'_{-}[\text{SRM-REC}_h].\mathit{to\text{-}be\text{-}delivered?}(s_p) = u'_{-}[\text{SRM-REC}_h].\mathit{archived\text{-}pkts?}(s_p)$, and $u'_{-}[\text{SRM-REC}_h].\mathit{to\text{-}be\text{-}delivered?}(s_p) = \emptyset$, it follows that $id(p) \in u'_{-}[\text{SRM-REC}_h].\mathit{delivered}(s_p)$. Since $\nu(t')$ does not affect $\text{SRM-REC}_h.\mathit{delivered}(s_p)$, it follows that $id(p) \in u'[\text{SRM-REC}_h].\mathit{delivered}(s_p)$.

Moreover, Lemma 4.31 implies that the timed execution fragment $\alpha_{u'w''}$ of α leading from u' to w'' contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. Thus, Lemma 4.21 implies that $id(p) \in w''[\text{SRM-REC}_h].\mathit{delivered}(s_p)$; that is, $h \in w''.\mathit{completed}(p)$.

Second, consider the case where $id(p) \in u'[\text{SRM-REC}_h].\mathit{scheduled\text{-}rqsts?}$. Let $v \in \mathit{states}(\text{SRM}_I)$ be the earliest state in α, such that $v \leq_\alpha u'$ and for any state $v' \in \mathit{states}(\text{SRM}_I)$ in α, such that $v \leq_\alpha v' \leq_\alpha u'$, it is the case that $id(p) \in v'[\text{SRM-REC}_h].\mathit{scheduled\text{-}rqsts?}$.

Since the discrete transition leading from u'_{-} to u' is a time passage action, which does not affect the variable $\text{SRM-REC}_h.\mathit{scheduled\text{-}rqsts?}$, it follows that $id(p) \in u'_{-}[\text{SRM-REC}_h].\mathit{scheduled\text{-}rqsts?}$. Thus, it follows that $v \leq_\alpha u'_{-}$. Since $u'_{-}.\mathit{now} \leq w'.\mathit{now} + \texttt{DET-BOUND}$, it follows that $v.\mathit{now} \leq w'.\mathit{now} + \texttt{DET-BOUND}$.

Since for any state $v' \in \mathit{states}(\text{SRM}_I)$ in α, such that $v \leq_\alpha v' \leq_\alpha u'$, it is the case that $id(p) \in v'[\text{SRM-REC}_h].\mathit{scheduled\text{-}rqsts?}$ and the $\texttt{crash}_h$ and $\texttt{rm-leave}_h$ actions reinitialize the variable $\text{SRM-REC}_h.\mathit{scheduled\text{-}rqsts?}$, it follows that the timed execution fragment $\alpha_{vu'}$ of α leading from v to u' contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions.

Thus, since neither $\alpha_{vu'}$ nor $\alpha_{u'w''}$ contain either $\texttt{crash}_h$ or $\texttt{rm-leave}_h$ actions, it follows that $\alpha_{vw''}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions.

Let $u'' \in \mathit{states}(\text{SRM}_I)$ be the earliest state in α, such that $w'.\mathit{now} + \Delta_L < u''.\mathit{now}$. Let $(u''_{-}, \nu(t''), u'')$, for $u''_{-} \in \mathit{states}(\text{SRM}_I)$ and $t'' \in \mathbb{R}^{\geq 0}, t'' > 0$, be the discrete transition in α leading to the particular occurrence of u'' in α. Since u'' is the earliest state in α, such that $w'.\mathit{now} + \Delta_L < u''.\mathit{now}$, it follows that $u''_{-}.\mathit{now} \leq w'.\mathit{now} + \Delta_L$. Since $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$ and $\texttt{REC-BOUND}(k^* + k) \geq 0$, it follows that $u \leq_\alpha u' \leq_\alpha u''$.

Since $\alpha \in \Delta_L\text{-}\mathit{src\text{-}recoverable\text{-}aexecs}_k(\text{SRM}_I)$, Constraint 4.7 implies that the timed execution fragment $\alpha_{wu''}$ of α leading from w to u'' contains neither $\texttt{crash}_{s_p}$ nor $\texttt{rm-leave}_{s_p}$ actions.

Moreover, the precondition of $\texttt{rm-send}_{s_p}(p)$ implies that $id(p) \notin w[\text{SRM}].\mathit{sent\text{-}pkts?}$. Thus, Invariants 4.15 and 4.26 imply that $id(p) \notin w[\text{SRM-REC}_h].\mathit{scheduled\text{-}rqsts?}$. It follows that $w <_\alpha v$. Since $w <_\alpha v$ and $\alpha_{wu''}$ contains neither $\texttt{crash}_{s_p}$ nor $\texttt{rm-leave}_{s_p}$ actions, it follows that the timed execution fragment $\alpha_{vu''}$ of α leading from v to u'' contains neither $\texttt{crash}_{s_p}$ nor $\texttt{rm-leave}_{s_p}$ actions.

Since $\alpha_{wu''}$ contains neither $\texttt{crash}_{s_p}$ nor $\texttt{rm-leave}_{s_p}$ actions, $w \leq_\alpha w'$, $id(p) \in w'[\text{SRM-REC}_{s_p}].\mathit{delivered}(s_p)$, and $w' \leq_\alpha v \leq_\alpha w''$, Lemma 4.21 implies that $id(p) \in v[\text{SRM-REC}_{s_p}].\mathit{delivered}(s_p)$.

Since $v.now \leq w'.now + \texttt{DET-BOUND}$ and $w'.now + \Delta_L < u''.now$, it follows that $v.now + \texttt{REC-BOUND}(k^*+k) < u''.now$. Thus, Lemma 4.30 implies that $id(p) \in u''[\text{SRM-REC}_h].delivered(s_p)$.

Since u'' is the earliest state in α, such that $w'.now + \Delta_L < u''.now$, it follows that $u'' \leq_\alpha w''$. Since $v \leq_\alpha u'$, $u \leq_\alpha u' \leq_\alpha u''$, $u'' \leq_\alpha w''$, and $\alpha_{vw''}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, it follows that the timed execution fragment $\alpha_{u''w''}$ of α leading from u'' to w'' contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. Since $id(p) \in u''[\text{SRM-REC}_h].delivered(s_p)$ and $\alpha_{u''w''}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, Lemma 4.21 implies that $id(p) \in w''[\text{SRM-REC}_h].delivered(s_p)$; that is, $h \in w''.completed(p)$. ❐

Lemma 4.34 *Let $k \in \mathbb{N}^+$, $\Delta_R = \texttt{REC-BOUND}(k^* + k)$, $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$, $p \in P_{\text{RM-CLIENT}}$, and α be any admissible timed execution of SRM_I in Δ_R-recoverable-aexecs$_k(\text{SRM}_I)$ that contains the transmission of p. Let (w, π, w'), for $w, w' \in states(\text{SRM}_I)$, $s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of SRM_I in α involving the transmission of p using the reliable multicast service. For any state w'' of SRM_I in α, such that $w'.now + \Delta_L < w''.now$, let $\alpha_{w'w''}$ be the timed execution fragment of α leading from w' to w''. If $h \in w''.intended(p)$, then it is the case that $h \in w''.completed(p)$.*

Proof: Suppose that $h \in w''.intended(p)$. Since $h \in w''.intended(p)$, it follows that $id(p) \in w''[\text{SRM-REC}_h].expected(s_p)$.

Let $u \in states(\text{SRM}_I)$ be the earliest state in α, such that $w'.now + \overline{d} < u.now$. Let $(u_-, \nu(t), u)$, for $u_- \in states(\text{SRM}_I)$ and $t \in \mathbb{R}^{\geq 0}, t > 0$, be the discrete transition in α leading to the particular occurrence of u in α. Since u is the earliest state in α, such that $w'.now + \overline{d} < u.now$, it follows that $u_-.now \leq w'.now + \overline{d}$. Let $u' \in states(\text{SRM}_I)$ be the earliest state in α, such that $w'.now + \texttt{DET-BOUND} < u'.now$. Let $(u'_-, \nu(t'), u')$, for $u'_- \in states(\text{SRM}_I)$ and $t' \in \mathbb{R}^{\geq 0}, t' > 0$, be the discrete transition in α leading to the particular occurrence of u' in α. Since u' is the earliest state in α, such that $w'.now + \texttt{DET-BOUND} < u'.now$, it follows that $u'_-.now \leq w'.now + \texttt{DET-BOUND}$. Moreover, since $\overline{d} \leq \texttt{DET-BOUND}$, it follows that $u \leq_\alpha u'$.

Since $\overline{d} \leq \texttt{DET-BOUND}$ and $id(p) \in w''[\text{SRM-REC}_h].expected(s_p)$, Lemma 4.32 implies that $id(p) \in u'[\text{SRM-REC}_h].expected(s_p)$. Thus, Constraint 4.4 implies that either $id(p) \in u'[\text{SRM-REC}_h].archived\text{-}pkts?(s_p)$ or $id(p) \in u'[\text{SRM-REC}_h].scheduled\text{-}rqsts?$.

First, consider the case where $id(p) \in u'[\text{SRM-REC}_h].archived\text{-}pkts?(s_p)$. Since $id(p) \in u'[\text{SRM-REC}_h].expected(s_p)$, $id(p) \in u'[\text{SRM-REC}_h].archived\text{-}pkts?(s_p)$, and the $\nu(t')$ action affects neither $\text{SRM-REC}_h.expected(s_p)$ nor $\text{SRM-REC}_h.archived\text{-}pkts?(s_p)$, it follows that $id(p) \in u'_-[\text{SRM-REC}_h].expected(s_p)$ and $id(p) \in u'_-[\text{SRM-REC}_h].archived\text{-}pkts?(s_p)$. Since $id(p) \in u'_-[\text{SRM-REC}_h].expected(s_p)$ and, consequently, $u'_-[\text{SRM-REC}_h].expected(s_p) \neq \emptyset$, Invariant 4.2 implies that $u'_-[\text{SRM-REC}_h].status = \texttt{member}$. Thus, Invariant 4.4 implies that $u'_-[\text{SRM-REC}_h].delivered(s_p) \cup u'_-[\text{SRM-REC}_h].to\text{-}be\text{-}delivered?(s_p) = u'_-[\text{SRM-REC}_h].archived\text{-}pkts?(s_p)$. However, from the precondition of $\nu(t')$, it follows that $u'_-[\text{SRM-REC}_h].to\text{-}be\text{-}delivered?(s_p) = \emptyset$. Since $id(p) \in u'_-[\text{SRM-REC}_h].archived\text{-}pkts?(s_p)$, $u'_-[\text{SRM-REC}_h].delivered(s_p) \cup u'_-[\text{SRM-REC}_h].to\text{-}be\text{-}delivered?(s_p) = u'_-[\text{SRM-REC}_h].archived\text{-}pkts?(s_p)$, and $u'_-[\text{SRM-REC}_h].to\text{-}be\text{-}delivered?(s_p) = \emptyset$, it follows that $id(p) \in u'_-[\text{SRM-REC}_h].delivered(s_p)$. Since $\nu(t')$ does not affect $\text{SRM-REC}_h.delivered(s_p)$, it follows that $id(p) \in u'[\text{SRM-REC}_h].delivered(s_p)$.

Moreover, Lemma 4.31 implies that the timed execution fragment $\alpha_{u'w''}$ of α leading from u' to w'' contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. Thus, Lemma 4.21 implies that $id(p) \in w''[\text{SRM-REC}_h].delivered(s_p)$; that is, $h \in w''.completed(p)$.

Second, consider the case where $id(p) \in u'[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$. Let $v \in \textit{states}(\text{SRM}_I)$ be the earliest state in α, such that $v \leq_\alpha u'$ and for any state $v'' \in \textit{states}(\text{SRM}_I)$ in α, such that $v \leq_\alpha v'' \leq_\alpha u'$, it is the case that $id(p) \in v''[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$. The precondition of $\texttt{rm-send}_{s_p}(p)$ implies that $id(p) \notin w[\text{SRM}].\textit{sent-pkts?}$. Thus, Invariants 4.15 and 4.26 imply that $id(p) \notin w[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$. It follows that $w <_\alpha v$.

Let (v_-, π, v), for $v_- \in \textit{states}(\text{SRM}_I)$ and $\pi \in \textit{acts}(\text{SRM}_I)$, be the discrete transition in α leading to the particular occurrence of v in α. By the definition of v, it follows that $id(p) \notin v_-[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$.

Since the discrete transition leading from u'_- to u' is a time passage action, which does not affect the variable $\text{SRM-REC}_h.\textit{scheduled-rqsts?}$, it follows that $id(p) \in u'_-[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$. Thus, it follows that $v \leq_\alpha u'_-$. Since $u'_-.\textit{now} \leq w'.\textit{now} + \texttt{DET-BOUND}$, it follows that $v.\textit{now} \leq w'.\textit{now} + \texttt{DET-BOUND}$. Since $v.\textit{now} \leq w'.\textit{now} + \texttt{DET-BOUND}$ and $w'.\textit{now} + \Delta_L < w''.\textit{now}$, it follows that $v.\textit{now} + \Delta_R < w''.\textit{now}$. Let $v' \in \textit{states}(\text{SRM}_I)$ be the earliest state in α, such that $v.\textit{now} + \Delta_R < v'.\textit{now}$. By the definition of v', it is the case that $v' \leq_\alpha w''$.

Since for any state $v'' \in \textit{states}(\text{SRM}_I)$ in α, such that $v \leq_\alpha v'' \leq_\alpha u'$, it is the case that $id(p) \in v''[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$ and the $\texttt{crash}_h$ and $\texttt{rm-leave}_h$ actions reinitialize the variable $\text{SRM-REC}_h.\textit{scheduled-rqsts?}$, it follows that the timed execution fragment $\alpha_{vu'}$ of α leading from v to u' contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. Since neither $\alpha_{vu'}$ nor $\alpha_{u'w''}$ contain either $\texttt{crash}_h$ or $\texttt{rm-leave}_h$ actions, it follows that $\alpha_{vw''}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. Thus, since $v' \leq_\alpha w''$ and $\alpha_{vw''}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, the timed execution fragment $\alpha_{vv'}$ of α leading from v to v' contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions.

Since $id(p) \notin v_-[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$, $id(p) \in v[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$, and v' is the earliest state in α, such that $v.\textit{now} + \Delta_R < v'.\textit{now}$, Constraint 4.8 implies that there exists $h' \in H, h' \neq h$ such that $id(p) \in v[\text{SRM-REC}_{h'}].\textit{delivered}(s_p)$ and the timed execution fragment $\alpha_{vv'}$ contains neither $\texttt{crash}_{h'}$ nor $\texttt{rm-leave}_{h'}$ actions. Let $h' \in H, h' \neq h$ be any such host.

Since $id(p) \notin v_-[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$, $id(p) \in v[\text{SRM-REC}_h].\textit{scheduled-rqsts?}$, $\alpha_{vv'}$ contains neither $\texttt{crash}_h$, $\texttt{rm-leave}_h$, $\texttt{crash}_{h'}$, nor $\texttt{rm-leave}_{h'}$ actions, $id(p) \in v[\text{SRM-REC}_{h'}].\textit{delivered}(s_p)$, $v.\textit{now} + \Delta_R < v'.\textit{now}$, Lemma 4.30 implies that $id(p) \in v'[\text{SRM-REC}_h].\textit{delivered}(s_p)$.

Since $v' \leq_\alpha w''$ and $\alpha_{vw''}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, it follows that the timed execution fragment $\alpha_{v'w''}$ of α leading from v' to w'' contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions. Since $id(p) \in v'[\text{SRM-REC}_h].\textit{delivered}(s_p)$, Lemma 4.21 implies that $id(p) \in w''[\text{SRM-REC}_h].\textit{delivered}(s_p)$; that is, $h \in w''.\textit{completed}(p)$. ❐

4.4.6 Static Timeliness Analysis

In this section, we show that when hosts neither crash nor leave the reliable multicast group and the number of packet drops pertaining to the transmission and, potentially, the recovery of any packet is bounded, SRM_I implements $\text{RM}_S(\Delta_L)$, for a particular $\Delta_L \in \mathbb{R}^{\geq 0}$. In particular, we show that any timed trace of SRM_I in the set $\textit{recoverable-attraces}_k(\text{SRM}_I)$, for some $k \in \mathbb{N}$, is also a timed trace of the specification automaton $\text{RM}_S(\Delta_L)$, for $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$. Thus, given Constraints 4.1, 4.2, 4.3, 4.4, 4.5, and 4.6 and assuming that the number of packet drops pertaining to the transmission and, potentially, the recovery of any packet is bounded by k, SRM_I implements the timely reliable multicast service specification $\text{RM}_S(\Delta_L)$.

The proof of this claim involves showing that the relation R of Definition 4.1 is a timed forward simulation relation from SRM_I to $\text{RM}_S(\Delta_L)$, under the aforementioned constraints and assumptions. The key part of the proof involves showing the correspondence of the time-passage steps. In particular, we show that active packets are delivered to all the hosts is their intended delivery sets within Δ_L time units.

Theorem 4.35 *Let $k \in \mathbb{N}^+$ and $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$. Then, it is the case that recoverable-attraces$_k$(SRM$_I$) $\subseteq$ attraces(RM$_S(\Delta_L)$).*

Proof: It suffices to show that the relation R of Definition 4.1 is a timed forward simulation relation from SRM$_I$ to RM$_S(\Delta_L)$, for any execution in the set *recoverable-attraces*$_k$(SRM$_I$).

The proof that R is indeed a timed forward simulation relation is identical to that of Lemma 4.11 with the exception that in this case showing the correspondence of the time passage transitions is nontrivial.

Consider any discrete transition $(u, \pi, u') \in \mathit{trans}(\mathrm{SRM}_I)$, where $\pi = \nu(t)$, for some $t \in \mathbb{R}^{\geq 0}$, that occurs in any admissible execution of SRM$_I$ in the set *recoverable-attraces*$_k$(SRM$_I$). It suffices to show that, for any reachable state s of RM$_S(\Delta_L)$ such that $(u, s) \in R$, there exists a timed execution fragment α of RM$_S(\Delta_L)$ such that $\alpha.\mathit{fstate} = s$, $\alpha.\mathit{lstate} = s'$, $\mathit{ttrace}(\alpha) = \mathit{ttrace}(u\pi u')$, the total amount of time-passage in α is the same as the total amount of time-passage in $u\pi u'$, and $(u', s') \in R$.

Let s be any reachable state of RM$_S(\Delta_L)$ such that $(u, s) \in R$. The timed execution fragment of RM$_S(\Delta_L)$ corresponding to the step (u, π, u') is comprised solely of the $\nu(t)$ action. We must show that the $\nu(t)$ action is enabled in s; that is, we must show that, for any active packet $p \in s.\mathit{active\text{-}pkts}$, it is the case that either $s.\mathit{now} + t \leq s.\mathit{trans\text{-}time}(p) + \Delta_L$ or $s.\mathit{intended}(p) \subseteq s.\mathit{completed}(p)$. Since $(u, s) \in R$, it suffices to show that, for any active packet $p \in u.\mathit{active\text{-}pkts}$, it is the case that either $u.\mathit{now} + t \leq u.\mathit{trans\text{-}time}(p) + \Delta_L$ or $u.\mathit{intended}(p) \subseteq u.\mathit{completed}(p)$.

Consider any active packet $p \in u.\mathit{active\text{-}pkts}$. It suffices to show that if $u.\mathit{trans\text{-}time}(p) + \Delta_L < u.\mathit{now} + t$, then $u.\mathit{intended}(p) \subseteq u.\mathit{completed}(p)$. Suppose that $u.\mathit{trans\text{-}time}(p) + \Delta_L < u.\mathit{now} + t$. It suffices to show that for any host $h \in u.\mathit{intended}(p)$, it is the case that $h \in u.\mathit{completed}(p)$.

Let $h \in H$ be any host in $u.\mathit{intended}(p)$. Since the action $\nu(t)$ of SRM$_I$ does not affect the derived history variable $\mathit{intended}(p)$, it follows that $h \in u'.\mathit{intended}(p)$. Moreover, since $u.\mathit{trans\text{-}time}(p) + \Delta_L < u.\mathit{now} + t$ and the action $\nu(t)$ increments the *now* variable by t time units, it follows that $u.\mathit{trans\text{-}time}(p) + \Delta_L < u'.\mathit{now}$. Since $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$, $u.\mathit{trans\text{-}time}(p) + \Delta_L < u'.\mathit{now}$, and $h \in u'.\mathit{intended}(p)$, Lemmas 4.18, 4.33, and 4.13 imply that $h \in u'.\mathit{completed}(p)$. Since the action $\nu(t)$ of SRM$_I$ does not affect the derived history variable SRM.$\mathit{completed}(p)$, it follows that $h \in u.\mathit{completed}(p)$. ❒

4.4.7 Dynamic Timeliness Analysis

We begin this section by showing that when sources remain members of the reliable multicast group for an amount of time $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$ past the transmission of any packet they send using the reliable multicast group, SRM$_I$ implements RM$_S(\Delta_L)$. In particular, we show that any timed trace of SRM$_I$ in the set Δ_L-*recoverable-attraces*$_k$(SRM$_I$), for $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$ and some $k \in \mathbb{N}$, is also a timed trace of the specification automaton RM$_S(\Delta_L)$. Thus, given Constraints 4.1, 4.2, 4.3, 4.4, and 4.7 and assuming that the number of packet drops pertaining to the transmission and, potentially, the recovery of any packet is bounded by k, SRM$_I$ implements the timely reliable multicast service specification RM$_S(\Delta_L)$. The proof of this claim is analogous to the proof of Theorem 4.35.

Theorem 4.36 *Let $k \in \mathbb{N}^+$ and $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$. Then, it is the case that Δ_L-src-recoverable-attraces$_k$(SRM$_I$) $\subseteq$ attraces(RM$_S(\Delta_L)$).*

Proof: The proof is analogous to that of Theorem 4.35. ❒

We strengthen the above result by weakening our assumption that sources neither crashing nor leaving the reliable multicast group. In particular, we show that SRM_I implements $\mathrm{RM}_S(\Delta_L)$, for $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$, if whenever a host $h \in H$ detects the loss of any packet $p \in P_{\text{RM-CLIENT}}$, there exists a host $h' \in H, h' \neq h$ that has already delivered p and remains a member of the reliable multicast group for at least $\Delta_R = \texttt{REC-BOUND}(k^* + k)$ time units. That is, given Constraints 4.1, 4.2, 4.3, 4.4, and 4.8 and assuming that the number of packet drops pertaining to the transmission and, potentially, the recovery of any packet is bounded by k, SRM_I implements the timely reliable multicast service specification $\mathrm{RM}_S(\Delta_L)$. The proof of this claim is analogous to the proof of Theorem 4.35.

Theorem 4.37 *Let $k \in \mathbb{N}^+$, $\Delta_R = \texttt{REC-BOUND}(k^* + k)$, and $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$. Then, it is the case that Δ_R-recoverable-attraces$_k(\mathrm{SRM}_I) \subseteq$ attraces$(\mathrm{RM}_S(\Delta_L))$.*

Proof: The proof is analogous to that of Theorem 4.35. ❒

Chapter 5

Packet Loss Locality

In this chapter, we make the case for exploiting packet loss locality in the loss recovery of reliable multicast protocols, such as SRM [13]. We claim that packet loss locality in IP multicast transmissions can be exploited by simple caching schemes. In such schemes, receivers cache information about the recovery of recently recovered packets and use this information to expedite the recovery of subsequent losses. We present a methodology for estimating the potential effectiveness of caching within multicast loss recovery. By applying this methodology to the IP multicast transmission traces of Yajnik *et al.* [41] and observing that IP multicast losses exhibit substantial locality, we establish that caching can be very effective.

5.1 Introduction

Recently, numerous retransmission-based reliable multicast protocols have been proposed [13, 16, 19, 20, 34, 35]. The challenge in designing such protocols lies in the requirements to scale to large multicast groups, to cater to a dynamic membership and network, and to minimize the recovery overhead. Most retransmission-based reliable multicast protocols treat losses independently and blindly repeat the recovery process for each loss. We propose the extension of reliable multicast protocols with caching schemes in which receivers cache information about the recovery of recently recovered packets and use this information to expedite the recovery of subsequent losses. Such schemes have the potential of substantially reduce recovery latency and overhead, in particular when packet losses exhibit locality — the property that losses suffered by a receiver at proximate times often occur on the same link of the IP multicast tree.

In this chapter, we present a methodology for estimating the degree to which IP multicast losses exhibit locality and quantifying the potential effectiveness of caching in multicast loss recovery. Our methodology involves evaluating the performance of a caching-based loss location estimation scheme. In this scheme, each receiver caches the locations of its most recent losses whose locations it has identified and estimates that its next loss occurs at the location that appears most frequently in this cache. We consider a loss location estimate to be a *hit* if it matches the location of the loss. The *hit rate* achieved by each receiver is an indication of the degree to which the losses suffered by each receiver exhibit locality. A *shared hit* corresponds to the case when the loss location estimates of all receivers sharing a loss are hits; that is, all such receivers estimate the same loss location and this loss location is correct. The *shared hit rate* indicates the potential effectiveness of a caching scheme that relies on the collaboration and coordination of all receivers that share losses.

We apply our evaluation methodology to the IP multicast transmission traces of Yajnik *et al.* [41] and observe the hit rates achieved by our loss location estimation scheme as a function of: the cache size, the delay in detecting losses, the delay in identifying a loss's location, and the precision of the loss location identification. As the delays in detecting losses and in identifying their locations increase, caches become populated by the locations of less recent losses and loss location estimates are made based on less recent information. Knowledge of the IP multicast tree topology may improve the precision with which the locations of losses are identified.

Our analysis reveals that the losses in the traces of Yajnik *et al.* exhibit substantial locality. The per-receiver hit rates achieved by our loss location estimation scheme in most cases exceed 40% and often exceed 80%. The shared hit rates range from 10% to 80% when the loss location identification is topology-oblivious and from 25% to 90% when it is topology-aware. The shared hit rates for a cache of size 10 exceed 35% (70%) for half the traces when the loss location identification is topology-oblivious (respectively, topology-aware). These observations suggest that exploiting packet loss locality through caching within either existing or novel reliable multicast protocols has the potential of substantially reducing recovery latency and overhead.

Recent studies of IP multicast transmission losses [1,15,41,42] have investigated whether losses in the multicast setting exhibit *temporal* and *spatial correlation*. Temporal correlation refers to the degree to which losses are bursty and spatial correlation refers to degree to which losses are pairwise shared between receivers. All such studies observe that although packet losses are clearly not independent, they exhibit low temporal and spatial correlation. Our observations do not contradict these results. Loosely speaking, these studies examine whether the *loss* of *consecutive (or, close-by) packets* is correlated whereas we examine whether the *location* of *consecutive (or, close-by) losses* is correlated. Notably, packet loss locality can be exploited in multicast loss recovery.

This chapter is organized as follows. Section 5.2 illustrates how caching can be incorporated within SRM in order to exploit locality. In Section 5.3, we present the IP multicast transmission trace data that we use in this chapter and describe how we interpret and represent it. Section 5.5 presents our analysis of locality and the effectiveness of caching in multicast loss recovery. Section 5.6 concludes the chapter and suggests future work directions.

5.2 Exploiting Locality Through Caching

In this section, we illustrate how caching can be used to exploit packet loss locality within the Scalable Reliable Multicast (SRM) protocol [13].

Packet recovery in SRM is initiated when a receiver detects a loss and schedules a retransmission *request* to be multicast in the near future. If the packet is received prior to the transmission of the scheduled request, then the scheduled request is canceled. If a request for the packet is received prior to the transmission of the scheduled request, then the scheduled request is postponed (suppressed and rescheduled). Upon receiving a request for a packet that has been received, a receiver schedules a retransmission of the requested packet (*reply*). If a reply for the same packet is received prior to the transmission of the scheduled reply, then the scheduled reply is canceled (suppressed). All requests and replies are multicast. SRM minimizes duplicate requests and replies using suppression. Unfortunately, suppression techniques delay the transmission of requests and replies so that only few (and, optimally, single) requests and replies are transmitted for each loss.

We propose enhancing SRM with a caching-based expedited recovery scheme [24]. This scheme operates roughly as follows. Each receiver caches the requestor and replier of the most recently recovered packet. A receiver considers itself to be optimal when its cached requestor is itself. Upon detecting losses, in addition to scheduling requests as is done in SRM, optimal receivers immediately unicast requests to their cached repliers. Upon receiving such a request, a receiver immediately multicasts a reply for the requested packet. A cache hit corresponds to the case when the unicast request is sent to a receiver that is capable of retransmitting the packet. Since unicast requests and the resulting retransmissions are not delayed for purposes of suppression, the recovery resulting from a hit incurs minimum latency. Moreover, it may suppress any requests and replies scheduled by SRM's recovery scheme, thus limiting the recovery overhead to one unicast request and one multicast reply. In the case of a miss, the recovery of a packet is carried out as prescribed by SRM's recovery scheme. The overhead associated with a miss is a single unicast request.

The above simple caching-based expedited recovery scheme associates loss locations with the requestor-replier pairs that recover the respective packets. This scheme may turn out to be too crude, in the sense that many requestor-replier pairs get associated with particular loss locations. To obtain more precise loss location identification, we propose employing a router-assisted scheme where routers annotate packets so that *turning point* routers [19, 34] are exposed. Turning points identify the subtrees of the IP multicast tree that are affected by each loss; thus, they identify loss locations precisely. This information can be used to associate sets of requestor-replier pairs to particular locations; thus, improving the effectiveness of caching.

SRM is highly resilient to group membership and network topology changes. Unfortunately, such resilience comes at the expense of performance. In static environments, other protocols [7, 16, 19, 34, 35] may outperform SRM by either *a priori* choosing designated repliers, arranging receivers in hierarchies, or extending the functionality of IP multicast routers so as to intelligently forward recovery packets. Our proposed caching-based expedited recovery scheme can substantially improve SRM's performance when the group membership and the network topology are static. Moreover, it may partially bridge the performance gap between SRM and hierarchical or router-assisted schemes, while still retaining SRM's resilience to dynamic environments.

Of course, many variations on the above caching scheme may be considered: caching several of the most recent requestor-replier pairs and choosing to recover from the most frequent such pair, multicasting the expedited request, *etc.* Moreover, similar caching schemes may benefit either other existing or novel reliable multicast protocols in similar ways.

5.3 IP Multicast Traces and Their Representation

We represent IP multicast traces by per-receiver sequences each of which indicates the locations at which the losses suffered by the particular receiver occur. We consider two such representations. The first representation is oblivious to the underlying IP multicast tree topology and identifies the location of each loss with the set of receivers that share the loss of the particular packet. The second representation takes into consideration the underlying IP multicast tree topology and identifies the location of each loss with an estimate of the actual link of the IP multicast tree that is responsible for the loss. We begin this section by describing the IP multicast trace data that we use throughout this chapter. We then describe how we interpret the trace data and produce our two trace representations.

5.3.1 Trace Data

We use 14 IP multicast transmission traces of Yajnik *et al.* [41]. These traces involve single-source IP multicast transmissions each originating in the *World Radio Network* (WRN), the *UC Berkeley Multimedia Seminar* (UCB), or the *Radio Free Vat* (RFV). In these IP multicast transmissions, packets are transmitted from the source at a constant rate. These packets are disseminated along an IP multicast tree to a subset of 17 research community hosts spread out throughout the US and Europe. These hosts constitute the receivers of the IP multicast transmission.

Table 5.1 IP Multicast Traces of Yajnik *et al.* [41].

	Source & Date	# of Rcvrs	Tree Depth	Period *(msec)*	Duration *(hr:min:sec)*	# of Pkts	# of Losses
1	RFV960419	12	6	80	1:00:00	45001	24086
2	RFV960508	10	5	40	1:39:19	148970	55987
3	UCB960424	15	7	40	1:02:29	93734	33506
4	WRN950919	8	4	80	0:23:31	17637	10276
5	WRN951030	10	4	80	1:16:02	57030	15879
6	WRN951101	9	5	80	0:55:40	41751	18911
7	WRN951113	12	5	80	1:01:55	46443	29686
8	WRN951114	10	4	80	0:51:23	38539	11803
9	WRN951128	9	4	80	0:59:56	44956	33040
10	WRN951204	11	5	80	1:00:32	45404	16814
11	WRN951211	11	4	80	1:36:42	72519	44649
12	WRN951214	7	4	80	0:51:38	38724	20872
13	WRN951216	8	3	80	1:06:56	50202	37833
14	WRN951218	8	3	80	1:33:20	69994	43578

The data collected from each of the IP multicast transmissions involves per-receiver sequences each of which indicates which packets were received and the order in which they were received by the respective receiver. These per-receiver sequences do not include the packet reception times. Yajnik *et al.* also provide the IP multicast tree topology for each of the IP multicast transmissions. This topology is presumed to be static (fixed) throughout the duration of the IP multicast transmission. Table 5.1 lists the source, date, number of receivers, IP multicast tree depth, packet transmission period, transmission duration, number of packets transmitted, and number of losses suffered for each of the 14 traces. For more information regarding the traces, see [41].

Henceforth in this chapter, we focus our attention on a single generic IP multicast transmission trace. This generic trace is intended to correspond to any single IP multicast transmission trace of Yajnik *et al.* [41]. Let $k \in \mathbb{N}$ be the finite number of packets transmitted during the IP multicast transmission. Moreover, let R be the finite set of receivers of the IP multicast transmission. For $I = \{1, \ldots, k\}$ and $i \in I$, we refer to the i-th packet transmitted during the IP multicast transmission as packet i.

As is traditionally done in the literature [1, 15, 41, 42], we represent the trace data by per-receiver binary sequences of length k. For $i \in I$ and $r \in R$, the i-th element of the binary sequence pertaining to receiver r indicates whether receiver r suffered the loss of the packet i. We represent these per-receiver binary sequences as a mapping $loss : R \to (I \to \{0, 1\})$, where:

$$loss(r)(i) = \begin{cases} 1 & \text{if receiver } r \text{ suffered the loss of packet } i \text{, and} \\ 0 & \text{otherwise.} \end{cases} \tag{5.1}$$

We represent the IP multicast tree, along which the k packets of the IP multicast transmission are disseminated, as a tuple $T = \langle N, s, L \rangle$ consisting of a set of nodes N, a root node $s \in N$, and a set of directed edges $L \subseteq N \times N$. The elements N, s, and L of T are further constrained to form a directed tree rooted at s in which all edges in L are directed away from s, there is a unique simple path from s to each other node in N, and the elements of R are exactly the leaf nodes of the tree (and, consequently, $R \subseteq N$).

In terms of the physical entities involved in the IP multicast transmission, the root node s

corresponds to the source of the IP multicast transmission, the internal nodes of T correspond to the IP multicast capable routers of the network that are used to disseminate the packets transmitted by s, and the leaf nodes of T correspond to the receivers of the IP multicast transmission. The edges of T correspond to the communication links that connect the source, routers, and receivers of the IP multicast transmission. Some edges of T may also correspond to series of communication links; that is, we abstractly represent chains of communication links as single communication links. Thus, T is absent of edge chains. We henceforth often refer to the edges of T as links.

For $n \in N$, we define $N_n \subseteq N$ to be the set of nodes in the subtree of T rooted at n, with $N_s = N$ and $N_r = \{r\}$, for $r \in R$. We define $R_n \subseteq R$ to be the set of leaf nodes in the subtree of T rooted at n, with $R_s = R$ and $R_r = \{r\}$, for $r \in R$. For any $l = \langle n, n' \rangle \in L$, we define $R_l \subseteq R$ to be the set of leaf nodes $R_{n'}$, with $R_l = \{n'\}$, when $n' \in R$. Moreover, we define $L_l \subseteq L$ to be the set of edges in the subtree of T rooted at n', with $L_l = \emptyset$, when $n' \in R$. Finally, we define $\mathit{spath}(n, n') \subseteq L$, for $n \in N$ and $n' \in N_n$, to be the set of edges of T that make up the unique simple path in T leading from n to n', with $\mathit{spath}(n, n) = \emptyset$.

5.3.2 Virtual Link Trace Representation

Our first representation is oblivious to the underlying IP multicast transmission tree. We identify the location at which a packet is dropped with the set of receivers that share the loss of the given packet. We henceforth refer to set of receivers that share the loss of a particular packet as the packet's *loss pattern*.

For $i \in I$, we define the *loss pattern* of packet i, denoted $\mathit{loss\text{-}patt}(i)$, as follows:

$$\mathit{loss\text{-}patt}(i) = \{r \in R \mid \mathit{loss}(r)(i) = 1\} \tag{5.2}$$

Note that, when a packet $i \in I$ is successfully received by each receiver in R, it is the case that $\mathit{loss\text{-}patt}(i) = \emptyset$.

Although many of the loss patterns observed in the trace data result from losses on multiple links of the underlying IP multicast tree, we attribute each distinct loss pattern to a loss on a single *virtual link*, which we identify with the loss pattern itself. Henceforth, we use $\text{V-Link} = \mathcal{P}(R)$ to denote the set of all virtual links pertaining to the IP multicast tree T.

Although virtual links do not correspond to actual links of the IP multicast tree, the loss patterns corresponding to two virtual links may be used to infer whether one is conceptually downstream or upstream of the other within a supposed virtual IP multicast tree. In particular, since losses closer to the source of the IP multicast tree affect a larger number of receivers, when the loss pattern of one virtual link is a subset of the loss pattern of another, we can infer that the former virtual link is conceptually downstream of the latter.

More precisely, letting $l, l' \in \text{V-Link}$, we say that the virtual link l is *downstream* (*upstream*) of the virtual link l' when $l \subset l'$ (respectively, $l \supset l'$). We use the notation $l \prec l'$ ($l \succ l'$) to denote that l is downstream (respectively, upstream) of l'. Moreover, we say that l is either equal to or downstream of (either equal to or upstream of) l' when $l \subseteq l'$ (respectively, $l \supseteq l'$). We use the notation $l \preceq l'$ ($l \succeq l'$) to denote that the virtual link l is either equal to or downstream of (respectively, either equal to or upstream of) the virtual link l'. When l and l' are neither equal to nor upstream/downstream of each other, we say that they are *incomparable*.

Given the above definition of virtual links, we represent the IP multicast transmission trace by per-receiver sequences of length k whose elements identify the virtual links, *i.e.*, loss patterns, on which the losses suffered by respective receivers occur. In particular, for $r \in R$ and $i \in I$, if the receiver r suffered the loss of the packet i, then the i-th element of the sequence pertaining to r identifies the virtual link on which the loss of packet i occurred.

We define the *virtual link trace representation* to be the mapping $\textit{v-link} : R \to (I \to \text{V-LINK} \cup \perp)$, such that, for $r \in R$ and $i \in I$,

$$\textit{v-link}(r)(i) = \begin{cases} \textit{loss-patt}(i) & \text{if } \textit{loss}(r)(i) = 1, \text{ and} \\ \perp & \text{otherwise.} \end{cases} \tag{5.3}$$

For $r \in R$ and $i \in I$, $\textit{v-link}(r)(i) \neq \emptyset$ if and only if receiver r suffered the loss of packet i; that is, $\textit{v-link}(r)(i) \neq \emptyset$ if and only if $\textit{loss}(r)(i) = 1$.

5.3.3 Concrete Link Trace Representation

Our second representation takes into account the underlying IP multicast transmission tree. In particular, we represent the IP multicast transmission trace by per-receiver sequences of length k whose elements are estimates of the links of the IP multicast tree responsible for the loss of the respective packets by the respective receivers.

We define the set of *concrete links*, C-LINK, to be set of links L of the IP multicast tree T. As in the case of virtual links, for $l, l' \in \text{C-LINK}$, we use the notation $l \prec l'$ ($l \succ l'$) to denote that l is downstream (respectively, upstream) of l'. In the case of concrete links, the notion of downstream and upstream is dictated by the IP multicast tree. In particular, a link l' is downstream of l if $l' \in L_l$ and l' is upstream of l if $l \in L_{l'}$. We use the notation $l \preceq l'$ ($l \succeq l'$) to denote that the virtual link l is either equal to or downstream of (respectively, either equal to or upstream of) the virtual link l'. When l and l' are neither equal to nor upstream/downstream of each other, we say that they are *incomparable*.

We define the *concrete link trace representation* to be the mapping $\textit{c-link} : R \to (I \to \text{C-LINK} \cup \perp)$, such that, for $r \in R$ and $i \in L$, $\textit{c-link}(r)(i)$ is an estimate of the link responsible for the loss of packet i by receiver r, if receiver r suffered the loss of packet i, and $\textit{c-link}(r)(i) = \perp$, otherwise. The rest of this section is devoted to precisely defining the mapping *c-link*.

To begin, given the IP multicast tree T and the trace data $\textit{loss}(r)(i)$, for $r \in R$ and $i \in I$, we estimate the set of links on which each packet is dropped. Several steps are involved in this estimation. We first define a random process that models the dissemination of a packet throughout the IP multicast tree T. Then, we estimate the *link loss probability* of each link $l \in L$ of the IP multicast tree; that is, the probability with which the link l drops packets. Finally, we estimate the set of links on which each packet is dropped. In particular, we define the *link loss combination mapping* $\textit{link-comb} : I \to \mathcal{P}(L)$, such that, for $i \in I$, $\textit{link-comb}(i)$ is an estimate of the set of links of the IP multicast tree on which the packet i is dropped while being disseminated throughout T. Recalling that, for $s, r \in N$, $\textit{spath}(s, r)$ is the set of links that make up the unique simple path in T leading from s to r, we define $\textit{c-link}(r)(i)$, for $r \in R$ and $i \in I$, as follows:

$$\textit{c-link}(r)(i) = \begin{cases} l & \text{if } \{l\} = \textit{spath}(s, r) \cap \textit{link-comb}(i), \text{ and} \\ \perp & \text{if } \textit{spath}(s, r) \cap \textit{link-comb}(i) = \emptyset. \end{cases} \tag{5.4}$$

For $r \in R$ and $i \in I$, it is the case that $\textit{c-link}(r)(i) \neq \perp$ if and only if receiver r suffered the loss of the packet i; that is, $\textit{c-link}(r)(i) \neq \perp$ if and only if $\textit{loss}(r)(i) = 1$.

We proceed by describing in detail the various steps leading up to the definition of the mapping *c-link*.

IP Multicast Transmission Process

As is commonly done in the literature [2, 3, 41], we model the dissemination of a single packet throughout the IP multicast tree T as a parameterized random process $\text{MCAST}_T[\alpha]$, where the parameter α of type $L \to [0,1]$ is a mapping from links to *link loss probabilities*. For any link $l = \langle n, n' \rangle \in L$, $\alpha(l)$ is the probability with which a packet that is received by n is dropped while being forwarded from n to n' along l — conversely, $1 - \alpha(l)$ is the probability with which a packet that is received by n is successfully forwarded from n to n' along l.

The process $\text{MCAST}_T[\alpha]$ disseminates the packet from the root s of T to the leaf nodes R of T according to the link loss probability mapping α. The outcome of the random process $\text{MCAST}_T[\alpha]$ consists of the values of the random variables $X_n \in \{0,1\}$, for $n \in N$. Each random variable X_n, for $n \in N$, indicates whether the node n received the packet transmitted; that is, $X_n = 1$ indicates that n received it and $X_n = 0$ indicates that n did not receive it.

The values of the random variables X_n, for $n \in N$, are calculated based on the link loss probability mapping α as follows: $X_s = 1$ and, for all $l = \langle n, n' \rangle \in L$, it is the case that:

$$X_{n'} = \begin{cases} 0 & \text{if } X_n = 0, \\ 0 & \text{with probability } \alpha(l), \text{ if } X_n = 1, \text{ and} \\ 1 & \text{with probability } 1 - \alpha(l), \text{ if } X_n = 1. \end{cases} \tag{5.5}$$

The loss pattern $Y \subseteq R$ resulting from the transmission of the packet using $\text{MCAST}_T[\alpha]$ is, thus, the set of receivers $\{r \in R \mid X_r = 0\}$.

Estimating Link Loss Probabilities

We model the complete IP multicast transmission resulting in the given trace as k repetitions of the same random process $\text{MCAST}_T[\hat{\alpha}]$, for some particular value $\hat{\alpha}$ of the link loss probability mapping parameter α. Given the IP multicast transmission tree topology T and the observed trace data $\mathit{loss}(r)(i)$, for $r \in R$ and $i \in I$, we can estimate the link loss probability mapping $\hat{\alpha}$ either by the method of Yajnik *et al.* [41] or the maximum-likelihood estimator method of Cáceres *et al.* [2]. We briefly describe the method of Yajnik *et al.* [41]; the method of Cáceres *et al.* [2], although more accurate, is substantially more involved.

For any $n \in N$, we define k_n to be the number of packets lost by all the receivers in R_n (the receivers that are descendants of the node n in T); that is,

$$k_n = |\{i \in I \mid R_n \subseteq \mathit{loss\text{-}patt}(i)\}| \tag{5.6}$$

For $l = \langle n, n' \rangle \in L$, the link loss probability $\hat{\alpha}(l)$ is defined as:

$$\hat{\alpha}(l) = \frac{k_{n'} - k_n}{k - k_n} \tag{5.7}$$

Presuming that it is much more likely for a packet to be dropped on l rather than on all downstream paths from n', $k_{n'} - k_n$ is the number of packets dropped on l. Similarly, presuming that it is much more likely for a packet to be dropped upstream of n than on all downstream paths from n, $k - k_n$ is the number of packets that are received by and forwarded downstream of n. Thus, $\hat{\alpha}(l)$ is an estimate of the ratio between the number of packets that are dropped on l and the number of packets successfully received by and forwarded downstream of n.

It is important to note that, since we use the same random process to model the dissemination of all the packets of the IP multicast transmission, we are indeed presuming that the loss characteristics of the links of the IP multicast tree T remain the same throughout the IP multicast transmission.

The method of Yajnik *et al.* [41] is known to yield biased link loss probability estimates [2]. However, for the IP multicast traces of Yajnik *et al.* [41], both the method of Yajnik *et al.* [41] and the maximum-likelihood estimator method of Cáceres *et al.* [2] yield similar link loss probability estimates. Throughout this chapter, we use the link loss probability estimates yielded by the method of Yajnik *et al.*

Estimating Loss Locations

In this section, we use the the IP multicast transmission process $\mathrm{MCAST}_T[\hat{\alpha}]$ and the observed trace data $loss(r)(i)$, for $r \in R$ and $i \in I$, to define the mapping $\textit{link-comb} : I \to \mathcal{P}(L)$; that is, to estimate the set of links of the IP multicast tree T that are responsible for the losses suffered by each receiver of the IP multicast transmission.

To begin, let $Y(i) \subseteq R$, for $i \in I$, denote the loss pattern resulting from the i-th repetition of the IP multicast transmission process $\mathrm{MCAST}_T[\hat{\alpha}]$. The loss pattern $Y(i)$ may result from several transmission scenarios; that is, scenarios in which the packet i is successfully forwarded on some links and dropped on others. For example, the loss pattern R may result either from losses on all the links leaving the source, from losses on each of the links leading to each of the receivers, or from losses on another combination of links.

Each transmission scenario resulting in $Y(i)$ may be identified by the set of links $C(i) \subseteq L$ on which the packet i is dropped. For any link $l = \langle n, n' \rangle \in C(i)$, where $n, n' \in N$, the packet i must have been successfully forwarded on the links leading from s to n — since the packet i was dropped by l, n must have received the packet i. Moreover, since the packet i is dropped on l, it is not received by n' and, thus, it is not forwarded on any of the links that form the subtree rooted at n'. Thus, $C(i)$ uniquely identifies a particular transmission scenario, *i.e.*, a particular outcome of the IP multicast transmission process $\mathrm{MCAST}_T[\hat{\alpha}]$. Since the set of links on which a packet is dropped uniquely identifies a packet's transmission scenario, we use this set of links to identify such a scenario. We refer to a set of links that identifies a particular scenario as a *link loss combination*.

We proceed by calculating the probability with which the link loss combination $C(i)$ of the packet i is a particular link loss combination $C \in L$ given that the loss pattern $Y(i)$ of the packet i is a particular loss pattern $Y \subseteq R$.

First, we define $C_T \subseteq \mathcal{P}(L)$ to be the set of all link loss combinations that are consistent with the IP multicast transmission tree T. C_T consists of the sets of links in which no link is downstream of another link in the set; that is,

$$C_T = \{C \subseteq L \mid \forall\, l, l' \in C, l' \notin L_l\} \tag{5.8}$$

recalling that $L_l \subseteq L$, for $l \in L$, denotes the set of links that are downstream of the link l in T.

For any loss pattern $Y \subseteq R$, we define $C_T(Y)$ to be the set of link loss combinations that result in the loss pattern Y; that is,

$$C_T(Y) = \{C \in C_T \mid Y = \cup_{l \in C} R_l\} \tag{5.9}$$

recalling that $R_l \subseteq R$, for $l \in L$, denotes the set of receivers that are downstream of the link l in T.

The probability $\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[C(i) = C]$ that $C(i)$ is a particular link loss combination $C \in C_T$ is given by:

$$\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[C(i) = C] = \prod_{l \in C} \hat{\alpha}(l) \prod_{l' \in U_C} (1 - \hat{\alpha}(l')) \tag{5.10}$$

with $U_C = L \backslash (C \cup_{l \in C} L_l)$. The probability $\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[C(i) = C]$ is equal to the product of the probabilities that the i-th packet is dropped on each of the links in C and successfully forwarded on each of the links that are neither links in C nor downstream of any of the links in C.

The probability $\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[Y(i) = Y]$ that $Y(i)$ is a particular loss pattern $Y \subseteq R$ is equal to the sum, over all link loss combinations $C \in C_T(Y)$ that result in the loss pattern Y, of the probability that $C(i)$ is the link loss combination C; that is,

$$\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[Y(i) = Y] = \sum_{C' \in C_T(Y)} \Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[C(i) = C'] \tag{5.11}$$

Thus, for any $C \in C_T$ and $Y \subseteq R$, the probability $\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[C(i) = C \mid Y(i) = Y]$ that $C(i)$ is a particular link loss combination $C \in C_T$, given that the loss pattern $Y(i)$ is a particular loss pattern $Y \subseteq R$, is given by:

$$\begin{aligned}\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[C(i) = C \mid Y(i) = Y] &= \frac{\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[C(i) = C \wedge Y(i) = Y]}{\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[Y(i) = Y]} \\ &= \frac{\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[Y(i) = Y \mid C(i) = C] \Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[C(i) = C]}{\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[Y(i) = Y]}\end{aligned} \tag{5.12}$$

However, letting $Y' \subseteq R$ be the loss pattern resulting from the link loss combination C, it is the case that:

$$\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[Y(i) = Y \mid C(i) = C] = \begin{cases} 1 & \text{if } Y = Y' \\ 0 & \text{otherwise} \end{cases} \tag{5.13}$$

Thus, it follows that:

$$\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[C(i) = C \mid Y(i) = Y] = \begin{cases} \frac{\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[C(i)=C]}{\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[Y(i)=Y]} & \text{if } Y = Y' \\ 0 & \text{otherwise} \end{cases} \tag{5.14}$$

We now describe how we probabilistically choose a particular link loss combination $C \in C_T(\textit{loss-patt}(i))$ to represent the transmission scenario pertaining to the packet i. We define the link loss combination sequence *link-comb* to be the mapping *link-comb* : $I \to \mathcal{P}(L)$ that identifies the link loss combination chosen to represent the transmission scenario pertaining to each packet. The probabilistic choice of a particular link loss combination to represent the transmission scenario of each packet i, for $i \in I$, is based on the probability of occurrence of each link loss combination resulting in the loss pattern *loss-patt*(i). In particular, for $i \in I$, we define *link-comb*(i) as follows:

$$\begin{aligned}\textit{link-comb}(i) = C, \text{ with probability } &\Pr_{\mathrm{MCAST}_T[\hat{\alpha}]}[C(i) = C \mid Y(i) = \textit{loss-patt}(i)], \\ &\text{for all } C \in C_T(\textit{loss-patt}(i)).\end{aligned} \tag{5.15}$$

For 13 out of the 14 IP multicast transmission traces of Yajnik *et al.* [41], more than 90% of the link combinations probabilistically chosen by Equation (5.15) have probabilities of occurrence (given by

Equation (5.14)) that exceed 95% and that are often very close to 100%. For the remaining trace, 85% of the chosen link combinations have probabilities of occurrence that exceed 98%. Thus, our estimates of the links responsible for the losses suffered by each receiver are predominantly accurate.

To conclude, recalling that $spath(s,r)$, for $s, r \in N$, is the set of links that make up the unique simple path in T leading from s to r, we define $c\text{-}link(r)(i)$, for $r \in R$ and $i \in I$, as follows:

$$c\text{-}link(r)(i) = \begin{cases} l & \text{if } \{l\} = spath(s,r) \cap link\text{-}comb(i), \text{ and} \\ \perp & \text{otherwise.} \end{cases} \tag{5.16}$$

For $r \in R$ and $i \in I$, it is the case that $c\text{-}link(r)(i) \neq \perp$ if and only if receiver r suffered the loss of the packet i; that is, $c\text{-}link(r)(i) \neq \perp$ if and only if $loss(r)(i) = 1$. If receiver r has received the packet i, then the packet i must have successfully been forwarded on each of the links on the path from the source s to r. It follows that $spath(s,r) \cap link\text{-}comb(i) = \emptyset$ and, consequently, $c\text{-}link(r)(i) = \perp$. Conversely, if r has suffered the loss of the packet i, then the packet i has been dropped on a single link $l \in L$ on the path from s to r. It follows that $spath(s,r) \cap link\text{-}comb(i) = \{l\}$ and, consequently, $c\text{-}link(r)(i) = l$.

5.4 Caching-Based Loss Location Estimation

During our generic IP multicast transmission, packets are transmitted by the source s and disseminated throughout the IP multicast tree T to the set R of receivers. Since packets may be dropped by the links of the IP multicast tree, the receivers may not receive all the packets transmitted by the source. In this section, we propose a caching-based loss location estimation scheme through which receivers estimate during the IP multicast transmission the locations of the losses they suffer.

To begin, we presume that receivers employ *loss detection* and *loss location identification* schemes. We presume that the loss detection scheme allows receivers to detect whether they have suffered the loss of individual packets, possibly with some delay. We presume that the loss location identification scheme allows receivers to identify the locations at which their losses occur, possibly with some additional delay. Here, the notion of a loss location is not limited to that of virtual or concrete links as introduced in Sections 5.3.2 and 5.3.3. Different implementations of the loss location identification scheme may identify loss locations in different ways. The precision of this identification depends on the loss location information that the scheme is able to gather during the IP multicast transmission. In SRM, for instance, receivers may identify the location of a loss by the receivers (requestor and replier pair) that carry out the recovery of the particular packet, *i.e.*, the first receiver to request the packet's retransmission and the first receiver to retransmit the packet. In router assisted reliable multicast protocols, receivers may be able to identify the exact links (or estimates of the links) on which losses occur.

Throughout this chapter, we consider two loss location identification schemes. The first loss location identification scheme is topology-oblivious and imprecisely identifies loss locations by their loss patterns. We model the behavior of this scheme by adopting the virtual link trace representation and presuming that the loss location identification scheme is accurate, *i.e.*, that it identifies loss locations by the virtual links responsible for the respective losses as prescribed by the virtual link trace representation. The second loss location identification scheme is topology-aware and precisely identifies loss locations by the links of the IP multicast tree on which the losses occur. We model the behavior of this scheme by adopting the concrete link trace representation and presuming that the loss location identification scheme is accurate, *i.e.*, that it identifies loss locations by the concrete links responsible for the respective losses as prescribed by the concrete link trace representation. Since the first loss location identification scheme is topology-oblivious and the second is topology-aware, we claim that they correspond to lower and upper precision bounds on any real-life implementation of a loss location identification scheme.

Our proposed caching-based loss location estimation scheme operates as follows. During the IP multicast transmission, each receiver caches the locations of its most recent losses whose locations it has identified using its loss location identification scheme. Upon detecting a loss, a receiver estimates the location of this loss to be the most frequent loss location in its cache; that is, the location responsible for the largest number of recent losses suffered by the given receiver. Since loss locations are identified by the loss location identification scheme, the precision of this caching-based loss location estimation scheme is dictated by the precision of the loss location identification scheme.

As explained in Section 5.2, receivers may benefit from loss location estimates by using them to streamline the recovery of losses and, possibly, reduce recovery latency and overhead. An accurate loss location estimate may allow a receiver to infer the set of receivers that have received and are thus capable of retransmitting the given packet. Thus, as opposed to sending a retransmission request to the whole group, a receiver suffering a loss may address its retransmission request to one or more of the receivers estimated to be capable of retransmitting the packet. A loss location estimate that overestimates the extent of the loss may lead the receiver to attempt to recover the given packet from receivers that are further away than required, thus unduly increasing recovery latency. Finally, a loss location estimate that underestimates the extent of the loss may lead the receiver to attempt to recover the packet from receivers that share the loss, thus preventing the successful recovery of the loss.

We claim that the accuracy of our proposed caching-based loss location estimation scheme is an indication of the degree to which IP multicast losses exhibit locality. We also claim that it is an indication of the potential benefit of incorporating a caching-based loss recovery scheme within either an existing or a novel reliable multicast protocol. Finally, we claim that the adoption of the virtual and the concrete link trace representations yields lower and upper bounds, respectively, on the performance of a caching-based loss recovery scheme that uses loss location estimates to guide the loss recovery. This claim ensues from the fact that the adoption of the virtual and the concrete link trace representations yields lower and upper loss location identification precision bounds, respectively.

In the rest of this section, we precisely define our proposed caching-based loss location estimation scheme and the metrics we will subsequently use to evaluate its performance. We first describe the notation that we use to denote the times at which receivers detect the losses they suffer and the times at which they identify the locations of these losses. We then define our caching-based loss location estimation scheme. Finally, we define the metrics that we will subsequently use to evaluate both its accuracy and the potential benefit of incorporating a caching-based loss recovery scheme within either an existing or a novel reliable multicast protocol.

5.4.1 Loss Detection and Loss Location Identification

In this section, we describe the notation that we use to denote the points in time at which each receiver detects the losses it suffers and identifies their respective locations. In order to estimate these loss detection and loss location identification times, we need to know the packet reception times. Since the IP multicast traces of Yajnik *et al.* [41] contain no timing information, we presume that all the packets received by each receiver incur the same transmission latency. Thus, since packets are transmitted periodically, we presume that each receiver is slated to receive packets periodically. We refer to the times at which each receiver is slated to receive IP multicast packets as *time slots*. We identify each such time slot by the identifier of the respective packet; that is, we say that a receiver is slated to receive packet $i \in I$ at time slot i.

We now introduce two mappings $dtime : R \rightarrow (I \rightarrow I \cup \perp)$ and $itime : R \rightarrow (I \rightarrow I \cup \perp)$. The first mapping *dtime* satisfies the following constraints:

1. For $r \in R$ and $i \in I$, $dtime(r)(i) = \perp$ if $loss(r)(i) = 0$ and $dtime(r)(i) \in I$ otherwise.
2. For $r \in R$ and $i \in I$, $dtime(r)(i) \neq \perp \Rightarrow i \leq dtime(r)(i)$.
3. For $r \in R$ and $i, i' \in I$, such that $dtime(r)(i) \neq \perp$, $dtime(r)(i') \neq \perp$, and $i < i'$, it is the case that $dtime(r)(i) \leq dtime(r)(i')$.

For $r \in R$ and $i \in I$, $dtime(r)(i)$ identifies the time slot at which receiver r detects the loss of packet i. The first constraint on *dtime* dictates that receivers associate loss detection times only with packets that they have lost. The second constraint dictates that receivers detect losses no earlier than the expected reception times of the respective packets. The third constraint dictates that losses are detected in the order in which the respective packets were transmitted.

The mapping *dtime* depends on the loss detection scheme employed by the receivers. Since each of the following sections will be considering different loss location identification schemes, we defer the definition of *dtime* to those sections.

The second mapping *itime* satisfies the following constraints:

1. For $r \in R$ and $i \in I$, $itime(r)(i) = \perp$ if $loss(r)(i) = 0$ and $itime(r)(i) \in I$ otherwise.
2. For $r \in R$ and $i \in I$, $itime(r)(i) \neq \perp \Rightarrow dtime(r)(i) \leq itime(r)(i)$.
3. For $r \in R$ and $i, i' \in I$, such that $itime(r)(i) \neq \perp$, $itime(r)(i') \neq \perp$, and $i < i'$, it is the case that $itime(r)(i) \leq itime(r)(i')$.

For $r \in R$ and $i \in I$, $itime(r)(i)$ identifies the time slot at which receiver r identifies the loss location of packet i. The first constraint on *itime* dictates that receivers associate loss location identification times only with packets that they have lost. The second constraint dictates that receivers identify loss locations no earlier than the points in time at which the respective losses are detected. The third constraint dictates that the loss locations are identified in the order in which the respective packets were transmitted.

Again, the mapping *itime* depends on the loss location identification scheme employed by the receivers. Since each of the following sections will be considering different loss location identification schemes, we defer the definition of *itime* to those sections.

5.4.2 Loss Location Estimation

Since our caching-based loss location estimation scheme is independent of the adopted trace representation (be that the virtual link, the concrete link, or any other analogous representation), we present our caching-based loss location estimation scheme using a generic link set and a corresponding generic link trace representation. We let LINK be this generic set of links on which losses may occur and $link : R \rightarrow (I \rightarrow \text{LINK} \cup \perp)$ be the mapping that identifies the links on which the losses suffered by the respective receivers occur. In the case of the virtual link trace representation, LINK and *link* correspond to V-LINK and *v-link*, respectively. In the case of the concrete link trace representation, LINK and *link* correspond to C-LINK and *c-link*, respectively.

At any time slot $i \in I$, a receiver $r \in R$ has cached the locations of its most recent losses whose locations it has identified. Letting $m \in \mathbb{N}^+$ denote the size of the cache of r, we define $iset_m(r)(i) \subseteq I$ to be the set of the m most recent losses suffered by r whose locations it has identified prior to time slot i; that is,

$$iset_m(r)(i) = \text{largest}_m \left(\{i' \in I \mid itime(r)(i') \in I \wedge itime(r)(i') < i\}\right) \tag{5.17}$$

where, for $S \subseteq \mathbb{N}$, $\text{largest}_m(S)$ denotes the set of the m largest elements of S, if $|S| \geq m$, and the set S, otherwise.

Let $\textsc{Cache}$ be the set of subsets of $I \times \textsc{Link}$ in which there is at most one element corresponding to each index $i \in I$; that is, in each subset of $I \times \textsc{Link}$ in $\textsc{Cache}$ there do not exist two elements $\langle i, l\rangle, \langle i', l'\rangle \in I \times \textsc{Link}$ such that $i = i'$ and $l \neq l'$.

For $r \in R$ and $i \in I$, we define $\mathit{cache}(r)(i) \in \textsc{Cache}$ to be the contents of the cache of r at time slot i; that is,

$$\mathit{cache}(r)(i) = \{\langle i', \mathit{link}(r)(i')\rangle \in I \times \textsc{Link} \mid i' \in \mathit{iset}_m(r)(i)\} \tag{5.18}$$

In particular, at time slot $\mathit{dtime}(r)(i)$, when receiver r detects the loss of the packet i, the contents of the cache of r are $\mathit{cache}(r)(\mathit{dtime}(r)(i))$.

For $C' \in \textsc{Cache}$ and $L' \in \mathcal{P}(\textsc{Link})$, we define the function $\mathit{most\text{-}frequent} : \textsc{Cache} \times \mathcal{P}(\textsc{Link}) \rightarrow \mathcal{P}(\textsc{Link})$ as follows:

$$\mathit{most\text{-}frequent}(C', L') = \operatorname*{argmax}_{l' \in L'} \left|\{\langle i, l\rangle \in C' \mid l = l'\}\right|, \tag{5.19}$$

where, for a non-empty finite set Z and a function $f : Z \rightarrow \mathbb{N}$, $\operatorname{argmax}_{z \in Z} f(z) = \{z \in Z \mid \forall\, z' \in Z : f(z') \leq f(z)\}$. The set $\mathit{most\text{-}frequent}(C', L')$ is comprised of the links in L' that appear most frequently in the tuples stored in the cache C'.

Moreover, we define the function $\mathit{most\text{-}recent} : \textsc{Cache} \times \mathcal{P}(\textsc{Link}) \rightarrow \mathcal{P}(\textsc{Link})$ as follows:

$$\mathit{most\text{-}recent}(C', L') = \operatorname*{argmax}_{l' \in L'} \max_{i' \in \{i \in I \mid \langle i, l'\rangle \in C'\}} i'. \tag{5.20}$$

Out of all the packets in C' dropped on links in L', the link $\mathit{most\text{-}recent}(C', L')$ is the link in L' on which the most recent packet was dropped.

Finally, we define the loss location estimate mapping $\mathit{est\text{-}link} : R \rightarrow (I \rightarrow \textsc{Link} \cup \perp)$ such that, for $r \in R$ and $i \in I$,

$$\mathit{est\text{-}link}(r)(i) = \begin{cases} l & \text{if } \mathit{loss}(r)(i) = 1 \\ & \quad \wedge \mathit{cache}(r)(\mathit{dtime}(r)(i)) \neq \emptyset \\ & \quad \wedge \{l\} = \mathit{most\text{-}recent}(\mathit{cache}(r)(\mathit{dtime}(r)(i)), \\ & \qquad\qquad \mathit{most\text{-}frequent}(\mathit{cache}(r)(\mathit{dtime}(r)(i)), \textsc{Link})), \\ & \text{and} \\ \perp & \text{otherwise.} \end{cases} \tag{5.21}$$

Thus, for $r \in R$ and $i \in I$, if $\mathit{loss}(r)(i) = 1$ and $\mathit{cache}(r)(\mathit{dtime}(r)(i)) \neq \emptyset$, then the link estimated by receiver r to be responsible for the loss of the packet i is the link occurring most frequently in its cache at the point in time when the loss of the packet i is detected, *i.e.*, the point in time $\mathit{dtime}(r)(i)$. Ties among links that populate the cache with equal cardinality are resolved by choosing the link on which the most recent loss, among all the losses occurring on contending links, occurs.

Loss Location Estimate Classification

For $r \in R$ and $i \in I$, such that $\mathit{link}(r)(i) \neq \perp$ and $\mathit{est\text{-}link}(r)(i) \neq \perp$, we classify the loss location estimate $\mathit{est\text{-}link}(r)(i)$ as:

- *high*, if $\mathit{link}(r)(i) \prec \mathit{est\text{-}link}(r)(i)$,

- *accurate*, if $\textit{est-link}(r)(i) = \textit{link}(r)(i)$,
- *low*, if $\textit{est-link}(r)(i) \prec \textit{link}(r)(i)$, and
- *incomparable*, otherwise.

We collectively refer to high, low, and incomparable link loss estimates as *inaccurate*.

Conceptually, a loss location estimate is high when the estimated location is upstream of the actual loss location. Since a loss at a location upstream of the actual loss location affects a larger subtree of the IP multicast tree, a high loss location estimate overestimates the extent of the loss. A loss location estimate is low when the estimated location is downstream of the actual loss location. Since a loss at a location downstream of the actual loss location affects a smaller subtree of the IP multicast tree, a low loss location estimate underestimates the extent of the loss. Incomparable loss location estimates may only arise in the case of the virtual link trace representation where loss locations are identified by the sets of receivers affected by the respective losses. When either the estimated or the actual loss locations are the result of simultaneous losses on multiple links of the IP multicast tree, the estimated and actual loss locations may be incomparable; that is, they may be neither equal, nor strict supersets or subsets of each other. In the case of the concrete link trace representation, each loss suffered by each receiver is identified by a single link of the IP multicast tree on the path from the source to the respective receiver. Since such links are always comparable, estimated and actual loss locations are never incomparable.

In a caching-based loss recovery scheme that uses loss location estimates to guide the recovery of losses, a high loss location estimate may lead a receiver to request the packet's retransmission from a receiver that is unnecessarily far away. Although the receiver of this request will presumably be capable of retransmitting the packet, the resulting recovery would incur unduly latency. Conversely, a low loss location estimate may lead a receiver to request the packet's retransmission from a receiver that has shared the given loss. Since the receiver of this request is incapable of retransmitting the packet, the packet's recovery would fail. In the case of an incomparable loss location estimate, it is unclear whether the packet's retransmission request will be addressed to a receiver that is capable of retransmitting the packet. It is thus questionable whether packet loss recoveries instigated by incomparable loss location estimates lead to successful loss recoveries.

Per-Receiver Hits and Per-Receiver Hit Rate

We consider accurate (inaccurate) loss location estimates to be cache *hits* (*misses*, respectively). For $r \in R$, we let $\textit{losses}(r) \subseteq I$ and $\textit{hits}(r) \subseteq I$, denote the set of losses suffered by r and the set of losses suffered by r whose locations are accurately estimated by r, respectively; that is,

$$\textit{losses}(r) = \{i \in I \mid \textit{loss}(r)(i) = 1\} \tag{5.22}$$

$$\textit{hits}(r) = \{i \in \textit{losses}(r) \mid \textit{est-link}(r)(i) = \textit{link}(r)(i)\} \tag{5.23}$$

For $r \in R$, we define the *hit rate* of receiver r, denoted $\textit{hit-rate}(r)$, to be the ratio of the number of hits to that of losses for r; that is,

$$\textit{hit-rate}(r) = \frac{|\textit{hits}(r)|}{|\textit{losses}(r)|}. \tag{5.24}$$

We claim that the per-receiver hit rate is an indication of the degree to which the losses suffered by individual receivers exhibit locality.

Consistent and Inconsistent Estimates

We define the set of all the losses suffered during the IP multicast transmission, denoted *losses*, as follows:

$$losses = \{\langle i, l\rangle \in I \times \textsc{Link} \mid \exists\, r \in R, link(r)(i) = l\}. \tag{5.25}$$

A tuple $\langle i, l\rangle \in I \times \textsc{Link}$ is in the set *losses* if the packet i was dropped on link l while being disseminated throughout the IP multicast tree. Since a single packet may be dropped on multiple links, the set *losses* may include several tuples pertaining to a single packet.

We refer to the scenarios in which the loss location estimates of all the receivers that share a loss are accurate as *consistent accurate estimates.* We define the set of consistent accurate estimates, denoted *cons-acc-ests*, as follows:

$$\begin{aligned} cons\text{-}acc\text{-}ests = \{&\langle i, l\rangle \in losses \mid \\ &\forall\, r \in R : link(r)(i) = \{l\} \Rightarrow est\text{-}link(r)(i) = link(r)(i)\}\end{aligned} \tag{5.26}$$

In the case of a consistent accurate estimate, the extent of the loss is accurately estimated by all receivers affected by the loss. In terms of a caching-based loss recovery scheme that uses loss location estimates to guide the recovery of losses, retransmission requests would presumably be addressed to receivers that are capable of retransmitting the packet and the recovery of the packet is successful. Furthermore, by not overestimating the extent of the loss, retransmission requests would presumably be addressed to the closest receivers capable of retransmitting the packet. Thus, the resulting recovery would incur the minimum recovery latency.

We refer to scenarios that do not constitute consistent accurate estimates and in which the loss location estimates of all the receivers affected by the loss are either high or accurate as *consistent high estimates.* We define the set of consistent high estimates, denoted *cons-high-ests*, as follows:

$$\begin{aligned} cons\text{-}high\text{-}ests = \{&\langle i, l\rangle \in losses \backslash cons\text{-}acc\text{-}ests \mid \\ &\forall\, r \in R : link(r)(i) = l \Rightarrow link(r)(i) \preceq est\text{-}link(r)(i)\}\end{aligned} \tag{5.27}$$

In the case of a consistent high estimate, some (possibly, all) of the receivers sharing the loss overestimate and no such receiver underestimates its extent. In terms of our supposed caching-based loss recovery scheme, retransmission requests would presumably be addressed to receivers that are part of a larger subtree of the IP multicast tree than required. Consequently, the recovery would be exposed to a larger region of the IP multicast tree than required and would incur unduly latency.

We refer to scenarios in which the loss location estimates of all the receivers that share a loss are low as *consistent low estimates.* We define the set of consistent high estimates, denoted *cons-low-ests*, as follows:

$$\begin{aligned} cons\text{-}low\text{-}ests = \{&\langle i, l\rangle \in losses \mid \\ &\forall\, r \in R : link(r)(i) = l \Rightarrow est\text{-}link(r)(i) \prec link(r)(i)\}\end{aligned} \tag{5.28}$$

In such scenarios, retransmission requests of our supposed caching-based loss recovery scheme would be addressed to receivers that share the loss and would consequently fail.

Finally, we refer to the remaining scenarios as *inconsistent estimates.* We define the set of inconsistent estimates, denoted *incons-ests*, as follows:

$$incons\text{-}ests = losses \backslash (cons\text{-}acc\text{-}ests \cup cons\text{-}high\text{-}ests \cup cons\text{-}low\text{-}ests) \tag{5.29}$$

In the case of an inconsistent estimate, the receivers affected by the loss estimate that the loss occurs on a combination of upstream, accurate, downstream, and, possibly, incomparable locations. The outcome of inconsistent estimates in terms of the loss recovery process depends on precisely how loss location estimates are used by the recovery scheme at hand. In particular, it depends on which of the receivers affected by the loss participate in the recovery by transmitting retransmission requests and whether such requests are based on either upstream, accurate, downstream, or incomparable loss location estimates.

Shared Hit Rate and Estimated Recovery Rate

We refer to consistent accurate estimates as *shared hits* and we define the *shared hit rate*, denoted *shared-hit-rate*, to be the ratio of the number of shared hits and the number of losses. More precisely, we define the shared hit rate, $\textit{shared-hit-rate} \in [0, 1]$, as follows:

$$\textit{shared-hit-rate} = \frac{|\textit{cons-acc-ests}|}{|\textit{losses}|} \tag{5.30}$$

We now estimate the number of losses that would be successfully recovered by a caching-based loss recovery scheme that uses the loss location estimates to guide the recovery of losses. As mentioned above, recoveries that are based on either consistent accurate or consistent high estimates are presumably successful. Conversely, packet loss recoveries based on consistent low estimates are presumably unsuccessful. The success or failure of a recovery based on inconsistent estimates depends on which of the affected receivers participate in the recovery process by transmitting a retransmission request and whether their loss location estimates are either high or accurate.

Since both consistent accurate and consistent high estimates result in successful recoveries, we define the *consistent estimate recovery rate*, denoted *cons-est-rec-rate*, to be the ratio of the number of consistent accurate and high estimates to the total number of losses. More precisely, we define the consistent estimate recovery rate, $\textit{cons-est-rec-rate} \in [0, 1]$, as follows:

$$\textit{cons-est-rec-rate} = \frac{|\textit{cons-acc-ests} \cup \textit{cons-high-ests}|}{|\textit{losses}|} \tag{5.31}$$

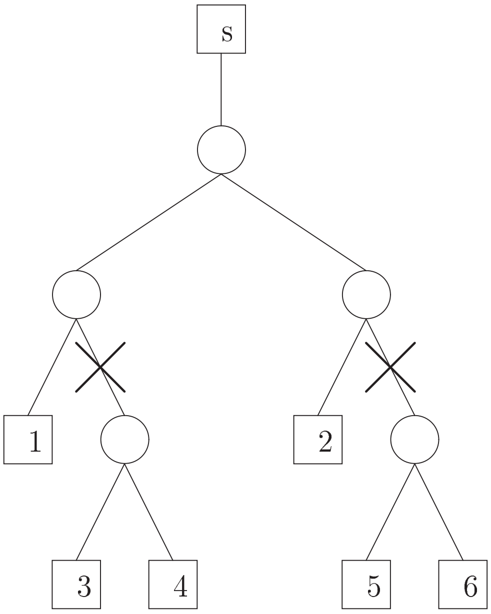
For each loss resulting in an inconsistent estimate, we calculate the ratio of the number of receivers that share the loss and produce either accurate or high estimates and the total number of receivers that share the given loss. This ratio corresponds to the probability with which the given packet is successfully recovered. Thus, we define the number of successful recoveries instigated by inconsistent estimates, denoted *incons-est-rec-count*, to be the sum of these ratios. More precisely, we define $\textit{incons-est-rec-count} \in \mathbb{R}^{\geq 0}$ as follows:

$$\textit{incons-est-rec-count} = \sum_{\langle i,l \rangle \in \textit{incons-ests}} \frac{|\{r \in R \mid \textit{link}(r)(i) = l \wedge \textit{link}(r)(i) \preceq \textit{est-link}(r)(i)\}|}{|\{r \in R \mid \textit{link}(r)(i) = l\}|} \tag{5.32}$$

We define the *inconsistent estimate recovery rate*, denoted *incons-est-rec-rate*, to be the ratio of the number of successful recoveries instigated by inconsistent estimates to the total number of losses. More precisely, we define the inconsistent estimate recovery rate, $\textit{incons-est-rec-rate} \in [0, 1]$, as follows:

$$\textit{incons-est-rec-rate} = \frac{\textit{incons-est-rec-count}}{|\textit{losses}|} \tag{5.33}$$

Figure 5.1 Example of a Lossy IP Multicast Transmission.



Finally, we define the overall *recovery rate*, denoted *rec-rate*, to be the estimate of the ratio of losses that would be successfully recovered by our supposed caching-based loss recovery scheme. More precisely, we define the recovery rate, $\textit{rec-rate} \in [0, 1]$, as follows:

$$\textit{rec-rate} = \textit{cons-rec-rate} + \textit{incons-rec-rate} \tag{5.34}$$

5.4.3 Discussion

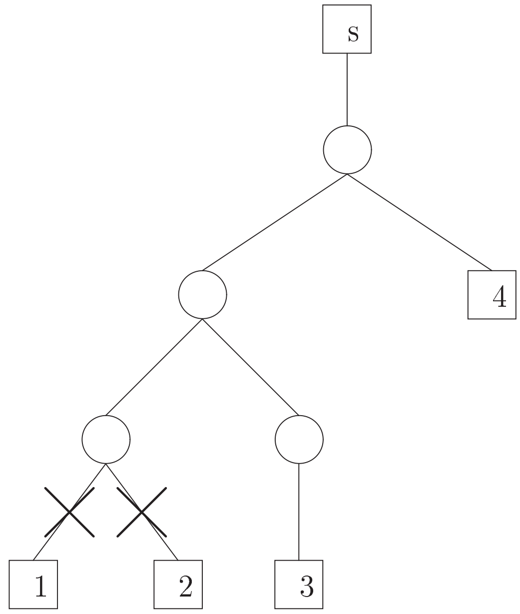
In the case of the virtual link trace representation, a loss location estimate is a hit only if the receiver accurately estimates the virtual link responsible for the loss, *i.e.*, the exact set of receivers that share the loss. In terms of the loss recovery process, however, a receiver need not estimate this exact set in order to benefit from a caching-based loss recovery scheme.

Consider, for instance, the lossy IP multicast transmission scenario shown in Figure 5.1. In this scenario, the source, which is the root of the IP multicast tree, transmits a packet and this packet is dropped on two distinct links of the IP multicast tree. Receivers 3 and 4 can recover the packet from receiver 1 and receivers 5 and 6 can recover the packet from receiver 2. Were receivers 3 and 4 to estimate that the loss is shared by receivers 3 and 4 only, they would thus be able to recover the packet from receiver 1. Analogously, were receivers 5 and 6 to estimate that the loss is shared by receivers 5 and 6 only, they would be able to recover the packet from receiver 2. Since the loss pattern corresponding to the given scenario involves the receivers 3, 4, 5, and 6, the scenario in which receivers 3 and 4 estimate the loss location to be the virtual link $\{3, 4\}$ and receivers 5 and 6 estimate the loss location to be the virtual link $\{5, 6\}$ would be deemed as leading to an unsuccessful recovery — the loss location estimates of all receivers affected by the losses would be considered low.

This example demonstrates that the performance of our loss location estimation scheme in the case of the virtual link representation may underestimate the expected effectiveness of caching within a multicast loss recovery scheme.

While adopting the virtual link trace representation may lead to us underestimating the benefit of incorporating caching in a multicast loss recovery scheme, adopting the concrete link trace

Figure 5.2 Example of a Lossy IP Multicast Transmission.



representation may lead to us overestimating it. This is due to several reasons. Firstly, by adopting the concrete link trace representation and presuming that loss locations are accurately identified, we essentially presume that each receiver is capable of identifying the exact links of the IP multicast tree responsible for the losses it suffers. In practice, however, such precision in identifying loss locations may not be realistic.

Secondly, even precise loss location estimates may not always result in optimal loss recovery. Consider, for instance, the transmission scenario depicted in Figure 5.2. In this scenario, the packet being transmitted is dropped on two distinct links leading to receivers 1 and 2. The receivers 3 and 4 are capable of retransmitting the packet and are equidistant from the receivers 1 and 2. Even when receivers 1 and 2 accurately estimate the links on which the packet is dropped, receiver 1 may request the packet from 3 and 2 may request it from 4, leading to two retransmissions. Although for the concrete link trace representation such estimates are considered hits, they do not lead to the desired recovery behavior involving a single request and a single reply.

5.5 Evaluating the Effectiveness of Caching

In this section, we analyze the performance of our caching-based loss location estimation scheme introduced in Section 5.4. We present and compare the performance of our loss location estimation scheme for several cache sizes. A cache of size 1 estimates that a loss occurs on the location of the most recent loss whose location has been identified. A cache of infinite size records the location of all prior losses whose locations have been identified. Estimates made based on an infinite cache correspond to the most frequent loss location identified by the receiver up to that point in the trace.

We analyze the performance of our loss location estimation scheme using both the virtual and the concrete link trace representations. As noted above, the virtual link representation may underestimate the expected effectiveness of caching in multicast loss recovery, while the concrete link representation may overestimate it. Our analysis indicates that the IP multicast transmission traces of Yajnik *et al.* [41] exhibit substantial locality and that caching within multicast loss recovery can indeed be very effective.

In Section 5.5.1, we present the per-receiver hit rates achieved by our caching-based loss location estimation scheme under the assumption that the detection of losses and the identification of their locations are both immediate. In Section 5.5.2, we present the per-receiver hit rates under the assumption that losses are detected upon the receipt of later packets and their locations are identified immediately. In Section 5.5.3, we evaluate the performance of our loss location estimation scheme as the delay in identifying loss locations increases. In Section 5.5.4, we observe the distribution of loss location estimate scenarios among consistent high, accurate, low, and incomparable estimates. While in Sections 5.5.1 through 5.5.3, we consider caches of size 1, 10, and infinity, in Section 5.5.5 we analyze the effect of the cache size on the shared hit rates.

5.5.1 Immediate Detection/Immediate Identification

We present the per-receiver hit rates achieved by our caching-based loss location estimation scheme, under the assumption that the detection of losses and the identification of their locations are both immediate; that is, we assume that the loss location estimation scheme is aware of the location of all losses that precede the loss whose location is being estimated.

In particular, we define $dtime(r)(i) \in I$ and $itime(r)(i) \in I \cup \perp$, for $r \in R$ and $i \in I \cup \perp$, as follows:

$$dtime(r)(i) = \begin{cases} \perp & \text{if } loss(r)(i) = 0\text{, and} \\ i & \text{otherwise.} \end{cases} \tag{5.35}$$

$$itime(r)(i) = dtime(r)(i) \tag{5.36}$$

Figure 5.3 presents the per-receiver hit rates for the virtual link trace representation for 6 out of the 14 traces. The per-receiver hit rates for the rest of the traces are similar. Each of the graphs in Figure 5.3 depict the per-receiver hit rates achieved using cache sizes of 1, 10, and infinity.

We observe that the cache of size 10 outperforms the cache of size 1 in most cases. As observed by the multicast loss studies of [1, 15, 41, 42], IP multicast transmissions involve a few highly lossy links that generate a large percentage of the losses and a large number of slightly lossy links. The larger the cache size, the less susceptible it is to sporadic losses (due to slightly lossy links) that may interrupt long sequences of losses on the same location (due to highly lossy links).

We also observe that caches of size 1 and 10 often outperform the infinite cache size. In fact, the infinite cache size performs as well as the others only for receivers whose losses are predominantly due to single locations. Consider, for instance, the hit rates achieved by receivers 2 and 3 of trace WRN951128. The caches of size 1 and 10 substantially outperform the infinite cache size for receiver 2. In the case of receiver 3, the hit rates achieved by caches of size 1 and 10 are comparable to those achieved by the infinite cache size. Figure 5.4 depicts the loss distributions for receivers 2 and 3 of trace WRN951128; that is, the percentage of losses suffered by each receiver that occur on each loss location. The loss percentages are shown in log scale. Three loss locations account for large percentages of the losses suffered by receiver 2. In this case, smaller cache sizes that can adapt quicker to changing loss conditions outperform the infinite cache. Conversely, the losses suffered by receiver 3 occur predominantly on a single location. In this case, the infinite cache size estimates that all losses occur at the highly lossy location and thus performs similarly to the smaller cache sizes.

Figure 5.5 presents the per-receiver hit rates for the concrete link trace representation for the same 6 traces. Again, the per-receiver hit rates for the rest of the traces are similar. The per-receiver hit rates for the concrete link trace representation are substantially higher than those for the virtual link trace representation. This is not surprising given the fact that in the case of the concrete

Figure 5.3 Virtual Link Trace Representation — Immediate Detection/Identification.

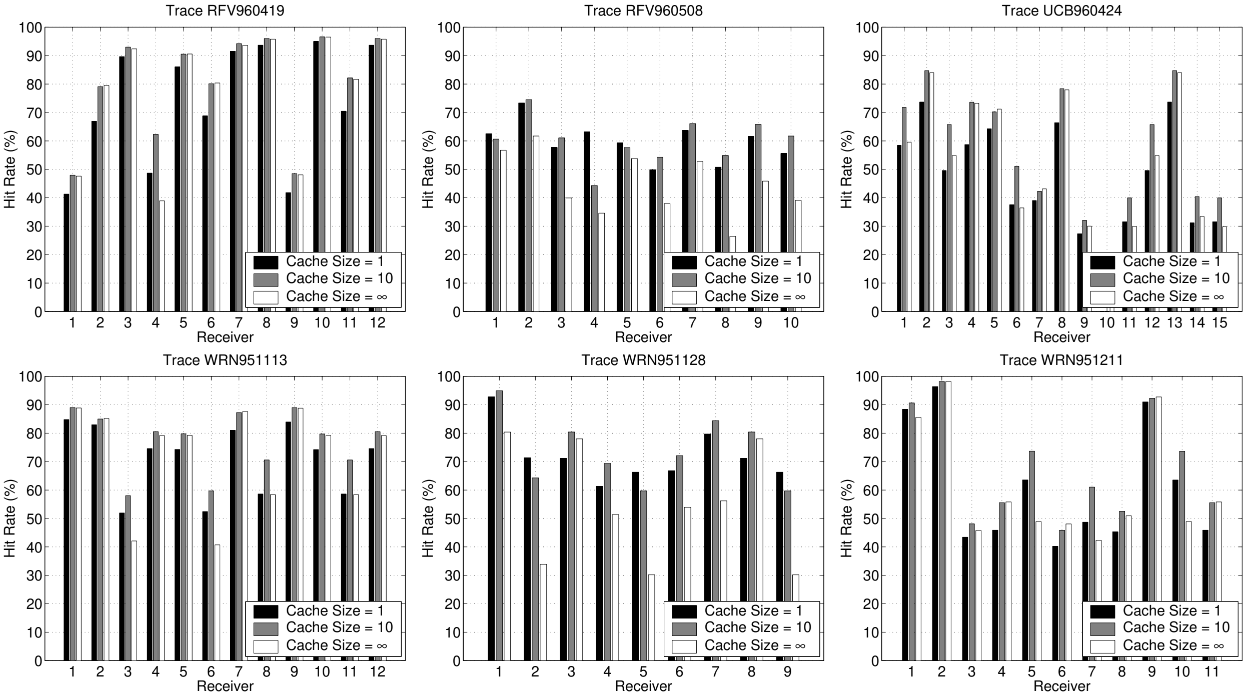
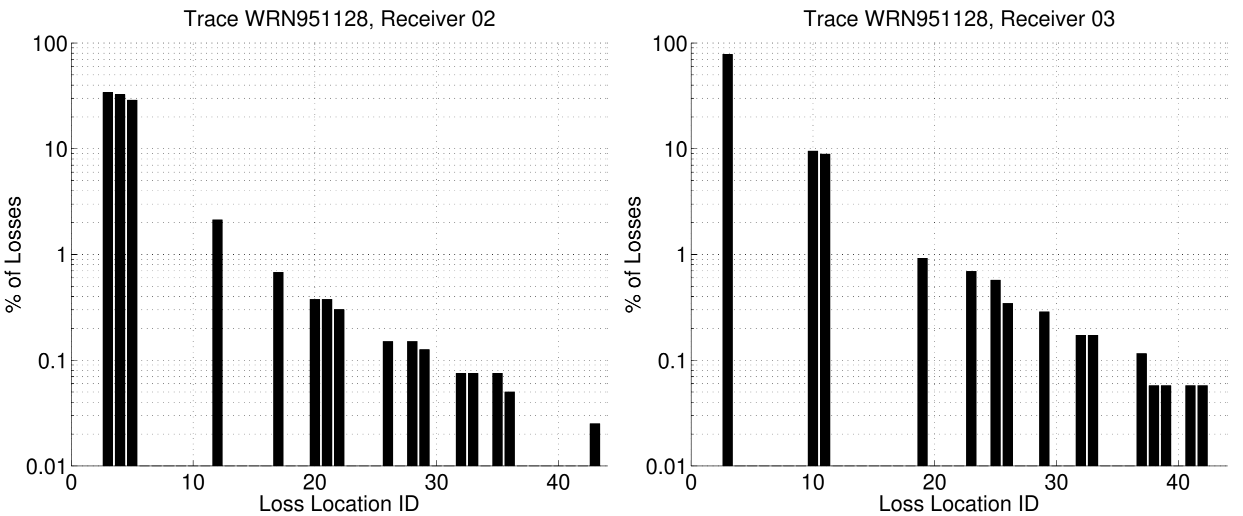


Figure 5.4 Virtual Link Trace Representation — Per-receiver Loss Distributions, Receivers 2 & 3, Trace WRN951128.

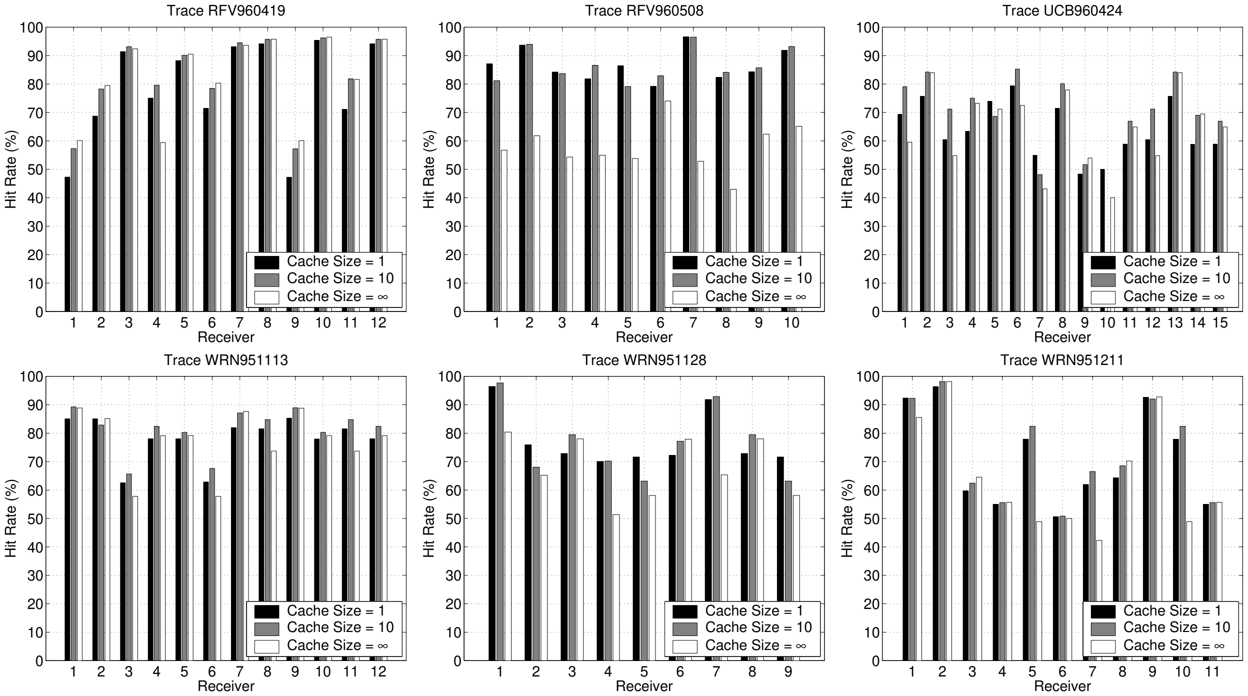


link representation each receiver suffers losses due to a small number of distinct locations — equal to the path length from the source to each receiver. Moreover, in the case of the concrete link trace representation, loss patterns resulting from simultaneous losses on highly lossy links are not misinterpreted as losses occurring at distinct locations; rather, each receiver attributes each loss to one of the IP multicast tree links that are on the path from the source to itself.

5.5.2 Delayed Detection/Immediate Identification

The packet loss locality exhibited in the previous section may not be exploitable, since losses may not be immediately detectable by the affected receivers. In many reliable multicast protocols,

Figure 5.5 Concrete Link Trace Representation — Immediate Detection/Identification.



receivers detect losses upon the receipt of later packets. Thus, when a receiver suffers a loss burst, it detects all the losses that are part of the burst all at once upon the receipt of the first packet following the loss burst. In this section, we observe the effect of delayed loss detection. In particular, we assume that: i) losses are detected upon the receipt of a later packet (*delayed detection*), and ii) the loss location estimation scheme is aware of the location of all losses that are detected earlier than the detection time of the loss whose location is being estimated (*immediate identification*).

In particular, we define $dtime(r)(i) \in I \cup \perp$ and $itime(r)(i) \in I \cup \perp$, for $r \in R$ and $i \in I$, as follows:

$$dtime(r)(i) = \begin{cases} \perp & \text{if } loss(r)(i) = 0\text{, and} \\ \min\{i' \in I \mid i < i' \wedge loss(r)(i') = 0\} & \text{otherwise.} \end{cases} \tag{5.37}$$

$$itime(r)(i) = dtime(r)(i) \tag{5.38}$$

Figure 5.6 presents the per-receiver hit rates of our loss location estimation scheme for the virtual link trace representation of 6 out of the 14 traces. By comparing the hit rates presented in Figures 5.3 and 5.6, we observe that the delay in detecting losses heavily affects the per-receiver hit rates of some traces; the trace RFV960508 is the most heavily affected trace and achieves the lowest hit rates of all 14 traces. This effect is due to loss bursts. With immediate detection, the estimate of the location of trailing losses within a burst is based on the location of the leading losses of the burst. In contrast, when losses are detected upon the receipt of a later packet, the losses that are part of the burst are detected simultaneously and their locations are all estimated based on the locations of losses suffered prior to the burst. Thus, the (in)correct estimate of the losses that are part of long loss bursts heavily affects the estimate hit rates.

Consider for instance the hit rates of receivers 3 and 4 of trace RFV960419. Figure 5.7 depicts the distribution of losses across loss bursts of increasing length for receivers 3 and 4 of trace RFV960419, *i.e.*, the percentage of losses that are part of loss bursts of increasing lengths. The loss percentages

Figure 5.6 Virtual Link Trace Representation — Delayed Detection/Immediate Identification.

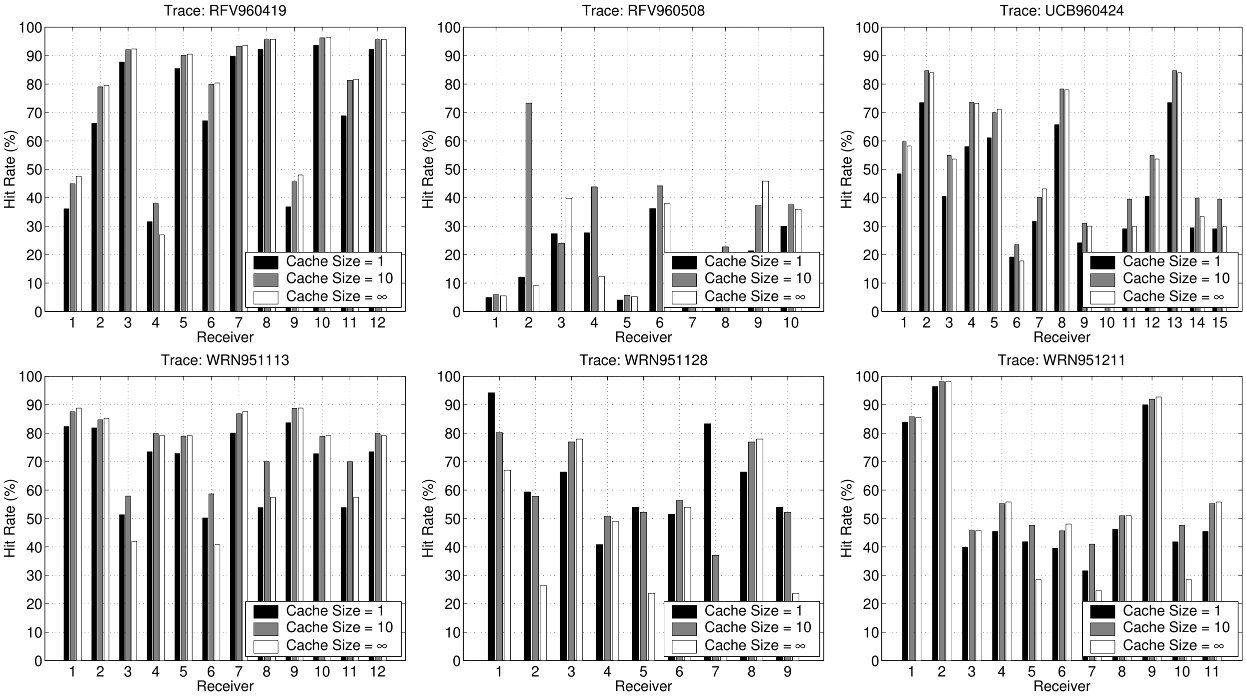
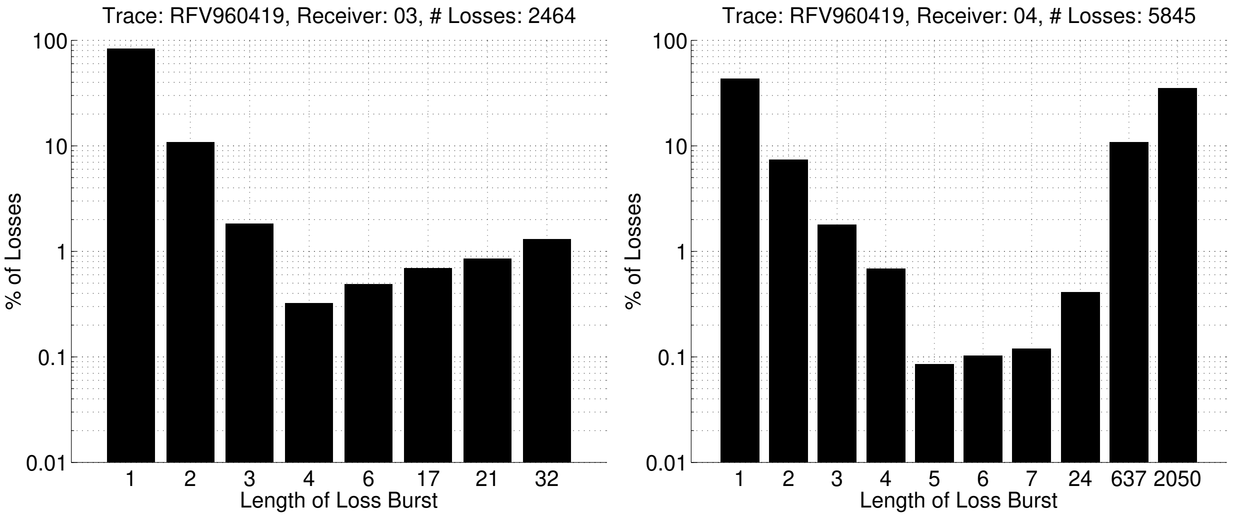


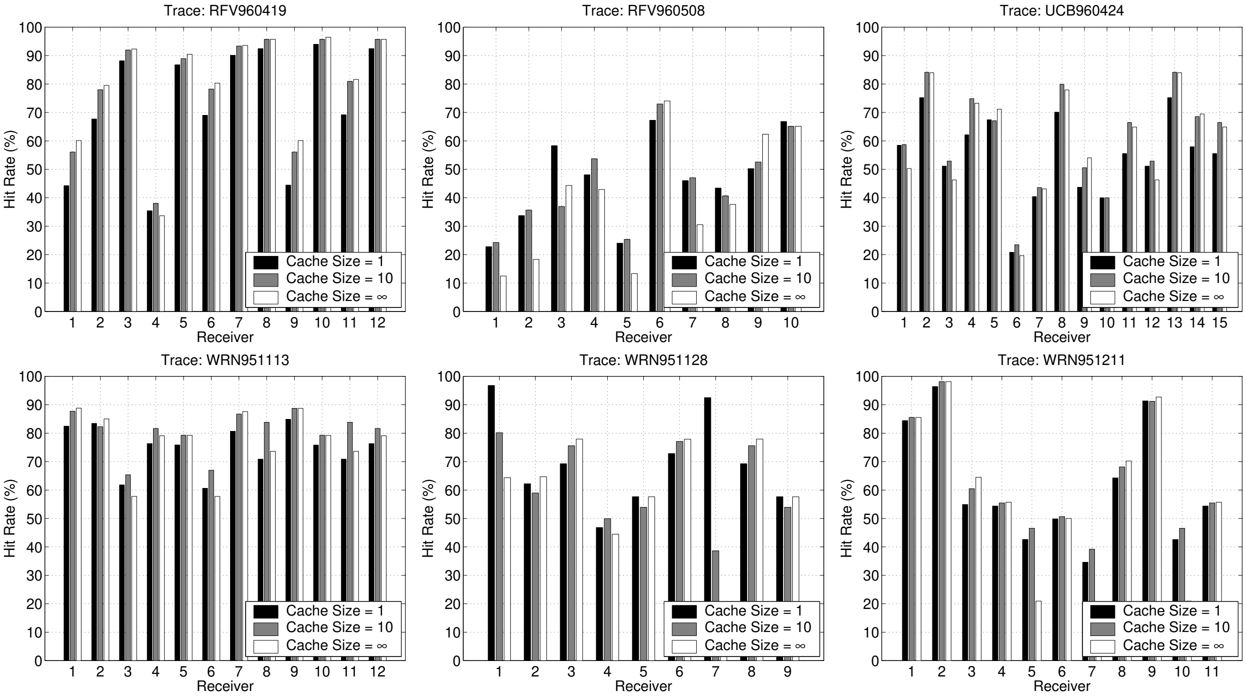
Figure 5.7 Loss Distribution wrt Burst Length, Receivers 3 & 4, Trace RFV960419.



are shown in log scale. Receiver 3 suffers predominantly isolated losses. Conversely, receiver 4 suffers a couple of long loss bursts. The adverse effect of these loss bursts on the hit rate of receiver 4 is evident when one compares receiver 4's hit rates in Figures 5.3 and 5.6. The hit rates of receiver 3 are barely affected by the delayed detection, while those of receiver 4 are nearly cut in half.

The adverse effect of the delay in detecting losses suggests that it would be beneficial to design schemes for detecting losses sooner. SRM's exchange of session messages is one such scheme. Session messages are used by receivers to periodically advertise the per-source transmission progress they have observed. Thus, receivers may discover losses by detecting discrepancies in the observed transmission progress of the receivers. When packets are transmitted at a fixed frequency, as is done in audio and video transmissions, an alternative approach may be to track the inter-packet

Figure 5.8 Concrete Link Trace Representation — Delayed Detection/Immediate Identification.



delays and to declare a packet missing when its arrival with respect to its predecessor has exceeded some jitter threshold. In order for such schemes to allow the early detection and recovery of packets, session and recovery packets must avoid the congested links responsible for the loss burst by, for example, using a source-based IP multicast tree implementation [37].

Early detection schemes may potentially allow the reliable multicast protocol to identify the location of the leading losses of a burst sooner, thus benefiting the location estimate of the trailing losses of the burst. Alternatively, it may be beneficial to treat all the losses that are part of particular loss bursts collectively. For instance, upon detection of a loss burst, a receiver could recover the first loss of the burst and, subsequently, recover the remaining losses of the burst in the manner in which the first loss of the burst was recovered.

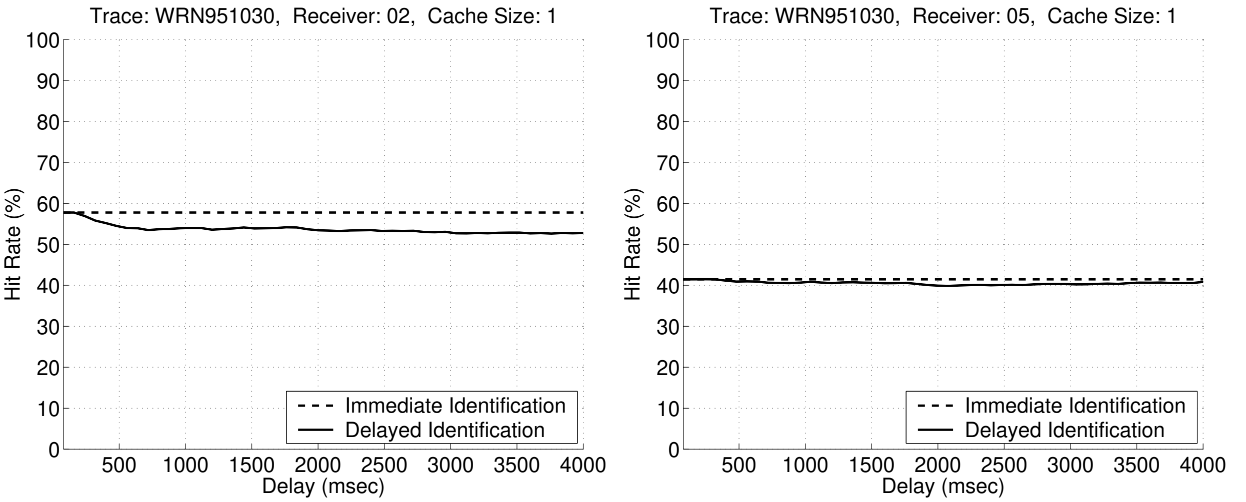
Figure 5.8 presents the hit rates of our loss location estimation scheme for the concrete link trace representation for the same 6 traces. The effects of delayed loss detection for the concrete link loss representation are similar to, yet less severe than, those observed for the virtual link loss representation.

5.5.3 Delayed Detection/Delayed Identification

In this section, we observe the degree to which the delay in identifying a loss's location affects the per-receiver hit rates of our loss location estimation scheme. We define the loss location identification delay to be the time that elapses from the time a loss is detected to the time its location is identified. We only consider delays that are multiples of the IP multicast transmission period $\Delta T \in \mathbb{R}^{\geq 0}$; that is, the identification delay is always presumed to be an integral number $\Delta i \in \mathbb{N}$ of time slots.

In particular, for $r \in R$, $i \in I$, and an identification delay of $\Delta i \Delta T$ time units, we define

Figure 5.9 Virtual Link Trace Representation — Estimation hit rates wrt loss identification delay, cache of size 1 (Trace WRN951030).



$dtime(r)(i) \in I \cup \perp$ and $itime(r)(i) \in I \cup \perp$ as follows:

$$dtime(r)(i) = \begin{cases} \perp & \text{if } loss(r)(i) = 0, \text{ and} \\ \min\{i' \in I \mid i < i' \wedge loss(r)(i') = 0\} & \text{otherwise.} \end{cases} \tag{5.39}$$

$$itime(r)(i) = \begin{cases} \perp & \text{if } loss(r)(i) = 0, \text{ and} \\ dtime(r)(i) + \Delta i & \text{otherwise.} \end{cases} \tag{5.40}$$

We first consider the virtual link trace representation. Figures 5.9 and 5.10 present the hit rates of a couple of receivers of trace WRN951030 with respect to the loss location identification delay for caches of size 1 and 10, respectively. These plots depict the per-receiver hit rates that are least and most affected by the loss location identification delay for the given trace. The plots for the remaining receivers and traces are similar. The dashed lines correspond to the hit rates achieved with delayed detection and immediate loss location identification (presented in Figure 5.6).

Figures 5.9 and 5.10 present the hit rates obtained for a delay of up to 4 seconds. We presume that a loss's location can be identified within the amount of time required to recover from a loss. Several reliable multicast protocols, such as SRM [13] and LMS [34], recover from the vast majority of losses well within 3–4 round-trip-times (RTTs), on average. Thus, presuming a 1 second RTT upper bound, a 4 second upper bound on the location identification delay is reasonable.

We observe that the hit rates of the loss location estimation scheme only slightly decrease as the loss location identification delay increases and the available loss location information becomes less recent. This is because 4 seconds is a short enough time interval for locality to still hold. The hit rates achieved with a cache of size 1 are more sensitive to the loss location identification delay. This is because the larger cache sizes favor the estimation of more lossy locations (links); that is, locations (links) that are probabilistically better candidates for being liable for losses.

We now consider the concrete link trace representation. Figures 5.11 and 5.12 present the hit rates of a couple of receivers of trace WRN951030 as a function of the loss location identification delay for caches of size 1 and 10, respectively. Again, these plots depict the per-receiver hit rates that are least and most affected by the loss location identification delay for the given trace. The effects of delayed loss location identification for the concrete link trace representation are similar to those observed for the virtual link trace representation.

Figure 5.10 Virtual Link Trace Representation — Estimation hit rates wrt loss identification delay, cache of size 10 (Trace WRN951030).

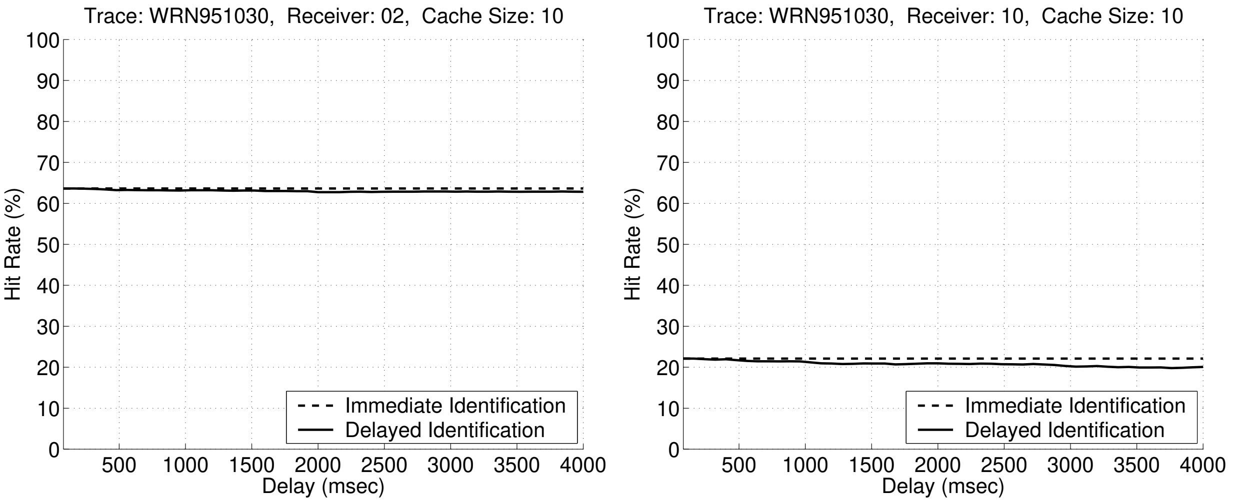
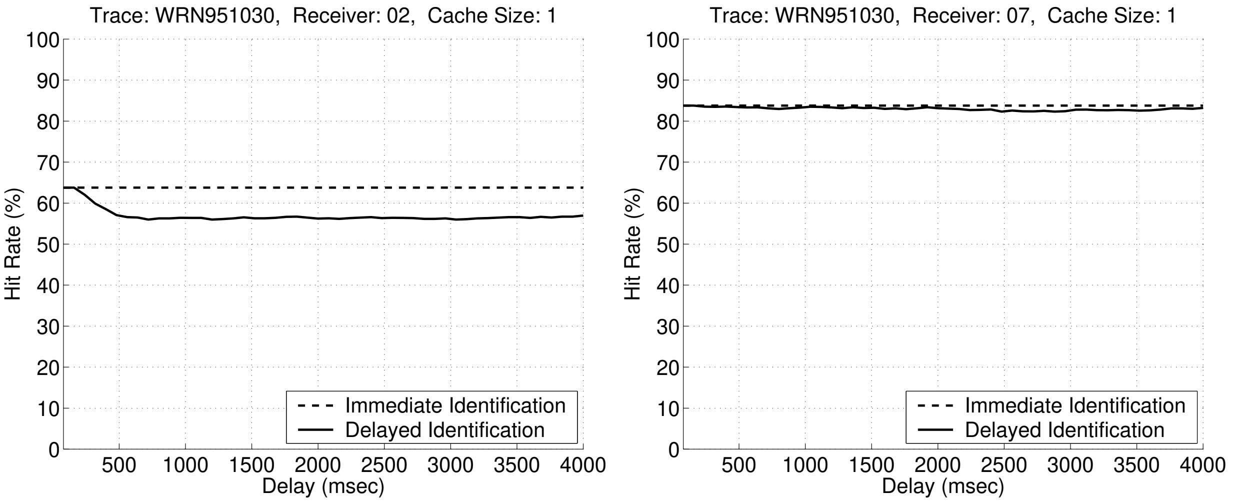


Figure 5.11 Concrete Link Trace Representation — Estimation hit rates wrt loss identification delay, cache of size 1 (Trace WRN951030).



5.5.4 Loss Location Scenario Distribution

In this section, we observe the distribution of loss location estimates among consistent high, accurate, low, and incomparable estimates. Throughout this section, we assume that losses are detected upon the receipt of later packets and that loss location identification is immediate; that is, in this section, we let $dtime(r)(i) \in I \cup \perp$ and $itime(r)(i) \in I \cup \perp$, for $r \in R$ and $i \in I$, be defined as in Section 5.5.2.

We first consider the virtual link trace representation. Figure 5.13 presents the distribution of the estimates of our loss location estimation scheme among consistent high/accurate/low and inconsistent estimate types. With a cache of size 10, the shared hit rates always exceed 10% and exceed 35% for half the traces.

We now present the average distribution of the inconsistent estimates of Figure 5.13 among high, accurate, and low estimates. For each inconsistent estimate produced by our loss location estimation scheme in each trace, we compute the percentage of receivers that share the loss and

Figure 5.12 Concrete Link Trace Representation — Estimation hit rates wrt loss identification delay, cache of size 10 (Trace WRN951030).

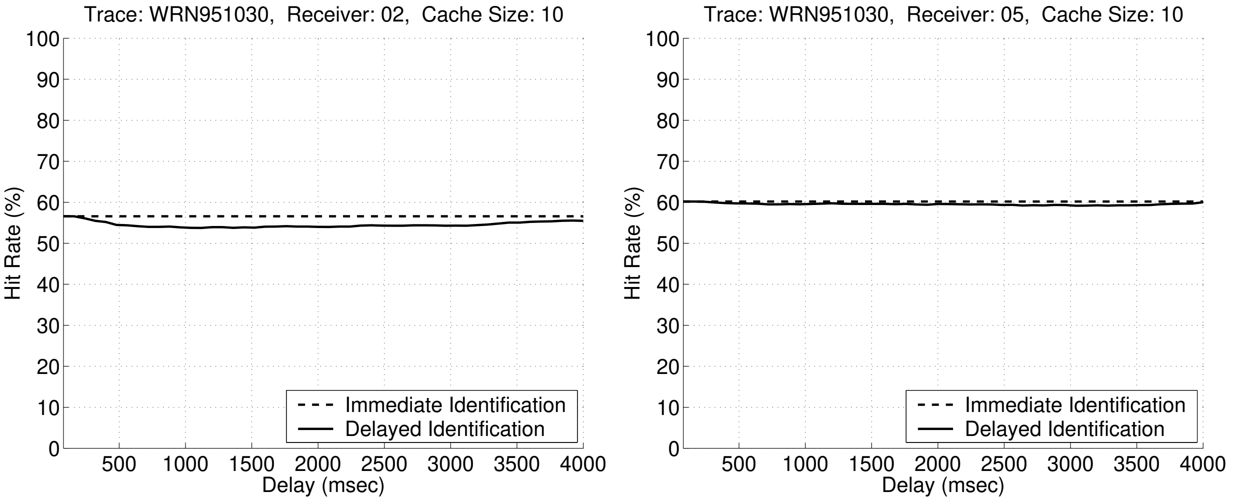


Figure 5.13 Virtual Link Trace Representation — Consistent High/Accurate/Low and Inconsistent Estimate Percentages.

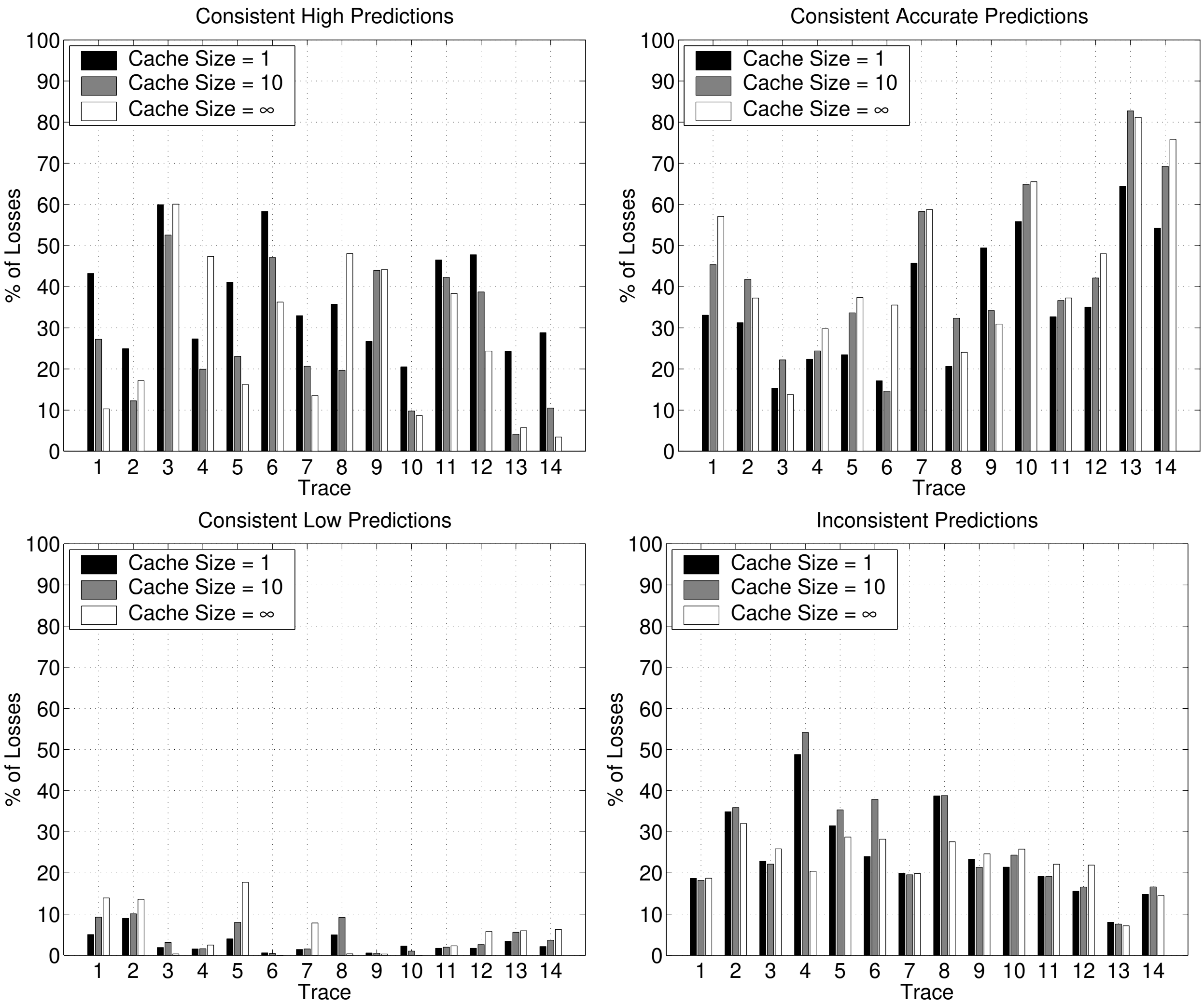
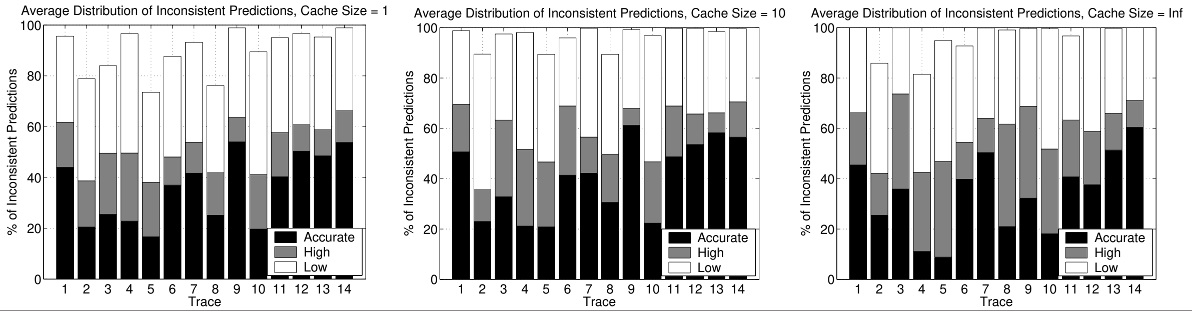


Figure 5.14 Virtual Link Trace Representation — Average Distribution of Inconsistent Estimates.



estimate upstream, accurate, or downstream locations. The averages of these percentages over all inconsistent estimates in each trace are presented in Figure 5.14. The distributions in Figure 5.14 do not add up to 100% because some of the loss location estimates are incomparable to the actual loss locations. The receivers that produce upstream and accurate estimates often account for more than half of the receivers sharing the loss. This indicates that more than half of the losses resulting in inconsistent estimates may be recovered through a caching-based expedited recovery scheme.

We now consider the concrete link trace representation. Figure 5.15 presents the distribution of the estimates among consistent high, accurate, low and inconsistent types. The shared hit rate is substantially higher in the case of the concrete rather than the virtual link trace representation.

The shared hit rates for all cache sizes exceed 25% for all traces. For most of the traces, the cache of size 10 outperforms the cache of size 1. Moreover, its shared hit rate exceeds 70% for half the traces. The infinite cache performs similarly to the cache of size 10. This indicates that, in the case of the concrete link trace representation, a single loss location is responsible for a large percentage of the losses suffered by most of the receivers.

We expect that as the size of the reliable multicast group increases and as the IP multicast transmissions become longer-lived, i) several links will be responsible for large percentages of the losses suffered by individual receivers, and ii) the links responsible for a large percentage of the losses suffered by individual receivers will change over time. Smaller cache sizes would in such cases be preferable so as to adapt quicker to changing loss characteristics and accommodate either multiple or a highly varying number of lossy links.

Figure 5.16 presents the average distribution of the inconsistent estimates of Figure 5.15 among high, accurate, and low estimates. Since loss locations in the concrete link trace representations are never incomparable, the distributions in Figure 5.15 always add up to 100%. Once again, the receivers that produce upstream and accurate estimates often account for more than half of the receivers sharing the loss.

A comparison of Figures 5.13 and 5.15 suggests that the precise identification of the links on which losses occur may be highly beneficial to the effectiveness of caching. Reliable multicast protocols that feature local recovery schemes may be particularly suitable both for precisely identifying the links on which losses occur and for effectively exploiting this information by recovering from losses locally.

Figure 5.17 presents the recovery rate of a caching-based loss recovery scheme as estimated by Equation (5.34) for each of the trace representations and cache sizes of 1, 10, and infinity. For both trace representations, the estimated percentage of losses that are successfully recoverable by a caching-based scheme is substantial. In the case of the virtual link trace representation, at least 65% of the losses in each trace are recoverable through a caching-based scheme. For more than half of the traces, this percentage is above 85%. In the case of the concrete link trace representation,

Figure 5.15 Concrete Link Trace Representation — Consistent High/Accurate/Low and Inconsistent Estimate Percentages.

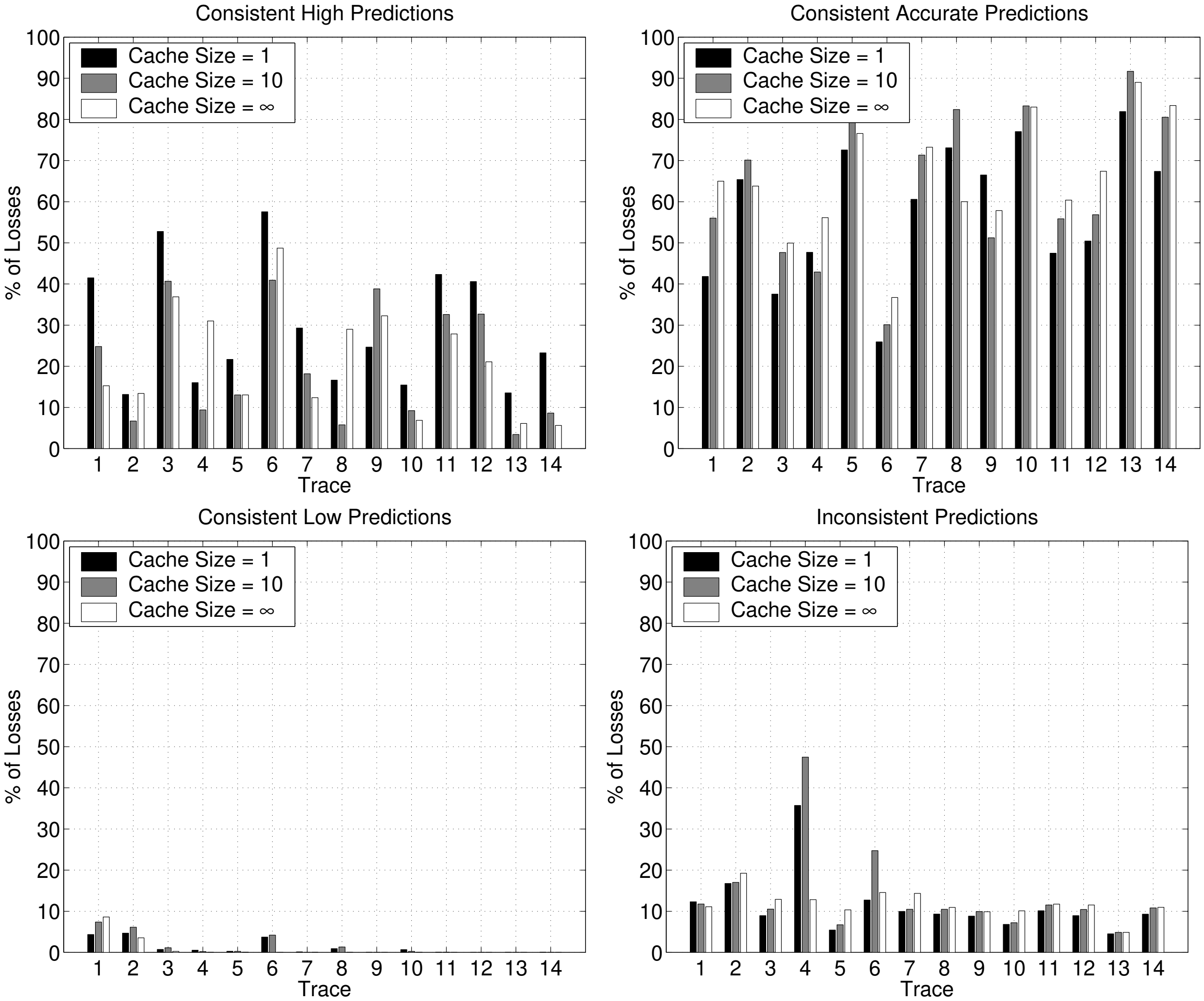


Figure 5.16 Concrete Link Trace Representation — Average Distribution of Inconsistent Estimates.

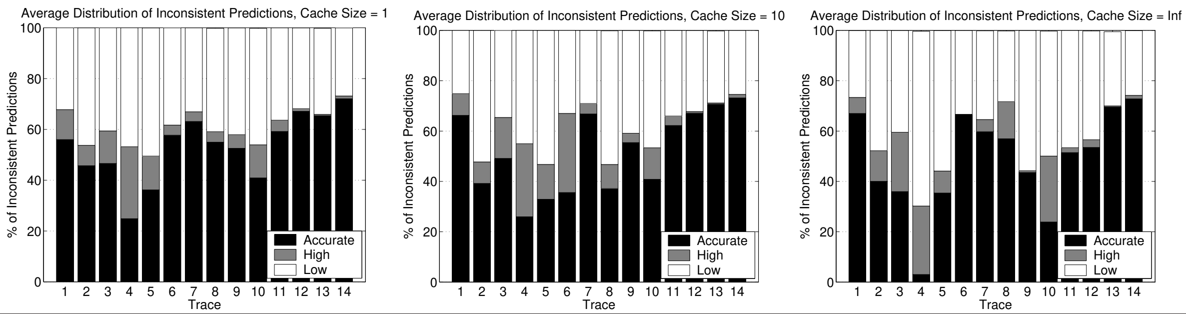
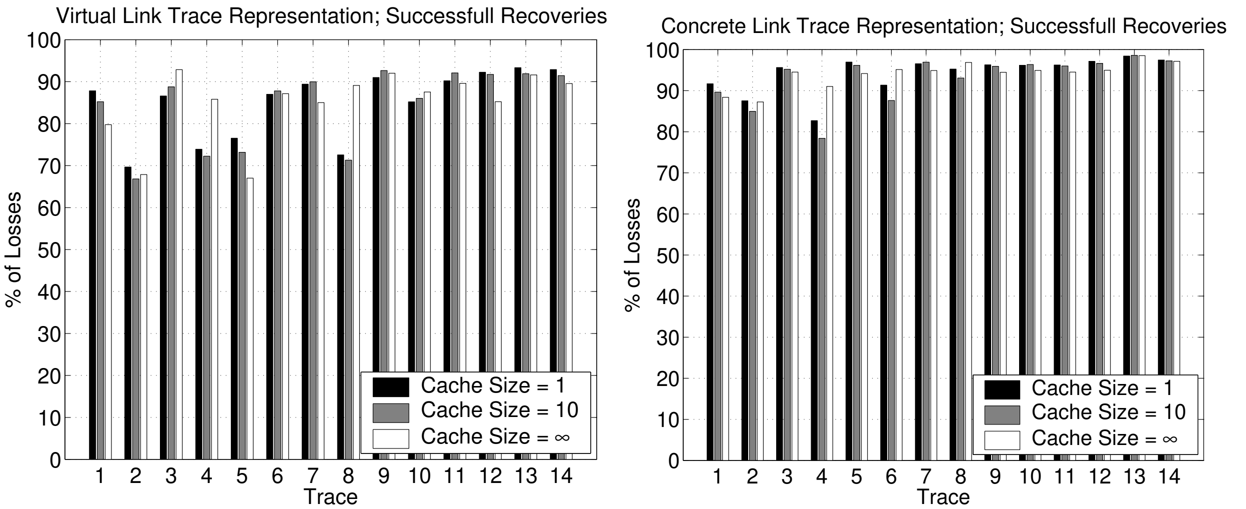


Figure 5.17 Virtual and Concrete Link Trace Representations — Percentage of Successful Expedited Recoveries.



at least 75% of the losses in each trace are recoverable through a caching-based scheme. For more than half of the traces, this percentage is above 95%.

5.5.5 Optimal Cache Size

Finally, we examine the effect of the cache size on the shared hit rate, again assuming that losses are detected upon the receipt of later packets and that loss location identification is immediate; that is, in this section, we let $dtime(r)(i) \in I \cup \perp$ and $itime(r)(i) \in I \cup \perp$, for $r \in R$ and $i \in I$, be defined as in Section 5.5.2.

Figure 5.18 presents the shared hit rates of our caching-based loss location estimation scheme for the virtual link trace representation for different cache sizes. We present the plots for 6 out of the 14 traces; the plots for the other traces are similar. Since an infinite cache size results in a estimation scheme that adapts slowly to changing loss conditions and performs poorly in the case of multiple highly lossy links, we restrict ourselves to relatively small cache sizes.

For many of the traces, the shared hit rate increases with the size of the cache and the infinite cache size outperforms the finite-size caches. Since larger cache sizes favor locations that are responsible for frequent losses, it follows that, for these traces, a large percentage of the losses occur on few links. For others traces, *e.g.*, UCB960424 and WRN951128, finite cache sizes outperform the infinite cache. Overall, caches of modest size, *e.g.*, 11, perform reasonably well for most of the traces.

Figure 5.19 presents the shared hit rate of the loss location estimation scheme for the concrete link trace representation for different cache sizes. Again, we present the plots for 6 out of the 14 traces; the plots for the other traces are similar.

Again, for many of the traces, the shared hit rate increases with the cache size. However, for some traces finite caches outperform the cache of infinite size. Overall, caches of modest size, *e.g.*, 11 or 15, perform reasonably well for most of the traces.

In summary, caches of modest size, *e.g.*, 11 or 15, perform well for most of the traces and both trace representations. This indicates that a caching-based loss recovery scheme would be effective and implementable without prohibitive resource requirements.

Figure 5.18 Virtual Link Trace Representation — Consistent accurate hit rates wrt cache size.

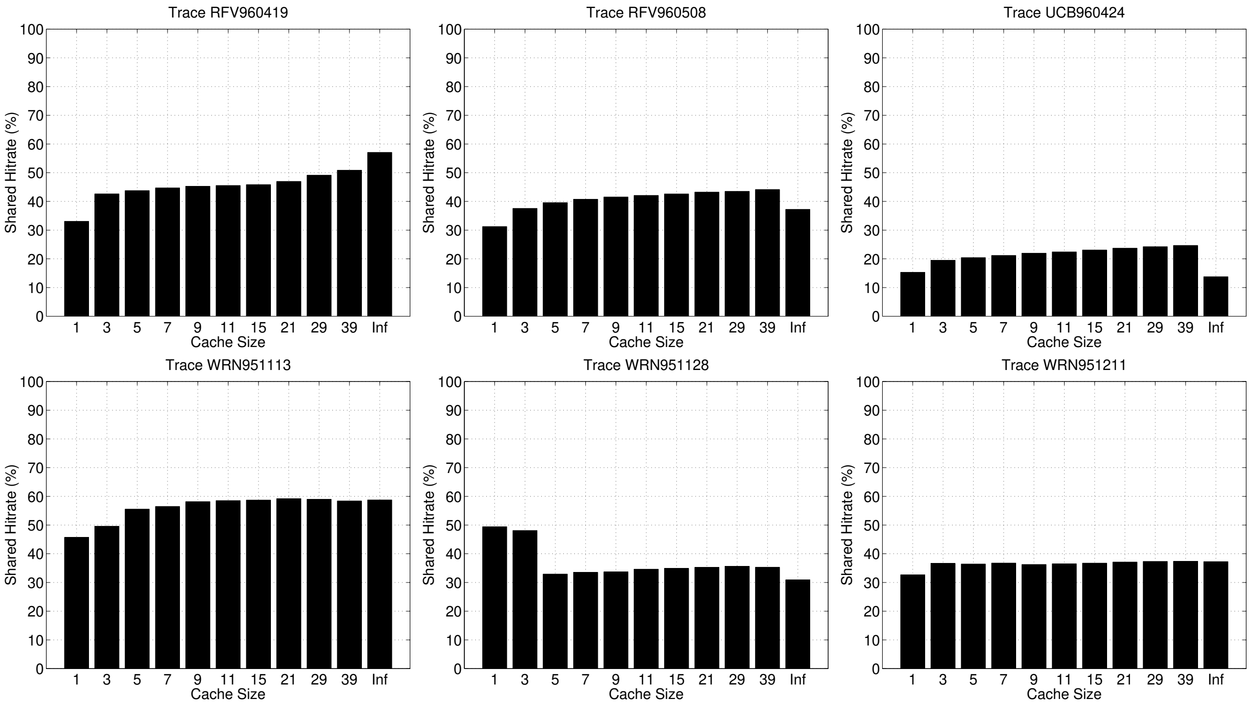
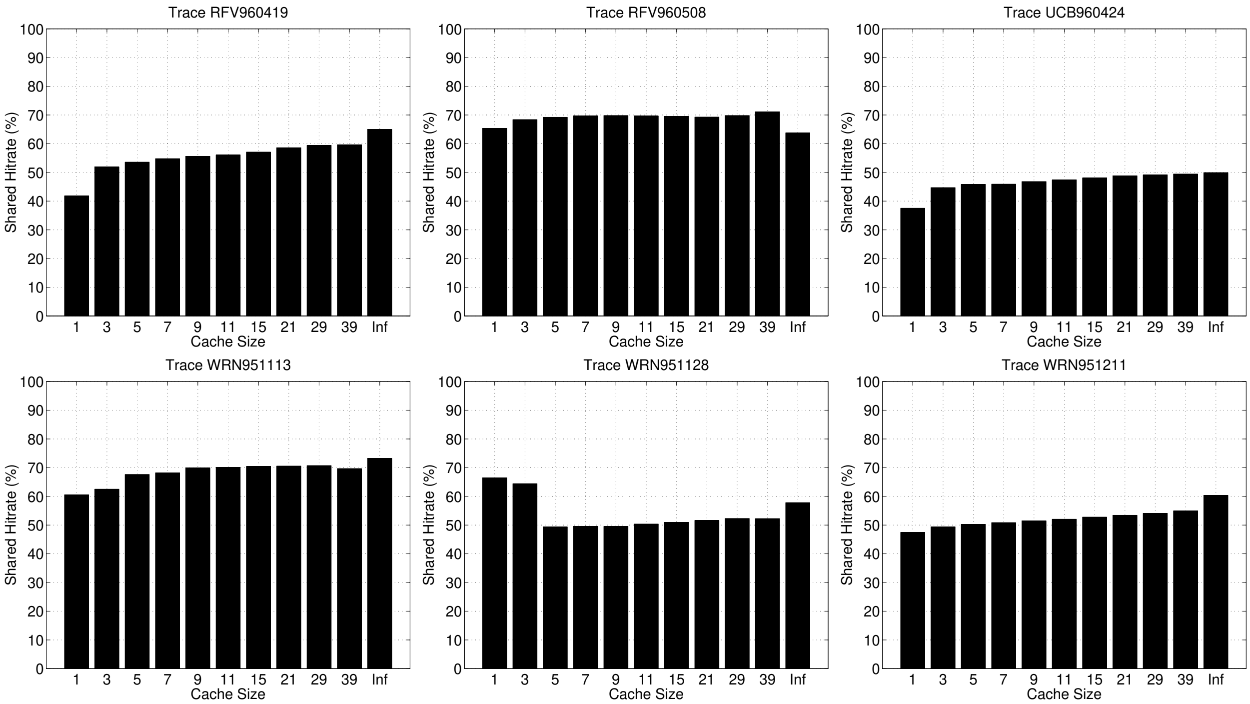


Figure 5.19 Concrete Link Trace Representation — Consistent accurate hit rates wrt cache size.



5.6 Summary, Conclusions, and Future Work

In this chapter, we proposed exploiting packet loss locality within existing or novel reliable multicast protocols through caching. We presented a methodology for estimating the potential effectiveness of caching in multicast loss recovery. Our methodology involved analyzing the performance of a caching-based loss location estimation scheme. We applied our methodology to the IP multicast transmission traces of Yajnik *et al.* [41] and observed that packet loss locality is indeed substantial.

Presuming immediate loss detection and loss location identification, per-receiver hit rates in most cases exceeded 40% and often exceeded 80%. The delay in detecting losses did not substantially affect the per-receiver hit rates, except in the cases where the receivers suffer long loss bursts. The delay in identifying the locations of losses did not substantially affect the hit rates of individual receivers. In most cases, a cache of size 10 outperformed a cache of size 1. The infinite cache performed similarly to the cache of size 10 only when the losses suffered by individual receivers occur predominantly at single locations.

We also observed substantial shared hit rates. In the case of the virtual link trace representation, shared hit rates ranged from 10% to 80%. The shared hit rate for a cache size of 10 exceeded 35% for half the traces. In the case of the concrete link trace representation, shared hit rates ranged from 25% to 90%. The shared hit rate for a cache size of 10 exceeded 70% for half the traces. In our analysis of the effect of cache size on the shared hit rate, caches of modest sizes, *e.g.*, 11 or 15, achieved high shared hit rates for most of the traces and both trace representations.

Most importantly, our estimate of the percentage of losses that would be recoverable through a caching-based recovery scheme was estimated at 65% (75%) for the virtual (respectively, concrete) link trace representation. This suggests that a caching-based loss recovery scheme can be very effective.

The work presented in this chapter may be extended in several directions. First, our methodology can be applied to IP multicast transmissions of larger group size and longer duration. Such work will reveal whether the effectiveness of caching scales. Second, caching schemes that exploit locality can be designed and incorporated in either existing or novel reliable multicast protocols. Finally, the effectiveness of such schemes can be evaluated through simulation or deployment and compared to the expected effectiveness indicated by our observations. The following chapter presents the caching-enhanced SRM protocol (CESRM), which exploits packet loss locality through a caching-based expedited recovery scheme.

Chapter 6

Caching-Enhanced Scalable Reliable Multicast

In this chapter, we present the Caching-Enhanced Scalable Reliable Multicast (CESRM) protocol. CESRM augments the functionality of SRM with a caching-based expedited recovery scheme that exploits packet loss locality in IP multicast transmission losses. CESRM's expedited recovery scheme operates in parallel to SRM's recovery scheme. In this scheme, each receiver caches the requestor/replier pairs that carry out the recovery of its recent losses. The requestor and the replier of the requestor/replier pair that appears most frequently in a receiver's cache, henceforth referred to as the *expeditious requestor* and *expeditious replier*, respectively, are responsible for expeditiously recovering each new loss. Thus, upon detecting a loss, if a receiver considers itself to be the expeditious requestor, then it initiates an expedited recovery for the given packet by unicasting an expedited request to the expeditious replier. The transmission of this expedited request is not delayed. Upon receiving this request, the expeditious replier immediately multicasts the requested packet. Since neither expedited requests nor expedited replies are delayed, packets that are successfully recovered by CESRM's expedited recovery scheme incur minimal recovery latency. When a packet's expedited recovery fails, CESRM falls back onto SRM's recovery scheme. By using SRM's recovery scheme as a fall-back recovery scheme, CESRM recovers from losses no later than SRM; that is, CESRM's recovery latency is bounded from above by SRM's recovery latency.

In CESRM, hosts opportunistically attempt to recover new losses in the manner in which recent losses were recovered. Thus, CESRM's expedited recovery scheme effectively operates in the spirit of the caching-based loss location estimation scheme introduced in Chapter 5. In the case of CESRM, however, the location of a particular loss is identified by the requestor/replier pair that carries out its recovery.

This chapter is organized as follows. To begin, after informally describing its functionality, we present a formal model of the CESRM protocol. We then carry out correctness and timeliness analyses of CESRM. These are analogous to the ones carried out for the SRM protocol in Chapter 3. The correctness analysis states that CESRM delivers appropriate packets to appropriate members of the reliable multicast group. The timeliness analysis states that, under certain timeliness and faultiness assumptions, CESRM guarantees the delivery of the appropriate packets to the appropriate members of the reliable multicast group within a finite amount of time. We also state the worst-case recovery latency incurred by either successful expedited recoveries or successful non-expedited first-round recoveries of CESRM. The substantial difference in the recovery latency afforded by such recoveries demonstrates the performance advantage of CESRM's expedited recovery scheme. Finally, we use trace-driven simulations to evaluate CESRM's performance and compare it to that of SRM.

6.1 Overview of Functionality of CESRM

CESRM extends SRM's loss recovery scheme by caching the optimal requestor/replier pair capable of repairing each prior loss and using this information to expedite the requests and replies of future losses. CESRM enhances the SRM's functionality in two ways: i) each member of the reliable multicast group maintains an optimal requestor/replier cache which is comprised of the optimal requestor/replier pair used to recover each recent loss, and ii) members that are deemed optimal requestors initiate the expedited recovery of future losses by unicasting requests to the optimal repliers upon the detection of a loss; subsequently, such requests induce the immediate multicast retransmission of the requested packets. We proceed by briefly describing CESRM's functionality beyond that of SRM's.

The determination of the optimal requestor/replier pairs is carried out as receivers overhear the request and replies multicast during the recovery of each loss. Requests are annotated with the requestor's distance estimate to the source of the packet. Replies are annotated with the requestor that induced the given reply, this requestor's distance estimate to the source of the packet, and the replier's distance estimate to the requestor that induced the given reply. The optimality of a given requestor/replier pair is based on the estimated recovery delay afforded by the given requestor/replier pair. We choose to represent the recovery delay as the sum of the distance estimate from the requestor to the source and twice the distance estimate from the requestor to the replier (inter-host distances are presumed to be symmetric). This measurement estimates the time that elapses from the transmission of the packet that results in the requestor detecting the given loss to the time the loss is recovered through a retransmission of the packet.

6.1.1 Expedited Recovery

Upon detecting that a packet p is missing, a host h schedules a request for p to be multicast as is done in the SRM protocol. In addition to scheduling this request for p, the host consults the optimal requestor/replier pair cache so as to determine whether it should also schedule an expedited request for the missing packet. If h is considered to be the optimal requestor for packets from the source of p, then it schedules the transmission of an expedited request for p for a point in time `RQST-DELAY` time units in the future. The transmission of an expedited request is delayed so as to avoid the transmission of extraneous requests when packets are temporarily presumed missing due to packet reordering.

Upon receiving an expedited request for a packet p, a host h immediately transmits an expedited reply for p provided it has previously either sent or received p, and a reply for p is neither scheduled, nor pending.

6.1.2 Maintaining the Optimal Requestor/Replier Selection

CESRM maintains per-source optimal requestor/replier pair caches which are comprised of the optimal requestor/replier pairs used to recover packet losses from the respective source. For simplicity, we presume that hosts archive the optimal requestor/replier pairs for all packets recovered. An implementation of CESRM would limit the size of each cache. The cache size would then constitute a parameter of the CESRM protocol. In addition to the optimal requestor/replier pair, each cache entry also records the distance of the optimal requestor to the packet's source and the distance of the optimal replier to the optimal requestor.

The optimal requestor/replier pair of a particular packet from a particular source is initially set upon the reception of a reply for the given packet. All replies are annotated with the requestor that induced them, this requestor's distance estimate to the packet's source, and the replier's distance estimate to the particular requestor. Upon receiving a reply for a particular packet, if an optimal requestor/replier pair for the given packet is not yet cached, then the requestor/replier pair of the reply is considered to be optimal and is cached. If an optimal requestor/replier pair is already cached, then the requestor/replier pair of the reply is considered optimal and replaces the cached pair only if the recovery delay it affords is smaller than that afforded by the cached requestor/replier pair. The recovery delay afforded by a requestor/replier pair is estimated as the sum of the distance estimate from the requestor to the source and the round-trip distance from the requestor to the replier.

CESRM uses two additional message types to maintain its optimal requestor/replier caches: *requestor* and *replier updates*. Suppose that a host h receives an expedited reply from the host r for a packet p. Moreover, suppose that the requestor inducting this reply is host q. After updating the cache based on the recovery delay afforded by the requestor/replier pair $\langle q, r\rangle$, h determines whether it is a preferable requestor for p than q by comparing the recovery delay afforded by the requestor/replier pair $\langle h, r\rangle$ to that afforded by $\langle q, r\rangle$. If the recovery delay afforded by $\langle h, r\rangle$ is smaller than that afforded by $\langle q, r\rangle$, then h schedules the transmission of a requestor update — a message whose purpose is to inform the multicast group members that h is a preferable requestor for p than q. Requestor updates are scheduled in the fashion in which SRM schedules requests. In our example, the requestor update is annotated with h, h's distance estimate to the source s_p of p, r, and r's distance estimate to h; that is, presuming symmetric inter-host distances, the requestor update is annotated with the tuple $\langle h, \hat{d}_{hs_p}, r, \hat{d}_{hr}\rangle$, where $\hat{d}_{hs_p}$ is h's distance estimate to s_p and $\hat{d}_{hr}$ is h's distance estimate to r.

If another requestor update is received prior to the scheduled transmission of the requestor update, then the scheduled requestor update is canceled. If no such update is received prior to the scheduled transmission time of the requestor update, then this update is multicast and $\langle h, r\rangle$ is recorded in h's cache as the optimal requestor/replier pair for p.

A host h' handles the reception of a requestor update for a packet p as follows. If h' suffered the loss of the original transmission of p, has since recovered p, and the requestor/replier pair of p is preferable to the requestor/replier pair already cached for p, then h' replaces its cached pair for p with the requestor/replier pair of p. Otherwise, h' discards the requestor update for p.

Now we describe the use of replier updates. Suppose that a host h receives an expedited reply from the host r for a packet p. Moreover, suppose that h received the original transmission of p; that is, h is capable of retransmitting p. Moreover, suppose that the requestor that induced this reply is host q. Upon receiving this expedited reply, h determines whether it is a preferable replier than r by comparing the recovery delay afforded by the requestor/replier pair $\langle q, h\rangle$ to that afforded by $\langle q, r\rangle$. If the recovery delay afforded by $\langle q, h\rangle$ is smaller than that afforded by $\langle q, r\rangle$, then h schedules the transmission of replier update — a message whose purpose is to inform the multicast group members of a preferable replier for p. Replier updates are scheduled in the fashion in which SRM schedules replies. In our example, the replier update is annotated with q, q's distance to the source of p, h, and h's distance to q; that is, the replier update is annotated with the tuple $\langle q, \hat{d}_{qs_p}, h, \hat{d}_{hq}\rangle$, where $\hat{d}_{qs_p}$ is q's distance estimate to s_p and $\hat{d}_{hq}$ is h's distance estimate to q.

If h receives another replier update for p prior to the transmission of the replier update it has scheduled for p, then it cancels its own replier update. If no such update is received prior to the scheduled transmission time of the replier update scheduled by h, then this update is multicast and $\langle q, h\rangle$ is recorded in h's cache as the optimal requestor/replier pair for p.

A host h' handles the reception of a replier update for a packet p as follows. If h' suffered the loss of the original transmission of p, has since recovered p, and the requestor/replier pair of p is preferable to the one already cached by h' for p, then h' replaces its cached pair for p with the requestor/replier pair of p. Otherwise, h' discards the requestor update for p.

6.1.3 Deducing the Optimal Requestor/Replier Pairs

Several strategies may be used to ascertain the optimal requestor/replier pair for a particular packet loss based on the archived requestor/replier pairs of recent losses. We begin by describing perhaps the simplest such strategy, which we refer to as the *most recent loss* strategy. In this strategy, the optimal requestor/replier pair for the given loss is chosen to be the pair for the most recent packet that was lost and, subsequently, recovered. In effect, this scheme presumes that the loss occurred at the same location as the loss that was most recently suffered and, subsequently, recovered. The rationale behind this scheme is that if indeed a loss occurs at the same location as the loss that was most recently suffered and, subsequently, recovered, then it may be recovered in the same manner.

More sophisticated strategies may take into account the cached requestor/replier pairs of a fixed number of most recent packets that have been lost and, subsequently, recovered. In effect, such a scheme caches the requestor/replier pairs of a fixed number of losses and uses this information to deduce the optimal requestor/replier pair for a new loss. One possible strategy in deducing the optimal requestor/replier pair for a new loss, which we refer to as the *most frequent loss* strategy, is to choose the pair that appears most frequently in the replier/requestor pair cache.

It is plausible that more sophisticated strategies may be able to better ascertain the optimal requestor/replier pair for a particular loss. For purposes of simplicity, in this chapter we model and analyze the most recent loss strategy. Our work focuses on the demonstration of our modeling and analysis techniques rather than the design of the best performing optimal requestor/replier pair selection strategy.

6.2 Formal Model of the CESRM Protocol

In this section, we present a formal model for the CESRM protocol. Since CESRM is a caching-enhanced version of SRM, it shares many of SRM's components. Figure 6.1 depicts the interaction of the components of CESRM and the environment. The client at each host is modeled by the RM-CLIENT$_h$ timed I/O automaton of Chapter 3. The reporting and membership components of Chapter 4 carry over unchanged to the specification of the CESRM protocol. The remaining components of SRM presented in Chapter 4 are enhanced so as to capture the enhanced functionality of CESRM. We proceed to specify the enhancements pertaining to the IP buffer component of SRM, the recovery component of SRM, and to the IP automaton. In the upcoming definitions and TIOA models of CESRM's components, the parts pertaining to SRM are typeset in gray and those pertaining to CESRM are typeset in black.

6.2.1 Preliminary Definitions

Figure 6.2 presents a list of set definitions that are used in the specification of the CESRM protocol. The sets *Pending-Rqsts*, *Scheduled-Rqsts*, *Pending-Repls*, and *Scheduled-Repls* are comprised of tuples corresponding to all possible pending requests, scheduled requests, pending replies, and scheduled replies, respectively. The tuples comprising the set *Scheduled-Repls* differ from those comprising the *Scheduled-Repls* set in Chapter 4. In the case of our specification of the CESRM protocol, such tuples include an additional component that corresponds to the distance estimate from the requestor of the packet to the packet's source.

Figure 6.1 Interface of all components involved in the reliable multicast service.

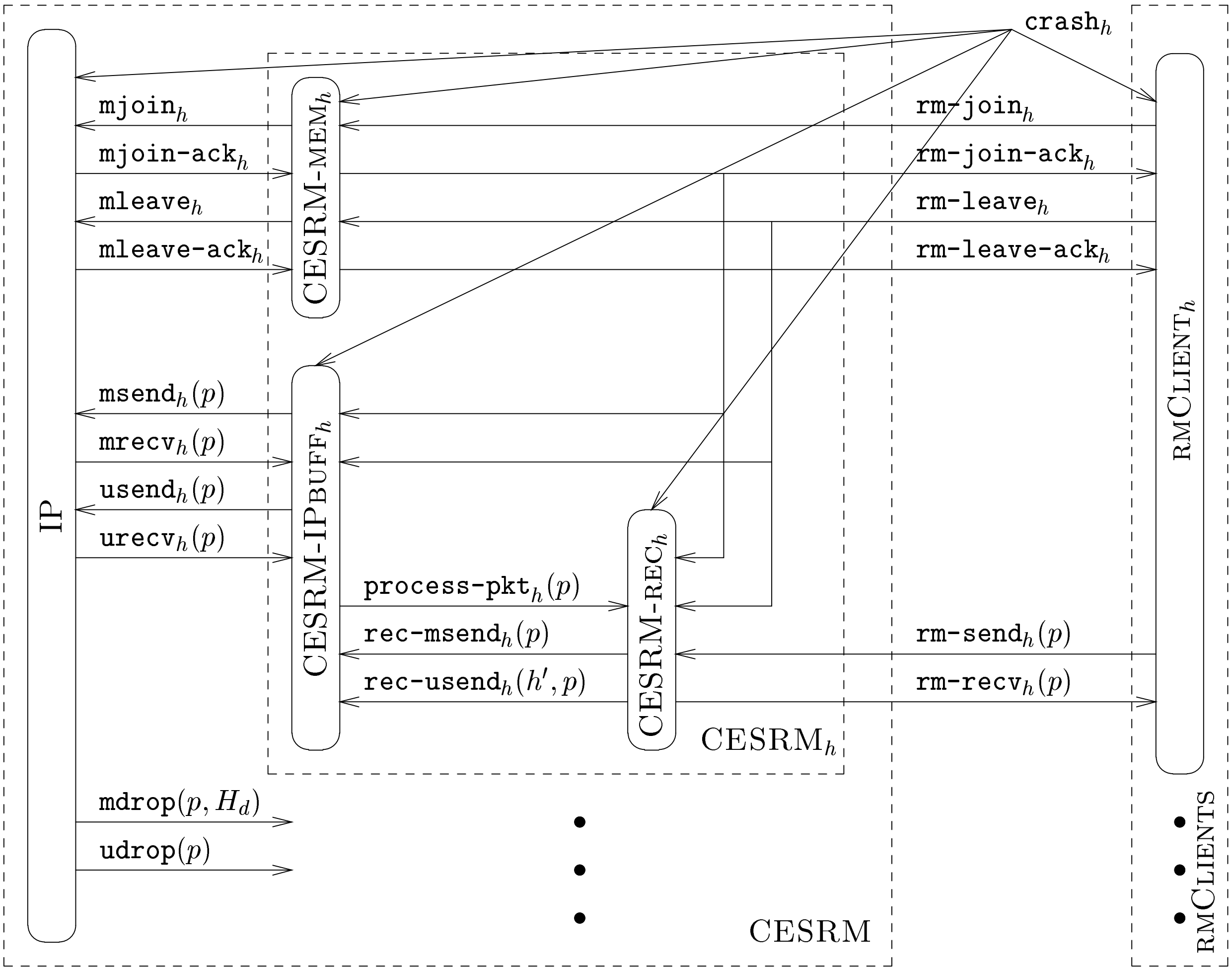


Figure 6.2 CESRM Preliminary Definitions

$\mathit{Pending\text{-}Rqsts} = \{\langle s, i, t\rangle \mid s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}\}$
$\mathit{Scheduled\text{-}Rqsts} = \{\langle s, i, t, k\rangle \mid s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}\}$
$\mathit{Pending\text{-}Repls} = \{\langle s, i, t\rangle \mid s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}\}$
$\mathit{Scheduled\text{-}Repls} = \{\langle s, i, t, q, d_{qs}\rangle \mid s, q \in H, i \in \mathbb{N}, t, d_{qs} \in \mathbb{R}^{\geq 0}\}$

$\mathit{Recovery\text{-}Tuples} = H \times \mathbb{R}^{\geq 0} \times H \times \mathbb{R}^{\geq 0}$
$\mathit{Expedited\text{-}Rqsts} = \{\langle s, i, t, \mathit{rec\text{-}tpl}\rangle \mid s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}, \mathit{rec\text{-}tpl} \in \mathit{Recovery\text{-}Tuples}\}$
$\mathit{Rqst\text{-}Updates} = \{\langle s, i, t, \mathit{rec\text{-}tpl}\rangle \mid s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}, \mathit{rec\text{-}tpl} \in \mathit{Recovery\text{-}Tuples}\}$
$\mathit{Repl\text{-}Updates} = \{\langle s, i, t, \mathit{rec\text{-}tpl}\rangle \mid s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}, \mathit{rec\text{-}tpl} \in \mathit{Recovery\text{-}Tuples}\}$

The set *Recovery-Tuples* is comprised of the set of all possible optimal recovery tuples. Such tuples consist of the optimal requestor, an estimate of this requestor's distance to the source of the packet in question, the optimal replier, and an estimate of this replier's distance to the optimal requestor. The sets *Expedited-Rqsts*, *Rqst-Updates*, and *Repl-Updates* are comprised of all tuples of the form $\langle s, i, t, \mathit{rec\text{-}tpl}\rangle$, where $s \in H$, $i \in \mathbb{N}$, $t \in \mathbb{R}^{\geq 0}$, and $\mathit{rec\text{-}tpl} \in \mathit{Recovery\text{-}Tuples}$. Such tuples specify the time t at which either an EXP-RQST, RQST-UPDATE, or REPL-UPDATE control packet pertaining to the recovery of the packet $\langle s, i\rangle$ is scheduled for transmission and the optimal requestor/replier pair *rec-tpl* pertaining to this recovery.

Figure 6.3 presents a list of set definitions that specify the format of the various types of packets used in our specification of the CESRM protocol. The set $P_{\text{RM-CLIENT}}$ represents the set of packets that may be transmitted by the client processes using the reliable multicast service. This set of packets is identical to that defined in Chapter 4. The set P_{CESRM} is comprised of all packets whose format is that used by the reliable multicast process. The format of each packet $p \in P_{\mathrm{CESRM}}$ depends on its type *type*(p).

`DATA` and `SESS` packets are unchanged from those defined in Chapter 4. `RQST` packets for the CESRM protocol are augmented with the additional operation *dist2src*(p). This operation extracts the distance of the sender of the request (the requestor) to the source of the packet being requested. `REPL` packets for the CESRM protocol are augmented with the additional operation *rec-tpl*(p). This operation extracts the recovery tuple pertaining to the particular reply.

In addition to the packet types of Chapter 4, we introduce the following additional packet types: `EXP-RQST`, `EXP-REPL`, `RQST-UPDATE`, and `REPL-UPDATE`.

When the packet p is an expedited request, that is, when *type*(p) = `EXP-RQST`, p supports the operations *sender*(p), *source*(p), *seqno*(p), *id*(p), and *rec-tpl*(p). These operations extract the sender, source, sequence number, identifier, and optimal recovery tuple of p, respectively. The optimal recovery tuple of p corresponds to the optimal recovery tuple known to the sender of p at the point in time when the transmission of p was scheduled.

When the packet p is an expedited request, that is, when *type*(p) = `EXP-REPL`, p supports the operations *sender*(p), *requestor*(p), *source*(p), *seqno*(p), *id*(p), *data*(p), *strip*(p), and *rec-tpl*(p). These operations extract the sender, requestor, source, sequence number, identifier, data segment, ADU packet, and optimal recovery tuple of p, respectively.

When the packet p is either a request or a reply update, that is, when *type*$(p) \in \{$`RQST-UPDATE`, `REPL-UPDATE`$\}$, p supports the operations *sender*(p), *source*(p), *seqno*(p), *id*(p), and *rec-tpl*(p). These operations extract the sender, source, sequence number, identifier, and optimal recovery tuple of p, respectively.

6.2.2 The IP Buffer Component — CESRM-IPBUFF$_h$

The CESRM-IPBUFF$_h$ timed I/O automaton specifies the IP buffer component of the reliable multicast process. Figures 6.4 and 6.5 present the signature, the variables, and the discrete transitions of CESRM-IPBUFF$_h$. The CESRM-IPBUFF$_h$ automaton augments the functionality of the SRM-IPBUFF$_h$ automaton of Section 4.3.3. In this section, we only describe the aspects of CESRM-IPBUFF$_h$ that pertain to its unicast functionality; that is, the interaction with CESRM-REC$_h$ and IP pertaining to the transmission of unicast packets.

Variables The set *usend-buff* is used to buffer all packets to be unicast using the underlying IP service.

Input Actions The input action $\mathtt{urecv}_h(p)$ models the reception of the unicast packet p from the underlying IP service. If the host h is a member of the reliable multicast group, then the $\mathtt{urecv}_h(p)$ action decapsulates the packet p and adds the result to the *recv-buff* buffer. Thus, the contents of the packet p may subsequently be processed by the other components of the reliable multicast process.

Each input action $\mathtt{rec\text{-}usend}_h(h', p)$ is performed by the recovery component at h so as to unicast the packet p using the underlying IP service to the host h'. If the host h is a member of the reliable multicast group, then CESRM-IPBUFF$_h$ encapsulates h, *seqno*, h', and p into a unicast packet, buffers this packet in *usend-buff* for unicast transmission using the underlying IP service, and increments *seqno*. In effect, the encapsulation of p annotates it with the host h, the value

Figure 6.3 CESRM Packet Definitions

$P_{\text{RM-CLIENT}}$ = Set of packets such that $\forall\, p \in P_{\text{RM-CLIENT}}$:
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$data(p) \in \{0,1\}^*$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$suffix(p) = \{\langle s,i\rangle \in H \times \mathbb{N} \mid source(p) = s \wedge seqno(p) \leq i\}$

$P_{\text{RM-CLIENT}}[h] = \{p \in P_{\text{RM-CLIENT}} \mid source(p) = h\}$

P_{CESRM} = Set of packets such that $\forall\, p \in P_{\text{CESRM}}$:
$type(p) \in \{\texttt{DATA}, \texttt{RQST}, \texttt{REPL}, \texttt{SESS}, \texttt{EXP-RQST}, \texttt{EXP-REPL}, \texttt{RQST-UPDATE}, \texttt{REPL-UPDATE}\}$

DATA :
$sender(p) \in H$
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$data(p) \in \{0,1\}^*$
$strip(p) \in P_{\text{RM-CLIENT}}$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$

RQST :
$sender(p) \in H$
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$dist2src(p) \in \mathbb{R}^{\geq 0}$

REPL :
$sender(p) \in H$
$requestor(p) \in H$
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$data(p) \in \{0,1\}^*$
$strip(p) \in P_{\text{RM-CLIENT}}$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$rec\text{-}tpl(p) \in Recovery\text{-}Tuples$

SESS :
$sender(p) \in H$
$time\text{-}sent(p) \in \mathbb{R}^{\geq 0}$
$dist\text{-}rprt?(p) \subseteq H$
$dist\text{-}rprt(p,h) \in \{\langle t,t'\rangle \mid t,t' \in \mathbb{R}^{\geq 0}\}$, for all $h \in H$
$seqno\text{-}rprts(p) \subseteq \{\langle s,i\rangle \mid s \in H, i \in \mathbb{N}\}$

EXP-RQST :
$sender(p) \in H$
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$rec\text{-}tpl(p) \in Recovery\text{-}Tuples$

EXP-REPL :
$sender(p) \in H$
$requestor(p) \in H$
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$data(p) \in \{0,1\}^*$
$strip(p) \in P_{\text{RM-CLIENT}}$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$rec\text{-}tpl(p) \in Recovery\text{-}Tuples$

RQST-UPDATE :
$sender(p) \in H$
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$rec\text{-}tpl(p) \in Recovery\text{-}Tuples$

REPL-UPDATE :
$sender(p) \in H$
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$rec\text{-}tpl(p) \in Recovery\text{-}Tuples$

$P_{\text{IPUCAST-CLIENT}}$ = Set of packets such that $\forall\, p \in P_{\text{IPUCAST-CLIENT}}$:
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$dest(p) \in H$
$strip(p) \in \{0,1\}^*$

$P_{\text{IPMCAST-CLIENT}}$ = Set of packets such that $\forall\, p \in P_{\text{IPMCAST-CLIENT}}$:
$source(p) \in H$
$seqno(p) \in \mathbb{N}$
$strip(p) \in \{0,1\}^*$

P_{IPMCAST} = Set of packets such that $\forall\, pkt \in P_{\text{IPMCAST}}$:
$strip(pkt) \in P_{\text{IPMCAST-CLIENT}}$
$intended(pkt) \subseteq H$
$completed(pkt) \subseteq H$
$dropped(pkt) \subseteq H$

Figure 6.4 The CESRM-IP$\textsc{buff}_h$ Automaton — Signature

Parameters:

$h \in H$

Actions:

input
`crash`$_h$
`rm-join-ack`$_h$
`rm-leave`$_h$
`mrecv`$_h(p)$, for $p \in P_{\text{IPmcast-Client}}$
`urecv`$_h(p)$, for $p \in P_{\text{IPucast-Client}}$
`rec-msend`$_h(p)$, for $p \in P_{\text{CESRM}}$
`rec-usend`$_h(h', p)$, for $h' \in H, h' \neq h, p \in P_{\text{CESRM}}$

output
`process-pkt`$_h(p)$, for $p \in P_{\text{CESRM}}$
`msend`$_h(p)$, for $p \in P_{\text{IPmcast-Client}}$
`usend`$_h(p)$, for $p \in P_{\text{IPucast-Client}}$

time-passage
$\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

Figure 6.5 The CESRM-IP$\textsc{buff}_h$ Automaton — Variables and Discrete Transitions

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status \in$ *CESRM-Status*, initially $status =$ `idle`
$seqno \in \mathbb{N}$, initially $seqno = 0$
$recv\text{-}buff \subseteq P_{\text{CESRM}}$, initially $recv\text{-}buff = \emptyset$
$msend\text{-}buff \subseteq P_{\text{IPmcast-Client}}$, initially $msend\text{-}buff = \emptyset$
$usend\text{-}buff \subseteq P_{\text{IPucast-Client}}$, initially $usend\text{-}buff = \emptyset$

Discrete Transitions:

input `crash`$_h$
eff $status :=$ `crashed`

input `rm-join-ack`$_h$
eff **if** $status \neq$ `crashed` **then** $status :=$ `member`

input `rm-leave`$_h$
eff **if** $status \neq$ `crashed` **then**
Reinitialize all variables except now and $seqno$.

input `mrecv`$_h(p)$
eff **if** $status =$ `member` **then** $recv\text{-}buff \cup= \{strip(p)\}$

input `urecv`$_h(p)$
eff **if** $status =$ `member` **then** $recv\text{-}buff \cup= \{strip(p)\}$

input `rec-msend`$_h(p)$
eff **if** $status =$ `member` **then**
$msend\text{-}buff \cup= \{comp\text{-}IPmcast\text{-}pkt(h, seqno, p)\}$
$seqno := seqno + 1$

input `rec-usend`$_h(h', p)$
eff **if** $status =$ `member` **then**
$usend\text{-}buff \cup= \{comp\text{-}IPucast\text{-}pkt(h, seqno, h', p)\}$
$seqno := seqno + 1$

output `process-pkt`$_h(p)$
pre $status =$ `member` $\wedge\, p \in recv\text{-}buff$
eff $recv\text{-}buff \setminus= \{p\}$

output `msend`$_h(p)$
pre $status =$ `member` $\wedge\, p \in msend\text{-}buff$
eff $msend\text{-}buff \setminus= \{p\}$

output `usend`$_h(p)$
pre $status =$ `member` $\wedge\, p \in usend\text{-}buff$
eff $usend\text{-}buff \setminus= \{p\}$

time-passage $\nu(t)$
pre $status =$ `crashed`
$\vee(recv\text{-}buff = \emptyset \wedge usend\text{-}buff = \emptyset \wedge msend\text{-}buff = \emptyset)$
eff $now := now + t$

of $seqno$, and the destination host h'. Since the variable $seqno$ is persistent across host joins and leaves, packets transmitted by the CESRM-IP$\textsc{buff}_h$ automata, for $h \in H$, are unique.

Output Actions The output action `usend`$_h(p)$ models the transmission of the packet p using the underlying IP unicast service. It is enabled when the host h is a member of the group and the packet p is in the *usend-buff* buffer. Its effects are to remove the packet p from the *usend-buff* buffer.

6.2.3 The Recovery Component — CESRM-$\textsc{rec}_h$

The CESRM-$\textsc{rec}_h$ timed I/O automaton specifies the recovery component of the reliable multicast service. Figure 6.6 presents the signature of CESRM-$\textsc{rec}_h$, that is, its parameters, and actions. Figure 6.7 presents the variables of CESRM-$\textsc{rec}_h$. Figures 6.8, 6.10, 6.11, 6.12, and 6.13 present the discrete transitions of CESRM-$\textsc{rec}_h$. Since the CESRM-$\textsc{rec}_h$ automaton is an enhancement

Figure 6.6 The CESRM-REC_h Automaton — Signature

Parameters:

$h \in H, C_1, C_2, C_3, D_1, D_2, D_3 \in \mathbb{R}^{\geq 0}$, `DFLT-DIST`, `RQST-DELAY` $\in \mathbb{R}^{\geq 0}$, `SESS-PERIOD` $\in \mathbb{R}^{+}$

Actions:

input
- $\texttt{crash}_h$
- $\texttt{rm-join-ack}_h$
- $\texttt{rm-leave}_h$
- $\texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$
- $\texttt{process-pkt}_h(p)$, for $p \in P_{\text{CESRM}}$

internal
- $\texttt{schdl-rqst}_h(s, i)$, for $s \in H, i \in \mathbb{N}$
- $\texttt{send-sess}_h$
- $\texttt{send-rqst}_h(s, i)$, for $s \in H, i \in \mathbb{N}$
- $\texttt{send-repl}_h(s, i)$, for $s \in H, i \in \mathbb{N}$
- $\texttt{send-exp-rqst}_h(s, i)$, for $s \in H, i \in \mathbb{N}$
- $\texttt{send-rqst-update}_h(s, i)$, for $s \in H, i \in \mathbb{N}$
- $\texttt{send-repl-update}_h(s, i)$, for $s \in H, i \in \mathbb{N}$

output
- $\texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$
- $\texttt{rec-msend}_h(p)$, for $p \in P_{\text{CESRM}}$
- $\texttt{rec-usend}_h(h', p)$, for $h' \in H, h' \neq h, p \in P_{\text{CESRM}}$

time-passage
- $\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

of the SRM-REC_h automaton of Section 4.3.4, we only describe the functionality of CESRM-REC_h beyond that of SRM-REC_h. Once again, in order to provide the appropriate context, the description of each of the parameters of CESRM-REC_h is deferred to appropriate places within the description of its variables and actions.

Variables

Each set *usend-buff*$(h') \subseteq P_{\text{CESRM}}$, for $h' \in H, h' \neq h$, is used to buffer the packets that are to be subsequently unicast to h' using the underlying IP service. More precisely, the set *usend-buff*(h') is comprised of the expedited requests to be unicast by the reliable multicast process at h to the host h'.

The variable *recovered-pkts?* $\subseteq H \times \mathbb{N}$ identifies the packets that have been received as either `REPL` or `EXP-REPL` packets; that is, packets whose original transmissions have been lost but have since been recovered.

The sets *expedited-rqsts* $\subseteq$ *Expedited-Rqsts*, *rqst-updates* $\subseteq$ *Rqst-Updates*, and *Repl-Updates* $\subseteq$ *Repl-Updates* are comprised of tuples that identify the packets for which either expedited requests, request updates, or reply updates, respectively, have been scheduled for transmission. Tuples comprising these sets identify each packet, its scheduled transmission time, and the optimal requestor/replier pair pertaining to the recovery of the given packet.

Each of the variables *rec-tpl*$(h', i') \in$ *Recovery-Tuples* $\cup \perp$, for $h' \in H, h' \neq h$ and $i' \in \mathbb{N}$, identifies the optimal requestor/replier pair for the packet $\langle h', i' \rangle$. The variable *rec-tpl*(h', i') is defined, *i.e.*, not equal to $\perp$, only if the packet $\langle h', i' \rangle$ is a proper packet that has been recovered and, consequently, whose optimal requestor/replier pair has been determined.

Input Actions

As in the case of the SRM-REC_h automaton, the input action $\texttt{process-pkt}_h(p)$ models the processing of the packet p by CESRM-REC_h. The packet p is processed only when the host h is a member of the reliable multicast group. We proceed by describing the effects of $\texttt{process-pkt}_h(p)$ depending on the type of the packet p. In our description of the effects of $\texttt{process-pkt}_h(p)$, we only describe the additional effects to those of the $\texttt{process-pkt}_h(p)$ action of SRM-REC_h. Throughout

Figure 6.7 The CESRM-REC$_h$ Automaton — Variables

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status \in$ *CESRM-Status*, initially $status =$ `idle`
$rep\text{-}deadline \in \mathbb{R}^{\geq 0} \cup \perp$, initially $rep\text{-}deadline = \perp$
$dist\text{-}rprt(h') \in \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0} \cup \perp$, for all $h' \in H, h' \neq h$, initially $dist\text{-}rprt(h') = \perp$
$dist(h') \in \mathbb{R}^{\geq 0} \times \mathbb{R}^{\geq 0}$, for all $h' \in H, h' \neq h$, initially $dist(h') = \langle 0, \texttt{DFLT-DIST} \rangle$
$min\text{-}seqno(h') \in \mathbb{N} \cup \perp$, for all $h' \in H$, initially $min\text{-}seqno(h') = \perp$, for all $h' \in H$
$max\text{-}seqno(h') \in \mathbb{N} \cup \perp$, for all $h' \in H$, initially $max\text{-}seqno(h') = \perp$, for all $h' \in H$
$archived\text{-}pkts \subseteq P_{\text{RM-CLIENT}} \times \mathbb{R}^{\geq 0}$, initially $archived\text{-}pkts = \emptyset$
$to\text{-}be\text{-}requested? \subseteq H \times \mathbb{N}$, initially $to\text{-}be\text{-}requested? = \emptyset$
$pending\text{-}rqsts \subseteq Pending\text{-}Rqsts$, initially $pending\text{-}rqsts = \emptyset$
$scheduled\text{-}rqsts \subseteq Scheduled\text{-}Rqsts$, initially $scheduled\text{-}rqsts = \emptyset$
$pending\text{-}repls \subseteq Pending\text{-}Repls$, initially $pending\text{-}repls = \emptyset$
$scheduled\text{-}repls \subseteq Scheduled\text{-}Repls$, initially $scheduled\text{-}repls = \emptyset$
$to\text{-}be\text{-}delivered \subseteq P_{\text{RM-CLIENT}}$, initially $to\text{-}be\text{-}delivered = \emptyset$
$msend\text{-}buff \subseteq P_{\text{CESRM}}$, initially $msend\text{-}buff = \emptyset$
$\boldsymbol{usend\text{-}buff(h') \subseteq P_{\text{CESRM}}}$, **for all** $\boldsymbol{h' \in H, h' \neq h}$, **initially** $\boldsymbol{usend\text{-}buff = \emptyset}$, **for all** $\boldsymbol{h' \in H, h' \neq h}$
$\boldsymbol{recovered\text{-}pkts? \subseteq H \times \mathbb{N}}$, **initially** $\boldsymbol{recovered\text{-}pkts? = \emptyset}$
$\boldsymbol{expedited\text{-}rqsts \subseteq Expedited\text{-}Rqsts}$, **initially** $\boldsymbol{expedited\text{-}rqsts = \emptyset}$
$\boldsymbol{rqst\text{-}updates \subseteq Rqst\text{-}Updates}$, **initially** $\boldsymbol{rqst\text{-}updates = \emptyset}$
$\boldsymbol{repl\text{-}updates \subseteq Repl\text{-}Updates}$, **initially** $\boldsymbol{repl\text{-}updates = \emptyset}$
$\boldsymbol{rec\text{-}tpl(h', i') \in Recovery\text{-}Tuples \cup \perp}$, **for** $\boldsymbol{h' \in H, h' \neq h}$ **and** $\boldsymbol{i' \in \mathbb{N}}$, **initially** $\boldsymbol{rec\text{-}tpl = \perp}$, **for** $\boldsymbol{h' \in H, h' \neq h}$ **and** $\boldsymbol{i' \in \mathbb{N}}$

Derived Variables:

$dist?(h') = d$, for $d \in \mathbb{R}^{\geq 0}$, such that $dist(h') = \langle t, d \rangle$, for some $t \in \mathbb{R}^{\geq 0}$, for all $h' \in H$
$dist\text{-}rprt = \cup_{h' \in H, h' \neq h, dist\text{-}rprt(h') \neq \perp} \{\langle h', t_{sent}, t_{rcvd} \rangle \mid dist\text{-}rprt(h') = \langle t_{sent}, t_{rcvd} \rangle\}$
$max\text{-}seqno = \cup_{h' \in H, h' \neq h, max\text{-}seqno(h') \neq \perp} \{\langle h', max\text{-}seqno(h') \rangle\}$
for all $h' \in H$, $proper?(h') = \begin{cases} \emptyset & \text{if } min\text{-}seqno(h') = \perp \\ \{\langle s, i \rangle \in H \times \mathbb{N} \mid s = h', min\text{-}seqno(h') \leq i\} & \text{otherwise} \end{cases}$
for all $h' \in H$, $window?(h') = \begin{cases} \emptyset & \text{if } min\text{-}seqno(h') = \perp \\ \{\langle s, i \rangle \in H \times \mathbb{N} \mid s = h', min\text{-}seqno(h') \leq i \leq max\text{-}seqno(h')\} & \text{otherwise} \end{cases}$
$archived\text{-}pkts? = \{\langle s, i \rangle \in H \times \mathbb{N} \mid \exists\, p \in P_{\text{RM-CLIENT}}, t \in \mathbb{R}^{\geq 0} : \langle p, t \rangle \in archived\text{-}pkts \wedge id(p) = \langle s, i \rangle\}$
$archived\text{-}pkts?(h') = \{\langle s, i \rangle \in archived\text{-}pkts? \mid s = h'\}$, for all $h' \in H$
$to\text{-}be\text{-}requested?(h') = \{\langle s, i \rangle \in to\text{-}be\text{-}requested? \mid s = h'\}$, for all $h' \in H$
$to\text{-}be\text{-}delivered? = \{\langle s, i \rangle \in H \times \mathbb{N} \mid \exists\, p \in to\text{-}be\text{-}delivered : \langle s, i \rangle = id(p)\}$
$to\text{-}be\text{-}delivered?(h') = \{\langle s, i \rangle \in to\text{-}be\text{-}delivered? \mid s = h'\}$, for all $h' \in H$
$scheduled\text{-}rqsts? = \{\langle s, i \rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N} : \langle s, i, t, k \rangle \in scheduled\text{-}rqsts\}$
$scheduled\text{-}rqsts?(h') = \{\langle s, i \rangle \in scheduled\text{-}rqsts? \mid s = h'\}$, for all $h' \in H$
$scheduled\text{-}repls? = \{\langle s, i \rangle \in H \times \mathbb{N} \mid \exists\, t, \boldsymbol{d_{qs}} \in \mathbb{R}^{\geq 0}, q \in H : \langle s, i, t, q, \boldsymbol{d_{qs}} \rangle \in scheduled\text{-}repls\}$
$scheduled\text{-}repls?(h') = \{\langle s, i \rangle \in scheduled\text{-}repls? \mid s = h'\}$, for all $h' \in H$
$pending\text{-}rqsts? = \{\langle s, i \rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0} : now \leq t \wedge \langle s, i, t \rangle \in pending\text{-}rqsts\}$
$pending\text{-}repls? = \{\langle s, i \rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0} : now \leq t \wedge \langle s, i, t \rangle \in pending\text{-}repls\}$
$\boldsymbol{recovered\text{-}pkts?(h') = \{\langle s, i \rangle \in recovered\text{-}pkts? \mid s = h'\}}$, **for all** $\boldsymbol{h' \in H}$
$\boldsymbol{expedited\text{-}rqsts? = \{\langle s, i \rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0}, rec\text{-}tpl \in Recovery\text{-}Tuples : \langle s, i, t, rec\text{-}tpl \rangle \in expedited\text{-}rqsts\}}$
$\boldsymbol{rqst\text{-}updates?(h') = \{\langle s, i \rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0}, rec\text{-}tpl \in Recovery\text{-}Tuples : \langle s, i, t, rec\text{-}tpl \rangle \in rqst\text{-}updates\}}$, **for all** $\boldsymbol{h' \in H}$
$\boldsymbol{repl\text{-}updates?(h') = \{\langle s, i \rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0}, rec\text{-}tpl \in Recovery\text{-}Tuples : \langle s, i, t, rec\text{-}tpl \rangle \in repl\text{-}updates\}}$, **for all** $\boldsymbol{h' \in H}$

our presentation of the effects of $\texttt{process-pkt}_h(p)$, we let $s_p \in H$ and $i_p \in \mathbb{N}$ denote the source and the sequence number pertaining to the packet p.

First, consider the case where p is a `DATA` packet. In addition to the effects of the $\texttt{process-pkt}_h(p)$ action of SRM-REC$_h$, the $\texttt{process-pkt}_h(p)$ of CESRM-REC$_h$ cancels any expedited requests for the packet p that are scheduled for transmission.

Second, consider the case where p is a `RQST` packet. In this case, the effects of the $\texttt{process-pkt}_h(p)$ action mimics the effects of the $\texttt{process-pkt}_h(p)$ action of SRM-REC$_h$ with the exception that when the request alerts the loss of the packet $\langle s_p, i_p \rangle$ and h considers itself to be the optimal requestor for $\langle s_p, i_p \rangle$, the $\texttt{process-pkt}_h(p)$ action also schedules the transmission of an expedited request for $\langle s_p, i_p \rangle$. The scheduling of an expedited request in this case is not necessary. The sender of the packet p has already detected the loss of $\langle s_p, i_p \rangle$ and is possibly a preferable requestor for $\langle s_p, i_p \rangle$. The alternative decision of opting out of recovering the packet $\langle s_p, i_p \rangle$ expeditiously is also plausible strategy for the CESRM protocol.

Third, consider the case where p is a `REPL` packet. In addition to the effects of the $\texttt{process-pkt}_h(p)$ action of SRM-REC$_h$, the $\texttt{process-pkt}_h(p)$ of CESRM-REC$_h$ cancels any expedited requests for the packet p that are scheduled for transmission and updates the optimal requestor/replier pair for the packet $\langle s_p, i_p \rangle$. If p is a packet that has been recovered by h and the optimal recovery tuple cached at h for p is either undefined or affords a worse recovery latency than the recovery tuple annotating p, then $\texttt{process-pkt}_h(p)$ sets the optimal recovery tuple for $\langle s_p, i_p \rangle$ to the one annotating p and cancels any scheduled requestor and replier updates for $\langle s_p, i_p \rangle$. These updates pertain to the requestor/replier tuple that has just been replaced and is, thus, considered to be stale.

Fourth, consider the case where p is a `EXP-RQST` packet. The packet p is processed only when the host h is a member of the reliable multicast group and the packet p is a proper packet; that is, when $status = \texttt{member}$ and $min\text{-}seqno(s_p) \neq \perp \wedge min\text{-}seqno(s_p) \leq i_p$, where $\langle s_p, i_p \rangle = id(p)$. If h considers itself to be the optimal replier for $\langle s_p, i_p \rangle$, the packet $\langle s_p, i_p \rangle$ is archived at h, and there are no pending replies for $\langle s_p, i_p \rangle$, then h cancels any scheduled replies for $\langle s_p, i_p \rangle$ and schedules the immediate transmission of an expedited reply for $\langle s_p, i_p \rangle$. In particular, the $\texttt{process-upkt}_h(p)$ action composes an expedited reply packet for $\langle s_p, i_p \rangle$ and adds it to the buffer *msend-buff*. The operation $comp\text{-}exp\text{-}repl\text{-}pkt(s_p, i_p, rec\text{-}tpl)$ composes an `EXP-REPL` packet from h for $\langle s_p, i_p \rangle$. Moreover, the $\texttt{process-upkt}_h(p)$ action adds a tuple corresponding to $\langle s_p, i_p \rangle$ to the set *pending-repls*. The reply abstinence timeout of this pending reply is set to $now + D_3 d_{rq}$, where d_{rq} is the distance estimate from the optimal replier r to the optimal requestor q of the given expedited recovery; namely, d_{rq} is the distance estimate from r to the host q whose expedited request induced the particular expedited reply for $\langle s_p, i_p \rangle$.

Fifth, consider the case where p is a `EXP-REPL` packet. In addition to the effects pertaining to a `REPL` packet of the $\texttt{process-pkt}_h(p)$ action of SRM-REC$_h$, the $\texttt{process-pkt}_h(p)$ cancels any expedited requests for p that are scheduled for transmission, updates the optimal recovery tuple for $\langle s_p, i_p \rangle$, and, when appropriate, schedules the transmission of either a requestor or a replier update packets. If the packet $\langle s_p, i_p \rangle$ has been recovered at h and either the cached optimal recovery tuple for $\langle s_p, i_p \rangle$ is undefined or it affords a worse recovery latency than the recovery tuple annotating p, then $\texttt{process-pkt}_h(p)$ sets the optimal recovery tuple for $\langle s_p, i_p \rangle$ to the one annotating p and cancels any scheduled requestor and replier updates for $\langle s_p, i_p \rangle$. These updates pertain to the requestor/replier tuple that has just been replaced and is, thus, considered to be stale. Moreover, if no requestor updates are scheduled for transmission, h is different from the requestor q of the cached optimal recovery tuple for $\langle s_p, i_p \rangle$, and h is preferable to q in terms of the afforded recovery latency, then $\texttt{process-pkt}_h(p)$ schedules the transmission of a requestor update packet. This control packet is used to inform the members of the reliable multicast group that shared the loss of $\langle s_p, i_p \rangle$ of a preferable requestor/replier pair.

If h has received the original transmission of the packet $\langle s_p, i_p \rangle$, that is, the packet $\langle s_p, i_p \rangle$ has been archived but is not recorded as having been recovered by h, no replier updates are scheduled for transmission, h is different from the replier r of the cached optimal recovery tuple for $\langle s_p, i_p \rangle$, and h is a preferable replier to r in terms of the afforded recovery latency, then $\texttt{process-pkt}_h(p)$ schedules the transmission of a replier update packet. This control packet is used to inform the members of the reliable multicast group that shared the loss of $\langle s_p, i_p \rangle$ of a preferable requestor/replier pair.

Sixth, consider the case where p is a `RQST-UPDATE` packet. The packet p is only processed when the host h is a member of the reliable multicast group and the packet $\langle s_p, i_p \rangle$ is proper. If p is a packet that has been recovered at the host h and either the optimal recovery tuple for p is undefined or it affords a worse recovery latency than the recovery tuple annotating p, then $\texttt{process-pkt}_h(p)$ sets the optimal recovery tuple for $\langle s_p, i_p \rangle$ to the one annotating p. Moreover, if there is a scheduled

Figure 6.8 The CESRM-REC_h Automaton — Discrete Transitions

```
input crash_h
eff  status := crashed

input rm-join-ack_h
eff  if status ≠ crashed then
       status := member
       rep-deadline :∈ now + (0, SESS-PERIOD]

input rm-leave_h
eff  if status ≠ crashed then
       Reinitialize all variables except now.

input rm-send_h(p)
eff  if status = member ∧ h = source(p) then
       ⟨s_p, i_p⟩ = id(p)
       \\ Record foremost DATA packet
       if min-seqno(s_p) =⊥ then min-seqno(s_p) := i_p
       \\ Only consider next packet
       if max-seqno(s_p) =⊥
          ∨i_p = max-seqno(s_p) + 1
       then
          max-seqno(s_p) := i_p
          \\ Archive packet
          archived-pkts ∪= {⟨p, now⟩}
          \\ Compose data packet
          msend-buff ∪= {comp-data-pkt(p)}

output rm-recv_h(p)
pre  status = member ∧ p ∈ to-be-delivered
       ∧(∄ p' ∈ to-be-delivered :
          source(p') = source(p) ∧ seqno(p') < seqno(p))
eff  to-be-delivered \= {p}

output rec-msend_h(p)
pre  status = member ∧ p ∈ msend-buff
eff  msend-buff \= {p}

output rec-usend_h(h', p)
pre  status = member ∧ p ∈ usend-buff(h')
eff  usend-buff(h') \= {p}
```

replier update that affords worse recovery latency, then $\texttt{process-pkt}_h(p)$ cancels this update. The action $\texttt{process-pkt}_h(p)$ also cancels any scheduled requestor updates; such updates are suppressed by the requestor update being processed in the spirit of SRM. Finally, $\texttt{process-pkt}_h(p)$ adds any trailing missing packets to the set *to-be-requested?*, so that a request for each of them may subsequently be scheduled.

Finally, the effects of the action $\texttt{process-pkt}_h(p)$ when p is a `REPL-UPDATE` packet are analogous to those of a `RQST-UPDATE` packet.

Output Actions

Each output action $\texttt{rec-usend}_h(h', p)$, for $h' \in H, h' \neq h, p \in P_{\text{CESRM}}$, hands off the packet p from CESRM-REC_h to CESRM-IPBUFF_h so that it may subsequently be unicast to h' using the underlying IP service. The precondition of the $\texttt{rec-usend}_h(h', p)$ action is that the host h is a member of the reliable multicast group and p is in the *usend-buff*(h') buffer. Its effects are to remove p from the *usend-buff*(h') buffer.

Internal Actions

The effects of the action $\texttt{schdl-rqst}_h(s, i)$, for $s \in H, s \neq h, i \in \mathbb{N}$, are augmented so as to schedule an expedited request for the packet $\langle s, i\rangle$. Such a request is scheduled in addition to one scheduled as part of the original SRM protocol. The operation *opt-rec-tpl*(s) determines the optimal requestor/replier pair for the source s given the archive of optimal requestor/replier pairs for each of the packets that have been recovered so far; the optimal requestor/replier pairs are recorded by the variables *rec-tpl*(s, i'), for $i' \in \mathbb{N}$. In this chapter, we designate the recovery tuple of the most recent packet that has been recovered by h to be the optimal recovery tuple; that is, we assume a cache of size 1. In particular, throughout this chapter, we let *opt-rec-tpl*$(s) =$ *rec-tpl*(s, i^*), where $i^* = \max\{i' \in \mathbb{N} \mid \textit{rec-tpl}(s, i') \neq \perp\}$.

Once the optimal requestor/replier pair for the packet $\langle s, i\rangle$ has been determined, if the host h

Figure 6.9 The CESRM-$\textsc{rec}_h$ Automaton — Discrete Transitions (Cont'd)

input $\texttt{process-pkt}_h(p)$
where $\mathit{type}(p) = \texttt{DATA}$
eff **if** $\mathit{status} = \texttt{member}$ **then**
$\langle s_p, i_p\rangle = \mathit{id}(p)$
\\ Record foremost DATA packet
if $h \neq s_p \wedge \mathit{min\text{-}seqno}(s_p) = \perp$ **then**
$\mathit{min\text{-}seqno}(s_p) := i_p$; $\mathit{max\text{-}seqno}(s_p) := i_p$
\\ Only consider proper packets
if $\mathit{min\text{-}seqno}(s_p) \neq \perp \wedge \mathit{min\text{-}seqno}(s_p) \leq i_p$ **then**
\\ Archive and deliver packet
if $h \neq s_p \wedge \langle s_p, i_p\rangle \notin \mathit{archived\text{-}pkts?}$ **then**
$\mathit{archived\text{-}pkts} \cup= \{\langle \mathit{strip}(p), \mathit{now}\rangle\}$
if $h \neq s_p$ **then** $\mathit{to\text{-}be\text{-}delivered} \cup= \{\mathit{strip}(p)\}$
\\ Pkt need not be requested
$\mathit{to\text{-}be\text{-}requested?} \setminus= \{\langle s_p, i_p\rangle\}$
\\ Cancel any scheduled requests and replies
$\mathit{scheduled\text{-}rqsts} \setminus= \{\langle s_p, i_p, t, k\rangle \mid t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}\}$
$\mathit{scheduled\text{-}repls} \setminus=$
$\{\langle s_p, i_p, t, q, \boldsymbol{d_{qs_p}}\rangle \mid t, \boldsymbol{d_{qs_p}} \in \mathbb{R}^{\geq 0}, q \in H\}$
\\ Cancel any pending requests
$\mathit{pending\text{-}rqsts} \setminus= \{\langle s_p, i_p, t\rangle \mid t \in \mathbb{R}^{\geq 0}\}$
\\ Discover any trailing missing packets
if $h \neq s_p \wedge \mathit{max\text{-}seqno}(s_p) < i_p$ **then**
$\mathit{to\text{-}be\text{-}requested?} \cup=$
$\{\langle s_p, i\rangle \mid i \in \mathbb{N}, \mathit{max\text{-}seqno}(s_p) < i < i_p\}$
$\mathit{max\text{-}seqno}(s_p) := i_p$
\\ **Cancel any scheduled expedited requests**
$\boldsymbol{\mathit{expedited\text{-}rqsts} \setminus= \{\langle s_p, i_p, t, \mathit{rec\text{-}tpl}\rangle \mid}$
$\boldsymbol{t \in \mathbb{R}^{\geq 0}, \mathit{rec\text{-}tpl} \in \mathit{Recovery\text{-}Tuples}\}}$

input $\texttt{process-pkt}_h(p)$
where $\mathit{type}(p) = \texttt{SESS}$
eff **if** $\mathit{status} = \texttt{member}$ **then**
$s_p := \mathit{sender}(p)$
if $\mathit{dist\text{-}rprt}(s_p) = \perp$ **then**
$\mathit{dist\text{-}rprt}(s_p) := \langle \mathit{time\text{-}sent}(p), \mathit{now}\rangle$
else
$\langle t_{sent}, t_{rcvd}\rangle := \mathit{dist\text{-}rprt}(s_p)$
if $t_{sent} \leq \mathit{time\text{-}sent}(p)$ **then**
$\mathit{dist\text{-}rprt}(s_p) := \langle \mathit{time\text{-}sent}(p), \mathit{now}\rangle$
if $h \in \mathit{dist\text{-}rprt?}(p)$ **then**
$\langle t_{sent}, t_{delayed}\rangle := \mathit{dist\text{-}rprt}(p, h)$
$\langle t_{rprt}, t_{dist}\rangle := \mathit{dist}(s_p)$
if $t_{rprt} \leq t_{sent}$ **then**
$t'_{dist} := (\mathit{now} - t_{delayed} - t_{sent})/2$
$\mathit{dist}(s_p) := \langle t_{sent}, t'_{dist}\rangle$
foreach $\langle h'', i''\rangle \in \mathit{seqno\text{-}rprts}(p)$ **do:**
if $\mathit{min\text{-}seqno}(h'') \neq \perp$ **then**
\\ Discover any trailing missing packets
if $h \neq h'' \wedge \mathit{max\text{-}seqno}(h'') < i''$ **then**
$\mathit{to\text{-}be\text{-}requested?} \cup=$
$\{\langle h'', i\rangle \mid i \in \mathbb{N}, \mathit{max\text{-}seqno}(h'') < i < i''\}$
$\mathit{max\text{-}seqno}(h'') := i''$

is the optimal requestor, then it schedules the transmission of an expedited request for $\langle s, i\rangle$ for a point in time that is `RQST-DELAY` time units in the future. Expedited requests are delayed in this fashion so as to prevent the premature transmission of expedited requests when packets are temporarily considered missing due to the reordering of packets within the transmission stream from the source.

Each internal action $\texttt{send-exp-rqst}_h(s, i)$, for $s \in H, i \in \mathbb{N}$, models the expiration of the transmission timeout of a scheduled expedited request for the packet $\langle s, i\rangle$. The precondition of $\texttt{send-exp-rqst}_h(s, i)$ is that the host h is a member of the reliable multicast group and a previously scheduled expedited request for the packet $\langle s, i\rangle$ has just expired; that is, there is a tuple $\langle s, i, t, \mathit{rec\text{-}tpl}\rangle$ in *expedited-rqsts* such that $t = \mathit{now}$. Let the tuple $\langle s, i, t, \mathit{rec\text{-}tpl}\rangle$ be the element of *expedited-rqsts* corresponding to the packet $\langle s, i\rangle$. $\texttt{send-exp-rqst}_h(s, i)$ composes an expedited request packet and adds this packet to the buffer *usend-buff*. The operation $\mathit{comp\text{-}exp\text{-}rqst\text{-}pkt}(s, i, \mathit{rec\text{-}tpl})$ composes an `EXP-RQST` packet for the packet $\langle s, i\rangle$. Finally, the action $\texttt{send-exp-rqst}_h(s, i)$ removes the tuple corresponding to $\langle s, i\rangle$ from the set *expedited-rqsts*.

Each internal action $\texttt{send-rqst-update}_h(s, i)$, for $s \in H, i \in \mathbb{N}$, models the expiration of the transmission timeout of a scheduled request update for the packet $\langle s, i\rangle$. The precondition of $\texttt{send-rqst-update}_h(s, i)$ is that the host h is a member of the reliable multicast group and a previously scheduled request update for the packet $\langle s, i\rangle$ has just expired; that is, there is a tuple $\langle s, i, t, \mathit{rec\text{-}tpl}\rangle$ in *rqst-updates* such that $t = \mathit{now}$. $\texttt{send-rqst-update}_h(s, i)$ sets the $\mathit{rec\text{-}tpl}(s, i)$ variable to the recovery tuple *rec-tpl* pertaining to the particular request update, composes a request update packet, and adds it to the buffer *msend-buff*. The operation $\mathit{comp\text{-}rqst\text{-}update\text{-}pkt}(s, i, \mathit{rec\text{-}tpl})$ composes a `RQST-UPDATE` packet from h for the packet $\langle s, i\rangle$. Finally, the action $\texttt{send-rqst-update}_h(s, i)$ removes the tuple $\langle s, i, t, \mathit{rec\text{-}tpl}\rangle$ from the set *rqst-updates*.

Each internal action $\texttt{send-repl-update}_h(s, i)$, for $s \in H, i \in \mathbb{N}$, models the expiration of the

Figure 6.10 The CESRM-REC_h Automaton — Discrete Transitions (Cont'd)

input $\texttt{process-pkt}_h(p)$
where $type(p) = \texttt{RQST}$
eff **if** $status = \texttt{member}$ **then**
$\langle s_p, i_p\rangle = id(p)$
`\\ Only consider proper packets`
if $min\text{-}seqno(s_p) \neq \perp \wedge min\text{-}seqno(s_p) \leq i_p$ **then**
if $\langle s_p, i_p\rangle \in archived\text{-}pkts?$ **then**
if $\langle s_p, i_p\rangle \notin scheduled\text{-}repls?$
$\wedge \langle s_p, i_p\rangle \notin pending\text{-}repls?$
then
`\\ Schedule a new reply`
$q := sender(p)$; $d_{qs_p} := dist2src(p)$
$d_{repl} := dist?(q)$
$t_{repl} :\in now + [D_1 d_{repl}, (D_1 + D_2) d_{repl}]$
$scheduled\text{-}repls \cup= \{\langle s_p, i_p, t_{repl}, q, d_{qs_p}\rangle\}$
else
if $h \neq s_p$ **then**
if $\langle s_p, i_p\rangle \notin scheduled\text{-}rqsts?$ **then**
`\\ Schedule a backed-off request`
$k_r := 2$; $d_r := dist?(s_p)$
$t_r :\in now + 2^{k_r - 1}[C_1 d_r, (C_1 + C_2) d_r]$
$scheduled\text{-}rqsts \cup= \{\langle s_p, i_p, t_r, k_r\rangle\}$
`\\ Pkt request has been scheduled`
$to\text{-}be\text{-}requested? \setminus= \{\langle s_p, i_p\rangle\}$
`\\ A request becomes pending`
$pending\text{-}rqsts \setminus= \{\langle s_p, i_p, t_*\rangle \mid t_* \in \mathbb{R}^{\geq 0}\}$
$t_r := now + 2^{k_r - 1} C_3 d_r$
$pending\text{-}rqsts \cup= \{\langle s_p, i_p, t_r\rangle\}$
`\\ Schedule an expedited request`
if $opt\text{-}rec\text{-}tpl(s_p) \neq \perp$ **then**
$\langle q, d_{qs}, r, d_{rq}\rangle := opt\text{-}rec\text{-}tpl(s_p)$
if $h = q$ **then**
$t := now + \texttt{RQST-DELAY}$
$expedited\text{-}rqsts \cup=$
$\{\langle s_p, i_p, t, \langle q, d_{qs}, r, d_{rq}\rangle\rangle\}$
else
if $\langle s_p, i_p\rangle \notin pending\text{-}rqsts?$ **then**
`\\ Backoff scheduled request`
choose $t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}$
where $\langle s_p, i_p, t, k\rangle \in scheduled\text{-}rqsts$
$scheduled\text{-}rqsts \setminus= \{\langle s_p, i_p, t, k\rangle\}$
$k_r := k + 1$; $d_r := dist?(s_p)$
$t_r :\in now + 2^{k_r - 1}[C_1 d_r, (C_1 + C_2) d_r]$
$scheduled\text{-}rqsts \cup= \{\langle s_p, i_p, t_r, k_r\rangle\}$
`\\ A request becomes pending`
$pending\text{-}rqsts \setminus= \{\langle s_p, i_p, t_*\rangle \mid t_* \in \mathbb{R}^{\geq 0}\}$
$t_r := now + 2^{k_r - 1} C_3 d_r$
$pending\text{-}rqsts \cup= \{\langle s_p, i_p, t_r\rangle\}$
`\\ Discover any trailing missing packets`
if $h \neq s_p \wedge max\text{-}seqno(s_p) < i_p$ **then**
$to\text{-}be\text{-}requested? \cup=$
$\{\langle s_p, i\rangle \mid i \in \mathbb{N}, max\text{-}seqno(s_p) < i < i_p\}$
$max\text{-}seqno(s_p) := i_p$

input $\texttt{process-pkt}_h(p)$
where $type(p) = \texttt{REPL}$
eff **if** $status = \texttt{member}$ **then**
$\langle s_p, i_p\rangle = id(p)$
`\\ Only consider proper packets`
if $min\text{-}seqno(s_p) \neq \perp \wedge min\text{-}seqno(s_p) \leq i_p$ **then**
`\\ A reply becomes pending`
$pending\text{-}repls \setminus= \{\langle s_p, i_p, t_*\rangle \mid t_* \in \mathbb{R}^{\geq 0}\}$
$t_{repl} := now + D_3\, dist?(requestor(p))$
$pending\text{-}repls \cup= \{\langle s_p, i_p, t_{repl}\rangle\}$
`\\ Archive and deliver packet`
if $h \neq s_p \wedge \langle s_p, i_p\rangle \notin archived\text{-}pkts?$ **then**
$recovered\text{-}pkts? \cup= \{\langle s_p, i_p\rangle\}$
$archived\text{-}pkts \cup= \{\langle strip(p), now\rangle\}$
if $h \neq s_p$ **then** $to\text{-}be\text{-}delivered \cup= \{strip(p)\}$
`\\ Pkt need not be requested`
$to\text{-}be\text{-}requested? \setminus= \{\langle s_p, i_p\rangle\}$
`\\ Cancel any scheduled requests and replies`
$scheduled\text{-}rqsts \setminus= \{\langle s_p, i_p, t, k\rangle \mid t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}\}$
$scheduled\text{-}repls \setminus=$
$\{\langle s_p, i_p, t, q, d_{qs_p}\rangle \mid t, d_{qs_p} \in \mathbb{R}^{\geq 0}, q \in H\}$
`\\ Cancel any pending requests`
$pending\text{-}rqsts \setminus= \{\langle s_p, i_p, t\rangle \mid t \in \mathbb{R}^{\geq 0}\}$
`\\ Discover any trailing missing packets`
if $h \neq s_p \wedge max\text{-}seqno(s_p) < i_p$ **then**
$to\text{-}be\text{-}requested? \cup=$
$\{\langle s_p, i\rangle \mid i \in \mathbb{N}, max\text{-}seqno(s_p) < i < i_p\}$
$max\text{-}seqno(s_p) := i_p$
`\\ Cancel any scheduled expedited requests`
$expedited\text{-}rqsts \setminus= \{\langle s_p, i_p, t, rec\text{-}tpl\rangle \mid$
$t \in \mathbb{R}^{\geq 0}, rec\text{-}tpl \in Recovery\text{-}Tuples\}$
`\\ Update requestor/replier state`
if $\langle s_p, i_p\rangle \in recovered\text{-}pkts?$ **then**
if $rec\text{-}tpl(s_p, i_p) = \perp$
$\vee rec\text{-}time(rec\text{-}tpl(p)) < rec\text{-}time(rec\text{-}tpl(s_p, i_p))$
then
$rec\text{-}tpl(s_p, i_p) := rec\text{-}tpl(p)$
`\\ Cancel any requestor updates`
$rqst\text{-}updates \setminus= \{\langle s_p, i_p, t, rec\text{-}tpl\rangle \mid$
$t \in \mathbb{R}^{\geq 0}, rec\text{-}tpl \in Recovery\text{-}Tuples\}$
`\\ Cancel any replier updates`
$repl\text{-}updates \setminus= \{\langle s_p, i_p, t, rec\text{-}tpl\rangle \mid$
$t \in \mathbb{R}^{\geq 0}, rec\text{-}tpl \in Recovery\text{-}Tuples\}$

transmission timeout of a scheduled request update for the packet $\langle s, i\rangle$. The precondition and the effects of a $\texttt{send-repl-update}_h(s, i)$ action are analogous to those of the $\texttt{send-rqst-update}_h(s, i)$ action described above.

Time Passage

The action $\nu(t)$ models the passage of t time units. If the host h has crashed, then time is allowed to elapse. Otherwise, time is prevented from elapsing while either there are packets in the delivery, IP unicast, and IP multicast transmission buffers or there are packets which have been declared missing

Figure 6.11 The CESRM-REC$_h$ Automaton — Discrete Transitions (Cont'd)

input process-pkt$_h(p)$
where $\textit{type}(p) =$ EXP-RQST
eff **if** $\textit{status} =$ member **then**
$\langle s_p, i_p\rangle = \textit{id}(p)$
\\ Only consider proper packets
if $\textit{min-seqno}(s_p) \neq \perp \wedge \textit{min-seqno}(s_p) \le i_p$ **then**
$\langle q, d_{qs_p}, r, d_{rq}\rangle := \textit{rec-tpl}(p)$
\\ Expedite a reply
if $h = r \wedge \langle s_p, i_p\rangle \in \textit{archived-pkts?}$
$\wedge \langle s_p, i_p\rangle \notin \textit{pending-repls?}$
then
\\ A reply becomes pending
$\textit{pending-repls} \setminus= \{\langle s_p, i_p, t_*\rangle \mid t_* \in \mathbb{R}^{\ge 0}\}$
$t_{repl} := \textit{now} + D_3 d_{rq}$
$\textit{pending-repls} \cup= \{\langle s_p, i_p, t_{repl}\rangle\}$
\\ Cancel any scheduled replies
$\textit{scheduled-repls} \setminus=$
$\{\langle s_p, i_p, t, q, d_{qs_p}\rangle \mid t, d_{qs_p} \in \mathbb{R}^{\ge 0}, q \in H\}$
\\ Compose EXP-REPL packet
$\textit{msend-buff} \cup=$
$\{\textit{comp-exp-repl-pkt}(s_p, i_p, \textit{rec-tpl}(p))\}$
\\ Discover any trailing missing packets
if $h \neq s_p \wedge \textit{max-seqno}(s_p) < i_p$ **then**
$\textit{to-be-requested?} \cup=$
$\{\langle s_p, i\rangle \mid i \in \mathbb{N}, \textit{max-seqno}(s_p) < i < i_p\}$
$\textit{max-seqno}(s_p) := i_p$

input process-pkt$_h(p)$
where $\textit{type}(p) =$ EXP-REPL
eff **if** $\textit{status} =$ member **then**
$\langle s_p, i_p\rangle = \textit{id}(p)$
\\ Only consider proper packets
if $\textit{min-seqno}(s_p) \neq \perp \wedge \textit{min-seqno}(s_p) \le i_p$ **then**
\\ A reply becomes pending
$\textit{pending-repls} \setminus= \{\langle s_p, i_p, t_*\rangle \mid t_* \in \mathbb{R}^{\ge 0}\}$
$t_{repl} := \textit{now} + D_3\, \textit{dist?}(\textit{requestor}(p))$
$\textit{pending-repls} \cup= \{\langle s_p, i_p, t_{repl}\rangle\}$
\\ Archive and deliver packet
if $h \neq s_p \wedge \langle s_p, i_p\rangle \notin \textit{archived-pkts?}$ **then**
$\textit{recovered-pkts?} \cup= \{\langle s_p, i_p\rangle\}$
$\textit{archived-pkts} \cup= \{\langle \textit{strip}(p), \textit{now}\rangle\}$
if $h \neq s_p$ **then** $\textit{to-be-delivered} \cup= \{p\}$
\\ Pkt need not be requested
$\textit{to-be-requested?} \setminus= \{\langle s_p, i_p\rangle\}$
\\ Cancel any requests/replies
$\textit{scheduled-rqsts} \setminus= \{\langle s_p, i_p, t, k\rangle \mid t \in \mathbb{R}^{\ge 0}, k \in \mathbb{N}\}$
$\textit{scheduled-repls} \setminus=$
$\{\langle s_p, i_p, t, q, d_{qs_p}\rangle \mid t, d_{qs_p} \in \mathbb{R}^{\ge 0}, q \in H\}$
\\ Cancel any pending requests
$\textit{pending-rqsts} \setminus= \{\langle s_p, i_p, t\rangle \mid t \in \mathbb{R}^{\ge 0}\}$
\\ Discover any trailing missing packets
if $\textit{max-seqno}(s_p) < i_p$ **then**
$\textit{to-be-requested?} \cup=$
$\{\langle s_p, i\rangle \mid i \in \mathbb{N}, \textit{max-seqno}(s_p) < i < i_p\}$
$\textit{max-seqno}(s_p) := i_p$
\\ Cancel any scheduled expedited requests
$\textit{expedited-rqsts} \setminus= \{\langle s_p, i_p, t, \textit{rec-tpl}\rangle \mid$
$t \in \mathbb{R}^{\ge 0}, \textit{rec-tpl} \in \textit{Recovery-Tuples}\}$
if $\langle s_p, i_p\rangle \in \textit{recovered-pkts?}$ **then**
if $\textit{rec-tpl}(s_p, i_p) = \perp$
$\vee \textit{rec-time}(\textit{rec-tpl}(p)) < \textit{rec-time}(\textit{rec-tpl}(s_p, i_p))$
then
$\textit{rec-tpl}(s_p, i_p) := \textit{rec-tpl}(p)$
\\ Cancel any requestor updates
$\textit{rqst-updates} \setminus= \{\langle s_p, i_p, t, \textit{rec-tpl}\rangle \mid$
$t \in \mathbb{R}^{\ge 0}, \textit{rec-tpl} \in \textit{Recovery-Tuples}\}$
\\ Cancel any replier updates
$\textit{repl-updates} \setminus= \{\langle s_p, i_p, t, \textit{rec-tpl}\rangle \mid$
$t \in \mathbb{R}^{\ge 0}, \textit{rec-tpl} \in \textit{Recovery-Tuples}\}$
$\langle q, d_{qs_p}, r, d_{rq}\rangle := \textit{rec-tpl}(s_p, i_p)$
if $\langle s_p, i_p\rangle \notin \textit{rqst-updates?} \wedge h \neq q$ **then**
\\ Schedule a requestor update
$\textit{cand-rec-tpl} := \langle h, \textit{dist?}(s_p), r, \textit{dist?}(r)\rangle$
$\textit{cand-rec-time} := \textit{rec-time}(\textit{cand-rec-tpl})$
$\textit{curr-rec-tpl} := \textit{rec-tpl}(s_p, i_p)$
$\textit{curr-rec-time} := \textit{rec-time}(\textit{curr-rec-tpl})$
if $\textit{cand-rec-time} < \textit{curr-rec-time}$ **then**
$d_{rqst} := \textit{dist?}(s_p)$
$t_{rqst} :\in \textit{now} + [C_1 d_{rqst}, (C_1 + C_2) d_{rqst}]$
$\textit{rqst-updates} \cup= \{\langle s_p, i_p, t_{rqst}, \textit{cand-rec-tpl}\rangle\}$
$\langle q, d_{qs_p}, r, d_{rq}\rangle := \textit{rec-tpl}(p)$
if $\langle s_p, i_p\rangle \in \textit{archived-pkts?} \setminus \textit{recovered-pkts?}$ **then**
if $\langle s_p, i_p\rangle \notin \textit{repl-updates?} \wedge h \neq r$ **then**
\\ Schedule a replier update
$\textit{cand-rec-tpl} := \langle q, d_{qs}, h, \textit{dist?}(q)\rangle$
$\textit{cand-rec-time} := \textit{rec-time}(\textit{cand-rec-tpl})$
if $\textit{cand-rec-time} < \textit{rec-time}(\textit{rec-tpl}(p))$ **then**
$d_{repl} := \textit{dist?}(q)$
$t_{repl} :\in \textit{now} + [D_1 d_{repl}, (D_1 + D_2) d_{repl}]$
$\textit{repl-updates} \cup= \{\langle s_p, i_p, t_{repl}, \textit{cand-rec-tpl}\rangle\}$

Figure 6.12 The CESRM-REC$_h$ Automaton — Discrete Transitions (Cont'd)

```
input process-pkt_h(p)
where type(p) = RQST-UPDATE
eff  if status = member then
       ⟨s_p, i_p⟩ = id(p)
       \\ Only consider proper packets
       if min-seqno(s_p) ≠⊥ ∧min-seqno(s_p) ≤ i_p then
         if ⟨s_p, i_p⟩ ∈ recovered-pkts? then
           \\ Update requestor state
           if rec-tpl(s_p, i_p) =⊥
             ∨rec-time(rec-tpl(p)) < rec-time(rec-tpl(s_p, i_p))
           then
             rec-tpl(s_p, i_p) := rec-tpl(p)
         \\ Remove worse replier update
         if ⟨s_p, i_p⟩ ∈ repl-updates? then
           choose t ∈ ℝ^≥0, rec-tpl ∈ Recovery-Tuples
             where ⟨s_p, i_p, t, rec-tpl⟩ ∈ repl-updates
           rec-time = rec-time(rec-tpl)
           if rec-time(rec-tpl(p)) < rec-time then
             repl-updates \= {⟨s_p, i_p, t, rec-tpl⟩}
         \\ Cancel any requestor updates
         rqst-updates \= {⟨s_p, i_p, t, rec-tpl⟩ |
           t ∈ ℝ^≥0, rec-tpl ∈ Recovery-Tuples}
         \\ Discover any trailing missing packets
         if max-seqno(s_p) < i_p then
           to-be-requested? ∪=
             {⟨s_p, i⟩ | i ∈ ℕ, max-seqno(s_p) < i ≤ i_p}
           max-seqno(s_p) := i_p
```

```
input process-pkt_h(p)
where type(p) = REPL-UPDATE
eff  if status = member then
       ⟨s_p, i_p⟩ = id(p)
       \\ Only consider proper packets
       if min-seqno(s_p) ≠⊥ ∧min-seqno(s_p) ≤ i_p then
         if ⟨s_p, i_p⟩ ∈ recovered-pkts? then
           \\ Update replier state
           if rec-tpl(s_p, i_p) =⊥
             ∨rec-time(rec-tpl(p)) < rec-time(rec-tpl(s_p, i_p))
           then
             rec-tpl(s_p, i_p) := rec-tpl(p)
         \\ Remove worse requestor update
         if ⟨s_p, i_p⟩ ∈ rqst-updates? then
           choose t ∈ ℝ^≥0, rec-tpl ∈ Recovery-Tuples
             where ⟨s_p, i_p, t, rec-tpl⟩ ∈ rqst-updates
           rec-time = rec-time(rec-tpl)
           if rec-time(rec-tpl(p)) < rec-time then
             rqst-updates \= {⟨s_p, i_p, t, rec-tpl⟩}
         \\ Cancel any replier updates
         repl-updates \= {⟨s_p, i_p, t, rec-tpl⟩ |
           t ∈ ℝ^≥0, rec-tpl ∈ Recovery-Tuples}
         \\ Discover any trailing missing packets
         if max-seqno(s_p) < i_p then
           to-be-requested? ∪=
             {⟨s_p, i⟩ | i ∈ ℕ, max-seqno(s_p) < i ≤ i_p}
           max-seqno(s_p) := i_p
```

but for which a request has yet to be scheduled; that is, while either the buffer *to-be-delivered*, the buffer *msend-buff*, the buffer *msend-buff*, or the set *to-be-requested?* is non-empty. Furthermore, time is prevented from elapsing past the scheduled transmission time of any requests, replies, expedited requests, request updates, and reply updates.

6.2.4 The IP Component — IP

In this section, we augment our abstract specification of the underlying IP multicast service IPMCAST of Chapter 4 to provide the IP unicast service, *i.e.*, the best-effort point-to-point communication service. Figures 6.14 and 6.15 present the signature, the variables, and the actions of the TIOA model IP of the IP service component that provides both multicast and unicast communication. We proceed by only describing the additions to the specification of the IPMCAST automaton of Chapter 4.

The set $upkts \subseteq P_{\text{IPUCAST-CLIENT}}$ is comprised of the packets that have been unicast and are pending delivery to their respective recipients. The action $\texttt{usend}_h(p)$, for $p \in P_{\text{IPUCAST-CLIENT}}$, models the unicast transmission of the packet p by the host h. When the host h is operational, the $\texttt{usend}_h(p)$ action adds the packet p to the set of pending unicast packets *upkts*. The action $\texttt{udrop}(p)$, for $p \in P_{\text{IPUCAST-CLIENT}}$, models the loss of the packet p. Its effects are to remove p from the set of pending unicast packets *upkts*. The action $\texttt{urecv}_h(p)$, for $p \in P_{\text{IPUCAST-CLIENT}}$, models the reception of the unicast packet p. The precondition of $\texttt{urecv}_h(p)$ is that h is operational, h is the intended recipient of p, and that p is a pending unicast packet. Its effects are to remove p from the set of pending unicast packets *upkts*.

Figure 6.13 The CESRM-REC_h Automaton — Discrete Transitions (Cont'd)

internal `schdl-rqst`$_h(s,i)$

pre $\mathit{status} = \texttt{member} \wedge \langle s,i\rangle \in \mathit{to\text{-}be\text{-}requested?}$

eff `\\ Schedule new request`

$k_r := 1;\ d_r := \mathit{dist?}(s)$

$t_r :\in \mathit{now} + 2^{k_r-1}[C_1 d_r, (C_1+C_2)d_r]$

$\mathit{scheduled\text{-}rqsts} \cup= \{\langle s,i,t_r,k_r\rangle\}$

`\\ Pkt request has been scheduled`

$\mathit{to\text{-}be\text{-}requested?} \setminus= \{\langle s,i\rangle\}$

`\\ Schedule an expedited request`

if $\mathit{opt\text{-}rec\text{-}tpl}(s) \neq \perp$ **then**

$\langle q, d_{qs}, r, d_{rq}\rangle := \mathit{opt\text{-}rec\text{-}tpl}(s)$

if $h = q$ **then**

$t := \mathit{now} + \texttt{RQST-DELAY}$

$\mathit{expedited\text{-}rqsts} \cup= \{\langle s,i,t,\langle q,d_{qs},r,d_{rq}\rangle\rangle\}$

internal `send-sess`$_h$

pre $\mathit{status} = \texttt{member} \wedge \mathit{now} = \mathit{rep\text{-}deadline}$

eff `\\ Compose session packet`

$\mathit{msend\text{-}buff} \cup=$

$\{\mathit{comp\text{-}sess\text{-}pkt}(h, \mathit{now}, \mathit{dist\text{-}rprt}, \mathit{max\text{-}seqno})\}$

`\\ Reset session packet deadline`

$\mathit{rep\text{-}deadline} := \mathit{now} + \texttt{SESS-PERIOD}$

internal `send-rqst`$_h(s,i)$

choose $t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}$

pre $\mathit{status} = \texttt{member}$

$\wedge t = \mathit{now} \wedge \langle s,i,t,k\rangle \in \mathit{scheduled\text{-}rqsts}$

eff `\\ Compose request packet`

$\mathit{msend\text{-}buff} \cup= \{\mathit{comp\text{-}rqst\text{-}pkt}(s,i,h,\boldsymbol{\mathit{dist?}(s)})\}$

`\\ Back-off scheduled request`

$\mathit{scheduled\text{-}rqsts} \setminus= \{\langle s,i,t,k\rangle\}$

$k_r := k+1;\ d_r := \mathit{dist?}(s)$

$t_r :\in \mathit{now} + 2^{k_r-1}[C_1 d_r, (C_1+C_2)d_r]$

$\mathit{scheduled\text{-}rqsts} \cup= \{\langle s,i,t_r,k_r\rangle\}$

`\\ A request becomes pending`

$\mathit{pending\text{-}rqsts} \setminus= \{\langle s,i,t_*\rangle \mid t_* \in \mathbb{R}^{\geq 0}\}$

$t_r := \mathit{now} + 2^{k_r-1} C_3 d_r$

$\mathit{pending\text{-}rqsts} \cup= \{\langle s,i,t_r\rangle\}$

internal `send-repl`$_h(s,i)$

choose $t \in \mathbb{R}^{\geq 0}, q \in H, \boldsymbol{d_{qs} \in \mathbb{R}^{\geq 0}}$

pre $\mathit{status} = \texttt{member}$

$\wedge t = \mathit{now} \wedge \langle s,i,t,q,\boldsymbol{d_{qs}}\rangle \in \mathit{scheduled\text{-}repls}$

eff `\\ Compose reply packet`

choose $p \in P_{\text{RM-CLIENT}}, t \in \mathbb{R}^{\geq 0}$

where $\langle p,t\rangle \in \mathit{archived\text{-}pkts} \wedge \mathit{id}(p) = \langle s,i\rangle$

$\mathit{msend\text{-}buff} \cup= \{\mathit{comp\text{-}repl\text{-}pkt}(p,q,\boldsymbol{d_{qs}},h,\boldsymbol{\mathit{dist?}(q)})\}$

`\\ A reply becomes pending`

$\mathit{pending\text{-}repls} \setminus= \{\langle s,i,t_*\rangle \mid t_* \in \mathbb{R}^{\geq 0}\}$

$t_{repl} := \mathit{now} + D_3\, \mathit{dist?}(r)$

$\mathit{pending\text{-}repls} \cup= \{\langle s,i,t_{repl}\rangle\}$

`\\ Cancel scheduled reply`

$\mathit{scheduled\text{-}repls} \setminus= \{\langle s,i,t,q,\boldsymbol{d_{qs}}\rangle\}$

internal `send-exp-rqst`$_h(s,i)$

choose $t \in \mathbb{R}^{\geq 0}, \mathit{rec\text{-}tpl} \in \mathit{Recovery\text{-}Tuples}$

pre $\mathit{status} = \texttt{member}$

$\wedge t = \mathit{now} \wedge \langle s,i,t,\mathit{rec\text{-}tpl}\rangle \in \mathit{expedited\text{-}rqsts}$

eff $\langle q, d_{qs}, r, d_{rq}\rangle = \mathit{rec\text{-}tpl}$

`\\ Compose EXP-RQST packet`

$\mathit{usend\text{-}buff}(r) \cup= \{\mathit{comp\text{-}exp\text{-}rqst\text{-}pkt}(s,i,\mathit{rec\text{-}tpl})\}$

`\\ Expedited request completed`

$\mathit{expedited\text{-}rqsts} \setminus= \{\langle s,i,t,\mathit{rec\text{-}tpl}\rangle\}$

internal `send-rqst-update`$_h(s,i)$

choose $t \in \mathbb{R}^{\geq 0}, \mathit{rec\text{-}tpl} \in \mathit{Recovery\text{-}Tuples}$

pre $\mathit{status} = \texttt{member}$

$\wedge t = \mathit{now} \wedge \langle s,i,t,\mathit{rec\text{-}tpl}\rangle \in \mathit{rqst\text{-}updates}$

eff `\\ Update optimal requestor/replier pair`

$\mathit{rec\text{-}tpl}(s,i) := \mathit{rec\text{-}tpl}$

`\\ Compose RQST-UPDATE packet`

$\mathit{msend\text{-}buff} \cup= \{\mathit{comp\text{-}rqst\text{-}update\text{-}pkt}(s,i,\mathit{rec\text{-}tpl})\}$

`\\ Request update completed`

$\mathit{rqst\text{-}updates} \setminus= \{\langle s,i,t,,\mathit{rec\text{-}tpl}\rangle\}$

internal `send-repl-update`$_h(s,i)$

choose $t \in \mathbb{R}^{\geq 0}, \mathit{rec\text{-}tpl} \in \mathit{Recovery\text{-}Tuples}$

pre $\mathit{status} = \texttt{member}$

$\wedge t = \mathit{now} \wedge \langle s,i,t,\mathit{rec\text{-}tpl}\rangle \in \mathit{repl\text{-}updates}$

eff `\\ Update optimal requestor/replier pair`

$\mathit{rec\text{-}tpl}(s,i) := \mathit{rec\text{-}tpl}$

`\\ Compose REPL-UPDATE packet`

$\mathit{msend\text{-}buff} \cup= \{\mathit{comp\text{-}repl\text{-}update\text{-}pkt}(s,i,\mathit{rec\text{-}tpl})\}$

`\\ Reply update completed`

$\mathit{repl\text{-}updates} \setminus= \{\langle s,i,t,,\mathit{rec\text{-}tpl}\rangle\}$

time-passage $\nu(t)$

pre $\mathit{status} = \texttt{crashed}$

$\vee(\mathit{to\text{-}be\text{-}requested?} = \emptyset \wedge \mathit{to\text{-}be\text{-}delivered} = \emptyset$

$\wedge \mathit{msend\text{-}buff} = \emptyset \wedge (\wedge_{h' \in H, h' \neq h} \mathit{usend\text{-}buff}(h') = \emptyset)$

$\wedge(\mathit{rep\text{-}deadline} = \perp \vee \mathit{now} + t \leq \mathit{rep\text{-}deadline})$

$\wedge$ no requests scheduled earlier than $\mathit{now}+t$

$\wedge$ no replies scheduled earlier than $\mathit{now}+t$

$\wedge$ no exp-rqsts scheduled earlier than $\mathit{now}+t$

$\wedge$ no rqst-updates scheduled earlier than $\mathit{now}+t$

$\wedge$ no repl-updates scheduled earlier than $\mathit{now}+t$)

eff $\mathit{now} := \mathit{now} + t$

Figure 6.14 The IP Automaton — Signature

Actions:

input

`crash`$_h$, for $h \in H$

`mjoin`$_h$, for $h \in H$

`mleave`$_h$, for $h \in H$

`usend`$_h(p)$, for $h \in H, p \in P_{\text{IPUCAST-CLIENT}}$

`msend`$_h(p)$, for $h \in H, p \in P_{\text{IPMCAST-CLIENT}}$

internal

`mgrbg-coll`(pkt), for $\mathit{pkt} \in P_{\text{IPMCAST}}$

output

`mjoin-ack`$_h$, for $h \in H$

`mleave-ack`$_h$, for $h \in H$

`urecv`$_h(p)$, for $h \in H, p \in P_{\text{IPMCAST-CLIENT}}$

`mrecv`$_h(p)$, for $h \in H, p \in P_{\text{IPMCAST-CLIENT}}$

`udrop`(p), for $p \in P_{\text{IPMCAST-CLIENT}}$

`mdrop`(p, H_d), for $p \in P_{\text{IPMCAST-CLIENT}}, H_d \subseteq H$

time-passage

$\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

Figure 6.15 The IP automaton — Variables and Discrete Transitions

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status(h) \in IPmcast\text{-}Status$, for all $h \in H$,
  initially $status(h) = \texttt{idle}$, for all $h \in H$
$upkts \subseteq P_{\text{IPUCAST-CLIENT}}$, initially $upkts = \emptyset$
$mpkts \subseteq P_{\text{IPMCAST}}$, initially $mpkts = \emptyset$

Derived Variables:

$up = \{h \in H | status(h) \neq \texttt{crashed}\}$
$idle = \{h \in H | status(h) = \texttt{idle}\}$
$joining = \{h \in H | status(h) = \texttt{joining}\}$
$leaving = \{h \in H | status(h) = \texttt{leaving}\}$
$members = \{h \in H | status(h) = \texttt{member}\}$

Discrete Transitions:

input $\texttt{crash}_h$
eff **if** $h \in up$ **then**
  $status(h) := \texttt{crashed}$
  foreach $pkt \in mpkts$ **do:**
    $intended(pkt) \setminus= \{h\}$

input $\texttt{mjoin}_h$
eff **if** $h \in idle$ **then**
  $status(h) := \texttt{joining}$

input $\texttt{mleave}_h$
eff **if** $h \in joining \cup members$ **then**
  $status(h) := \texttt{leaving}$
  foreach $pkt \in mpkts$ **do:**
    $intended(pkt) \setminus= \{h\}$

input $\texttt{usend}_h(p)$
eff **if** $h \in up$ **then**
  $upkts \cup= \{p\}$

input $\texttt{msend}_h(p)$
eff **if** $h \in up$ **then**
  $mpkts \cup= \{\langle p, members, \{h\}, \emptyset\rangle\}$

internal $\texttt{mgrbg-coll}(p)$
choose $pkt \in P_{\text{IPMCAST}}$
pre $pkt \in mpkts \wedge p = strip(pkt)$
  $\wedge intended(pkt) \subseteq (completed(pkt) \cup dropped(pkt))$
eff $mpkts \setminus= \{pkt\}$

output $\texttt{mjoin-ack}_h$
pre $h \in joining$
eff $status(h) := \texttt{member}$

output $\texttt{mleave-ack}_h$
pre $h \in leaving$
eff $status(h) := \texttt{idle}$

input $\texttt{urecv}_h(p)$
pre $h \in up \wedge h = dest(p) \wedge p \in upkts$
eff $upkts \setminus= \{p\}$

output $\texttt{mrecv}_h(p)$
choose $pkt \in P_{\text{IPMCAST}}$
pre $pkt \in mpkts \wedge p = strip(pkt)$
  $\wedge h \neq source(p) \wedge h \in members \setminus dropped(pkt)$
eff $completed(pkt) \cup= \{h\}$

input $\texttt{udrop}(p)$
pre $p \in upkts$
eff $upkts \setminus= \{p\}$

output $\texttt{mdrop}(p, H_d)$
choose $pkt \in P_{\text{IPMCAST}}$
pre $pkt \in mpkts \wedge p = strip(pkt)$
  $\wedge H_d \subseteq members \setminus (completed(pkt) \cup dropped(pkt))$
eff $dropped(pkt) \cup= H_d$

time-passage $\nu(t)$
pre None
eff $now := now + t$

6.3 CESRM Correctness

In this section, we analyze the correctness of our model of the CESRM protocol against the reliable multicast service specification of Chapter 3.

As in the case of the SRM protocol, our model of the CESRM protocol involves the CESRM processes at each host and the underlying IP multicast service; that is, the automaton $\prod_{h\in H} \text{CESRM}_h \times \text{IP}$, where $\text{CESRM}_h = \text{CESRM-MEM}_h \times \text{CESRM-IPBUFF}_h \times \text{CESRM-REC}_h$. We define the automaton CESRM to be the composition $\prod_{h\in H} \text{CESRM}_h \times \text{IP}$ after hiding all output actions that are not output actions of the specification $\text{RM}(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$; that is, $\text{CESRM} = hide_\Phi(\prod_{h\in H} \text{CESRM}_h \times \text{IP})$, with $\Phi = out(\prod_{h\in H} \text{CESRM}_h \times \text{IP}) \setminus out(\text{RM}(\Delta))$.

Furthermore, we let CESRM_I and $\text{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, denote the implementation and the specification of the reliable multicast service each composed with all the client automata; that is, $\text{CESRM}_I = \text{CESRM} \times \text{RMCLIENTS}$ and $\text{RM}_S(\Delta) = \text{RM}(\Delta) \times \text{RMCLIENTS}$.

The correctness analysis of CESRM_I follows precisely the correctness analysis of SRM_I in Chapter 4. We proceed by adapting the correctness analysis of SRM_I presented in Section 4.4 to the specifics of CESRM_I.

6.3.1 Correctness Analysis Preliminaries

In this section, we adapt the invariants and lemmas of Section 4.4.3 to the CESRM_I automaton. The proofs of most such invariants and lemmas carry over from Section 4.4 practically unchanged. The key to this realization is that: i) the effects of a $\texttt{process-pkt}_h(p)$, for $p \in P_{\text{CESRM}}$, such that $type(p) = \texttt{EXP-RQST}$, is analogous to the processing of a RQST packet and the transmission of a reply to this request, and ii) the effects of a $\texttt{process-pkt}_h(p)$, for $p \in P_{\text{CESRM}}$, such that $type(p) = \texttt{EXP-REPL}$, is analogous to the processing of a REPL packet.

We begin by stating the transmission integrity property of the IP component along the lines of Lemma 4.1. This property states that any packet that is received by a client of the IP component must have previously been transmitted by a client of the IP component.

Lemma 6.1 (IP Transmission Integrity) *For any timed trace β of* IP*, it is the case that:*

1. *any $\texttt{mrecv}_h(p)$ action, for $h \in H$ and $p \in P_{\text{IPMCAST-CLIENT}}$, in β is preceded in β by a $\texttt{msend}_{h'}(p)$ action, for some $h' \in H$, and*
2. *any $\texttt{urecv}_h(p)$ action, for $h \in H$ and $p \in P_{\text{IPUCAST-CLIENT}}$, in β is preceded in β by a $\texttt{usend}_{h'}(p)$ action, for some $h' \in H$.*

Proof: The proof of the first claim is identical to the proof of Lemma 4.1. The proof of the second claim is analogous.

Let α be any timed execution of IP such that $\beta = ttrace(\alpha)$. Consider a particular occurrence of an action $\texttt{urecv}_h(p)$ in α, for $h \in H$ and $p \in P_{\text{IPUCAST-CLIENT}}$. Let $(u, \texttt{urecv}_h(p), u') \in trans(\text{IP})$ be the discrete transition in α corresponding to the particular occurrence of the action $\texttt{urecv}_h(p)$ in α. From the precondition of $\texttt{urecv}_h(p)$, it is the case that $p \in u.upkts$. However, p may be added to *upkts* only by the occurrence of an action $\texttt{usend}_{h'}(p)$, for some $h \in H$. It follows that the occurrence of any action $\texttt{urecv}_h(p)$ in α is preceded by the occurrence of an action $\texttt{usend}_{h'}(p)$, for some $h' \in H$. ❒

Invariant 6.1 *For $h, h' \in H$ and any reachable state u of CESRM-REC_h, it is the case that $u.window?(h') \subseteq u.proper?(h')$.*

Proof: Follows directly from the definitions of the derived variables $\text{CESRM-REC}_h.window?(h')$ and $\text{CESRM-REC}_h.proper?(h')$. ❒

Invariant 6.2 *For $h, h' \in H$ and any reachable state u of CESRM-REC_h, if $u.status \neq \texttt{member}$, then $u.expected(h') = \emptyset$ and $u.delivered(h') = \emptyset$.*

Proof: The proof is identical to that of Invariant 4.2. ❒

Invariant 6.3 *For $h, h' \in H$ and any reachable state u of CESRM-REC_h, it is the case that:*

1. *$u.min\text{-}seqno(h') \neq \perp \Leftrightarrow u.max\text{-}seqno(h') \neq \perp$ and*
2. *$u.min\text{-}seqno(h') \neq \perp \ \Rightarrow u.min\text{-}seqno(h') \leq u.max\text{-}seqno(h')$.*

Proof: The proof is identical to that of Invariant 4.3. ❒

Invariant 6.4 *For $h, h' \in H$ and any reachable state u of CESRM-REC_h, it is the case that:*

1. $u.delivered(h') \cup u.to\text{-}be\text{-}delivered?(h') \subseteq u.archived\text{-}pkts?(h')$ *and*
2. $u.status = \texttt{member} \Rightarrow u.delivered(h') \cup u.to\text{-}be\text{-}delivered?(h') = u.archived\text{-}pkts?(h')$.

Proof: The proof is identical to that of Invariant 6.4. ❒

Invariant 6.5 *For* $h, h' \in H$ *and any reachable state* u *of* CESRM-REC$_h$*, it is the case that* $u.archived\text{-}pkts?(h') \subseteq u.window?(h')$.

Proof: The proof is similar to the induction used in the proof of Invariant 6.5. In this case, we must also consider the discrete transitions involving a $\texttt{process-pkt}_h(p)$ action, for $p \in P_{\text{CESRM}}$, such that $type(p) \in \{\texttt{EXP-RQST}, \texttt{EXP-REPL}, \texttt{RQST-UPDATE}, \texttt{REPL-UPDATE}\}$; the other actions introduced in CESRM-REC$_h$ do not affect the variables $min\text{-}seqno(h')$, $max\text{-}seqno(h')$, and $archived\text{-}pkts?(h')$. In the case of a `EXP-RQST` packet, the $\texttt{process-pkt}_h(p)$ action does not affect $archived\text{-}pkts?(h')$ and may only increase $max\text{-}seqno(h')$. In the case of a `EXP-REPL` packet, the effects of $\texttt{process-pkt}_h(p)$ with respect to the variables $min\text{-}seqno(h')$, $max\text{-}seqno(h')$, and $archived\text{-}pkts?(h')$ are similar to those in the case of a `REPL` packet. In the case of either a `RQST-UPDATE` or a `REPL-UPDATE` packet, the $\texttt{process-pkt}_h(p)$ action does not affect $archived\text{-}pkts?(h')$ and may only increase $max\text{-}seqno(h')$. Thus, it follows that the invariant assertion holds following the processing of either `EXP-RQST`, `EXP-REPL`, or `RQST-UPDATE`, `REPL-UPDATE` packets. ❒

Invariant 6.6 *For* $h, h' \in H$ *and any reachable state* u *of* CESRM-REC$_h$*, it is the case thatu.to-be-delivered?*$(h') \subseteq u.window?(h')$.

Proof: Follows directly from Invariants 6.4 and 6.5. ❒

Invariant 6.7 *For* $h, h' \in H$ *and any reachable state* u *of* CESRM-REC$_h$*, it is the case thatu.delivered*$(h') \subseteq u.window?(h')$.

Proof: Follows directly from Invariants 6.4 and 6.5. ❒

Invariant 6.8 *For* $h \in H$, $p \in P_{\text{RM-CLIENT}}$*, and any reachable state* u *of* CESRM-REC$_h$*, if* $p \in u.to\text{-}be\text{-}delivered$*, then* $u.min\text{-}seqno(source(p)) \neq \perp$ *and* $u.min\text{-}seqno(source(p)) \leq seqno(p)$.

Proof: The proof is identical to that of Invariant 4.8. ❒

Invariant 6.9 *For* $h, h' \in H$ *and any reachable state* u *of* CESRM-REC$_h$*, it is the case that:*

1. $u.min\text{-}seqno(h') = \perp \Rightarrow u.expected(h') = \emptyset$,
2. $u.delivered(h') \subseteq u.expected(h')$,
3. $h = h' \wedge u.status \neq \texttt{crashed} \Rightarrow u.expected(h') = u.proper?(h')$, *and*
4. $u.expected(h') \neq \emptyset \Rightarrow u.expected(h') = u.proper?(h')$

Proof: The proof is identical to that of Invariant 4.9; the effects of processing a `EXP-REPL` packet are identical to those of processing a `REPL` packet with respect to the relevant variables of CESRM-REC$_h$. ❒

Invariant 6.10 *Let $h \in H$ and u be any reachable state u of* CESRM-REC$_h$*. For any $p \in P_{\text{CESRM}}$, such that $type(p) \in \{\texttt{DATA}, \texttt{REPL}\}$ and $p \in u.msend\text{-}buff$, it is the case that $id(p) \in u.archived\text{-}pkts?$.*

Proof: The proof is identical to that of Invariant 4.10. ❒

Invariant 6.11 *For $h \in H$, $p \in P_{\text{RM-CLIENT}}$, and any reachable state u of* CESRM-REC$_h$*, if $p \in u.to\text{-}be\text{-}delivered$, then $source(p) \neq h$.*

Proof: From the effects of the $process\text{-}pkt_h(p)$ action, for $h \in H$ and $p \in P_{\text{CESRM}}$, it follows that a packet p may be added to *to-be-delivered* only if $source(p) \neq h$. ❒

Invariant 6.12 *For $h, h' \in H$ and any reachable state u of* CESRM-REC$_h$*, if $u.expected(h') \neq \emptyset$, then $u.to\text{-}be\text{-}delivered?(h') \subseteq u.expected(h')$.*

Proof: The proof is identical to that of Invariant 4.12. ❒

Invariant 6.13 *For $h, h' \in H$ and any reachable state u of* CESRM-REC$_h$*, it is the case that $u.to\text{-}be\text{-}requested?(h') \subseteq u.window?(h')$.*

Proof: The proof is similar to the induction used in the proof of Invariant 4.13. In this case, we must also consider the discrete transitions involving a $\texttt{process-pkt}_h(p)$ action, for $p \in P_{\text{CESRM}}$, such that $type(p) \in \{\texttt{EXP-RQST}, \texttt{EXP-REPL}, \texttt{RQST-UPDATE}, \texttt{REPL-UPDATE}\}$; the other actions introduced in CESRM-REC$_h$ do not affect the variables $to\text{-}be\text{-}requested?(h')$, $min\text{-}seqno(h')$, and $max\text{-}seqno(h')$. In the case of `EXP-RQST`, `EXP-REPL`, `RQST-UPDATE`, and `REPL-UPDATE` packets, the $\texttt{process-pkt}_h(p)$ action adds elements to $to\text{-}be\text{-}requested?(h')$ only when trailing missing packets are discovered. In such cases, it also increases the value of $max\text{-}seqno(h')$ to account for the packets that is has detected to have been transmitted by h'. Thus, following such a $\texttt{process-pkt}_h(p)$ action, the invariant assertion still holds. ❒

Invariant 6.14 *For $h, h' \in H$ and any reachable state u of* CESRM-REC$_h$*, it is the case that $u.scheduled\text{-}repls?(h') \subseteq u.archived\text{-}pkts?(h')$.*

Proof: The proof is similar to the induction used in the proof of Invariant 4.14. In this case, we must also consider the discrete transitions involving a $\texttt{process-pkt}_h(p)$ action, for $p \in P_{\text{CESRM}}$, such that $type(p) \in \{\texttt{EXP-RQST}, \texttt{EXP-REPL}\}$; the other actions introduced in CESRM-REC$_h$ do not affect the variables $scheduled\text{-}repls?(h')$ and $archived\text{-}pkts?(h')$. In the case of an `EXP-RQST` packet, the $\texttt{process-pkt}_h(p)$ action may only remove elements from $scheduled\text{-}repls?(h')$ and may only add elements to $archived\text{-}pkts?(h')$. From the induction hypothesis, it follows that the invariant assertion holds following the particular $\texttt{process-pkt}_h(p)$ action. In the case of a `EXP-REPL` packet, the effects of $\texttt{process-pkt}_h(p)$ with respect to the variables $scheduled\text{-}repls?(h')$ and $archived\text{-}pkts?(h')$ packet are identical to those in the case of a `REPL` packet. Thus, it follows that the invariant assertion holds following the particular $\texttt{process-pkt}_h(p)$ action. ❒

Invariant 6.15 *For $h, h' \in H$ and any reachable state u of* CESRM-REC$_h$*, it is the case that $u.scheduled\text{-}rqsts?(h') \subseteq u.window?(h')$.*

Proof: The proof is similar to the induction used in the proof of Invariant 4.14. In this case, we must also consider the discrete transitions involving a $\texttt{process-pkt}_h(p)$ action, for $p \in P_{\text{CESRM}}$, such that $type(p) \in \{\texttt{EXP-RQST}, \texttt{EXP-REPL}, \texttt{RQST-UPDATE}, \texttt{REPL-UPDATE}\}$; the other actions introduced in CESRM-REC$_h$ do not affect the variables *scheduled-rqsts?*(h') and *window?*(h'). In the case of either EXP-RQST, RQST-UPDATE, or REPL-UPDATE packets, the $\texttt{process-pkt}_h(p)$ action does not affect *scheduled-rqsts?*(h') and may only add elements to *window?*(h'). Thus, the induction hypothesis implies that the invariant assertion holds following the occurrence of $\texttt{process-pkt}_h(p)$. In the case of a EXP-REPL packet, the $\texttt{process-pkt}_h(p)$ action may only remove elements from *scheduled-rqsts?*(h') and may only add elements to *window?*(h'). Thus, the induction hypothesis implies that the invariant assertion holds following the occurrence of $\texttt{process-pkt}_h(p)$. ❐

Invariant 6.16 *For* $h, h' \in H$ *and any reachable state* u *of* CESRM$_I$*, it is the case that* u[CESRM-REC$_h$]*.recovered-pkts?*$(h') \subseteq u$[CESRM-REC$_h$]*.archived-pkts?*(h')*.*

Proof: Follows by a simple induction on the length of any finite admissible execution of CESRM$_I$ leading to u. The key point to the induction is that the only actions that affect the variable CESRM-REC$_h$*.recovered-pkts?*(h') are the actions $\texttt{rm-leave}_h$ and $\texttt{process-pkt}_h(p)$ action, for $p \in P_{\text{CESRM}}$, such that $type(p) \in \{\texttt{REPL}, \texttt{EXP-REPL}\}$. The action $\texttt{rm-leave}_h$ reinitializes the variables CESRM-REC$_h$*.recovered-pkts?*(h') and CESRM-REC$_h$*.archived-pkts?*(h'). Thus, the invariant assertion holds following the occurrence of $\texttt{rm-leave}_h$. In the case of the $\texttt{process-pkt}_h(p)$ action, whenever $\texttt{process-pkt}_h(p)$ adds $id(p)$ to CESRM-REC$_h$*.recovered-pkts?*(h'), it also adds it to CESRM-REC$_h$*.archived-pkts?*(h'). Thus, the induction hypothesis implies that the invariant assertion holds following the occurrence of $\texttt{process-pkt}_h(p)$. ❐

Invariant 6.17 *For* $h, h' \in H$ *and any reachable state* u *of* CESRM$_I$*, it is the case that* u[CESRM-REC$_h$]*.repl-updates?*$(h') \subseteq u$[CESRM-REC$_h$]*.window?*(h')*.*

Proof: Follows by a simple induction on the length of any finite admissible execution of CESRM$_I$ leading to u. The key point to the induction is that the only actions that affect the variable CESRM-REC$_h$*.repl-updates?*(h') are the actions $\texttt{rm-leave}_h$ and $\texttt{process-pkt}_h(p)$ action, for $p \in P_{\text{CESRM}}$, such that $type(p) = \texttt{EXP-REPL}$. The action $\texttt{rm-leave}_h$ reinitializes the variables CESRM-REC$_h$*.repl-updates?*(h') and CESRM-REC$_h$*.window?*(h'). Thus, the invariant assertion holds following the occurrence of $\texttt{rm-leave}_h$. The $\texttt{process-pkt}_h(p)$ action may add $id(p)$ to CESRM-REC$_h$*.repl-updates?*(h') only if $id(p) \in$ CESRM-REC$_h$*.recovered-pkts?*(h'). Invariants 6.16 and 6.5 imply that $id(p) \in$ CESRM-REC$_h$*.window?*(h'). Thus, the induction hypothesis implies that the invariant assertion holds following the occurrence of $\texttt{process-pkt}_h(p)$. ❐

Invariant 6.18 *For* $h, h' \in H$ *and any reachable state* u *of* CESRM$_I$*, it is the case that* u[CESRM-REC$_h$]*.rqst-updates?*$(h') \subseteq u$[CESRM-REC$_h$]*.window?*(h')*.*

Proof: Follows by a simple induction on the length of any finite admissible execution of CESRM$_I$ leading to u. The key point to the induction is that the only actions that affect the variable CESRM-REC$_h$*.rqst-updates?*(h') are the actions $\texttt{rm-leave}_h$ and $\texttt{process-pkt}_h(p)$ action, for $p \in P_{\text{CESRM}}$, such that $type(p) = \texttt{EXP-REPL}$. The action $\texttt{rm-leave}_h$ reinitializes the variables CESRM-REC$_h$*.rqst-updates?*(h') and CESRM-REC$_h$*.window?*(h'). Thus, the invariant assertion holds following the occurrence of $\texttt{rm-leave}_h$. The $\texttt{process-pkt}_h(p)$ action may add $id(p)$ to CESRM-REC$_h$*.rqst-updates?*(h') only if $id(p) \in$ CESRM-REC$_h$*.recovered-pkts?*(h'). Lemmas 6.16 and 6.5 imply that $id(p) \in$ CESRM-REC$_h$*.window?*(h'). Thus, the induction hypothesis implies that the invariant assertion holds following the occurrence of $\texttt{process-pkt}_h(p)$. ❐

Invariant 6.19 *For $h, h' \in H$ and any reachable state u of* CESRM-REC$_h$*, it is the case that* $u.\textit{to-be-requested?}(h') \cap u.\textit{archived-pkts?}(h') = \emptyset$.

Proof: The proof is similar to the induction used in the proof of Invariant 4.14. In this case, we must also consider the discrete transitions involving a $\texttt{process-pkt}_h(p)$ action, for $p \in P_{\text{CESRM}}$, such that $\textit{type}(p) \in \{\texttt{EXP-RQST}, \texttt{EXP-REPL}, \texttt{RQST-UPDATE}, \texttt{REPL-UPDATE}\}$; the other actions introduced in CESRM-REC$_h$ do not affect the variables $\textit{to-be-requested?}(h')$ and $\textit{archived-pkts?}(h')$. In the case of either `EXP-RQST`, `RQST-UPDATE`, or `REPL-UPDATE` packets, the $\texttt{process-pkt}_h(p)$ may only add the identifiers of trailing missing packets from h' to $\textit{to-be-requested?}(h')$ and does not affect $\textit{archived-pkts?}(h')$. Trailing packets are not in the current window of h' and, thus, Invariant 6.5 implies that the identifiers of trailing packets from h' are not in $\textit{archived-pkts?}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds following the occurrence of $\texttt{process-pkt}_h(p)$. In the case of a `EXP-REPL` packet, the $\texttt{process-pkt}_h(p)$ action may only add the identifiers of trailing missing packets from h' to $\textit{to-be-requested?}(h')$ and adds the identifier of p to $\textit{archived-pkts?}(h')$. Once again, Invariant 6.5 implies that the identifiers of any trailing missing packets are not in $\textit{archived-pkts?}(h')$. Moreover, Invariant 6.13 implies that the identifier of p is not in $\textit{to-be-requested?}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds following the occurrence of $\texttt{process-pkt}_h(p)$. ❒

Invariant 6.20 *For $h, h' \in H$ and any reachable state u of* CESRM-REC$_h$*, it is the case that* $u.\textit{scheduled-rqsts?}(h') \cap u.\textit{archived-pkts?}(h') = \emptyset$.

Proof: The proof is similar to the induction used in the proof of Invariant 4.14. In this case, we must also consider the discrete transitions involving a $\texttt{process-pkt}_h(p)$ action, for $p \in P_{\text{CESRM}}$, such that $\textit{type}(p) = \texttt{EXP-REPL}$; the other actions introduced in CESRM-REC$_h$ do not affect the variables $\textit{scheduled-rqsts?}(h')$ and $\textit{archived-pkts?}(h')$. In this case, the $\texttt{process-pkt}_h(p)$ action removes the element $\textit{id}(p)$ from $\textit{scheduled-rqsts?}(h')$ whenever it adds it to $\textit{archived-pkts?}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds following the occurrence of $\texttt{process-pkt}_h(p)$. ❒

Invariant 6.21 *For $h, h' \in H$ and any reachable state u of* CESRM-REC$_h$*, it is the case that* $u.\textit{to-be-requested?}(h') \cap u.\textit{scheduled-rqsts?}(h') = \emptyset$.

Proof: The proof is similar to the induction used in the proof of Invariant 4.18. In this case, we must also consider the discrete transitions involving a $\texttt{process-pkt}_h(p)$ action, for $p \in P_{\text{CESRM}}$, such that $\textit{type}(p) \in \{\texttt{EXP-RQST}, \texttt{EXP-REPL}, \texttt{RQST-UPDATE}, \texttt{REPL-UPDATE}\}$; the other actions introduced in CESRM-REC$_h$ do not affect the variables $\textit{to-be-requested?}(h')$ and $\textit{scheduled-rqsts?}(h')$. In the case of either `EXP-RQST`, `RQST-UPDATE`, or `REPL-UPDATE` packets, the $\texttt{process-pkt}_h(p)$ may only add the identifiers of trailing missing packets from h' to $\textit{to-be-requested?}(h')$ and does not affect $\textit{scheduled-rqsts?}(h')$. Trailing packets are not in the current window of h' and, thus, Invariant 6.15 implies that the identifiers of trailing packets from h' are not in $\textit{scheduled-rqsts?}(h')$. Thus, the induction hypothesis implies that the invariant assertion holds following the occurrence of $\texttt{process-pkt}_h(p)$. In the case of a `EXP-REPL` packet, the effects of the $\texttt{process-pkt}_h(p)$ action with respect to the $\textit{to-be-requested?}(h')$ and $\textit{scheduled-rqsts?}(h')$ variables are identical to those of a `REPL` packet. Thus, the inductive reasoning for the case of a `EXP-REPL` packet is identical to that of a `REPL` packet in the proof of Invariant 4.18. ❒

Invariant 6.22 *Let u be any reachable state of* CESRM-REC$_h$*. For $s \in H$, $i \in \mathbb{N}$, $t, t' \in \mathbb{R}^{\geq 0}$, and $k \in \mathbb{N}^+$, if $\langle s, i, t \rangle \in \textit{pending-rqsts}$ and $\langle s, i, t', k \rangle \in \textit{scheduled-rqsts}$, then $t < t'$.*

Proof: The proof is identical to that of Invariant 4.19. ❐

Invariant 6.23 *Let u be any reachable state of* CESRM-REC$_h$. *For $h, s \in H$ and $i \in \mathbb{N}$, if the action* $\texttt{send-rqst}_h(s,i)$ *is enabled in u, i.e., $u.Pre(\texttt{send-rqst}_h(s,i)) = \texttt{True}$, then $\langle s,i\rangle \notin$ u.pending-rqsts?.*

Proof: The proof is identical to that of Invariant 4.20. ❐

Lemma 6.2 *Let $u, u' \in$ states(CESRM$_I$) be any reachable states of CESRM$_I$, α be any timed execution fragment of CESRM$_I$, such that $u = \alpha$.fstate and $u' = \alpha$.lstate. It is the case that u[CESRM].sent-pkts $\subseteq u'$[CESRM].sent-pkts.*

Proof: The proof is identical to that of Lemma 4.2. ❐

Invariant 6.24 *Let $u \in$ states(CESRM$_I$) be any reachable state of CESRM$_I$. For any $s \in H$ and $i, i' \in \mathbb{N}, i \leq i'$, if $\langle s,i\rangle \in u$[CESRM].sent-pkts?(s) and $\langle s,i'\rangle \in u$[CESRM].sent-pkts?$(s)$, then it is the case that $\langle s,i''\rangle \in u$[CESRM].sent-pkts?$(s)$, for any $i'' \in \mathbb{N}, i \leq i'' \leq i'$.*

Proof: The proof is identical to that of Invariant 4.21. ❐

Lemma 6.3 *Let $s, h \in H$, $i \in \mathbb{N}$, and $u \in$ states(CESRM$_I$) be any reachable state of CESRM$_I$, such that $\langle s,i\rangle \in u$[CESRM-REC$_h$].archived-pkts?. Moreover, let α be any timed execution fragment of CESRM$_I$ that starts in u, does not contain a* $\texttt{rm-leave}_h$ *action, and ends in some $u' \in$ states(CESRM$_I$). Then, it is the case that $\langle s,i\rangle \in u'$[CESRM-REC$_h$].archived-pkts?.*

Proof: The proof is identical to that of Lemma 4.3. ❐

Lemma 6.4 *Let $h \in H$, $i \in \mathbb{N}$, and $u \in$ states(CESRM$_I$) be any reachable state of CESRM$_I$, such that u[CESRM-MEM$_h$].status = $\texttt{crashed}$. Moreover, let α be any timed execution fragment of CESRM$_I$ that starts in u and ends in some $u' \in$ states(CESRM$_I$). Then, it is the case that u'[CESRM-MEM$_h$].status = $\texttt{crashed}$.*

Proof: The proof is identical to that of Lemma 4.4. ❐

Lemma 6.5 *Let $s, h \in H$, $i \in \mathbb{N}$, and $u \in$ states(CESRM$_I$) be any reachable state of CESRM$_I$, such that $\langle s,i\rangle \in u$[CESRM-REC$_h$].scheduled-rqsts?. Moreover, let α be any timed execution fragment of CESRM$_I$ that starts in u, does not contain a* $\texttt{rm-leave}_h$ *action, and ends in some $u' \in$ states(CESRM$_I$). Then, either $\langle s,i\rangle \in u'$[CESRM-REC$_h$].scheduled-rqsts? or $\langle s,i\rangle \in u'$[CESRM-REC$_h$].archived-pkts?.*

Proof: The proof is identical to that of Lemma 4.5. ❐

Lemma 6.6 *Let $s, h \in H$, $i \in \mathbb{N}$, $t \in \mathbb{R}^{\geq 0}$, $k \in \mathbb{N}^+$, and $u \in$ states(CESRM$_I$) be any reachable state of CESRM$_I$, such that u[CESRM-REC$_h$].status = $\texttt{member}$ and $\langle s,i,t,k\rangle \in u$[CESRM-REC$_h$].scheduled-rqsts. Moreover, let α be any timed execution fragment of CESRM$_I$ that starts in u, contains neither* $\texttt{crash}_h$ *nor* $\texttt{rm-leave}_h$ *actions, and ends in some $u' \in$ states(CESRM$_I$), such that $t < u'$.now and $\langle s,i,t',k'\rangle \in u'$[CESRM-REC$_h$].scheduled-rqsts, for $t' \in \mathbb{R}^{\geq 0}$ and $k' \in \mathbb{N}^+$. Then, it is the case that $k < k'$.*

Proof: The proof is identical to that of Lemma 4.6. ❐

Lemma 6.7 *The occurrence of either a* $\texttt{send-rqst}_h(s,i)$, $\texttt{send-repl}_h(s,i)$, $\texttt{send-exp-rqst}_h(s,i)$, $\texttt{send-exp-repl}_h(s,i)$, $\texttt{send-rqst-update}_h(s,i)$, *or* $\texttt{send-repl-update}_h(s,i)$ *action, for* $h, s \in H$, *and* $i \in \mathbb{N}$, *in any admissible timed execution* α *of* CESRM_I *is instantaneously succeeded in* α *by the occurrence of either a* $\texttt{crash}_h$, $\texttt{rm-leave}_h$, *or* $\texttt{rec-msend}_h(p)$ *action, for* $p \in P_{\mathrm{CESRM}}$, $id(p) = \langle s, i \rangle$, *and* $type(p)$ *equal to either* RQST, REPL, EXP-RQST, EXP-REPL, RQST-UPDATE, *or* REPL-UPDATE, *respectively.*

Proof: The proof is identical to that of Lemma 4.7. ❐

Lemma 6.8 *Let* α *be any admissible execution of* CESRM_I *containing the discrete transition* (u, π, u'), *for* $u, u' \in states(\mathrm{CESRM}_I)$, $h \in H$, $p \in P_{\mathrm{RM\text{-}CLIENT}}$, $\langle s_p, i_p \rangle = id(p)$, *and* $\pi = \texttt{rm-send}_h(p)$. *If either* $u[\mathrm{CESRM\text{-}REC}_h].\mathit{min\text{-}seqno}(s_p) = \perp$ *or* $u[\mathrm{CESRM\text{-}REC}_h].\mathit{min\text{-}seqno}(s_p) \neq \perp \wedge i_p = u[\mathrm{CESRM\text{-}REC}_h].\mathit{max\text{-}seqno}(s_p) + 1$, *then the discrete transition* (u, π, u') *is instantaneously succeeded in* α *by the occurrence of either a* $\texttt{crash}_h$, $\texttt{rm-leave}_h$, *or* $\texttt{rec-msend}_h(pkt)$ *action, for* $pkt \in P_{\mathrm{CESRM}}$, *such that* $pkt = \mathit{comp\text{-}data\text{-}pkt}(p)$.

Proof: The proof is identical to that of Lemma 4.10. ❐

Lemma 6.9 *The occurrence of an action* $\texttt{rec-msend}_h(p)$, *for* $h \in H$ *and* $p \in P_{\mathrm{CESRM}}$, *in any admissible timed execution* α *of* CESRM_I *is instantaneously succeeded in* α *by the occurrence of either a* $\texttt{crash}_h$, $\texttt{rm-leave}_h$, *or* $\texttt{msend}_h(pkt)$ *action, for* $pkt \in P_{\mathrm{IPMCAST\text{-}CLIENT}}$, *such that* $strip(pkt) = p$.

Proof: The proof is identical to that of Lemma 4.8. ❐

Lemma 6.10 *The occurrence of an action* $\texttt{mrecv}_h(pkt)$, *for* $h \in H$ *and* $pkt \in P_{\mathrm{IPMCAST\text{-}CLIENT}}$, *in a state* $u \in states(\mathrm{CESRM}_I)$ *in any admissible timed execution* α *of* CESRM_I, *such that* $u[\mathrm{CESRM\text{-}IPBUFF}_h].status = \texttt{member}$, *is instantaneously succeeded in* α *by the occurrence of either a* $\texttt{crash}_h$, $\texttt{rm-leave}_h$, *or* $\texttt{process-pkt}_h(p)$ *action, for* $p \in P_{\mathrm{CESRM}}$, *such that* $p = strip(pkt)$.

Proof: The proof is identical to that of Lemma 4.9. ❐

We now present some invariants pertaining to the CESRM_I automaton.

Invariant 6.25 *For* $h \in H$ *and any reachable state* u *of* CESRM_I, *it is the case that:*

1. $u[\mathrm{RM\text{-}CLIENT}_h].status = \texttt{idle} \Leftrightarrow u[\mathrm{CESRM\text{-}MEM}_h].status = \texttt{idle}$,
2. $u[\mathrm{RM\text{-}CLIENT}_h].status = \texttt{member} \Leftrightarrow u[\mathrm{CESRM\text{-}MEM}_h].status = \texttt{member}$,
3. $u[\mathrm{RM\text{-}CLIENT}_h].status = \texttt{crashed} \Leftrightarrow u[\mathrm{CESRM\text{-}MEM}_h].status = \texttt{crashed}$,
4. $u[\mathrm{RM\text{-}CLIENT}_h].status = \texttt{joining} \Leftrightarrow u[\mathrm{CESRM\text{-}MEM}_h].status \in Joining$, *and*
5. $u[\mathrm{RM\text{-}CLIENT}_h].status = \texttt{leaving} \Leftrightarrow u[\mathrm{CESRM\text{-}MEM}_h].status \in Leaving$.

Proof: The proof is identical to that of Invariant 4.22. ❐

Invariant 6.26 *For $h \in H$ and any reachable state u of* CESRM_I, *it is the case that* $u[\text{RM-CLIENT}_h].seqno = u[\text{CESRM-REC}_h].max\text{-}seqno(h)$.

Proof: The proof is identical to that of Invariant 4.23. ❐

Invariant 6.27 *For $h \in H$ and any reachable state u of* CESRM_I, *it is the case that:*

1. $u[\text{CESRM-MEM}_h].status = \texttt{crashed} \Leftrightarrow u[\text{CESRM-IPBUFF}_h].status = \texttt{crashed}$ $\wedge u[\text{CESRM-MEM}_h].status = \texttt{member} \Leftrightarrow u[\text{CESRM-IPBUFF}_h].status = \texttt{member}$ *and*
2. $u[\text{CESRM-MEM}_h].status = \texttt{crashed} \Leftrightarrow u[\text{CESRM-REC}_h].status = \texttt{crashed}$ $\wedge u[\text{CESRM-MEM}_h].status = \texttt{member} \Leftrightarrow u[\text{CESRM-REC}_h].status = \texttt{member}$.

Proof: The proof is identical to that of Invariant 4.24. ❐

Invariant 6.28 *For $h \in H$ and any reachable state u of* CESRM_I, *it is the case that, for any packet* $p \in u[\text{CESRM-REC}_h].msend\text{-}buff$:

1. $type(p) = \texttt{SESS} \Rightarrow \forall \langle h', i' \rangle \in seqno\text{-}rprts(p), \langle h', i' \rangle \in u[\text{CESRM-REC}_h].window?(h')$, *and*
2. $type(p) \neq \texttt{SESS} \Rightarrow id(p) \in u[\text{CESRM-REC}_h].window?(source(p))$.

Proof: The proof is similar to the induction used in the proof of Invariant 6.14. In this case, we must also consider the discrete transitions involving either $\texttt{send-rqst-update}_h(s, i)$, $\texttt{send-repl-update}_h(s, i)$, or $\texttt{process-pkt}_h(p)$ actions, for $s \in H$, $i \in \mathbb{N}$ and $p \in P_{\text{CESRM}}$, such that $type(p) \in \{\texttt{EXP-RQST}, \texttt{EXP-REPL}, \texttt{RQST-UPDATE}, \texttt{REPL-UPDATE}\}$; the other actions introduced in CESRM-REC_h do not affect the variables $\text{CESRM-REC}_h.msend\text{-}buff$ and $\text{CESRM-REC}_h.window?(h')$.

The action $\texttt{send-rqst-update}_h(s, i)$ adds a $\texttt{RQST-UPDATE}$ packet for $\langle s, i \rangle$ to $\text{CESRM-REC}_h.msend\text{-}buff$ and does not affect $\text{CESRM-REC}_h.window?(h')$. The precondition of $\texttt{send-rqst-update}_h(s, i)$ implies that $\langle s, i \rangle \in u_k[\text{CESRM-REC}_h].rqst\text{-}updates?(s)$. Thus, Invariant 6.18 implies that $\langle s, i \rangle \in u_k[\text{CESRM-REC}_h].window?(s)$. Since $\texttt{send-rqst-update}_h(s, i)$ does not affect $\text{CESRM-REC}_h.window?(h')$, it follows that $\langle s, i \rangle \in u[\text{CESRM-REC}_h].window?(s)$.

The action $\texttt{send-repl-update}_h(s, i)$ adds a $\texttt{REPL-UPDATE}$ packet for $\langle s, i \rangle$ to $\text{CESRM-REC}_h.msend\text{-}buff$ and does not affect $\text{CESRM-REC}_h.window?(h')$. The precondition of $\texttt{send-repl-update}_h(s, i)$ implies that $\langle s, i \rangle \in u_k[\text{CESRM-REC}_h].repl\text{-}updates?(s)$. Thus, Invariant 6.17 implies that $\langle s, i \rangle \in u_k[\text{CESRM-REC}_h].window?(s)$. Since $\texttt{send-repl-update}_h(s, i)$ does not affect $\text{CESRM-REC}_h.window?(h')$, it follows that $\langle s, i \rangle \in u[\text{CESRM-REC}_h].window?(s)$.

The $\texttt{process-pkt}_h(p)$ action does not affect $\text{CESRM-REC}_h.msend\text{-}buff$ and may only add elements to $\text{CESRM-REC}_h.window?(h')$. Thus, the induction hypothesis implies that the invariant assertion holds following the occurrence of $\texttt{process-pkt}_h(p)$. ❐

Invariant 6.29 *For any reachable state u of* CESRM_I, *it is the case that, for all $h, h' \in H$,* $u[\text{CESRM-REC}_h].window?(h') \subseteq u[\text{CESRM}].sent\text{-}pkts?(h')$.

Proof: The proof is analogous to the induction used in the proof of Invariant 6.29. ❐

Invariant 6.30 *For any reachable state u of* CESRM_I, *it is the case that, for all $h, h' \in H$,* $u[\text{CESRM-REC}_h].archived\text{-}pkts?(h') \subseteq u[\text{CESRM}].sent\text{-}pkts?(h')$.

Proof: Follows directly from Invariants 6.5 and 6.29. ❐

Invariant 6.31 *For $h, h' \in H$ and any reachable state u of* CESRM_I*, it is the case that* $u[\text{CESRM-}\textsc{rec}_h].\textit{to-be-delivered?}(h') \subseteq u[\text{CESRM}].\textit{sent-pkts?}(h')$.

Proof: Follows directly from Invariants 6.4 and 6.30. ❐

6.3.2 Correctness Analysis

In this section, we show that our reliable multicast implementation CESRM_I indeed implements the reliable multicast service specification $\text{RM}_S(\infty)$. We begin by defining a relation R from CESRM_I to $\text{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$. This relation is identical to that relating CESRM_I to $\text{RM}_S(\Delta)$ in Section 4.4.4. We repeat it here for completeness.

Definition 6.1 *Let R be the relation between states of* CESRM_I *and* $\text{RM}_S(\Delta)$*, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, such that for any states u and s of* CESRM_I *and* $\text{RM}_S(\Delta)$*, respectively, $(u, s) \in R$ provided that, for all $h, h' \in H$ and $p \in P_{\text{RM-}\textsc{Client}}$, such that $\langle s_p, i_p \rangle = id(p)$, it is the case that:*

$$
\begin{aligned}
s.\textit{now} &= u.\textit{now} \\
s[\text{RM-}\textsc{Client}_h].\textit{status} &= u[\text{RM-}\textsc{Client}_h].\textit{status} \\
s[\text{RM-}\textsc{Client}_h].\textit{seqno} &= u[\text{RM-}\textsc{Client}_h].\textit{seqno} \\
s[\text{RM}(\Delta)].\textit{status}(h) &= \begin{cases} \texttt{idle} & \textit{if } u[\text{CESRM-}\textsc{mem}_h].\textit{status} = \texttt{idle} \\ \texttt{joining} & \textit{if } u[\text{CESRM-}\textsc{mem}_h].\textit{status} \in \textit{Joining} \\ \texttt{leaving} & \textit{if } u[\text{CESRM-}\textsc{mem}_h].\textit{status} \in \textit{Leaving} \\ \texttt{member} & \textit{if } u[\text{CESRM-}\textsc{mem}_h].\textit{status} = \texttt{member} \\ \texttt{crashed} & \textit{if } u[\text{CESRM-}\textsc{mem}_h].\textit{status} = \texttt{crashed} \end{cases} \\
s[\text{RM}(\Delta)].\textit{trans-time}(p) &= u[\text{CESRM-}\textsc{rec}_{s_p}].\textit{trans-time}(p) \\
s[\text{RM}(\Delta)].\textit{expected}(h, h') &= u[\text{CESRM-}\textsc{rec}_h].\textit{expected}(h') \\
s[\text{RM}(\Delta)].\textit{delivered}(h, h') &= u[\text{CESRM-}\textsc{rec}_h].\textit{delivered}(h')
\end{aligned}
$$

The following lemma states that the relation R of Definition 6.1 is a timed forward simulation relation from CESRM_I to $\text{RM}_S(\infty)$.

Lemma 6.11 *R is a timed forward simulation relation from* CESRM_I *to* $\text{RM}_S(\infty)$.

Proof: The proof is identical to that of Lemma 4.11. ❐

Theorem 6.12 $\text{CESRM}_I \leq \text{RM}_S(\infty)$

Proof: Follows directly from Lemma 6.11. ❐

6.4 CESRM Timeliness

In this section, we prove some timeliness guarantees of CESRM. We begin by showing that when hosts neither crash nor leave the reliable multicast group and the number of packet drops pertaining to the transmission and, potentially, the recovery of any packet is bounded, CESRM_I implements $\mathrm{RM}_S(\Delta_L)$, for a particular $\Delta_L \in \mathbb{R}^{\geq 0}$.

We then strengthen this timeliness guarantee by weakening the assumption that hosts neither crash nor leave the reliable multicast group. Our weaker assumption states that only reliable multicast transmission sources (as opposed to all hosts) neither crash nor leave the reliable multicast group. This weaker assumption is also reasonably practical, since in a real-life system it may be possible to design sources to be highly robust to failures (*e.g.*, through transparent replication).

We further strengthen our timeliness guarantee by once again weakening our assumption that sources neither crash nor leave the reliable multicast group. Our new assumption states that whenever a loss is detected by any host h, there is some other host h' that has delivered the packet (and is thus capable of retransmitting it) and neither crashes nor leaves for a sufficiently long period of time $\Delta_R \in \mathbb{R}^{\geq 0}$ thereafter. By choosing Δ_R to be long enough so that h can recover the packet from h', we show that h recovers the given packet within $\Delta_L = \texttt{DET-BOUND} + \Delta_R$ time units, where $\texttt{DET-BOUND} \in \mathbb{R}^{\geq 0}$ is an upper bound on the amount of time needed for h to detect the given loss.

We conclude our timeliness analysis by comparing the worst-case recovery latency incurred by successful expedited and non-expedited first-round recoveries. In particular, we show that successful expedited recoveries complete within at most $\texttt{DET-BOUND} + \texttt{RQST-DELAY} + 2\overline{d}$ time units, where $\overline{d}$ is an upper bound on the inter-host transmission latency. Furthermore, we show that successful non-expedited first-round recoveries complete within at most $\texttt{DET-BOUND} + (C_1+C_2)\overline{d} + \overline{d} + (D_1+D_2)\overline{d} + \overline{d}$ time units. This analysis reveals that, for typical SRM request and reply scheduling parameter values [13], the worst-case recovery latency of packets recovered by expedited rather than first-round recoveries in CESRM is reduced by roughly $3\overline{RTT}$, where $\overline{RTT} = 2\,\overline{d}$ is an upper bound on the inter-host round-trip-time.

6.4.1 Timeliness Analysis Preliminaries

The execution Constraints 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 4.7, and 4.8 defined for the SRM_I automaton in Section 4.4.5 carry over to the CESRM_I automaton unchanged. We proceed by defining bounded unicast transmission latency and resolution execution constraints for CESRM_I; these constraints are analogous to Constraints 4.1 and 4.2.

Constraint 6.1 (Bounded Unicast Transmission Latency) *Let α be any admissible timed execution of* CESRM_I *and h, h' be any two distinct hosts in H. The transmission latency incurred by any packet sent by h using the IP unicast service and received by h' in α lies in the interval $[\underline{d}, \overline{d}]$; that is, for any packet $p \in P_{\mathrm{IPUCAST\text{-}CLIENT}}$ multicast by h in α, the time elapsing from the time of occurrence of the action $\texttt{usend}_h(p)$ in α to that of any action $\texttt{urecv}_{h'}(p)$ in α lies in the interval $[\underline{d}, \overline{d}]$.*

Constraint 6.2 (Bounded Unicast Transmission Resolution) *Let α be any admissible timed execution of* CESRM_I*. The fate of any packet sent using the IP unicast service is resolved within at most $\overline{d}$ time units past its transmission time; that is, letting $p \in P_{\mathrm{IPUCAST\text{-}CLIENT}}$ be any packet unicast in α, (u, π, u'), for $u, u' \in states(\mathrm{CESRM}_I)$, $s_p = source(p)$, and $\pi = \texttt{usend}_{s_p}(p)$, be the discrete transition involving the transmission of p in α, and $d_p = dest(p)$ be the destination of p, it is the case that either a $\texttt{crash}_{d_p}$, $\texttt{urecv}_{s_p}(p)$, or $\texttt{udrop}(p)$ action occurs no later than $\overline{d}$ time units after the particular occurrence of the discrete transition (u, π, u') in α.*

We now define CESRM$_I$ versions of the execution sets defined for the SRM$_I$ automaton in Section 4.4.5.

Let $aexecs_k(\mathrm{CESRM}_I)$, for $k \in \mathbb{N}^+$, be the set of admissible timed executions of CESRM$_I$ in which the number of drops suffered by IP packets pertaining to the transmission and, potentially, the recovery of any packet $p \in P_{\text{RM-CLIENT}}$ is at most k. That is, $\alpha \in aexecs_k(\mathrm{CESRM}_I)$ if and only if, for any $p \in P_{\text{RM-CLIENT}}$, α contains at most k either $\texttt{mdrop}(pkt, H_d)$, for $pkt \in P_{\text{IPMCAST-CLIENT}}$ and $H_d \subseteq H$, such that $strip(pkt) \in P_{\text{CESRM}}[p]$, or $\texttt{udrop}(pkt')$, for $pkt' \in P_{\text{IPUCAST-CLIENT}}$, such that $strip(pkt') \in P_{\text{CESRM}}[p]$, actions.

Let $timely\text{-}aexecs(\mathrm{CESRM}_I)$, for $\Delta \in \mathbb{R}^{\geq 0}$, be the set of all admissible timed executions of CESRM$_I$ in $aexecs(\mathrm{CESRM}_I)$ that satisfy Constraints 4.1, 4.2, 4.3, 4.4, 6.1, and 6.2. Let $recoverable\text{-}aexecs(\mathrm{CESRM}_I)$ be the subset of the admissible timed executions of CESRM$_I$ in $timely\text{-}aexecs(\mathrm{CESRM}_I)$ that satisfy Constraints 4.5 and 4.6. Let $\Delta_L\text{-}src\text{-}recoverable\text{-}aexecs(\mathrm{CESRM}_I)$, for some $\Delta_L \in \mathbb{R}^{\geq 0}$, be the subset of the admissible timed executions of CESRM$_I$ in $timely\text{-}aexecs(\mathrm{CESRM}_I)$ that satisfy Constraint 4.7, for some $\Delta_L \in \mathbb{R}^{\geq 0}$. Let $\Delta_R\text{-}recoverable\text{-}aexecs(\mathrm{CESRM}_I)$, for some $\Delta_R \in \mathbb{R}^{\geq 0}$, be the subset of the admissible timed executions of CESRM$_I$ in $timely\text{-}aexecs(\mathrm{CESRM}_I)$ that satisfy Constraint 4.8, for some $\Delta_R \in \mathbb{R}^{\geq 0}$. Moreover, for $k \in \mathbb{N}^+$, let $timely\text{-}aexecs_k(\mathrm{CESRM}_I) = timely\text{-}aexecs(\mathrm{CESRM}_I) \cap aexecs_k(\mathrm{CESRM}_I)$, $recoverable\text{-}aexecs_k(\mathrm{CESRM}_I) = recoverable\text{-}aexecs(\mathrm{CESRM}_I) \cap aexecs_k(\mathrm{CESRM}_I)$, $\Delta_L\text{-}src\text{-}recoverable\text{-}aexecs_k(\mathrm{CESRM}_I) = \Delta_L\text{-}src\text{-}recoverable\text{-}aexecs(\mathrm{CESRM}_I) \cap aexecs_k(\mathrm{CESRM}_I)$, and $\Delta_R\text{-}recoverable\text{-}aexecs_k(\mathrm{CESRM}_I) = \Delta_R\text{-}recoverable\text{-}aexecs(\mathrm{CESRM}_I) \cap aexecs_k(\mathrm{CESRM}_I)$.

The sets of admissible timed traces corresponding to each of the above admissible timed execution sets of CESRM$_I$ are defined analogously to the respective sets defined for the SRM$_I$ automaton in Section 4.4.5.

We now proceed by restating the preliminary lemmas of Section 4.4.5 for the CESRM$_I$ automaton.

Lemma 6.13 *Let α be any admissible timed execution of* CESRM$_I$ *that satisfies Constraint 4.1 and contains the occurrence of a discrete transition (u, π, u'), for $u, u' \in states(\mathrm{CESRM}_I)$, $h \in H$, $p \in P_{\text{IPMCAST-CLIENT}}$, and $\pi = \texttt{mrecv}_h(p)$. Then, any $\texttt{mrecv}_{h'}(p)$ action, for $h' \in H$, in α occurs no earlier and no later than $\overline{d} - \underline{d}$ time units from the discrete transition (u, π, u') in α.*

Proof: The proof is identical to that of Lemma 4.17. ❐

Lemma 6.14 *Let α be any admissible timed execution of* CESRM$_I$ *that contains the transmission of a packet $p \in P_{\text{RM-CLIENT}}$. For any state $u \in states(\mathrm{CESRM}_I)$ in α, if $u.trans\text{-}time(p) \neq \perp$, then $u.trans\text{-}time(p) = \alpha.trans\text{-}time(p)$.*

Proof: The proof is identical to that of Lemma 4.18. ❐

Lemma 6.15 *Let $h, h' \in H$, $\alpha \in aexecs(\mathrm{CESRM}_I)$, $u, u' \in states(\mathrm{CESRM}_I)$ be any states in α, such that $u \leq_\alpha u'$, and $\alpha_{uu'}$ be the finite execution fragment of α leading from u to u'. If $u[\text{CESRM-REC}_h].expected(h') \neq \emptyset$ and $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, then it is the case that $u[\text{CESRM-REC}_h].expected(h') = u'[\text{CESRM-REC}_h].expected(h')$.*

Proof: The proof is identical to that of Lemma 4.19. ❐

Lemma 6.16 *Let $h, h' \in H$, $\alpha \in aexecs(\mathrm{CESRM}_I)$, $u, u' \in states(\mathrm{CESRM}_I)$ be any states in α, such that $u \leq_\alpha u'$, and $\alpha_{uu'}$ be the execution fragment of α leading from u to u'. If $\alpha_{uu'}$ contains neither* $\texttt{crash}_h$ *nor* $\texttt{rm-leave}_h$ *actions, then it is the case that $u[\text{CESRM-REC}_h].expected(h') \subseteq u'[\text{CESRM-REC}_h].expected(h')$.*

Proof: The proof is identical to that of Lemma 4.20. ❒

Lemma 6.17 *Let $h, h' \in H$, $\alpha \in aexecs(\mathrm{CESRM}_I)$, $u, u' \in states(\mathrm{CESRM}_I)$ be any states in α, such that $u \leq_\alpha u'$, and $\alpha_{uu'}$ be the finite execution fragment of α leading from u to u'. If $\alpha_{uu'}$ contains neither* $\texttt{crash}_h$ *nor* $\texttt{rm-leave}_h$ *actions, then it is the case that $u[\text{CESRM-REC}_h].delivered(h') \subseteq u'[\text{CESRM-REC}_h].delivered(h')$.*

Proof: The proof is identical to that of Lemma 4.21. ❒

Lemma 6.18 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of CESRM_I in timely-aexecs(CESRM_I) that contains the transmission of p. Let (w, π, w'), for $w, w' \in states(\mathrm{CESRM}_I)$, $s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of CESRM_I involving the transmission of p using the reliable multicast service. For any states u, u' of CESRM_I in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. For any $k \in \mathbb{N}^+$ and $h \in H, h \neq s_p$, suppose that $\alpha_{uu'}$ contains neither* $\texttt{crash}_h$ *nor* $\texttt{rm-leave}_h$ *actions and h schedules k-th and $k+1$-st round requests for p in $\alpha_{uu'}$. Let $t_k, t_{k+1} \in \mathbb{R}^{\geq 0}$ be the points in time in $\alpha_{uu'}$ at which the host h schedules its k-th and $k+1$-st round requests for p, respectively. Then, it is the case that $t_{k+1} \leq t_k + 2^{k-1}(C_1 + C_2)\overline{d}$.*

Proof: The proof is identical to that of Lemma 4.22. ❒

Corollary 6.19 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of CESRM_I in timely-aexecs(CESRM_I) that contains the transmission of p. Let (w, π, w'), for $w, w' \in states(\mathrm{CESRM}_I)$, $p \in P_{\text{RM-CLIENT}}$, $s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of CESRM_I involving the transmission of p using the reliable multicast service. For any states u, u' of CESRM_I in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. For any $k \in \mathbb{N}^+$ and $h \in H, h \neq s_p$, suppose that $\alpha_{uu'}$ contains neither* $\texttt{crash}_h$ *nor* $\texttt{rm-leave}_h$ *actions and contains the discrete transition in which h detects the loss of p. Moreover, suppose that, following the detection of p in $\alpha_{uu'}$, h schedules a $k+1$-st round request for p in $\alpha_{uu'}$. Let $t_{k+1} \in \mathbb{R}^{\geq 0}$ be the point in time in $\alpha_{uu'}$ at which the host h schedules its $k+1$-st round request for p. Then, it is the case that $t_{k+1} \leq \alpha_{uu'}.det\text{-}time_h(p) + (2^k - 1)(C_1 + C_2)\overline{d}$.*

Proof: The proof is identical to that of Corollary 4.23. ❒

Lemma 6.20 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of CESRM_I in timely-aexecs(CESRM_I) that contains the transmission of p. Let (w, π, w'), for $w, w' \in states(\mathrm{CESRM}_I)$, $s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of CESRM_I involving the transmission of p using the reliable multicast service. For any states u, u' of CESRM_I in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. For any $k \in \mathbb{N}^+$ and $h \in H, h \neq s_p$, suppose that $\alpha_{uu'}$ contains neither* $\texttt{crash}_h$ *nor* $\texttt{rm-leave}_h$ *actions and h schedules k-th and $k+1$-st round requests for p in $\alpha_{uu'}$. Let $t_k, t_{k+1} \in \mathbb{R}^{\geq 0}$ be the points in time in $\alpha_{uu'}$ at which the host h schedules its k-th and $k+1$-st round requests for p, respectively. Then, it is the case that $t_k + 2^{k-1}C_3\underline{d} < t_{k+1}$.*

Proof: The proof is identical to that of Lemma 4.24. ❐

Lemma 6.21 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of* CESRM_I *in timely-aexecs*(CESRM_I) *that contains the transmission of p. Let (w, π, w'), for $w, w' \in$ states*(CESRM_I), *$s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of* CESRM_I *in α involving the transmission of p using the reliable multicast service. For any states u, u' of* CESRM_I *in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. Moreover, let $h \in H$ be any member of the reliable multicast group in u, such that $id(p) \in u[\text{CESRM-REC}_h].expected(s_p)$ and $id(p) \notin u[\text{CESRM-REC}_h].scheduled\text{-}rqsts?$.*

For $k \in \mathbb{N}^+, k \geq k^*_{rqst}$, suppose that $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, h schedules k-th and $k+1$-st round requests for p in $\alpha_{uu'}$, and h either sends or receives its k-th and $k+1$-st round requests for p at the points in time $t_{k+1}, t_{k+2} \in \mathbb{R}^{\geq 0}$ in $\alpha_{uu'}$.

Then, the k-th and $k+1$-st round requests of h for p are distinct.

Proof: The proof is identical to that of Lemma 4.25. ❐

Lemma 6.22 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of* CESRM_I *in timely-aexecs*(CESRM_I) *that contains the transmission of p. Let (w, π, w'), for $w, w' \in$ states*(CESRM_I), *$s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of* CESRM_I *involving the transmission of p using the reliable multicast service. For any states u, u' of* CESRM_I *in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. For any $k \in \mathbb{N}^+$ and $h, h' \in H, h \neq h'$, suppose that $u[\text{CESRM-MEM}_h].status = \texttt{member}$, $u[\text{CESRM-MEM}_{h'}].status = \texttt{member}$, $\alpha_{uu'}$ contains neither $\texttt{crash}_h$, $\texttt{rm-leave}_h$, $\texttt{crash}_{h'}$, nor $\texttt{rm-leave}_{h'}$ actions, h schedules k-th and $k+1$-st round requests for the packet p in $\alpha_{uu'}$, h either sends or receives its k-th round request for p and schedules its $k+1$-st round request for p at the point in time $t_{k+1} \in \mathbb{R}^{\geq 0}$ in $\alpha_{uu'}$, and $t_{k+1} + \overline{d} < u'.now$. Then, h' may receive the k-th round request of h for p no later than $t_{k+1} + \overline{d}$ in α.*

Proof: The proof is identical to that of Lemma 4.26. ❐

Lemma 6.23 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of* CESRM_I *in timely-aexecs*(CESRM_I) *that contains the transmission of p. Let (w, π, w'), for $w, w' \in$ states*(CESRM_I), *$s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of* CESRM_I *involving the transmission of p using the reliable multicast service. For any states u, u' of* CESRM_I *in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. For any $k \in \mathbb{N}^+$ and $h, h' \in H, h \neq h'$, suppose that $u[\text{CESRM-MEM}_h].status = \texttt{member}$, $u[\text{CESRM-MEM}_{h'}].status = \texttt{member}$, $id(p) \in u[\text{CESRM-REC}_{h'}].archived\text{-}pkts?$, $\alpha_{uu'}$ contains neither $\texttt{crash}_h$, $\texttt{rm-leave}_h$, $\texttt{crash}_{h'}$, nor $\texttt{rm-leave}_{h'}$ actions, h' receives a request for p from h at time $t' \in \mathbb{R}^{\geq 0}$ in $\alpha_{uu'}$, and $t' + (D_1 + D_2)\overline{d} < u'.now$. Then, the reply of h' pertaining to this particular request of h for p is either sent or received by h' no later than $t' + (D_1 + D_2)\overline{d}$ in α.*

Proof: The proof is identical to that of Lemma 4.27. ❐

Lemma 6.24 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of* CESRM_I *in timely-aexecs*(CESRM_I) *that contains the transmission of p. Let (w, π, w'), for $w, w' \in$ states*(CESRM_I), *$s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of* CESRM_I *involving the transmission of p using the reliable multicast service. For any states u, u' of* CESRM_I

in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. For any $k \in \mathbb{N}^+$ and $h, h' \in H, h \neq h'$, suppose that $u[\text{CESRM-MEM}_h].status = \texttt{member}$, $u[\text{CESRM-MEM}_{h'}].status = \texttt{member}$, $id(p) \in u[\text{CESRM-REC}_{h'}].archived\text{-}pkts?$, $\alpha_{uu'}$ contains neither $\texttt{crash}_h$, $\texttt{rm-leave}_h$, $\texttt{crash}_{h'}$, nor $\texttt{rm-leave}_{h'}$ actions, h' receives a request for p from h at time $t' \in \mathbb{R}^{\geq 0}$ in $\alpha_{uu'}$, and $t' + (D_1 + D_2)\overline{d} + \overline{d} - \underline{d} + D_3\overline{d} < u'.now$. Then, the reply abstinence period of the reply of h' pertaining to this particular request of h for p expires no later than $t' + (D_1 + D_2)\overline{d} + \overline{d} - \underline{d} + D_3\overline{d}$ in α.

Proof: The proof is identical to that of Lemma 4.28. ❒

Lemma 6.25 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of CESRM_I in timely-aexecs(CESRM_I) that contains the transmission of p. Let (w, π, w'), for $w, w' \in states(\text{CESRM}_I)$, $s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of CESRM_I in α involving the transmission of p using the reliable multicast service. For any states u, u' of CESRM_I in α, such that $w' \leq_\alpha u \leq_\alpha u'$, let $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. Moreover, let $q, r \in H, q \neq r$ be any members of the reliable multicast group in u, such that $id(p) \in u[\text{CESRM-REC}_q].expected(s_p)$, $id(p) \notin u[\text{CESRM-REC}_q].scheduled\text{-}rqsts?$, and $id(p) \in u[\text{CESRM-REC}_r].delivered(s_p)$.*

For $k \in \mathbb{N}^+, k \geq k^*_{repl}$, suppose that $\alpha_{uu'}$ contains neither $\texttt{crash}_q$, $\texttt{rm-leave}_q$, $\texttt{crash}_r$, nor $\texttt{rm-leave}_r$ actions, q schedules k-th, $k+1$-st, and $k+2$nd round requests for the packet p in $\alpha_{uu'}$, q either sends or receives its k-th and $k+1$-st round requests for p at the points in time $t_{k+1}, t_{k+2} \in \mathbb{R}^{\geq 0}$ in $\alpha_{uu'}$, r receives the k-th and $k+1$-st round requests of q for p in $\alpha_{uu'}$, and r either sends or receives the replies pertaining to the k-th and $k+1$-st round requests of q for p in $\alpha_{uu'}$.

Then, the replies of r pertaining to the k-th and $k+1$-st round requests of q for p are distinct.

Proof: The proof is identical to that of Lemma 4.29. ❒

Lemma 6.26 *Let $k \in \mathbb{N}^+$, $p \in P_{\text{RM-CLIENT}}$, and α be any admissible timed execution of CESRM_I in timely-aexecs$_k(\text{CESRM}_I)$ that contains the transmission of p. Let (w, π, w'), for $w, w' \in states(\text{CESRM}_I)$, $s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of CESRM_I in α involving the transmission of p using the reliable multicast service. Suppose that the host $h \in H$ schedules a request for p following the transmission of p in α. Let $u \in states(\text{CESRM}_I)$ be the first state in α such that $id(p) \in u[\text{CESRM-REC}_h].scheduled\text{-}rqsts?(s_p)$, $u' \in states(\text{CESRM}_I)$ be any state in α such that $u.now + \texttt{REC-BOUND}(k^* + k) < u'.now$, and $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. Suppose that $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions, there exists a host $h' \in H, h' \neq h$, such that $id(p) \in u[\text{CESRM-REC}_{h'}].delivered(s_p)$, and $\alpha_{uu'}$ contains neither $\texttt{crash}_{h'}$, nor $\texttt{rm-leave}_{h'}$ actions. Then, it is the case that $id(p) \in u'[\text{CESRM-REC}_h].delivered(s_p)$.*

Proof: The proof is identical to that of Lemma 4.30. ❒

Lemma 6.27 *Let $p \in P_{\text{RM-CLIENT}}$ and α be any admissible timed execution of CESRM_I in timely-aexecs(CESRM_I) that contains the transmission of p. Let (w, π, w'), for $w, w' \in states(\text{CESRM}_I)$, $s_p = source(p)$, and $\pi = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of CESRM_I involving the transmission of p using the reliable multicast service. Let $h \in H$, u, u' be any states of CESRM_I in α, such that $w'.now + \overline{d} < u.now$ and $u \leq_\alpha u'$, and $\alpha_{uu'}$ be the timed execution fragment of α leading from u to u'. If $id(p) \in u'[\text{CESRM-REC}_h].expected(s_p)$, then $\alpha_{uu'}$ contains neither $\texttt{crash}_h$ nor $\texttt{rm-leave}_h$ actions.*

Proof: The proof is identical to that of Lemma 4.31. ❒

Lemma 6.28 *Let* $p \in P_{\text{RM-CLIENT}}$ *and* α *be any admissible timed execution of* CESRM_I *in timely-aexecs*(CESRM_I) *that contains the transmission of* p. *Let* (w, π, w'), *for* $w, w' \in$ *states*(CESRM_I), $s_p = source(p)$, *and* $\pi = \texttt{rm-send}_{s_p}(p)$, *be the discrete transition of* CESRM_I *in* α *involving the transmission of* p *using the reliable multicast service. Let* $h \in H$, u, u' *be any states of* CESRM_I *in* α, *such that* $w'.now + \overline{d} < u.now$ *and* $u \leq_\alpha u'$. *If* $id(p) \in u'[\text{CESRM-REC}_h].expected(s_p)$, *then it is the case that* $id(p) \in u[\text{CESRM-REC}_h].expected(s_p)$.

Proof: The proof is identical to that of Lemma 4.32. ❒

Lemma 6.29 *Let* $k \in \mathbb{N}^+$, $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$, $p \in P_{\text{RM-CLIENT}}$, *and* α *be any admissible timed execution of* CESRM_I *in* Δ_L*-src-recoverable-aexecs*$_k(\text{CESRM}_I)$ *that contains the transmission of* p. *Let* (w, π, w'), *for* $w, w' \in$ *states*(CESRM_I), $s_p = source(p)$, *and* $\pi = \texttt{rm-send}_{s_p}(p)$, *be the discrete transition of* CESRM_I *in* α *involving the transmission of* p *using the reliable multicast service. For any state* w'' *of* CESRM_I *in* α, *such that* $w'.now + \Delta_L < w''.now$, *let* $\alpha_{w'w''}$ *be the timed execution fragment of* α *leading from* w' *to* w''. *If* $h \in w''.intended(p)$, *then it is the case that* $h \in w''.completed(p)$.

Proof: The proof is identical to that of Lemma 4.33. ❒

Lemma 6.30 *Let* $k \in \mathbb{N}^+$, $\Delta_R = \texttt{REC-BOUND}(k^* + k)$, $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$, $p \in P_{\text{RM-CLIENT}}$, *and* α *be any admissible timed execution of* CESRM_I *in* Δ_R*-recoverable-aexecs*$_k(\text{CESRM}_I)$ *that contains the transmission of* p. *Let* (w, π, w'), *for* $w, w' \in$ *states*(CESRM_I), $s_p = source(p)$, *and* $\pi = \texttt{rm-send}_{s_p}(p)$, *be the discrete transition of* CESRM_I *in* α *involving the transmission of* p *using the reliable multicast service. For any state* w'' *of* CESRM_I *in* α, *such that* $w'.now + \Delta_L < w''.now$, *let* $\alpha_{w'w''}$ *be the timed execution fragment of* α *leading from* w' *to* w''. *If* $h \in w''.intended(p)$, *then it is the case that* $h \in w''.completed(p)$.

Proof: The proof is identical to that of Lemma 4.34. ❒

6.4.2 Static and Dynamic Timeliness Analysis

The static and dynamic timeliness guarantees for the SRM_I automaton presented in Sections 4.4.6 and 4.4.7 carry over to the CESRM_I automaton unchanged. In this section, we restate those guarantees for the CESRM_I automaton.

When hosts neither crash nor leave the reliable multicast group and the number of packet drops pertaining to the transmission and, potentially, the recovery of any packet is bounded, CESRM_I implements $\text{RM}_S(\Delta_L)$, for a particular $\Delta_L \in \mathbb{R}^{\geq 0}$. In particular, any timed trace of CESRM_I in the set *recoverable-attraces*$_k(\text{CESRM}_I)$, for some $k \in \mathbb{N}$, is also a timed trace of the specification automaton $\text{RM}_S(\Delta_L)$, for $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$. Thus, given Constraints 4.1, 4.2, 4.3, 4.4, 4.5, 4.6, 6.1, and 6.2 and assuming that the number of packet drops pertaining to the transmission and, potentially, the recovery of any packet is bounded by k, CESRM_I implements the timely reliable multicast service specification $\text{RM}_S(\Delta_L)$.

Theorem 6.31 *Let* $k \in \mathbb{N}^+$ *and* $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$. *Then, it is the case that recoverable-attraces*$_k(\text{CESRM}_I) \subseteq$ *attraces*$(\text{RM}_S(\Delta_L))$.

Proof: The proof is analogous to that of Theorem 4.35. ❒

In terms of the dynamic timeliness guarantees, the following lemma states that when sources remain members of the reliable multicast group for an amount of time $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$ past the transmission of any packet they send using the reliable multicast group, CESRM_I implements $\text{RM}_S(\Delta_L)$. In particular, any timed trace of CESRM_I in the set Δ_L-*recoverable-attraces*$_k(\text{CESRM}_I)$, for $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$ and some $k \in \mathbb{N}$, is also a timed trace of the specification automaton $\text{RM}_S(\Delta_L)$.

Theorem 6.32 *Let* $k \in \mathbb{N}^+$ *and* $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$. *Then, it is the case that* Δ_L-*src-recoverable-attraces*$_k(\text{CESRM}_I) \subseteq$ *attraces*$(\text{RM}_S(\Delta_L))$.

Proof: The proof is analogous to that of Theorem 4.35. ❒

We strengthen the above result by weakening our assumption that sources neither crashing nor leaving the reliable multicast group. In particular, we show that CESRM_I implements $\text{RM}_S(\Delta_L)$, for $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$, if whenever a host $h \in H$ detects the loss of any packet $p \in P_{\text{RM-CLIENT}}$, there exists a host $h' \in H, h' \neq h$ that has already delivered p and remains a member of the reliable multicast group for at least $\Delta_R = \texttt{REC-BOUND}(k^* + k)$ time units.

Theorem 6.33 *Let* $k \in \mathbb{N}^+$, $\Delta_R = \texttt{REC-BOUND}(k^* + k)$, *and* $\Delta_L = \texttt{DET-BOUND} + \texttt{REC-BOUND}(k^* + k)$. *Then, it is the case that* Δ_R-*recoverable-attraces*$_k(\text{CESRM}_I) \subseteq$ *attraces*$(\text{RM}_S(\Delta_L))$.

Proof: The proof is analogous to that of Theorem 4.35. ❒

6.4.3 Expedited Versus Non-Expedited Recovery Timeliness Analysis

In this section, we compare the worst-case recovery latency incurred by successful expedited and non-expedited first-round recoveries of CESRM. In particular, we show that successful expedited recoveries complete within at most $\texttt{DET-BOUND} + +\texttt{RQST-DELAY} + 2\overline{d}$ time units, where $\overline{d}$ is an upper bound on the inter-host transmission latency. Furthermore, we show that successful non-expedited first-round recoveries complete within at most $\texttt{DET-BOUND} + (C_1 + C_2)\overline{d} + \overline{d} + (D_1 + D_2)\overline{d} + \overline{d}$ time units. These bounds reveal that, for typical SRM request and reply scheduling parameter values [13], the worst-case recovery latency of packets recovered by expedited rather than first-round recoveries in CESRM is reduced by roughly $3\overline{RTT}$, where $\overline{RTT} = 2\ \overline{d}$ is an upper bound on the inter-host round-trip-time.

Let α be any admissible timed execution of CESRM_I, $p \in P_{\text{RM-CLIENT}}$ be any packet transmitted in α, and (u, π, u'), for $u, u' \in$ *states*(CESRM_I) and $\pi \in$ *acts*(CESRM_I), be any discrete transition of CESRM_I in α, such that $\pi = \texttt{process-pkt}_h(pkt)$, for $h \in H$ and $pkt \in P_{\text{CESRM}}$, such that $type(pkt) = \texttt{EXP-REPL}$ and $id(pkt) = id(p)$. If the discrete transition (u, π, u') in α culminates the recovery of the packet p by h, *i.e.*, $id(p) \in u[\text{CESRM-REC}_h]$.*scheduled-rqsts?* and $id(p) \in u'[\text{CESRM-REC}_h]$.*archived-pkts?*, then we say that the discrete transition (u, π, u') involves an expeditious recovery of p by h in α.

The following theorem states that, in any admissible timed execution α of CESRM_I in the set *timely-aexecs*(CESRM_I), any packet that is expeditiously recovered is done so within at most $\texttt{DET-BOUND} + \texttt{RQST-DELAY} + 2\overline{d}$ time units from the time the particular packet is transmitted.

Theorem 6.34 *Let* α *be any admissible timed execution of* CESRM_I *in timely-aexecs*(CESRM_I) *and* p *be any packet transmitted in* α *that is expeditiously recovered by a host* $h \in H$ *in* α.

Let (w, π_w, w'), for $w, w' \in states(\mathrm{CESRM}_I)$, $s_p = source(p)$, and $\pi_w = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of CESRM_I in α involving the transmission of p in α and (u, π_u, u'), for $u, u' \in states(\mathrm{CESRM}_I)$ and $\pi_u = \texttt{process-pkt}_h(pkt)$, such that $type(pkt) = \texttt{EXP-REPL}$ and $id(pkt) = id(p)$, be the discrete transition of CESRM_I in α involving the expeditious recovery of p by h in α. Then, it is the case that $u.now \leq w.now + \texttt{DET-BOUND} + \texttt{RQST-DELAY} + 2\overline{d}$.

Proof: From the precondition of the $\texttt{process-pkt}_h(pkt)$ action, it follows that $u[\text{CESRM-IPBUFF}_h].recv\text{-}buff \neq \emptyset$. Since time is not allowed to elapse in CESRM-IPBUFF_h while $\text{CESRM-IPBUFF}_h.recv\text{-}buff$ is non-empty and $\texttt{EXP-REPL}$ packets are sent using the IP multicast service, it follows that the $\texttt{process-pkt}_h(pkt)$ action immediately succeeds a $\texttt{mrecv}_h(ip\text{-}mpkt)$ action in α, for $ip\text{-}mpkt \in P_{\text{IPMCAST-CLIENT}}$, such that $strip(ip\text{-}mpkt) = pkt$. Let $(u_1, \texttt{mrecv}_h(ip\text{-}mpkt), u_1')$ be the discrete transition in α involving the occurrence of this $\texttt{mrecv}_h(ip\text{-}mpkt)$ action.

Lemma 6.1 implies that this $\texttt{mrecv}_h(ip\text{-}mpkt)$ action is preceded in α by a $\texttt{msend}_{h'}(ip\text{-}mpkt)$ action, for some $h' \in H, h' \neq h$. Let $(u_2, \texttt{msend}_{h'}(ip\text{-}mpkt), u_2')$ be the discrete transition in α involving the occurrence of this $\texttt{msend}_{h'}(ip\text{-}mpkt)$ action. Constraint 4.1 implies that the time elapsing between these $\texttt{msend}_{h'}(ip\text{-}mpkt)$ and $\texttt{mrecv}_h(ip\text{-}mpkt)$ actions is at most $\overline{d}$ time units.

From the precondition of the $\texttt{msend}_{h'}(ip\text{-}mpkt)$ action, it follows that $ip\text{-}mpkt \in u_2[\text{CESRM-IPBUFF}_{h'}].msend\text{-}buff$. Since time is not allowed to elapse in $\text{CESRM-IPBUFF}_{h'}$ while $\text{CESRM-IPBUFF}_{h'}.msend\text{-}buff$ is non-empty, it follows that the $\texttt{msend}_{h'}(ip\text{-}mpkt)$ action immediately succeeds a $\texttt{rec-msend}_{h'}(pkt)$ action. Let $(u_3, \texttt{rec-msend}_{h'}(pkt), u_3')$ be the discrete transition in α involving the occurrence of this $\texttt{rec-msend}_{h'}(pkt)$ action.

From the precondition of the $\texttt{rec-msend}_{h'}(pkt)$ action, it follows that $pkt \in u_3[\text{CESRM-REC}_{h'}].msend\text{-}buff$. Since $\texttt{EXP-REPL}$ packets may only be added to $\text{CESRM-REC}_{h'}.msend\text{-}buff$ through the occurrence of the $\texttt{process-pkt}_{h'}(pkt')$ action, where $type(pkt') = \texttt{EXP-RQST}$ and $id(pkt') = id(pkt)$, it follows that the $\texttt{rec-msend}_{h'}(pkt)$ immediately succeeds a $\texttt{process-pkt}_{h'}(pkt')$ action in α. Let $(u_4, \texttt{process-pkt}_{h'}(pkt'), u_4')$ be the discrete transition in α involving the occurrence of this $\texttt{process-pkt}_{h'}(pkt')$ action.

From the precondition of the $\texttt{process-pkt}_{h'}(pkt')$ action, it follows that $pkt' \in u_4[\text{CESRM-IPBUFF}_{h'}].recv\text{-}buff$. Since time is not allowed to elapse in $\text{CESRM-IPBUFF}_{h'}$ while $\text{CESRM-IPBUFF}_{h'}.recv\text{-}buff$ is non-empty and $\texttt{EXP-RQST}$ packets are sent using the IP unicast service, it follows that the $\texttt{process-pkt}_{h'}(pkt')$ action immediately succeeds a $\texttt{urecv}_{h'}(ip\text{-}upkt)$ action in α, for $ip\text{-}upkt \in P_{\text{IPUCAST-CLIENT}}$, such that $strip(ip\text{-}upkt) = pkt'$. Let $(u_5, \texttt{urecv}_{h'}(ip\text{-}upkt), u_5')$ be the discrete transition in α involving the occurrence of this $\texttt{urecv}_{h'}(ip\text{-}upkt)$ action.

Lemma 6.1 implies that this $\texttt{urecv}_{h'}(ip\text{-}upkt)$ action is preceded in α by a $\texttt{usend}_{h''}(ip\text{-}upkt)$ action, for some $h'' \in H, h'' \neq h'$. Let $(u_6, \texttt{usend}_{h''}(ip\text{-}upkt), u_6')$ be the discrete transition in α involving the occurrence of this $\texttt{usend}_{h''}(ip\text{-}upkt)$ action. Constraint 6.1 implies that the time elapsing between these $\texttt{usend}_{h''}(ip\text{-}upkt)$ and $\texttt{urecv}_{h'}(ip\text{-}upkt)$ actions is at most $\overline{d}$ time units.

From the precondition of the $\texttt{usend}_{h''}(ip\text{-}upkt)$ action, it follows that $ip\text{-}upkt \in u_6[\text{CESRM-IPBUFF}_{h''}].usend\text{-}buff$. Since time is not allowed to elapse in $\text{CESRM-IPBUFF}_{h'}$ while $\text{CESRM-IPBUFF}_{h''}.usend\text{-}buff$ is non-empty, it follows that the $\texttt{usend}_{h''}(ip\text{-}upkt)$ action immediately succeeds a $\texttt{rec-usend}_{h''}(h', pkt')$ action. Let $(u_7, \texttt{rec-usend}_{h''}(h', pkt'), u_7')$ be the discrete transition in α involving the occurrence of this $\texttt{rec-usend}_{h''}(h', pkt')$ action.

From the precondition of the $\texttt{rec-usend}_{h''}(h', pkt')$ action, it follows that $pkt' \in u_7[\text{CESRM-REC}_{h''}].usend\text{-}buff(h')$. Since time is not allowed to elapse in $\text{CESRM-REC}_{h''}$ while $\text{CESRM-REC}_{h''}.usend\text{-}buff(h')$ is non-empty and the only action that may add an $\texttt{EXP-RQST}$ packet to $\text{CESRM-REC}_{h''}.usend\text{-}buff(h')$ is the $\texttt{send-exp-rqst}_{h''}(s, i)$ action, for $\langle s, i\rangle = id(pkt')$, it follows that the $\texttt{rec-usend}_{h''}(h', pkt')$ action immediately succeeds a $\texttt{send-exp-rqst}_{h''}(s,i)$ action, for $\langle s,i\rangle = id(pkt')$, in α. Let $(u_8, \texttt{send-exp-rqst}_{h''}(s,i), u_8')$ be the discrete transition in α involving the occurrence of this $\texttt{send-exp-rqst}_{h''}(s,i)$ action.

From the precondition of the $\texttt{send-exp-rqst}_{h''}(s,i)$ action, it follows that $\langle s,i\rangle \in u_8[\text{CESRM-}\textsc{rec}_{h''}].\textit{expedited-rqsts?}$. The only actions that may add an element pertaining to the packet $\langle s,i\rangle$ to the set $\text{CESRM-}\textsc{rec}_{h''}.\textit{expedited-rqsts}$ are the actions $\texttt{schdl-rqst}_{h''}(s,i)$ and $\texttt{process-pkt}_{h''}(pkt'')$, for $type(pkt'') = \texttt{RQST}$ and $id(pkt'') = \langle s,i\rangle$. Let (u_9, π, u_9') be the discrete transition in α involving the occurrence of either such action. Either of these actions schedule the expedited request for $\langle s,i\rangle$ for a point in time that is $\texttt{RQST-DELAY}$ time units in the future. Since time is not allowed to elapse past the time such an expedited request is scheduled for transmission, the discrete transition $(u_8, \texttt{send-exp-rqst}_{h''}(s,i), u_8')$ occurs exactly $\texttt{RQST-DELAY}$ time units after the occurrence of the discrete transition (u_9, π, u_9').

In the case of a $\texttt{schdl-rqst}_{h''}(s,i)$ action, the precondition of the $\texttt{schdl-rqst}_{h''}(s,i)$ action implies that $\langle s,i\rangle \in u_9[\text{CESRM-}\textsc{rec}_{h''}].\textit{to-be-requested?}$. Invariant 6.21 implies that $\langle s,i\rangle \notin u_9[\text{CESRM-}\textsc{rec}_{h''}].\textit{scheduled-rqsts?}$. In the case of a $\texttt{process-pkt}_{h''}(pkt'')$ action, for $type(pkt'') = \texttt{RQST}$ and $id(pkt'') = \langle s,i\rangle$, the effects of $\texttt{process-pkt}_{h''}(pkt'')$ imply that $\langle s,i\rangle \notin u_9[\text{CESRM-}\textsc{rec}_{h''}].\textit{scheduled-rqsts?}$. Moreover, the effects of either $\texttt{schdl-rqst}_{h''}(s,i)$ or $\texttt{process-pkt}_{h''}(pkt'')$ imply that $\langle s,i\rangle \in u_9'[\text{CESRM-}\textsc{rec}_{h''}].\textit{scheduled-rqsts?}$.

In either case, it follows that $\langle s,i\rangle \notin u_9[\text{CESRM-}\textsc{rec}_{h''}].\textit{scheduled-rqsts?}$ and $\langle s,i\rangle \in u_9'[\text{CESRM-}\textsc{rec}_{h''}].\textit{scheduled-rqsts?}$; that is, h'' initiates the recovery of $\langle s,i\rangle$ through the discrete transition (u_9, π, u_9'). Thus, Constraint 4.4 implies that $u_9.now = u_9'.now \le w.now + \texttt{DET-BOUND} = w'.now + \texttt{DET-BOUND}$.

It follows that the discrete transition $(u, \texttt{process-pkt}_h(pkt), u')$ occurs at most $\texttt{DET-BOUND} + \texttt{RQST-DELAY} + 2\overline{d}$ time units after the occurrence of the discrete transition $(w, \texttt{rm-send}_{s_p}(p), w')$; that is, $u.now \le w.now + \texttt{DET-BOUND} + \texttt{RQST-DELAY} + 2\overline{d}$. ❒

The following theorem states that, in any admissible timed execution α of CESRM_I in the set $\textit{timely-aexecs}(\text{CESRM}_I)$, any packet that is recovered by a particular host by a reply to a 1st-round request of the same host is done so within at most $\texttt{DET-BOUND} + (C_1 + C_2)\overline{d} + \overline{d} + (D_1 + D_2)\overline{d} + \overline{d}$ time units from the time the particular packet is transmitted.

Theorem 6.35 *Let α be any admissible timed execution of CESRM_I in $\textit{timely-aexecs}(\text{CESRM}_I)$ and p be any packet transmitted in α that is expeditiously recovered by a host $h \in H$ in α. Let (w, π_w, w'), for $w, w' \in \textit{states}(\text{CESRM}_I)$, $s_p = \textit{source}(p)$, and $\pi_w = \texttt{rm-send}_{s_p}(p)$, be the discrete transition of CESRM_I in α involving the transmission of p in α and (u, π_u, u'), for $u, u' \in \textit{states}(\text{CESRM}_I)$ and $\pi_u = \texttt{process-pkt}_h(pkt)$, such that $type(pkt) = \texttt{REPL}$ and $id(pkt) = id(p)$, be the discrete transition of CESRM_I in α involving the recovery of p by h in α. Letting $r = sender(pkt)$, if the packet pkt is a reply of r to the 1st-round request of h for p, then it is the case that $u.now \le w.now + \texttt{DET-BOUND} + (C_1 + C_2)\overline{d} + \overline{d} + (D_1 + D_2)\overline{d} + \overline{d}$.*

Proof: Suppose that h recovers p through the reception of a reply of r to the 1st-round request of h for p. In this scenario, prior to the occurrence of the discrete transition (u, π_u, u') in α, the host h initiates the recovery of p through the occurrence of either a $\texttt{schdl-rqst}_h(s_p, i_p)$ action, for $\langle s_p, i_p\rangle = id(p)$, or a $\texttt{process-pkt}_h(pkt')$ action, for $type(pkt') = \texttt{RQST}$ and $id(pkt') = id(p)$. In the former case, $\text{CESRM-}\textsc{rec}_h$ initiates the recovery of p by scheduling a 1st-round request for p, while in the latter, $\text{CESRM-}\textsc{rec}_h$ initiates the recovery of p by scheduling a 2-nd round request for p. Constraint 4.4 implies that h initiates the recovery of p within at most $\texttt{DET-BOUND}$ time units past the transmission time of p. Moreover, Constraint 4.1 and Lemma 6.18 imply that r receives the

1-st round request of h for p no later than $(C_1 + C_2)\overline{d} + \overline{d}$ time units past the particular occurrence of either the $\texttt{schdl-rqst}_h(s_p, i_p)$ or the $\texttt{process-pkt}_h(pkt')$ action.

The 1-st round request of h for p is received by r while either i) a reply for p is already scheduled, ii) a reply for p is already pending, or iii) a reply for p is neither scheduled, nor pending. In either of these cases, r transmits its reply to the 1-st round request of h for p no later than $(D_1 + D_2)\overline{d}$ time units past the point in time at which it receives the 1-st round request of h for p. The reply of r to the 1-st round request of h for p is thus received by h no later than $\overline{d}$ thereafter.

Thus, it follows that h receives r's reply to its 1-st round request for p at most $\texttt{DET-BOUND} + (C_1 + C_2)\overline{d} + \overline{d} + (D_1 + D_2)\overline{d} + \overline{d}$ time units past the transmission of p. ❐

Floyd *et al.* [13] analyzed the performance of SRM under a variety of request and reply timing parameter settings. The optimal such settings depend on the topology, the session's density, and the loss characteristics of the links comprising the underlying IP multicast distribution tree. Given the typical parameter values used by Floyd *et al.* [13] of $C_1 = C_2 = 2$ and $D_1 = D_2 = 1$, Theorem 6.35 implies that the worst-case 1-st round recovery latency is $\texttt{DET-BOUND} + 8\overline{d}$, or $\texttt{DET-BOUND} + 4\overline{RTT}$, where $\overline{RTT} = 2\overline{d}$ denotes the worst-case round-trip-time between members of the reliable multicast group. In contrast, Theorem 6.34 implies that the worst-case expedited recovery latency is $\texttt{DET-BOUND} + \texttt{RQST-DELAY} + 2\overline{d}$, or $\texttt{DET-BOUND} + \texttt{RQST-DELAY} + \overline{RTT}$.

Recall that $\texttt{RQST-DELAY}$ involves the delay used to avoid prematurely transmitting expedited requests for packets that are temporarily considered missing due to packet reordering. Presuming that this delay is insignificant compared to the worst-case round trip delay, *i.e.*, $\texttt{RQST-DELAY} \ll \overline{RTT}$, the worst-case recovery latency of packets recovered by expedited rather than 1-st round recoveries is reduced by roughly $3\overline{RTT}$.

The question that remains is how often expedited recoveries are successful. CESRM operates in the spirit of our caching-based loss location estimation scheme introduced in Chapter 5. In CESRM, receivers cache the optimal requestor/replier pair engaged in the recovery of their most recent losses and attempt to recover losses using the optimal requestor/replier pair of the most recent loss whose optimal requestor/replier pair has been identified. In effect, CESRM identifies loss locations by their optimal requestor/replier pairs.

In our analysis of the IP multicast traces of Yajnik *et al.* [41] in Chapter 5, we estimated the percentage of losses that may be successfully recovered using a caching-based loss recovery scheme. This estimate included the consistent accurate estimates, the consistent high estimates, and the average percentage of estimates comprising inconsistent estimates. By presuming that it is highly likely that distinct losses on the same link will give rise to the same optimal requestor/replier pairs, we claim that this estimate is a rough indication of the percentage of expedited recoveries initiated by CESRM that are successful.

Thus, Figure 5.17 indicates that, for all of the IP multicast traces of Yajnik *et al.* [41], the percentage of losses that are expeditiously recoverable by CESRM should exceed 65%. Thus, presuming that the loss of recovery packets is infrequent, CESRM may afford a significant reduction is recovery latency for a large percentage of the losses.

6.5 CESRM Trace-Driven Simulations

In this section, we evaluate the performance of CESRM and compare it to that of SRM using trace-driven simulations. In these simulations, we reenact, as faithfully as possible, the 14 IP multicast transmissions that result in the traces collected by Yajnik *et al.* [41]. Thus, our simulations exhibit the packet loss locality exhibited by the actual IP multicast transmissions. We repeat our simulations using either SRM or CESRM as the packet loss recovery scheme and observe the recovery latency and overhead of each protocol. Our simulations show that, for the particular IP multicast transmissions, CESRM reduces the average recovery time of SRM by an average of 50%. Furthermore, CESRM sends fewer packet retransmissions than SRM — between 40% and 75% of the number of retransmissions sent by SRM. Finally, CESRM sends roughly as many control packets as SRM, but a large percentage of these are unicast whereas all of SRM's control packets are multicast. We conclude that, CESRM's overhead is significantly smaller than that of SRM.

We begin this section by describing the setup of our simulations. We then present the simulation results. We conclude by summarizing the results of our simulation-based evaluation of CESRM.

6.5.1 Simulation Setup

Following the presentation approach of Chapter 5, we collectively describe the setup of our simulations by describing the setup for simulating a single generic IP multicast transmission. This generic simulation is intended to correspond to the simulation of any single IP multicast transmission of Yajnik *et al.* [41]. From Chapter 5, recall that $k \in \mathbb{N}$ denotes the number of packets transmitted during the IP multicast transmission, R denotes the finite set of receivers of the IP multicast transmission, $I = \{1, \ldots, k\}$, and $loss : R \rightarrow (I \rightarrow \{0, 1\})$ is a mapping that represents per-receiver binary sequences, each of which indicates which of the packets the respective receiver failed to receive. Moreover, recall that we represent the IP multicast tree by a tuple $T = \langle N, s, L \rangle$ comprised of a set of nodes N, a root node $s \in N$, and a set of directed edges $L \subseteq N \times N$ — the elements of T satisfy several constraints, which are presented in Chapter 5. The packet transmission period is denoted by $\Delta T \in \mathbb{R}^{\geq 0}$. We also presume that the concrete link trace representation $c\text{-}link : R \rightarrow (I \rightarrow \text{C-LINK} \cup \perp)$, for $\text{C-LINK} = L$, accurately estimates the set of links responsible for each of the losses suffered during the particular IP multicast transmission.

Our generic simulation involves setting up the IP multicast tree T and disseminating k packets from the root of the tree, which corresponds to the IP multicast transmission source, to the tree's leaf nodes, which correspond to the receivers of the IP multicast transmission. Recall that the IP multicast tree is presumed to remain fixed throughout the duration of the IP multicast transmission. Since the IP multicast trace information of Yajnik *et al.* [41] contains no link delay and bandwidth info, we arbitrarily choose the delay and the bandwidth of each link in T to be $20\,ms$ and and $1.5\,Mbps$, respectively. Since the depth of the IP multicast tree involved in any of the IP multicast traces of Yajnik *et al.* [41] ranges from 3 to 7, the RTTs between the source and receivers in any trace ranges from $120\,ms$ to $280\,ms$.

We also presume that payload carrying packets, *i.e.*, original packets and retransmissions, are $1\,KB$ in size and control packets, *e.g.*, packet retransmission requests, are $0\,KB$. Since the IP multicast transmission period of any of the IP multicast transmission traces of Yajnik *et al.* [41] is either $40\,ms$ or $80\,ms$, the bandwidth required for the original transmissions is either $200\,Kbps$ or $400\,Kbps$. Thus, our choice of $1.5\,Mbps$ for the link bandwidth is sufficient to guarantee that no packets are dropped due to congestion. The simulation is carried out with the typical SRM scheduling parameter settings $C_1, C_2 = 2$, $C_3 = 1.5$, $D_1, D_2 = 1$, and $D_3 = 1.5$. Session packets are transmitted with a period of $1\,s$.

So as to focus our attention on the performance of CESRM packet loss recovery scheme, rather than that of the inter-host distance estimation scheme through session packet exchange, we presume that the session packet exchange is lossless. Since none of the session packets are dropped throughout our simulation, the inter-host distances are accurately and promptly calculated. Moreover, we sufficiently delay the beginning IP multicast transmission so that, prior to the beginning of the IP multicast transmission, receivers have a chance to exchange session messages and, thus, estimate their distance to each other.

We inject losses in the simulated IP multicast transmission according to the concrete link trace representation *c-link*. Recall that *c-link* estimates the links responsible for each of the losses suffered during the actual IP multicast transmission that resulted in the respective trace of Yajnik *et al.* [41]. Thus, by injecting losses in this fashion, we capture and reproduce the locality present in the actual IP multicast transmission.

So as to compare the performance of CESRM to that of SRM, we repeat the simulation twice; one simulation employs CESRM as the packet loss recovery scheme and the other employs SRM. We first conduct these simulations under the assumption that the packet loss recovery is lossless; that is, that none of the recovery packets (control packets and retransmissions) are dropped. This is precisely the assumption under which Floyd *et al.* [13] conducted the performance analysis of SRM. In order to obtain a more realistic evaluation of CESRM, we repeat the simulations while introducing losses to the packet loss recovery. So as to abide by the loss characteristics of the links of the IP multicast tree throughout which the IP multicast packets are disseminated, recovery packets are dropped according to the link loss probability estimates calculated in Chapter 5 for each of the links of the IP multicast tree.

6.5.2 Lossless Recovery Results

In this section, we assume that the packet loss recovery is lossless; that is, none of the recovery packets are dropped.

Figure 6.16 presents the per-receiver average normalized recovery times achieved by SRM and CESRM for 6 out of the 14 traces of Yajnik *et al.* [41] — the average normalized recovery times for the simulations corresponding to the remaining traces are similar. The recovery time of each receiver is normalized by the receiver's RTT distance estimate to the source of the IP multicast transmission and is, thus, quoted in units of RTT. From Figure 6.16, we can see that the caching-based expedited recovery scheme employed by CESRM substantially reduces the average normalized recovery time.

Figure 6.17 depicts the percentage by which the per-receiver average normalized recovery times are reduced using CESRM as opposed to SRM. Clearly, the use of expedited recoveries has a substantial effect on the per-receiver average normalized recovery times. For most of the receivers, the average normalized recovery times for CESRM are between 40% and 70% less than those of SRM.

Figure 6.17 depicts the difference in the average normalized recovery times between expedited and non-expedited recoveries of CESRM. For the scheduling parameters used in our simulations, Theorems 6.34 and 6.35 imply that the difference between the worst-case recovery latency between expedited and non-expedited successful recoveries is $3\overline{RTT}$. Figure 6.17 reveals that, in the particular simulations, the difference in the average normalized recovery latency between expedited and non-expedited successful recoveries ranges from 1 to 2 RTTs.

Figure 6.19 depicts the number of request packets sent by each of the receivers for 6 out of the 14 traces of Yajnik *et al.* [41] for either the SRM or the CESRM protocols — the number of request packets sent by each receiver for the simulations driven by the remaining traces are similar. The bars corresponding to the number of request packets sent by each receiver in the case of the CESRM protocol are split in two components. One component corresponds to the number of requests that are multicast as part of the regular recovery process of CESRM, which mimics that of SRM. The other component corresponds to the number of requests unicast as part of the expedited recovery process carried out by CESRM. The source of the IP multicast transmission corresponds to receiver 0.

Figure 6.16 Per-Receiver Average Normalized Recovery Times; Lossless Recovery

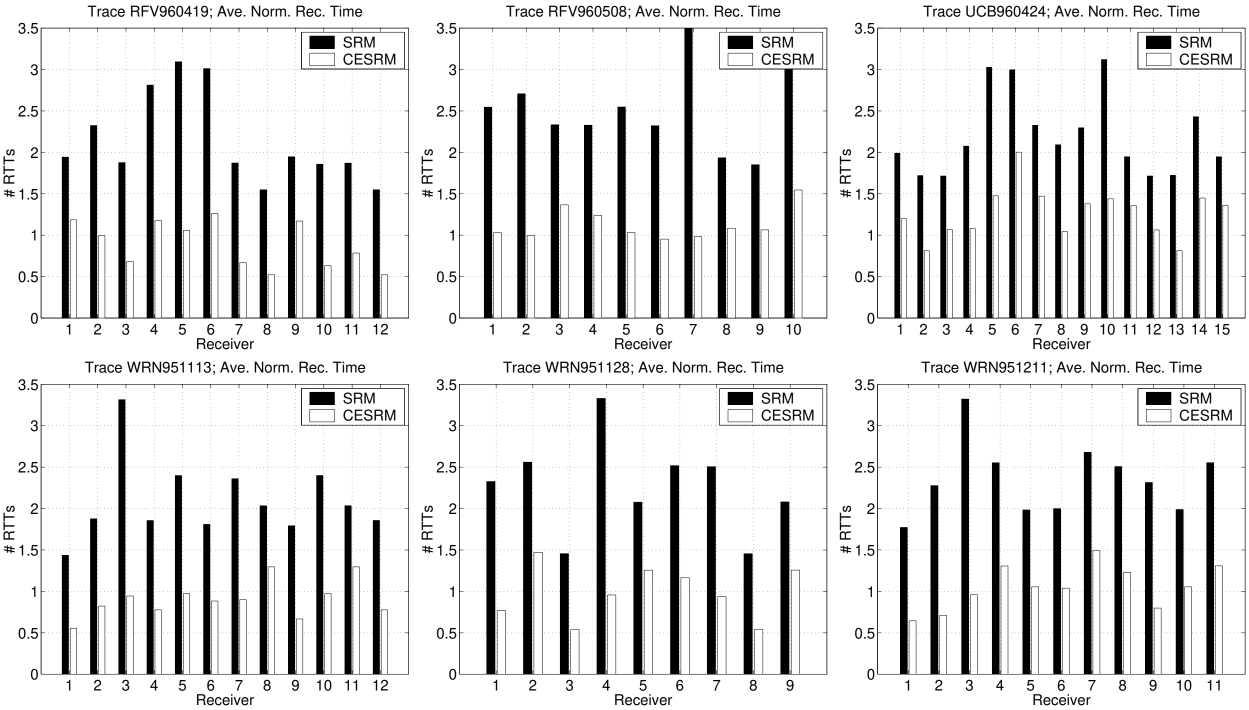


Figure 6.17 Percent Reduction in Per-Receiver Average Normalized Recovery Times; Lossless Recovery

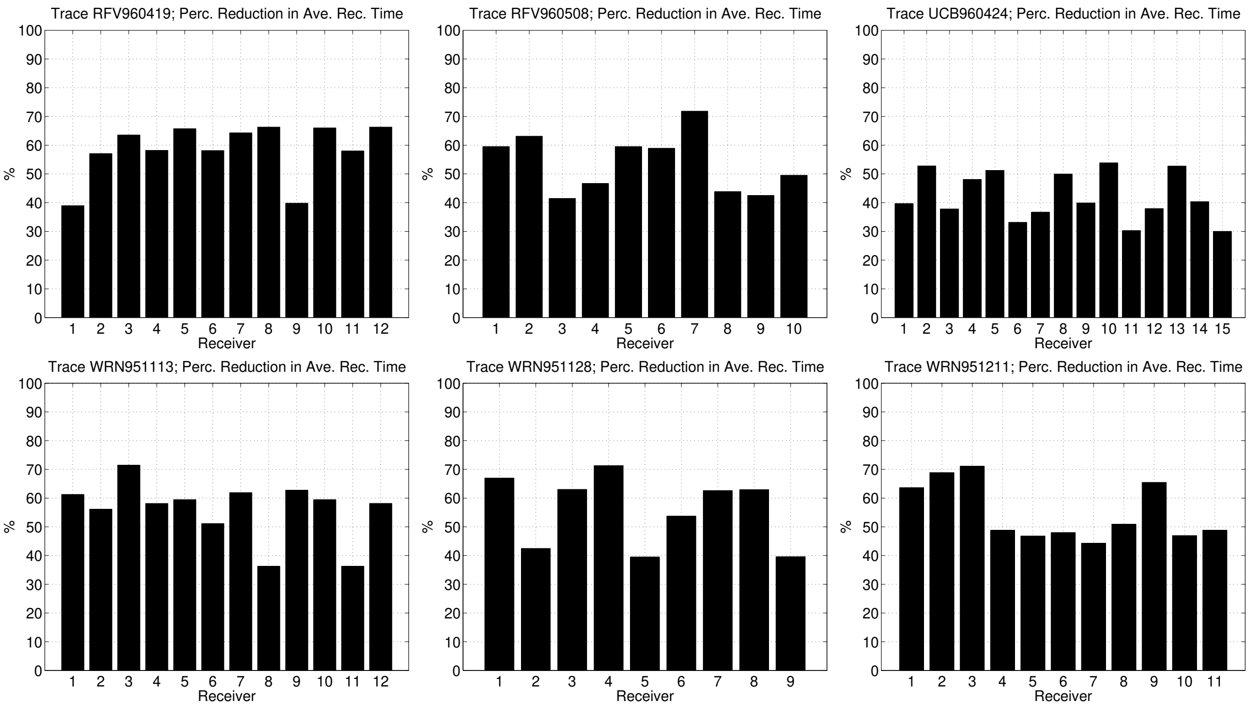


Figure 6.18 Difference in Average Normalized Recovery Times Between Expedited and Non-Expedited Recoveries of CESRM

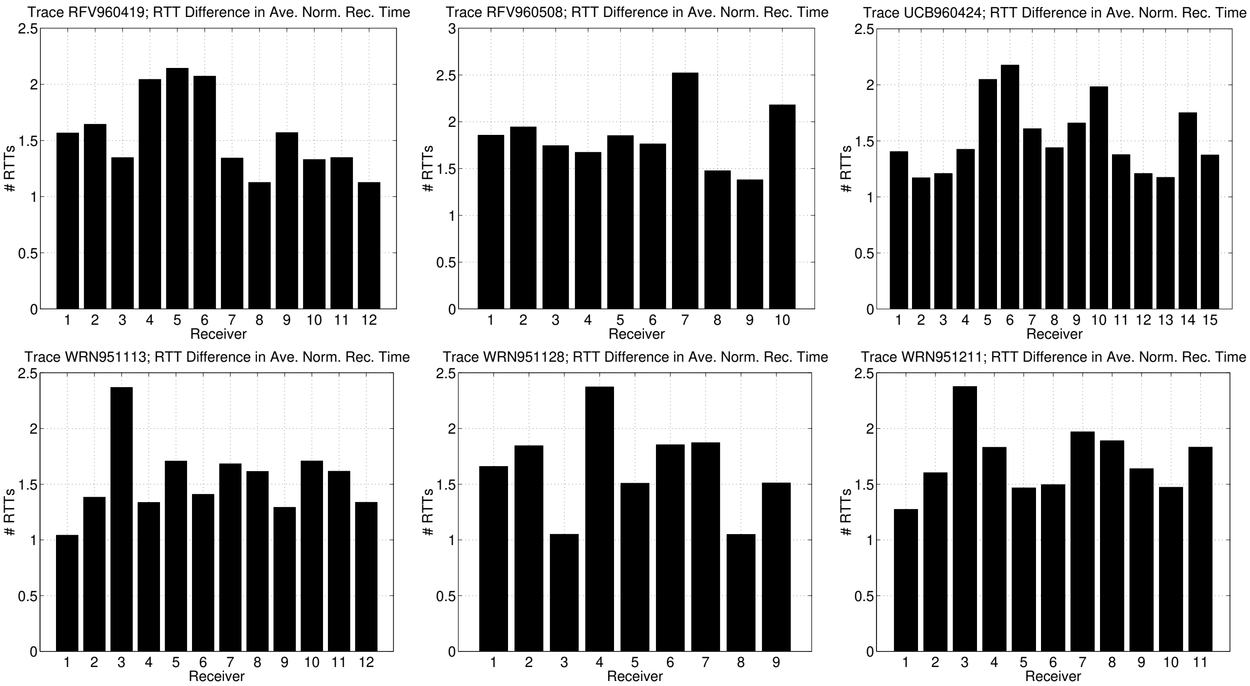


Figure 6.19 reveals that for most receivers in each of the simulations, the number of requests sent by CESRM are most often less than those sent by SRM. For some of the receivers the number of requests sent by CESRM exceeds that sent by SRM. Notably, however, a large portion of the number of requests sent by CESRM are unicast from particular requestors to particular repliers, rather than multicast to the entire group. Since unicast transmissions are substantially less costly than multicast transmissions, the overhead in terms of the number of requests incurred by CESRM is less than that incurred by SRM.

Figure 6.20 depicts the number of reply packets sent by each of the receivers for 6 out of the 14 traces of Yajnik *et al.* [41] for either the SRM or the CESRM protocols — the number of reply packets sent by each receiver for the simulations driven by the remaining traces are similar. The bars corresponding to the number of reply packets sent by each receiver in the case of the CESRM protocol are split in two components. One component corresponds to the number of replies that are multicast as part of the regular recovery process of CESRM, which mimics that of SRM. The other component corresponds to the number of expedited replies multicast as part of the expedited recovery process carried out by CESRM. Again, the source of the IP multicast transmission corresponds to receiver 0.

Figure 6.19 reveals that for most receivers in each of the simulations, the number of replies sent by CESRM are substantially less than those sent by SRM. The overhead incurred by replies is very important since reply packets, *i.e.*, packet retransmissions, carry the data. They are thus, not only multicast to the entire group, but their transmission is also substantially more costly than that of control packets, *e.g.*, requests, which do not carry data.

Figure 6.21 depicts the number of update packets sent by each of the receivers for 6 out of the 14 traces of Yajnik *et al.* [41] in the case of the CESRM protocol — the number of update packets sent by each receiver for the simulations driven by the remaining traces are similar. Once again, the source of the IP multicast transmission corresponds to receiver 0.

Comparing Figures 6.19, 6.20, and 6.21, it is clear that the number of update packets are at least

Figure 6.19 Number of Request Packets for SRM and CESRM

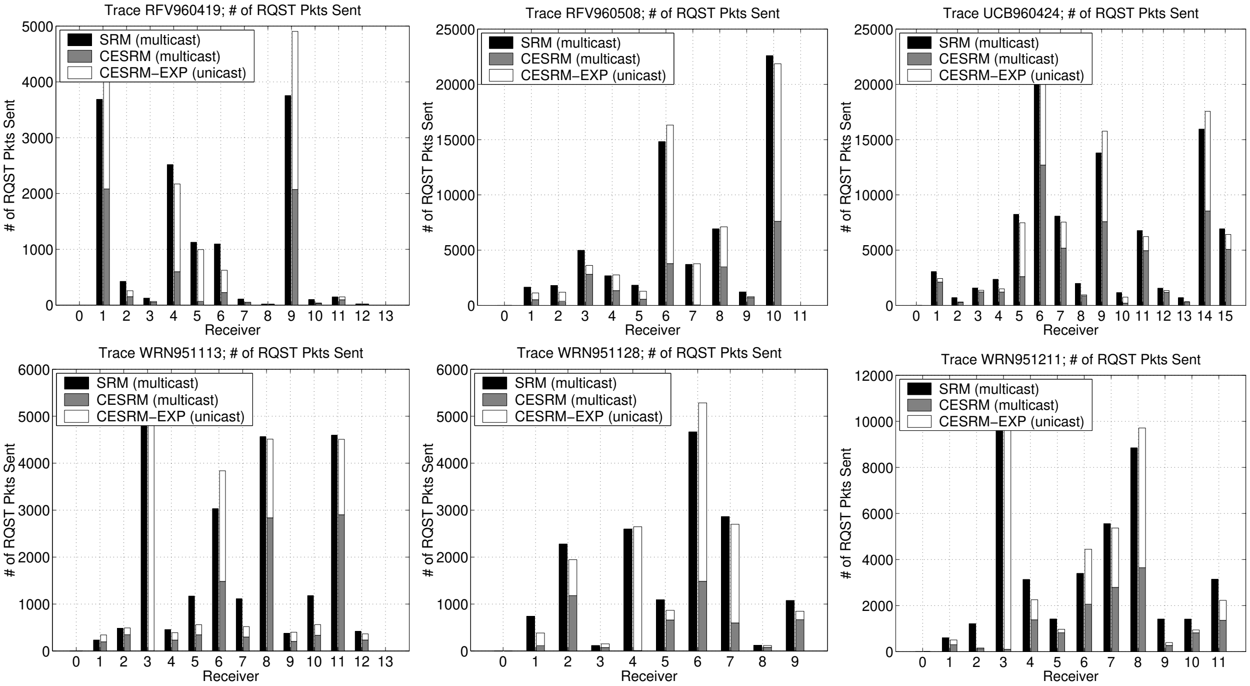


Figure 6.20 Number of Reply Packets for SRM and CESRM

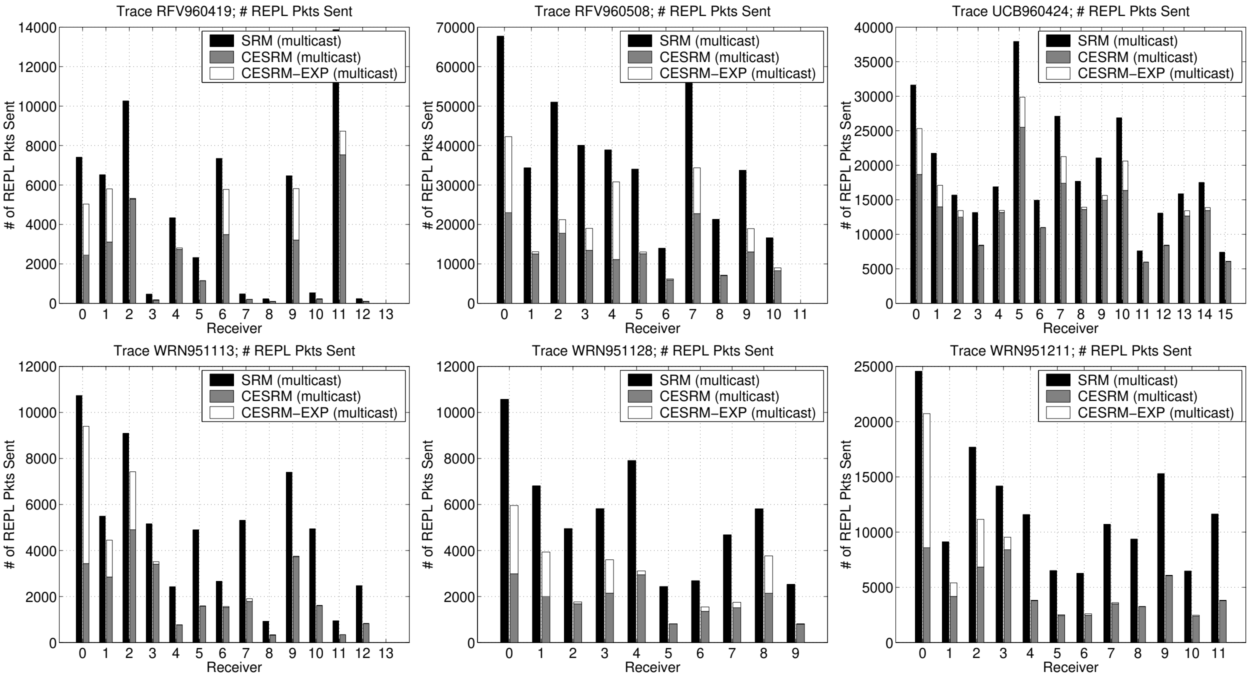
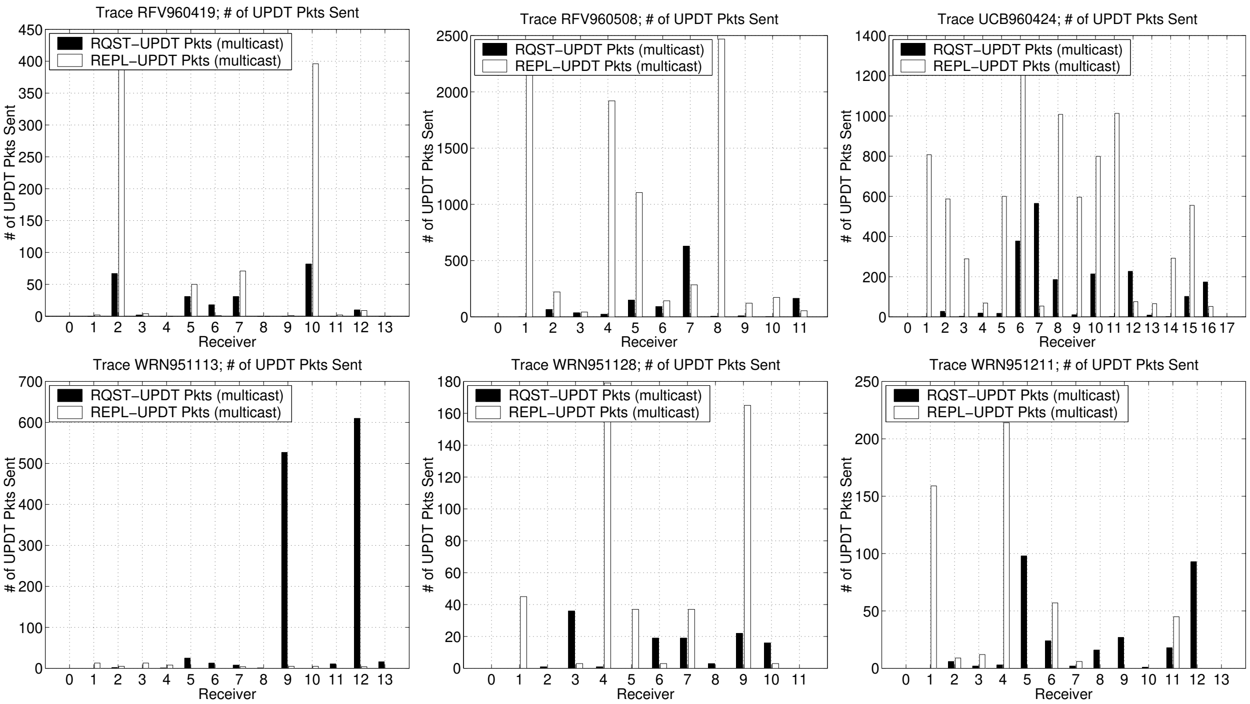


Figure 6.21 Number of Update Packets for CESRM



an order of magnitude fewer packets that either request or reply packets. As compared to requests and replies, they thus introduce a substantially smaller overhead.

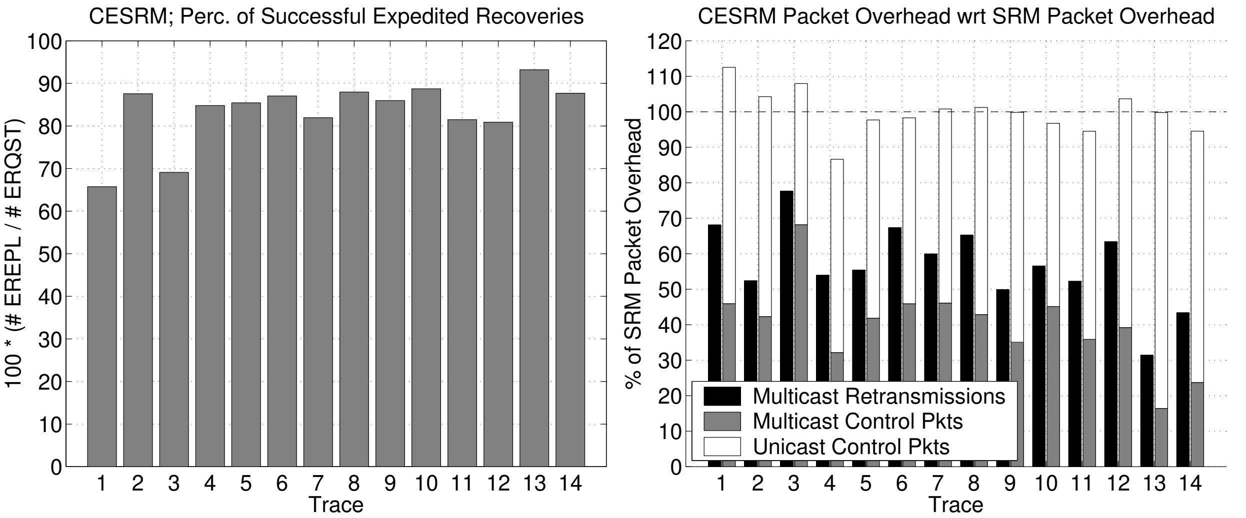
Figure 6.22 includes two plots pertaining to the performance of CESRM. The first plot depicts the percentage of successful expedited recoveries achieved by CESRM for each of the simulations driven by the respective IP multicast transmission traces of Yajnik *et al.* [41]. We consider an expedited recovery to be successful when the expedited request induces the transmission of an expedited reply. Thus, the percentage of successful expedited recoveries is given by the ratio of the number of expedited requests to the number of expedited replies transmitted during the simulation. Figure 6.22 reveals that a substantial percentage of expedited recoveries are successful. This percentage exceeds 65% for all the simulations and exceeds 80% for all but two of the simulations.

The second plot of Figure 6.22 depicts the overall overhead of CESRM in terms of the number of recovery packets sent as a percentage of the respective overhead of SRM. The overhead of CESRM would thus amount to 100% when it is equal to the overhead of SRM for the respective simulation. We split the overhead of CESRM into that associated with retransmission packets and control packets. Since we presume that unicast packets introduce substantially less overhead as compared to that introduced by multicast packets, we distinguish between the number of unicast and multicast control packets.

Figure 6.22 reveals that the number of retransmissions sent by CESRM is substantially less than that sent by SRM. For all the simulations, the retransmission overhead of CESRM is less than 80% of that of SRM. For 9 out of 14 of the simulations, the retransmission overhead of CESRM is less than 60% of that of SRM. Since retransmissions are the only packets that carry data, they are substantially more costly than control packets. Thus, the fact that CESRM sends substantially fewer retransmissions than SRM is a significant performance improvement with respect to SRM.

In terms of the number of control packets, for all but four of the simulations, the overhead of CESRM is either comparable to or less than that of SRM. For the remaining four simulations, although the total number of control packets is more than that of SRM, a large percentage of the

Figure 6.22 CESRM Performance; Percentage of Successful Expedited Recoveries and Overall Packet Overhead



packets sent by CESRM are unicast rather than multicast packets. Thus, even in the cases where the number of control packets sent by CESRM is larger than that sent by SRM, the overhead of CESRM associated with the transmission of control packets is presumably less than that of SRM.

6.5.3 Lossy Recovery Results

In this section, we assume that the packet loss recovery is lossy. In particular, in the simulations whose results are presented in this section, recovery packets are dropped according to the link loss probability estimates calculated in Chapter 5 for links of the IP multicast tree of each of the IP multicast transmissions. Thus, these simulations attempt to capture the loss characteristics of the links of the IP multicast tree of each of the IP multicast transmissions of Yajnik *et al.* [41].

Figure 6.23 presents the per-receiver average normalized recovery times achieved by SRM and CESRM for 6 out of the 14 traces of Yajnik *et al.* [41] — the average normalized recovery times for the simulations corresponding to the remaining traces are similar. Figure 6.23 reveals that the caching-based expedited recovery scheme employed by CESRM substantially reduces the average normalized recovery time. As a result of the losses suffered during the loss recovery, the average normalized recovery time of both SRM and CESRM is greater than that observed when the packet loss recovery process is lossless.

Figure 6.24 depicts the percentage by which the per-receiver average normalized recovery times is reduced using CESRM as opposed to SRM. Once again, the use of expedited recoveries has a substantial effect on the per-receiver average normalized recovery times. For most of the receivers, the average normalized recovery times for CESRM are between 30% and 60% less than those of SRM.

Figure 6.25 depicts the difference between the average normalized recovery time of expedited and non-expedited recoveries of CESRM. Comparing Figures 6.18 and 6.25, it is clear that the introduction of losses in the packet loss recovery process increases this difference. This effect is expected since the loss of recovery packets may result in the failure of initial recovery rounds and, consequently, the increase of the recovery latency associated with non-expedited recoveries.

Figures 6.26 and 6.27 depict the number of request and reply packets, respectively, sent by each of the receivers for 6 out of the 14 traces of Yajnik *et al.* [41] for either the SRM or the CESRM

Figure 6.23 Per-Receiver Average Normalized Recovery Times; Lossless Recovery

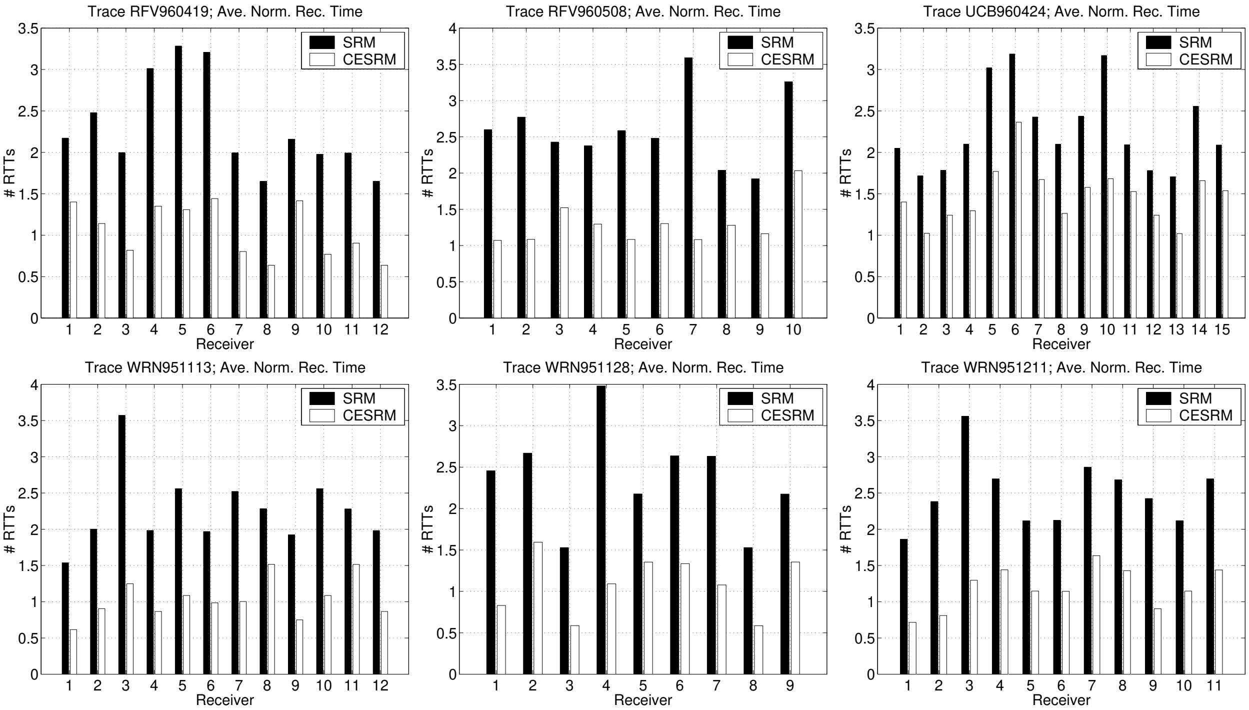


Figure 6.24 Percent Reduction in Per-Receiver Average Normalized Recovery Times; Lossless Recovery

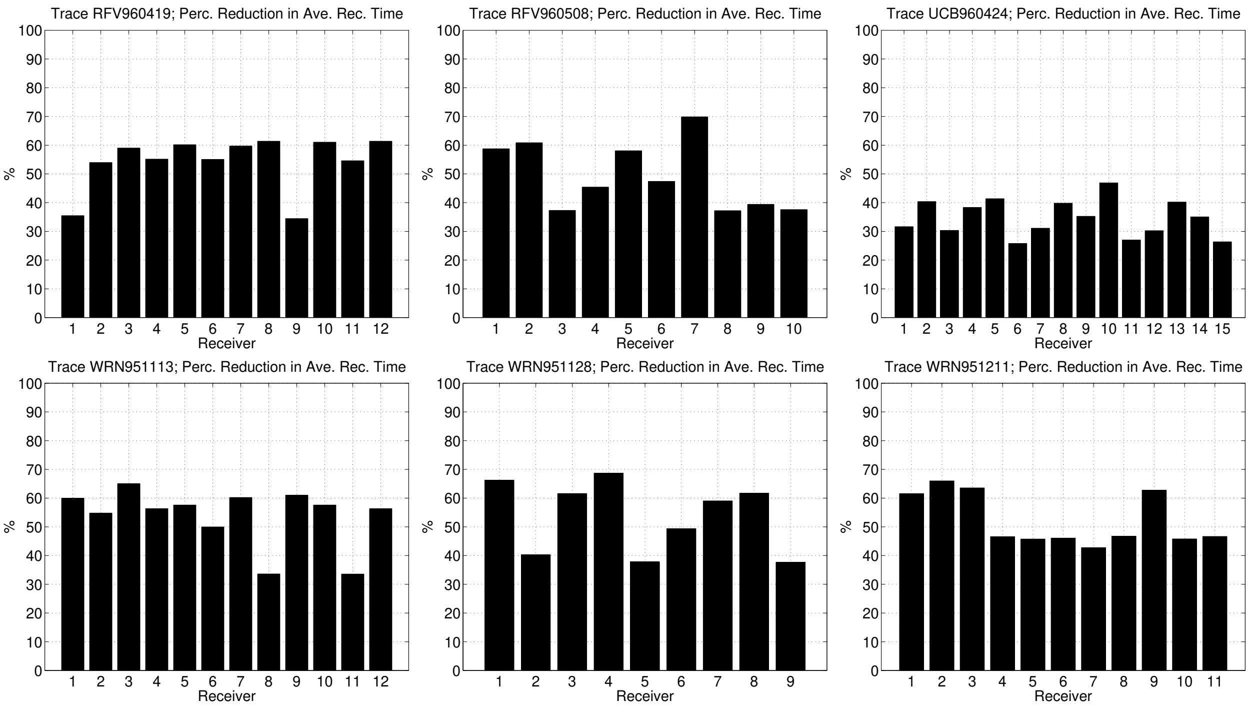


Figure 6.25 Difference in Average Normalized Recovery Times Between Expedited and Non-Expedited Recoveries of CESRM

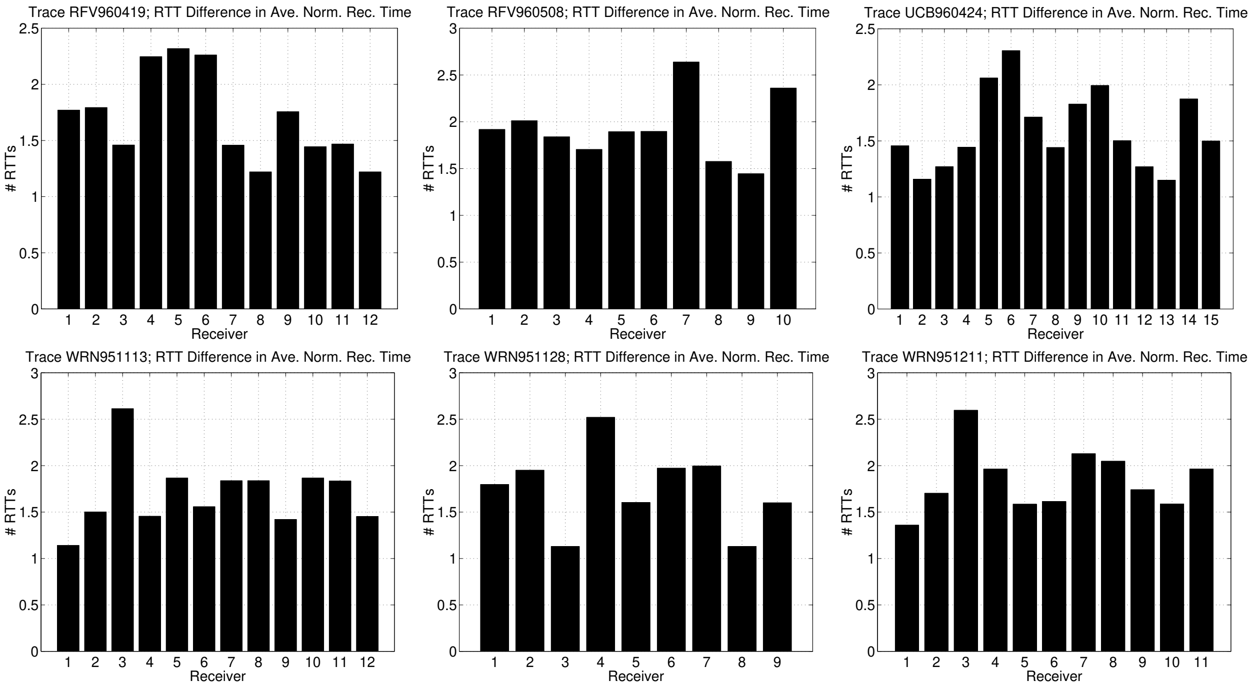
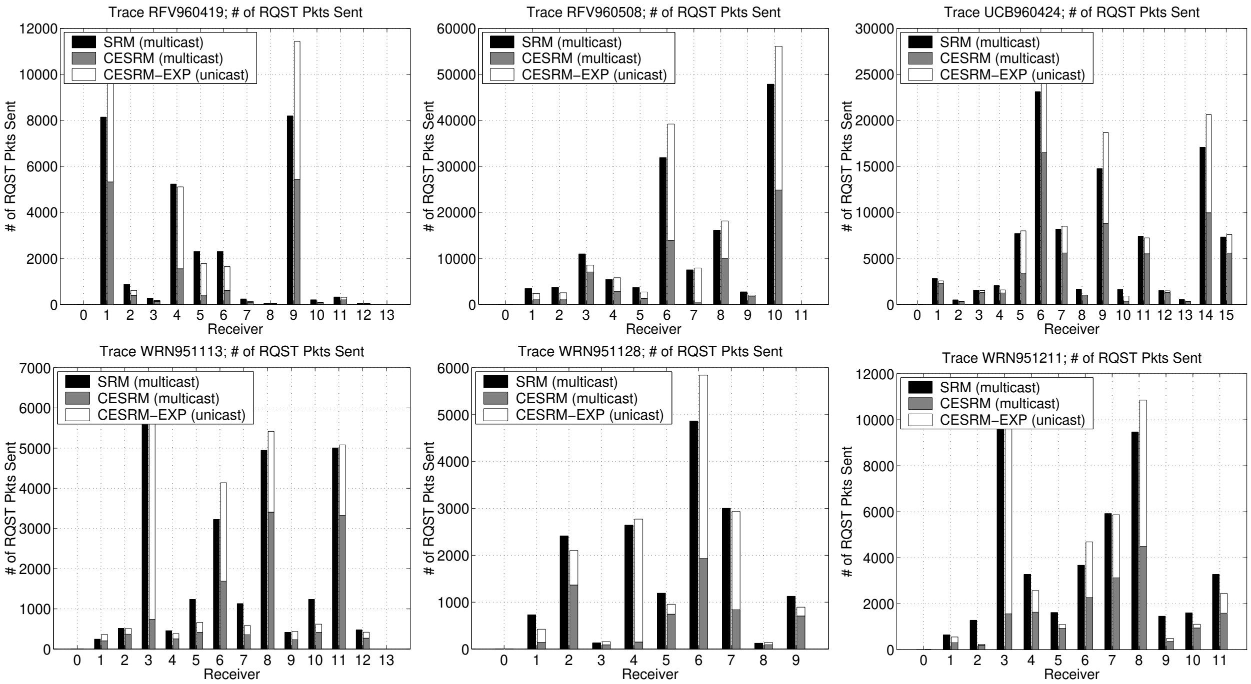


Figure 6.26 Number of Request Packets for SRM and CESRM



protocols — the number of request and reply packets sent by each receiver for the simulations driven by the remaining traces are similar. These plots reveal that losses in the packet loss recovery process result in an increase of the number of request and reply packets sent by both SRM and CESRM. However, this increase is not substantial.

Figure 6.28 depicts the number of update packets sent by each of the receivers for 6 out of the

Figure 6.27 Number of Reply Packets for SRM and CESRM

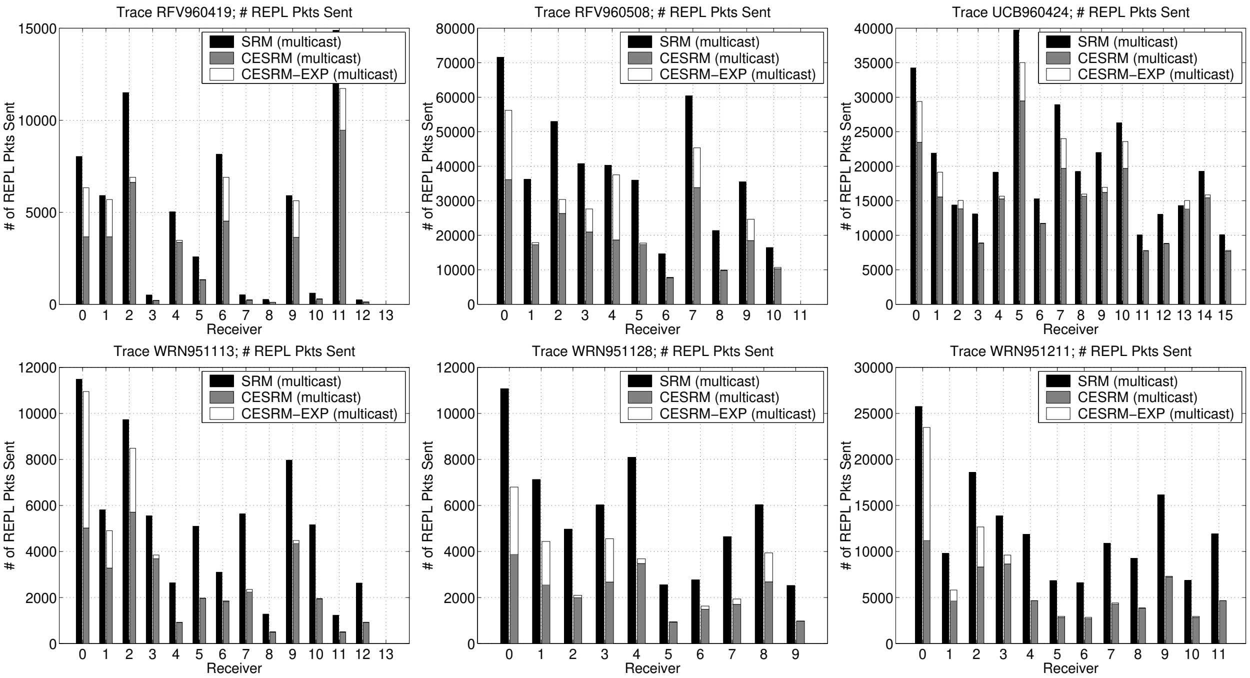
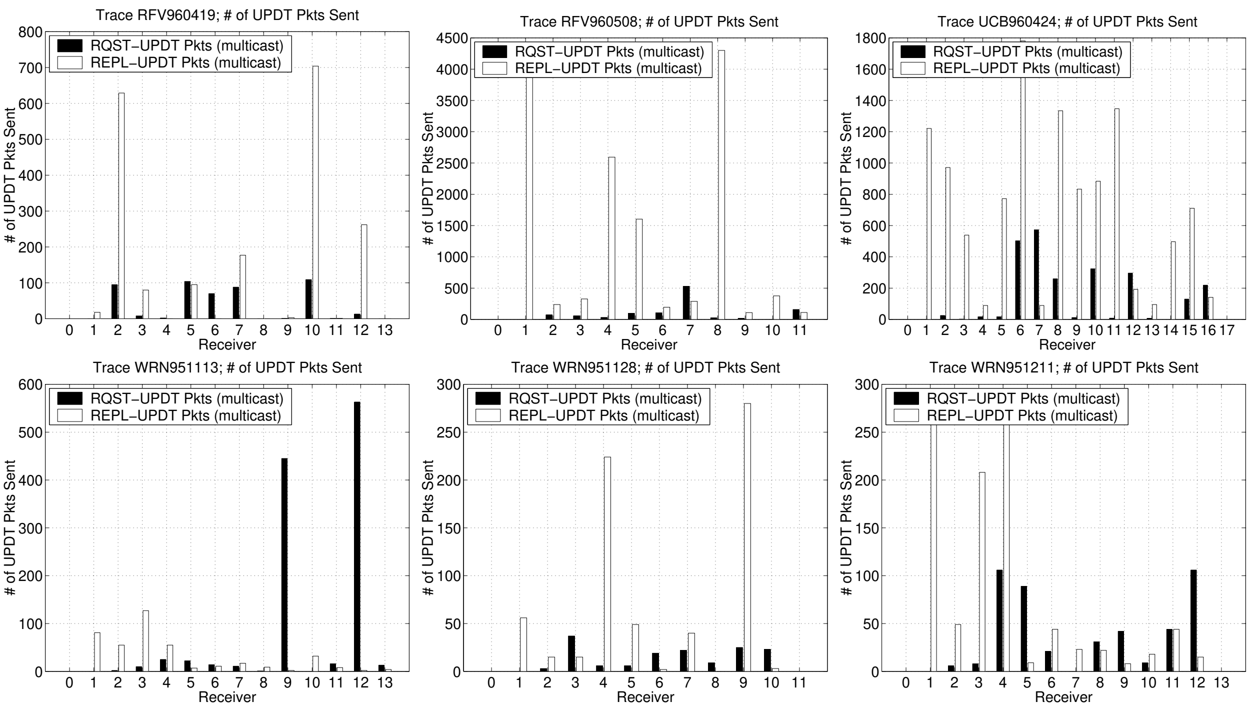


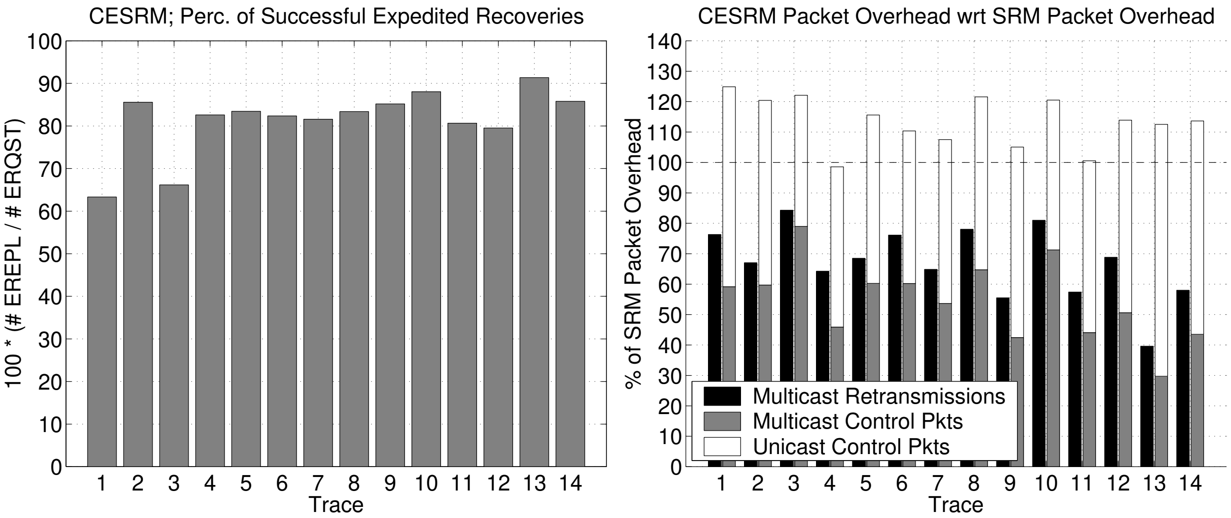
Figure 6.28 Number of Update Packets for CESRM



14 traces of Yajnik *et al.* [41] by CESRM — the number of update packets sent by each receiver for the simulations driven by the remaining traces are similar. These plots reveal that losses in the packet loss recovery process do not substantially increase the number of update packets sent by CESRM. In fact, for some of the simulations the number of update packets sent by CESRM is actually reduced.

Figure 6.29 includes two plots pertaining to the performance of CESRM. The first plot depicts the

Figure 6.29 CESRM Performance; Percentage of Successful Expedited Recoveries and Overall Packet Overhead



percentage of successful expedited recoveries achieved by CESRM for each of the simulations driven by the respective IP multicast transmission traces of Yajnik *et al.* [41]. This percentage exceeds 60% for all the simulations and exceeds 80% for all but three of the simulations. Thus, even in the case when recovery packets suffer losses, a substantial number of the expedited recoveries initiated by CESRM are successful.

The second plot of Figure 6.29 depicts the overall overhead of CESRM in terms of the number of recovery packets sent as a percentage of the respective overhead of SRM. Once again, we split the overhead of CESRM into that associated with retransmission packets and control packets and distinguish between the number of unicast and multicast control packets. Once again, the number of retransmissions sent by CESRM is substantially less than that sent by SRM. In particular, for all but one of the simulations, the number of retransmissions sent by CESRM is less than 80% of the number of retransmissions sent by SRM; for half the simulations, this percentage is below 70% (often, substantially so). Now, consider the number of control packets sent by CESRM. Since losses may cause the failure of a larger percentage of the expedited recoveries, the control packets pertaining to a larger percentage of the expedited recoveries are sent in vain. CESRM must then recover the particular losses using SRM's recovery scheme; thus, in addition to the packets pertaining to the expedited recoveries, CESRM incurs the overhead pertaining to SRM's recovery scheme. Indeed, as shown in Figure 6.29, the number of control packets sent by CESRM is larger than that sent by SRM. However, as in the lossless recovery case, a large percentage of the packets sent by CESRM are unicast rather than multicast packets. Thus, once again, the overhead of CESRM associated with the transmission of control packets is presumably less than that of SRM.

6.5.4 Summary of Simulation Results

Our simulations reveal that more than 65% (60%, when recoveries are lossy) of expedited recoveries are successful. Thanks to such expedited recoveries, CESRM reduces the overall average recovery time of SRM by an average of roughly 50% (40%, when recoveries are lossy). We further observe that these performance gains do not introduce additional packet overhead. On the contrary, in all of our simulations, CESRM reduces the total number of packet retransmissions. Moreover, the number of control packets of CESRM is comparable to that of SRM. In the case of CESRM, however, a large percentage of the control packets are unicast. SRM, in contrast, uses IP multicast to transmit all control packets. We conclude that CESRM effectively incurs less overhead than SRM while substantially reducing recovery latency.

Chapter 7

Reliable Multicast Using Light-Weight Multicast Services

In this chapter, we model and analyze the router-assisted reliable multicast protocol based on the Light-weight Multicast Services [32–34]. This protocol, which we will henceforth refer to as LMS, exploits the augmented functionality of IP multicast routers so as to intelligently forward retransmission requests and constrain the transmission of replies within the subtrees of the IP multicast tree affected by the respective losses.

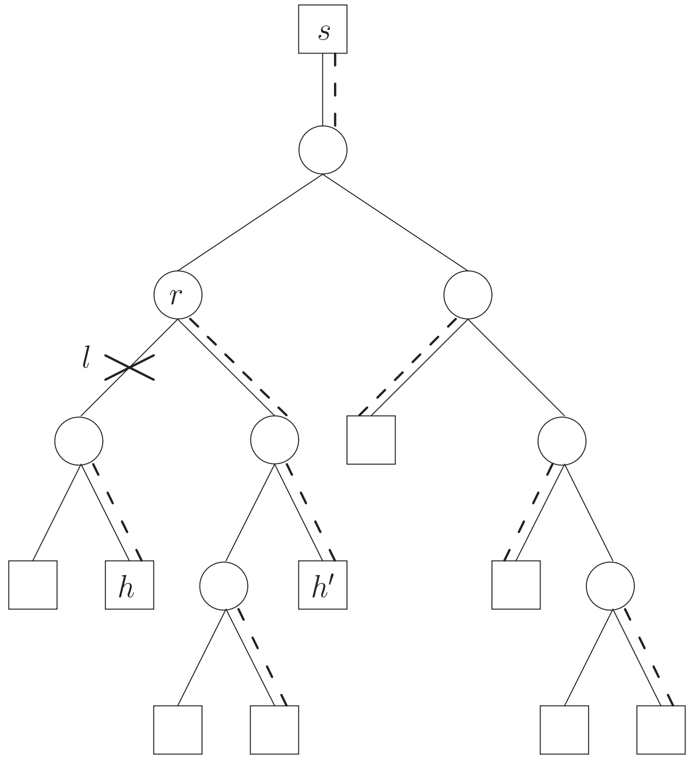
We begin by informally describing the protocol. We then present a formal model of the reliable multicast protocol and the enhanced functionality of the underlying IP multicast routers. We then state the correctness of the protocol; that is, that it is a faithful implementation of the reliable multicast service specification of Chapter 3 with no timeliness guarantees. We conclude by conducting an informal timeliness analysis of LMS in which we: i) state the worst-case recovery latency of LMS when recoveries proceed smoothly, ii) state the worst-case recovery latency of LMS in scenarios that demonstrate LMS's lack of robustness to highly dynamic and faulty environments, and iii) compare its performance to that of both SRM and CESRM.

7.1 Overview of LMS

LMS is a router-assisted reliable multicast protocol that introduces and exploits additional functionality in the underlying IP multicast routers. In our work, we presume that the IP multicast service builds and maintains a shared IP multicast tree. Moreover, we presume that each IP multicast router that is part of the IP multicast tree knows which of its links (network interfaces) leads to each reliable multicast transmission source. For any source s and any router r, we refer to the link of r that leads to s as the upstream link of r for s. For simplicity, we henceforth presume that there exists only a single reliable multicast transmission source s. Moreover, we think of the shared IP multicast tree as a per-source IP multicast tree rooted at s. Within this tree, the notions of upstream and downstream are dictated by the upstream links maintained for the source s by each IP multicast router.

In LMS, each IP multicast router selects one of its descendant members to be in charge of conducting transport layer duties for the subtree originating at the respective router. This member is denoted the *replier* of the respective subtree and the respective router. Members of the reliable multicast group that are willing to perform transport layer duties periodically advertise themselves as repliers. Moreover, each replier estimates the cost that is associated with serving as a replier and advertises

Figure 7.1 Example of LMS recovery hierarchy based on replier links.



it to the IP multicast routers by multicasting a *refresh* packet. A particular member's cost of serving as a replier may correspond, for instance, to loss rate or its distance to the source.

Each IP multicast router maintains in soft state their link that leads to the replier that affords the minimum cost and this minimum cost. We refer to this link and the associated cost as the particular IP multicast router's *replier link* and *replier cost*, respectively. Presuming that the source affords (and advertises) a replier cost of 0, IP multicast routers that are adjacent to the reliable transmission source always adopt their upstream link and the cost of 0 as their replier link and cost, respectively.

Upon either updating or refreshing its replier link and the associated replier cost, an IP multicast router sends (propagates) a refresh packet upstream. By having members of the reliable multicast group advertise their cost of serving as repliers and having the IP multicast routers propagating their replier cost upstream, LMS builds a hierarchy of repliers. Each such replier is responsible for performing transport layer duties on behalf of particular IP multicast subtrees. Figure 7.1 depicts an example of this hierarchy. The solid links correspond to the links that form the IP multicast tree and the dashed links correspond to the replier links dictating the replier hierarchy. For example, in the IP multicast dissemination tree depicted in Figure 7.1, the host h' serves as the replier for the subtree rooted at the IP multicast router r.

Upon detecting the loss of a packet p, a host h multicasts a request with a hop-by-hop designation such that all IP multicast routers process it. After multicasting a retransmission request, h schedules the transmission of another request for p for an appropriate point in time in the future. The reliable multicast transmission of p is guaranteed by LMS by having h keep multicasting and rescheduling requests for p until p is recovered. These requests are rescheduled either at fixed or exponentially increasing intervals.

A router r processes a retransmission request for a packet p according to the link l on which it arrives. If it arrives on the upstream link, *i.e.*, the link leading to the source, then the router knows that the request is destined for its replier and forwards the request on the replier link. If the request arrives on the replier link, then the router forwards the request upstream toward the source. In this case, the replier h' of the subtree rooted at r has shared the loss and serves as the designated requestor for the given loss on behalf of the subtree rooted at r. By forwarding the request upstream, the router r is attempting to reach either the replier of an encompassing IP multicast subtree that has received the packet or the actual source of the packet. Finally, if a request arrives at any other link, the router forwards the request along the replier link, thus calling upon the replier to perform its transport layer duties. In this case, the router annotates the forwarded request with fields that identify the router r and the link l. Papadopoulos *et al.* refer to r as the *turning point* because it is at the router r where the request for p stops moving upstream toward the source and starts moving downstream toward the replier. Analogously, we henceforth refer to the link l as the *turning point link.* The turning point link is the link on which a reply to the request must be forwarded in order to reach the subtree of receivers suffering the loss of the given packet.

Suppose that a host h' receives a request for a packet p and let r and l be the turning point router and link, respectively, pertaining to this request. The host h' processes this request as follows.If it has not received p but has already initiated the recovery of p, then it discards the request for p. If it has neither received p nor already initiated the recovery of p, then it initiates its recovery by transmitting a request for p. If h' has either sent or received p, then it encapsulates p into a unicast packet and unicasts it to the turning point router r. Upon receiving this unicast packet, r decapsulates it and forwards p on the turning point link l as a regular multicast packet. In so doing, r effectively subcasts p down the IP multicast subtree reached through l.

7.1.1 LMS's Weakness

In effect, LMS uses the enhanced IP multicast router functionality to introduce a recovery hierarchy. This hierarchy is very effective in achieving localized recovery and, thus, reducing recovery exposure. However, this hierarchy is relatively static and may not fare well in highly dynamic environments where reliable multicast group members may either crash or leave the reliable multicast group unexpectedly. In such environments, the replier state maintained by the IP multicast routers may often become stale and, thus, either prolong or inhibit packet loss recovery until the replier state in refreshed.

We proceed to describe an example of such behavior. Suppose that a packet p is dropped on the link l of the IP multicast tree. Based on replier state maintained by the IP multicast tree routers, let h be the replier that is responsible for requesting p on behalf of all the members suffering the loss of p. Moreover, let h' be the replier that is responsible of replying to the requests sent by h for packet dropped on l. If either h or h' either crash or leave the reliable multicast group unexpectedly, then all recovery attempts to recover p will fail until the replier state at the appropriate IP multicast routers becomes stale and is refreshed. Since h is the replier responsible for sending the request for p on behalf of all the hosts that shared the loss of p, when it either crashes or leaves the reliable multicast group no request for p will be transmitted beyond the IP multicast subtree sharing the loss. Since h' is the replier that is responsible for replying to the requests of h for packets dropped on l, if it either fails or leaves the reliable multicast group then h's requests for p will go unanswered.

Since the members that suffered the loss of p will keep sending requests until the packet is recovered, p will eventually be recovered when the replier state is updated. However, the resulting recovery may incur a substantial delay.

Unfortunately, the recovery of a packet may similarly be delayed even when particular reliable multicast group members leave gracefully (by multicasting flush packets that are intended to alert the IP multicast routers of stale replier state). When these flush packets are either dropped or do not reach the appropriate IP multicast routers in time for their replier state to be flushed, some of the attempts to recover particular packets may be unsuccessful due again to stale replier state information.

7.1.2 Improving LMS's Robustness to Leaves and Failures

Realizing this weakness, Papadopoulos *et al.* [32, 34] have proposed modifying LMS slightly so as to improve its robustness to replier leaves and crashes. To begin, Papadopoulos *et al.* propose that, after several failed recovery attempts, reliable multicast group members alert the appropriate IP multicast routers that their replier state has become stale. Upon being alerted to stale state, IP multicast routers flush their replier state and begin soliciting replier costs from all downstream links. Moreover, until their replier state has been refreshed, they forward all requests upstream.

Although forwarding requests for a given packet upstream allows the packet's recovery to proceed uninhibited, it potentially exposes the packet's recovery to a larger than required IP multicast subtree. So as to mitigate such unnecessary exposure, Papadopoulos *et al.* propose that IP multicast routers maintain redundant (secondary) replier state. This redundant replier state allows IP multicast routers to promptly delegate the transport duties of the subtree they root to alternate repliers. In particular, once an IP multicast router is alerted to the fact that its replier state has become stale, it may switch to its secondary (now, primary) replier state. Thus, instead of being forwarded upstream, subsequently received requests are forwarded on the secondary replier link. Presuming that the secondary replier hasn't either crashed or left the reliable multicast group, subsequent recovery attempts may thus proceed uninhibited.

To our understanding, however, it is not clear whether and, if so, how the members of the reliable multicast group may ascertain which IP multicast router(s) must either flush or switch to their secondary replier state. For a loss on a link l, the IP multicast router that must flush/switch its replier state is either i) the IP multicast router that is immediately downstream of l, or ii) the IP multicast router that is upstream of l and whose replier does not lie in the IP multicast subtree emanating from l. However, the members of the reliable multicast group that have suffered the loss on l are not aware of which these routers are. Therefore, the only option would be for them to multicast a flush/switch packet. Were they to attempt this, then all of their upstream IP multicast routers would flush/switch their replier state. Instructing all upstream IP multicast routers to switch to their secondary replier state is a plausible solution. However, it may potentially be quite costly since the given IP multicast routers would need to update their secondary replier state. Instructing all the upstream IP multicast routers to flush their replier state would force all requests to be forwarded upstream to the source and induce the IP multicast routers that lead to the source to update their replier state. Both of these options seem to introduce substantial overhead and, thus, do not provide a satisfactory solution to the LMS robustness concerns.

7.2 Formal Model of LMS

In this section, we present a formal model of LMS. Figure 7.2 depicts the interaction of the components of LMS and the environment. The client at each host is modeled by the RM-CLIENT$_h$ timed I/O automaton of Chapter 3. The reporting, membership, and IP buffer components of LMS are identical to those of our model of the CESRM protocol of Chapter 6. In the rest of this section, we present the recovery component pertaining to the LMS reliable multicast protocol and a new model of the IP multicast service component. In addition to refining the behavior of the IP multicast service models used for SRM and CESRM in Chapters 4 and 6, this new IP multicast

Figure 7.2 Interface of the components of the reliable multicast service involving the LMS reliable multicast protocol.

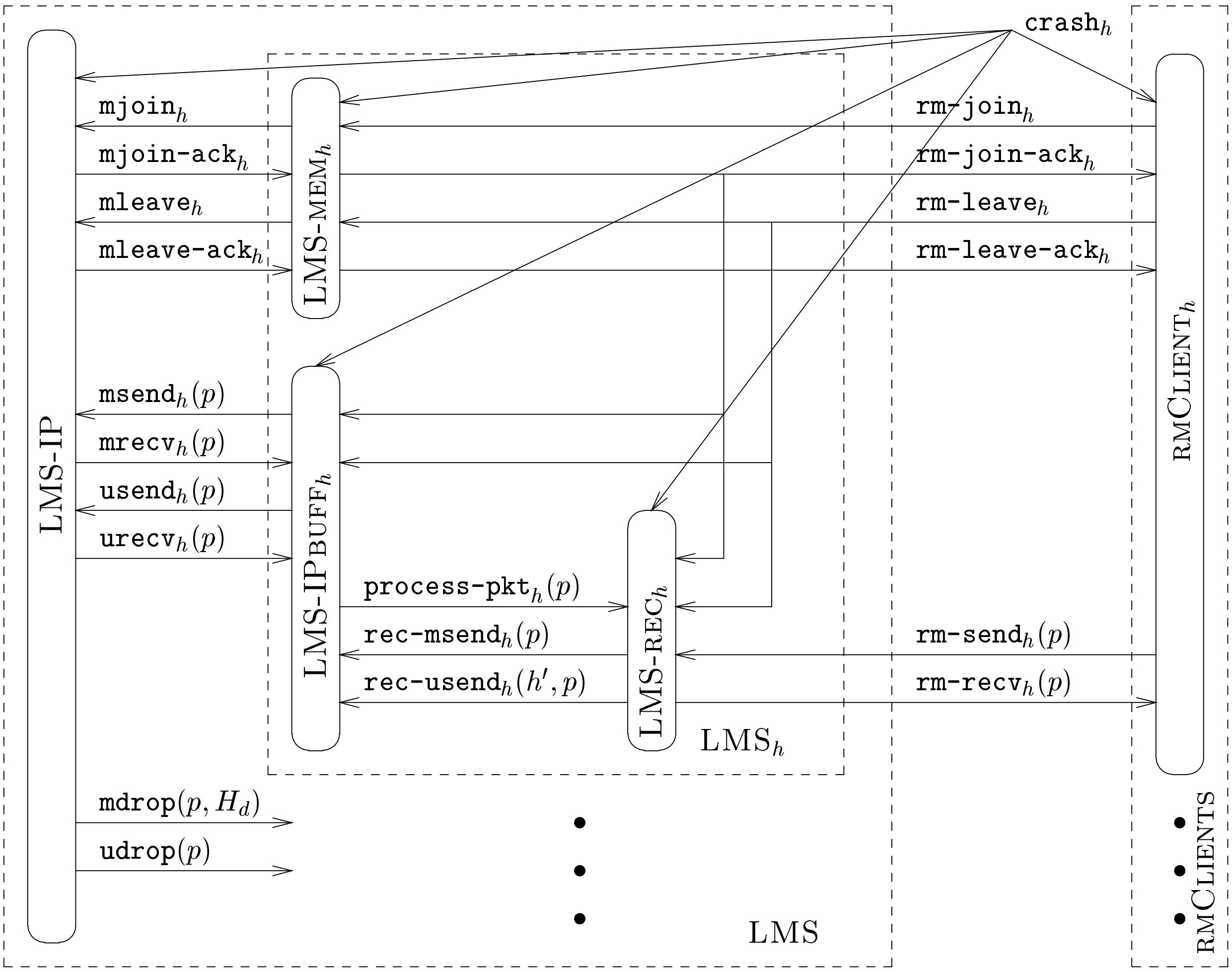


Figure 7.3 Preliminary Definitions for LMS

LMS-Rec$_h$ Automaton

$\textit{LMS-Status} = \{\texttt{idle}, \texttt{member}, \texttt{crashed}\}$
$\textit{Scheduled-Rqsts} = \{\langle s, i, t, k\rangle \mid s \in H, i \in \mathbb{N}, t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}\}$

LMS-IP Automaton

$\textit{IPmcast-Status} = \{\texttt{idle}, \texttt{joining}, \texttt{leaving}, \texttt{member}, \texttt{crashed}\}$
H, set of hosts.
R, set of IP multicast capable routers.
$N = H \cup R$, set of IP multicast capable nodes.
$L = N \times N$, set of bidirectional links interconnecting IP multicast capable nodes.
$L_H = \{\{n, n'\} \in L \mid n \in H\}$
$L_R = \{\{n, n'\} \in L \mid n \in R\}$
$L_n = \{\{n', n''\} \in L \mid n' = n\}$, for $n \in N$.
$L_{nR} = \{\{n', n''\} \in L \mid n' = n, n'' \in R\}$, for $n \in N$.
$L_{nH} = \{\{n', n''\} \in L \mid n' = n, n'' \in H\}$, for $n \in N$.

model precisely specifies the behavior of the enhanced router functionality introduced by LMS. Figures 7.3 and 7.4 contain several set and packet definitions, respectively, used in the specification of LMS.

Figure 7.4 Packet Definitions for LMS

$P_{\text{RM-CLIENT}} : \forall\, p \in P_{\text{RM-CLIENT}},$
$\quad source(p) \in H$
$\quad seqno(p) \in \mathbb{N}$
$\quad data(p) \in \{0,1\}^*$
$\quad id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$\quad suffix(p) = \{\langle s,i\rangle \in H \times \mathbb{N} \mid source(p) = s \wedge seqno(p) \leq i\}$

$P_{\text{IPUCAST-CLIENT}} : \forall\, p \in P_{\text{IPUCAST-CLIENT}},$
$\quad source(p) \in H$
$\quad tp\text{-}router(p) \in R$
$\quad tp\text{-}link(p) \in L_R$
$\quad dest(p) \in H$
$\quad strip(p) \in P_{\text{LMS}}$

$P_{\text{IPMCAST-CLIENT}} : \forall\, p \in P_{\text{IPMCAST-CLIENT}},$
$\quad source(p) \in H$
$\quad cost(p) \in \mathbb{R}^{\geq 0}$
$\quad tp\text{-}router(p) \in R$
$\quad tp\text{-}link(p) \in L_R$
$\quad strip(p) \in P_{\text{LMS}}$
$\quad type(p) = \{\texttt{DATA}, \texttt{RQST}, \texttt{REPL}, \texttt{REFRESH}, \texttt{SOLICIT}, \texttt{FLUSH}, \texttt{PRUNE}\}$

$P_{\text{LMS}} : \forall\, p \in P_{\text{LMS}},$
$\quad type(p) \in \{\texttt{DATA}, \texttt{RQST}, \texttt{REPL}, \texttt{SESS}\}$
$\quad\quad \texttt{DATA}:$
$\quad\quad\quad sender(p) \in H$
$\quad\quad\quad source(p) \in H$
$\quad\quad\quad seqno(p) \in \mathbb{N}$
$\quad\quad\quad strip(p) \in P_{\text{RM-CLIENT}}$
$\quad\quad\quad id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$\quad\quad \texttt{RQST}:$
$\quad\quad\quad sender(p) \in H$
$\quad\quad\quad source(p) \in H$
$\quad\quad\quad seqno(p) \in \mathbb{N}$
$\quad\quad\quad tp\text{-}router(p) \in R$
$\quad\quad\quad tp\text{-}link(p) \in L$
$\quad\quad\quad id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$\quad\quad \texttt{REPL}:$
$\quad\quad\quad sender(p) \in H$
$\quad\quad\quad source(p) \in H$
$\quad\quad\quad seqno(p) \in \mathbb{N}$
$\quad\quad\quad tp\text{-}router(p) \in R$
$\quad\quad\quad tp\text{-}link(p) \in L$
$\quad\quad\quad strip(p) \in P_{\text{RM-CLIENT}}$
$\quad\quad\quad id(p) \in H \times \mathbb{N} : id(p) = \langle source(p), seqno(p)\rangle$
$\quad\quad \texttt{REFRESH}:$
$\quad\quad\quad sender(p) \in H$
$\quad\quad\quad source(p) \in H$
$\quad\quad\quad cost(p) \in \mathbb{R}^{\geq 0}$
$\quad\quad \texttt{SOLICIT}:$
$\quad\quad\quad source(p) \in H$
$\quad\quad\quad seqno(p) \in \mathbb{N}$

Figure 7.5 The LMS-Rec$_h$ Automaton — Signature

Parameters:

$h \in H, \texttt{DFLT-COST}, \texttt{RQST-DELAY}, \texttt{RQST-TIMEOUT}, \texttt{REFRESH-PERIOD} \in \mathbb{R}^+$

Actions:

input
$\quad \texttt{crash}_h$
$\quad \texttt{rm-join-ack}_h$
$\quad \texttt{rm-leave}_h$
$\quad \texttt{rm-send}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$
$\quad \texttt{process-pkt}_h(p)$, for $p \in P_{\text{LMS}}$

output
$\quad \texttt{rm-recv}_h(p)$, for $p \in P_{\text{RM-CLIENT}}$
$\quad \texttt{rec-msend}_h(p)$, for $p \in P_{\text{LMS}}$
$\quad \texttt{rec-usend}_h(h', p)$, for $h' \in H, h' \neq h, p \in P_{\text{LMS}}$

internal
$\quad \texttt{update-cost}_h(s)$, for $s \in H$
$\quad \texttt{send-refresh}_h(s)$, for $s \in H$
$\quad \texttt{send-rqst}_h(s, i)$, for $s \in H, i \in \mathbb{N}$

time-passage
$\quad \nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

7.2.1 The Recovery Component — LMS-rec$_h$

The LMS-rec$_h$ timed I/O automaton specifies the recovery component of the LMS protocol. Figure 7.5 presents the signature of LMS-rec$_h$, that is, its parameters and its actions. Figure 7.6 presents the variables of LMS-rec$_h$. Figures 7.7 and 7.8 present the discrete transitions of LMS-rec$_h$. Throughout this section, we only describe the functionality of LMS-rec$_h$ that is either new or different from that of either SRM-rec$_h$ or CESRM-rec$_h$ presented in Chapters 4 and 6, respectively. Once again, in order to provide the appropriate context, the description of each of the parameters of LMS-rec$_h$ is deferred to appropriate places within the description of its variables and actions.

Figure 7.6 The LMS-Rec$_h$ Automaton — Variables

Variables:

$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status \in LMS\text{-}Status$, initially $status =$ `idle`
$cost(s) \in \mathbb{R}^{\geq 0}$, for all $s \in H$, initially $cost(s) =$ `DFLT-COST`, for all $s \in H$
$refresh\text{-}deadline(s) \in \mathbb{R}^{\geq 0} \cup \perp$, for all $s \in H$, initially $refresh\text{-}deadline(s) = \perp$, for all $s \in H$
$min\text{-}seqno(h') \in \mathbb{N} \cup \perp$, for all $h' \in H$, initially $min\text{-}seqno(h') = \perp$, for all $h' \in H$
$max\text{-}seqno(h') \in \mathbb{N} \cup \perp$, for all $h' \in H$, initially $max\text{-}seqno(h') = \perp$, for all $h' \in H$
$archived\text{-}pkts \subseteq P_{\text{RM-CLIENT}} \times \mathbb{R}^{\geq 0}$, initially $archived\text{-}pkts = \emptyset$
$scheduled\text{-}rqsts \subseteq Scheduled\text{-}Rqsts$, initially $scheduled\text{-}rqsts = \emptyset$
$to\text{-}be\text{-}delivered \subseteq P_{\text{RM-CLIENT}}$, initially $to\text{-}be\text{-}delivered = \emptyset$
$msend\text{-}buff \subseteq P_{\text{LMS}}$, initially $msend\text{-}buff = \emptyset$
$usend\text{-}buff(h') \subseteq P_{\text{LMS}}$, for all $h' \in H, h' \neq h$, initially $usend\text{-}buff = \emptyset$, for all $h' \in H, h' \neq h$
$recovered\text{-}pkts? \subseteq H \times \mathbb{N}$, initially $recovered\text{-}pkts? = \emptyset$

Derived Variables:

for all $h' \in H$, $proper?(h') = \begin{cases} \emptyset & \text{if } min\text{-}seqno(h') = \perp \\ \{\langle s,i\rangle \in H \times \mathbb{N} \mid s = h', min\text{-}seqno(h') \leq i\} & \text{otherwise} \end{cases}$

for all $h' \in H$, $window?(h') = \begin{cases} \emptyset & \text{if } min\text{-}seqno(h') = \perp \\ \{\langle s,i\rangle \in H \times \mathbb{N} \mid s = h', min\text{-}seqno(h') \leq i \leq max\text{-}seqno(h')\} & \text{otherwise} \end{cases}$

$archived\text{-}pkts? = \{\langle s,i\rangle \in H \times \mathbb{N} \mid \exists\, p \in P_{\text{RM-CLIENT}}, t \in \mathbb{R}^{\geq 0} : \langle p,t\rangle \in archived\text{-}pkts \wedge id(p) = \langle s,i\rangle\}$
$archived\text{-}pkts?(h') = \{\langle s,i\rangle \in archived\text{-}pkts? \mid s = h'\}$, for all $h' \in H$
$scheduled\text{-}rqsts? = \{\langle s,i\rangle \in H \times \mathbb{N} \mid \exists\, t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N} : \langle s,i,t,k\rangle \in scheduled\text{-}rqsts\}$
$scheduled\text{-}rqsts?(h') = \{\langle s,i\rangle \in scheduled\text{-}rqsts? \mid s = h'\}$, for all $h' \in H$
$to\text{-}be\text{-}delivered? = \{\langle s,i\rangle \in H \times \mathbb{N} \mid \exists\, p \in to\text{-}be\text{-}delivered : \langle s,i\rangle = id(p)\}$
$to\text{-}be\text{-}delivered?(h') = \{\langle s,i\rangle \in to\text{-}be\text{-}delivered? \mid s = h'\}$, for all $h' \in H$
$recovered\text{-}pkts?(h') = \{\langle s,i\rangle \in recovered\text{-}pkts? \mid s = h'\}$, for all $h' \in H$
$sources = \{h' \in H \mid archived\text{-}pkts?(h') \neq \emptyset\}$

Variables

Each variable $cost(s)$, for $s \in H$, denotes the cost associated with h serving as a replier for packets transmitted by the source s. Each variable $cost(s)$, for $s \in H$, is initialized to `DFLT-COST`, where `DFLT-COST` $\in \mathbb{R}^+$ specifies the default replier cost of h.

Each variable $refresh\text{-}deadline(s)$, for $s \in H$, denotes the time at which h must send its next refresh packet to the IP multicast communication service — a member of the reliable multicast group periodically sends refresh packets so as to advertise its cost of serving as a replier for packets transmitted by the source s. Each variable $refresh\text{-}deadline(s)$, for $s \in H$, is initialized to $\perp$.

The derived variable *sources* denotes the set of IP multicast transmission sources that h is aware of. The host h becomes aware of a particular source s upon receiving either an original transmission or a retransmission of a packet originally transmitted by s. Since h archives all such packets, *sources* is the set of sources some of whose packets h has archived. It is initially equal to the empty set.

Actions

The internal action $\texttt{update-cost}_h(s)$, for $s \in H$, updates the cost of h serving as a replier for packets transmitted by the source s. The action $\texttt{update-cost}_h(s)$ is enabled when the host h is a member of the reliable multicast group and h is aware of the source s. The effects of $\texttt{update-cost}_h(s)$ are to nondeterministically set the variable $cost(s)$. For simplicity, we have chosen not to model the manner in which the cost of h serving as a replier for packets transmitted by s is calculated. By forcing hosts to advertise a finite cost for serving as repliers, we are effectively disallowing hosts from avoiding to become repliers. However, the larger the cost of h serving as a replier for s, the less likely it is that h is selected to serve as a replier for s.

The internal action $\texttt{send-refresh}_h(s)$, for $s \in H$, models the expiration of the refresh timeout for the source s and the composition of a refresh packet intended to advertise the cost of h serving as a replier for s. The action $\texttt{send-refresh}_h(s)$ is enabled when the host h is a member of the reliable multicast group, h is aware of the source s, the refresh timeout for s has previously been set and has expired, *i.e.*, $\mathit{refresh\text{-}deadline}(s) \neq \perp$ and $\mathit{now} = \mathit{refresh\text{-}deadline}(s)$. The effects of $\texttt{send-refresh}_h(s)$ are to compose a refresh packet, to add it to the *msend-buff* buffer, and to reset the refresh timeout to a point in time REFRESH-PERIOD time units in the future. The parameter REFRESH-PERIOD specifies the period with which LMS-REC_h transmits refresh packets which advertise the current cost of having h serve as a replier for s.

The internal action $\texttt{send-rqst}_h(s, i)$, for $s \in H$ and $i \in \mathbb{N}$, models the expiration of the request transmission timeout, the transmission of a request for the packet $\langle s, i\rangle$, and the scheduling the transmission of another request for the packet $\langle s, i\rangle$ for an appropriate time in the future. The action $\texttt{send-rqst}_h(s, i)$ is enabled when the host h is a member of the reliable multicast group and the transmission time of a scheduled request for the packet $\langle s, i\rangle$ has arrived; that is, $\mathit{status} = \texttt{member}$, $t = \mathit{now}$, and $\langle s, i, t, k\rangle \in \mathit{scheduled\text{-}rqsts}$. The effects of $\texttt{send-rqst}_h(s, i)$ are to compose a request for the packet $\langle s, i\rangle$, to add it to the IP multicast transmission buffer of LMS-Rec_h, and to reschedule the request for an appropriate time in the future by updating the request tuple in *scheduled-rqsts* pertaining to the packet $\langle s, i\rangle$. Papadopoulos *et al.* [32, 34] propose two request rescheduling schemes. The first involves scheduling the transmission of the next request for a point in time that is RQST-TIMEOUT time units in the future. This scheme results in requests being transmitted at fixed intervals. The second involves scheduling the transmission of the next request for a point in time that is exponentially further (than the previous request) in the future; that is, for a point in time $\mathit{now} + 2^{k_r-1}\texttt{RQST-TIMEOUT}$, where $k_r = k + 1$ and k is the back-off used to schedule the previous request. This scheme results in requests being transmitted at exponentially increasing intervals. The pseudo-code of Figure 7.7 implements the second scheme.

The input action $\texttt{process-pkt}_h(p)$ models the processing of the packet p by LMS-REC_h. The packet p is processed only when the host h is a member of the reliable multicast group. We proceed by describing the effects of $\texttt{process-pkt}_h(p)$ depending on the type of the packet p. Throughout our presentation of the effects of $\texttt{process-pkt}_h(p)$, we let $s_p \in H$ and $i_p \in \mathbb{N}$ denote the source and the sequence number pertaining to the packet p.

First, consider the case where p is a DATA packet. If p is the foremost packet from s_p, $\texttt{process-pkt}_h(p)$ records its reception. If h is not the source of p and p is not already archived, then $\texttt{process-pkt}_h(p)$ archives p. Moreover, $\texttt{process-pkt}_h(p)$ cancels any scheduled requests for p. Finally, if p is a proper packet, then $\texttt{process-pkt}_h(p)$ adds p to the packet delivery set *to-be-delivered* and schedules the request for any trailing missing packets for RQST-DELAY time units in the future. The requests for these missing packets are delayed so as to avoid the transmission of extraneous requests when packets are temporarily presumed missing due to packet reordering. The parameter RQST-DELAY specifies the amount of time that such requests must be delayed to avoid the transmission of such extraneous requests. It is important to note that LMS-Rec_h archives all packets and not only proper packets as done by the recovery components of our models of the SRM and CESRM protocols in Sections 4.3.4 and 6.2.3, respectively. This is done because in the case of LMS a host may serve as a replier for improper packets; that is, packets that it need not deliver to its reliable multicast client.

Second, consider the case where p is a RQST packet. If the packet $\langle s_p, i_p\rangle$ has been archived, then $\texttt{process-pkt}_h(p)$ composes a reply packet for the packet $\langle s_p, i_p\rangle$ and adds it to the unicast transmission buffer *usend-buff*. The destination of this reply is the turning point router annotating p. The reply is also annotated with the turning point link annotating p. Upon receiving this unicast reply, the turning point router will forward the reply along this turning point link, which presumably leads to the requestor that instigated the reply. If the packet $\langle s_p, i_p\rangle$ has not been archived

Figure 7.7 The LMS-Rec$_h$ Automaton — Discrete Transitions

input $\texttt{crash}_h$

eff $\mathit{status} := \texttt{crashed}$

input $\texttt{rm-join-ack}_h$

eff **if** $\mathit{status} \neq \texttt{crashed}$ **then**
$\quad \mathit{status} := \texttt{member}$
$\quad \mathit{refresh\text{-}deadline} :\in \mathit{now} + (0, \texttt{REFRESH-PERIOD}]$

input $\texttt{rm-leave}_h$

eff **if** $\mathit{status} \neq \texttt{crashed}$ **then**
$\quad$ Reinitialize all variables except now.

input $\texttt{rm-send}_h(p)$

eff **if** $\mathit{status} = \texttt{member} \wedge h = \mathit{source}(p)$ **then**
$\quad \langle s_p, i_p \rangle = \mathit{id}(p)$
$\quad$ `\\ Record foremost DATA packet`
$\quad$ **if** $\mathit{min\text{-}seqno}(s_p) = \perp$ **then** $\mathit{min\text{-}seqno}(s_p) := i_p$
$\quad$ `\\ Only consider next packet`
$\quad$ **if** $\mathit{max\text{-}seqno}(s_p) = \perp$
$\quad\quad \vee i_p = \mathit{max\text{-}seqno}(s_p) + 1$
$\quad$ **then**
$\quad\quad \mathit{max\text{-}seqno}(s_p) := i_p$
$\quad\quad$ `\\ Archive packet`
$\quad\quad \mathit{archived\text{-}pkts} \cup= \{\langle p, \mathit{now} \rangle\}$
$\quad\quad$ `\\ Compose data packet`
$\quad\quad \mathit{msend\text{-}buff} \cup= \{\mathit{comp\text{-}data\text{-}pkt}(p)\}$

internal $\texttt{update-cost}_h(s)$

pre $\mathit{status} = \texttt{member} \wedge s \in \mathit{sources}$

eff `\\ Update cost`
$\quad \mathit{cost}(s) :\in \mathbb{R}^{\geq 0}$

internal $\texttt{send-refresh}_h(s)$

pre $\mathit{status} = \texttt{member} \wedge s \in \mathit{sources}$
$\quad \wedge \mathit{refresh\text{-}deadline}(s) \neq \perp \wedge \mathit{refresh\text{-}deadline}(s) = \mathit{now}$

eff `\\ Compose refresh packet`
$\quad \mathit{msend\text{-}buff} \cup= \{\mathit{comp\text{-}refresh\text{-}pkt}(h, \mathit{now}, s, \mathit{cost}(s))\}$
$\quad$ `\\ Reset refresh deadline`
$\quad \mathit{refresh\text{-}deadline}(s) := \mathit{now} + \texttt{REFRESH-PERIOD}$

internal $\texttt{send-rqst}_h(s, i)$

choose $t \in \mathbb{R}^{\geq 0}, k \in \mathbb{N}$

pre $\mathit{status} = \texttt{member}$
$\quad \wedge t = \mathit{now} \wedge \langle s, i, t, k \rangle \in \mathit{scheduled\text{-}rqsts}$

eff `\\ Compose request packet`
$\quad \mathit{msend\text{-}buff} \cup= \{\mathit{comp\text{-}rqst\text{-}pkt}(s, i)\}$
$\quad$ `\\ Back-off scheduled request`
$\quad \mathit{scheduled\text{-}rqsts} \setminus= \{\langle s, i, t, k \rangle\}$
$\quad k_r := k + 1;\ t_r := \mathit{now} + 2^{k_r - 1}\texttt{RQST-TIMEOUT}$
$\quad \mathit{scheduled\text{-}rqsts} \cup= \{\langle s, i, t_r, k_r \rangle\}$

time-passage $\nu(t)$

pre $\mathit{status} = \texttt{crashed}$
$\quad \vee(\mathit{to\text{-}be\text{-}delivered} = \emptyset$
$\quad\quad \wedge \mathit{msend\text{-}buff} = \emptyset \wedge (\wedge_{h' \in H, h' \neq h} \mathit{usend\text{-}buff}(h') = \emptyset)$
$\quad\quad \wedge_{s \in \mathit{sources}} (\mathit{refresh\text{-}deadline}(s) = \perp$
$\quad\quad\quad\quad \vee \mathit{now} + t \leq \mathit{refresh\text{-}deadline}(s))$
$\quad\quad \wedge$ no requests scheduled earlier than $\mathit{now} + t$)

eff $\mathit{now} := \mathit{now} + t$

output $\texttt{rm-recv}_h(p)$

pre $\mathit{status} = \texttt{member} \wedge p \in \mathit{to\text{-}be\text{-}delivered}$
$\quad \wedge(\nexists\, p' \in \mathit{to\text{-}be\text{-}delivered}:$
$\quad\quad \mathit{source}(p') = \mathit{source}(p) \wedge \mathit{seqno}(p') < \mathit{seqno}(p))$

eff $\mathit{to\text{-}be\text{-}delivered} \setminus= \{p\}$

output $\texttt{rec-msend}_h(p)$

pre $\mathit{status} = \texttt{member} \wedge p \in \mathit{msend\text{-}buff}$

eff $\mathit{msend\text{-}buff} \setminus= \{p\}$

output $\texttt{rec-usend}_h(h', p)$

pre $\mathit{status} = \texttt{member} \wedge p \in \mathit{usend\text{-}buff}(h')$

eff $\mathit{usend\text{-}buff}(h') \setminus= \{p\}$

---

and for which there is no scheduled request, then $\texttt{process-pkt}_h(p)$ schedules the immediate transmission of a request for the packet $\langle s_p, i_p \rangle$. This is done by adding the tuple $\langle s_p, i_p, \mathit{now}, 0 \rangle$ to the set *scheduled-rqsts* of scheduled requests. Finally, if the packet $\langle s_p, i_p \rangle$ is a proper packet, then $\texttt{process-pkt}_h(p)$ schedules the immediate request for any trailing missing packets. Here, $\texttt{process-pkt}_h(p)$ does not delay the transmission of these requests by `RQST-DELAY` time units; we presume that, by the time p is scheduled, transmitted, and received by h, a sufficient amount of time has elapsed such that the premature transmission of requests as a result of packet reordering is highly unlikely.

Third, consider the case where p is a `REPL` packet. The effects of $\texttt{process-pkt}_h(p)$ in the case of `REPL` packet are similar to those when p is a `DATA` packet. The only difference is that, if h is not the source of p and p is not already archived, then in addition to archiving p, $\texttt{process-pkt}_h(p)$ also records that p has been recovered.

Finally, consider the case where p is a `SOLICIT` packet. This action models the solicitation of an updated cost of h serving as a replier for packets transmitted by s_p. If h is aware of the source s_p, then it composes a refresh packet including the current cost of h serving as a replier for s_p and adds it to the multicast buffer *msend-buff*. Moreover, $\texttt{process-pkt}_h(p)$ resets the refresh timeout for s_p to a point in time `REFRESH-PERIOD` time units in the future.

Figure 7.8 The LMS-Rec$_h$ Automaton — Discrete Transitions

```
input process-pkt_h(p)
where type(p) = DATA
eff  if status = member then
       ⟨s_p, i_p⟩ = id(p)
       \\ Record foremost DATA packet
       if h ≠ s_p ∧ min-seqno(s_p) =⊥ then
         min-seqno(s_p) := i_p; max-seqno(s_p) := i_p
       \\ Archive the packet
       if h ≠ s_p ∧ ⟨s_p, i_p⟩ ∉ archived-pkts? then
         archived-pkts ∪= {⟨strip(p), now⟩}
       \\ Cancel any scheduled requests
       scheduled-rqsts \= {⟨s_p, i_p, t, k⟩ | t ∈ ℝ^{≥0}, k ∈ ℕ}
       \\ Only consider proper packets
       if min-seqno(s_p) ≠⊥ ∧min-seqno(s_p) ≤ i_p then
         \\ Deliver proper packet
         if h ≠ s_p then to-be-delivered ∪= {strip(p)}
         \\ Discover any trailing missing packets
         if h ≠ s_p ∧ max-seqno(s_p) < i_p then
           foreach i ∈ ℕ : max-seqno(s_p) < i < i_p do:
             \\ Schedule a delayed request
             scheduled-rqsts ∪=
               {⟨s_p, i, now + RQST-DELAY, 0⟩}
           max-seqno(s_p) := i_p
```

```
input process-pkt_h(p)
where type(p) = RQST
eff  if status = member then
       ⟨s_p, i_p⟩ = id(p)
       if ⟨s_p, i_p⟩ ∈ archived-pkts? then
         \\ Compose reply packet
         choose p' ∈ P_RM-CLIENT, t ∈ ℝ^{≥0}
           where ⟨p', t⟩ ∈ archived-pkts ∧ id(p') = ⟨s_p, i_p⟩
         usend-buff ∪=
           {comp-repl-pkt(p', tp-router(p), tp-link(p))}
       else
         if h ≠ s_p ∧ ⟨s_p, i_p⟩ ∉ scheduled-rqsts? then
           \\ Schedule an immediate request
           scheduled-rqsts ∪= {⟨s_p, i_p, now, 0⟩}
       \\ Only consider proper packets
       if min-seqno(s_p) ≠⊥ ∧min-seqno(s_p) ≤ i_p then
         \\ Discover any trailing missing packets
         if h ≠ s_p ∧ max-seqno(s_p) < i_p then
           foreach i ∈ ℕ : max-seqno(s_p) < i < i_p do:
             \\ Schedule an immediate request
             scheduled-rqsts ∪= {⟨s_p, i, now, 0⟩}
           max-seqno(s_p) := i_p
```

```
input process-pkt_h(p)
where type(p) = REPL
eff  if status = member then
       ⟨s_p, i_p⟩ = id(p)
       \\ Archive the packet
       if h ≠ s_p ∧ ⟨s_p, i_p⟩ ∉ archived-pkts? then
         recovered-pkts? ∪= {⟨s_p, i_p⟩}
         archived-pkts ∪= {⟨strip(p), now⟩}
       \\ Cancel any scheduled requests
       scheduled-rqsts \= {⟨s_p, i_p, t, k⟩ | t ∈ ℝ^{≥0}, k ∈ ℕ}
       \\ Only consider proper packets
       if min-seqno(s_p) ≠⊥ ∧min-seqno(s_p) ≤ i_p then
         \\ Deliver proper packet
         if h ≠ s_p then to-be-delivered ∪= {strip(p)}
         \\ Discover any trailing missing packets
         if h ≠ s_p ∧ max-seqno(s_p) < i_p then
           foreach i ∈ ℕ : max-seqno(s_p) < i < i_p do:
             \\ Schedule an immediate request
             scheduled-rqsts ∪= {⟨s_p, i, now, 0⟩}
           max-seqno(s_p) := i_p
```

```
input process-pkt_h(p)
where type(p) = SOLICIT
eff  if status = member then
       s_p = source(p)
       if s_p ∈ sources then
         \\ Compose refresh packet
         msend-buff ∪=
           {comp-refresh-pkt(h, now, s_p, cost(s_p))}
         \\ Reset refresh deadline
         refresh-deadline(s_p) := now + REFRESH-PERIOD
```

7.2.2 The Light-Weight Multicast Services Component — LMS-IP

In this section, we give an abstract specification of the IP communication service enhanced with the Light-Weight Multicast Services (LMS) [32,34]. We model the LMS-enhanced IP multicast service by the timed I/O automaton LMS-IP. Figure 7.9 presents the signature of LMS-IP, Figure 7.10 lists the variables and derived variables of LMS-IP, and Figures 7.11, 7.12, and 7.13 specify the discrete transitions of LMS-IP.

It is important to note that LMS-IP models the dissemination tree used by the IP multicast communication service to disseminate IP multicast packets to the members of the IP multicast group. In particular, LMS-IP models the routers and the bidirectional links that form the IP multicast dissemination tree and the hop-by-hop transmission of packets from one router of the tree to the next and, finally, to the members of the IP multicast group. In terms of faults, we only consider host crashes and packet drops on the bidirectional links interconnecting the hosts to their respective gateway routers and the routers among themselves.

Figure 7.9 The LMS-IP Automaton — Signature

Parameters:

`REPL-TIMEOUT` $\in \mathbb{R}^+$

Actions:

input
- $\texttt{crash}_h$, for $h \in H$
- $\texttt{mjoin}_h$, for $h \in H$
- $\texttt{mleave}_h$, for $h \in H$
- $\texttt{usend}_h(p)$, for $h \in H, p \in P_{\text{IPUCAST-CLIENT}}$
- $\texttt{msend}_h(p)$, for $h \in H, p \in P_{\text{IPMCAST-CLIENT}}$

internal
- $\texttt{mprop}_{nl}(p)$, for $n \in N, l \in L, p \in P_{\text{IPMCAST-CLIENT}}$

output
- $\texttt{mjoin-ack}_h$, for $h \in H$
- $\texttt{mleave-ack}_h$, for $h \in H$
- $\texttt{urecv}_n(p)$, for $n \in N, p \in P_{\text{IPUCAST-CLIENT}}$
- $\texttt{mrecv}_h(p)$, for $h \in H, p \in P_{\text{IPMCAST-CLIENT}}$
- $\texttt{udrop}(p)$, for $p \in P_{\text{IPUCAST-CLIENT}}$
- $\texttt{mdrop}_{nl}(p)$, for $n \in N, l \in L, p \in P_{\text{IPMCAST-CLIENT}}$

time-passage
- $\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$

Variables

The variable *now* $\in \mathbb{R}^{\geq 0}$ denotes the time that has elapsed since the beginning of an execution of LMS-IP. Each variable $\mathit{status}(h) \in$ *IPmcast-Status*, for $h \in H$, denotes the IP multicast membership status of the host h as already described in Section 4.3.5. The set $\mathit{routers} \subseteq R$ consists of the routers that are part of the IP multicast tree. Each set $\mathit{links}(n) \subseteq L_n$, for $n \in N$, consists of the links connecting the node n to its neighbor nodes in the IP multicast tree.

Each variable $\mathit{upstream\text{-}link}(r,s) \in L_r \cup \{\perp\}$, for $r \in R, s \in H$, is the upstream link of r for s; that is, the link of r that is believed to lead to the source s. Each variable $\mathit{repl\text{-}state}(r,s) \in \{L_r \times \mathbb{R}^{\geq 0} \cup \{\infty\} \times \mathbb{R}^{\geq 0} \cup \{\infty\}\} \cup \{\perp\}$, for all $r \in R, s \in H$, involves the soft state maintained by r for the source s. This state is a tuple involving the replier link $\mathit{repl\text{-}link}(r,s)$ of r for s, the cost $\mathit{repl\text{-}cost}(r,s)$ of recovering the packet from the replier reached through the replier link, and the expiration time $\mathit{repl\text{-}timeout}(r,s)$ of the replier state maintained by r for s.

The set $\mathit{upkts} \subseteq P_{\text{IPUCAST-CLIENT}}$ consists of the unicast packets that have been sent by clients of the IP unicast communication service and whose delivery is still pending. Each variable $\mathit{mqueue}(n,l) : \mathit{QueueOf}(P_{\text{IPMCAST-CLIENT}})$, for all $n \in N, l \in L_n$, consists of the packets that are pending transmission at node n along the link l. The variable $\mathit{mqueue}(n,l)$ is presumed to be a FIFO queue. For any packet $p \in P_{\text{IPMCAST-CLIENT}}$, the operation $\mathit{enqueue}(p, \mathit{mqueue}(n,l))$ adds the packet p to the end of the queue $\mathit{mqueue}(n,l)$. The operation $\mathit{dequeue}(\mathit{mqueue}(n,l))$ removes and returns the packet at the front of the queue $\mathit{mqueue}(n,l)$. The operation $\mathit{head}(\mathit{mqueue}(n,l))$ returns the packet at the front of the queue $\mathit{mqueue}(n,l)$ without actually removing it from the queue.

The derived variables $\mathit{up} \subseteq H$, $\mathit{idle} \subseteq H$, $\mathit{joining} \subseteq H$, $\mathit{leaving} \subseteq H$, and $\mathit{members} \subseteq H$ are as defined in Section 4.3.5. The set *up* denotes the set of hosts that are operational. The sets *idle*, *joining*, *leaving*, and *members* denote the set of hosts that are idle, joining, leaving, and members of the IP multicast group, respectively. Each derived variable $\mathit{sources}(r)$, for $r \in R$, consists of the IP multicast transmission sources that the router r is aware of; that is, the sources for which the router r maintains upstream link state.

We presume that the state of LMS-IP satisfies the following constraints:

1. The set of IP multicast tree links maintained by each member h of the IP multicast group includes exactly two links: the link $\{h,h\}$ which abstractly models the connection between the IP multicast process and its client (*i.e.*, reliable multicast) process on h and a link $\{h,r\}$, for some $r \in \mathit{routers}$, which corresponds to a link connecting h to its gateway router r. We refer to the transmission queue corresponding to the link $\{h,h\}$ as the delivery queue.
2. The set of IP multicast links maintained by each host h' that is not a member of the IP multicast group is empty.

Figure 7.10 The LMS-IP automaton — Variables

Variables:
$now \in \mathbb{R}^{\geq 0}$, initially $now = 0$
$status(h) \in IPmcast\text{-}Status$, for all $h \in H$, initially $status(h) = \mathtt{idle}$, for all $h \in H$
$routers \subseteq R$, initially $routers = \emptyset$
$links(n) \subseteq L_n$, for all $n \in N$, initially $links(n) = \emptyset$, for all $n \in N$
$upstream\text{-}link(r, s) \in L_r \cup \{\perp\}$, for all $r \in R, s \in H$, initially $upstream\text{-}link(r, s) = \perp$, for all $r \in R, s \in H$
$repl\text{-}state(r, s) \in \{L_r \times \mathbb{R}^{\geq 0} \cup \{\infty\} \times \mathbb{R}^{\geq 0}\} \cup \{\perp\}$, for all $r \in R, s \in H$,
initially $repl\text{-}state(r, s) = \perp$, for all $r \in R, s \in H$
$repl\text{-}state(r, s) = \langle repl\text{-}link(r, s), repl\text{-}cost(r, s), repl\text{-}timeout(r, s)\rangle$, for all $r \in R, s \in H$
$upkts \subseteq P_{\text{IPUCAST-CLIENT}}$, initially $upkts = \emptyset$
$mqueue(n, l) : QueueOf(P_{\text{IPMCAST-CLIENT}})$, for all $n \in N, l \in L_n$, initially $mqueue(n, l) = \emptyset$, for all $n \in N, l \in L_n$
Derived Variables:
$up = \{h \in H \mid status(h) \neq \mathtt{crashed}\}$
$idle = \{h \in H \mid status(h) = \mathtt{idle}\}$
$joining = \{h \in H \mid status(h) = \mathtt{joining}\}$
$leaving = \{h \in H \mid status(h) = \mathtt{leaving}\}$
$members = \{h \in H \mid status(h) = \mathtt{member}\}$
for all $r \in R$, $sources(r) = \{s \in H \mid upstream\text{-}link(r, s) \neq \perp\}$
State Constraints:
$\forall\, h \in members, links(h) = \{\{h, h\}, \{h, r\}\}$, for some $r \in routers$
$\forall\, h \in H \backslash members, links(h) = \emptyset$
$\forall\, r \in R \backslash routers, links(r) = \emptyset$
$\forall\, r \in R \backslash routers, s \in H, upstream\text{-}link(r, s) = \perp \wedge repl\text{-}state(r, s) = \perp$
$\forall\, n, n' \in members \cup routers, \{n, n'\} \in links(n) \Leftrightarrow \{n, n'\} \in links(n')$
The nodes $members \cup routers$ and the links $\cup_{r \in routers} links(r)$ form a spanning tree of $members \cup routers$ with $members$ as the set of leaf nodes.

3. The set of IP multicast links maintained by each router r that is not part of the IP multicast tree is empty.
4. For each router r that is not part of the IP multicast tree, the upstream link and the replier state is undefined.
5. Any two nodes n, n' that are either members of the IP multicast group, or routers that are part of the IP multicast tree, are mutually aware of an IP multicast link connecting them.
6. The IP multicast tree nodes (*i.e.*, the members of the IP multicast group and the routers that are part of the IP multicast tree) and their IP multicast links form a spanning tree.

Actions

The input action $\mathtt{crash}_h$ models the crashing of the host h. The $\mathtt{crash}_h$ action sets the variable $status(h)$ to $\mathtt{crashed}$, thus recording the fact that the host h has crashed. Moreover, $\mathtt{crash}_h$ reinitializes the links and flushes the IP multicast transmission queues of h. By reinitializing the links of h, we ensure that no other packets are propagated to h. By flushing the IP multicast transmission queues of h, we ensure that none of the packets in the IP multicast transmission queues of h get propagated after h has crashed.

The input action $\mathtt{mjoin}_h$ models the request of the client at h to join the IP multicast group. The $\mathtt{mjoin}_h$ action is effective only while the host is idle with respect to the IP multicast group. When effective, the $\mathtt{mjoin}_h$ action sets the $status(h)$ variable to $\mathtt{joining}$, thus recording the fact that the host h has initiated the process of joining the IP multicast group. If the client is either a member of or in the process of joining the IP multicast group, then the $\mathtt{mjoin}_h$ action is superfluous. If the client is already in the process of leaving the group, then the $\mathtt{mjoin}_h$ action is discarded so as to allow the process of leaving the IP multicast group to complete.

The output action $\mathtt{mjoin\text{-}ack}_h$ acknowledges the join request of the client at h. The $\mathtt{mjoin\text{-}ack}_h$ action is enabled only when the host is in the process of joining the IP multicast group. The

Figure 7.11 The LMS-IP automaton — Discrete Transitions

Discrete Transitions:

input $\texttt{crash}_h$
eff $status(h) :=$ `crashed`
  $\backslash\backslash$ `Reinitialize the set of links of` h
  $links(h) := \emptyset$
  $\backslash\backslash$ `Flush the queues of` h
  foreach $l \in L_h$ **do:** $mqueue(h,l) := \emptyset$

input $\texttt{mjoin}_h$
eff **if** $h \in idle$ **then** $status(h) :=$ `joining`

output $\texttt{mjoin-ack}_h$
pre $h \in joining$
eff $status(h) :=$ `member`
  $\backslash\backslash$ `Extend IP multicast tree to include` h
  $2b\text{-}added :\subseteq R\backslash routers$
  $routers \cup= 2b\text{-}added$
  foreach $n \in routers \cup members$ **do:**
    $links(n) :\subseteq \{\{n,n'\} \in L \mid n' \in routers \cup members\}$
    foreach $l \in L_n\backslash links(n)$ **do:** $mqueue(n,l) := \emptyset$
  such that IP multicast tree constraint is satisfied.

input $\texttt{mleave}_h$
eff **if** $h \in joining \cup members$ **then**
    $status(h) :=$ `leaving`
    $\backslash\backslash$ `Choose the link to the gateway router of` h
    choose $l \in L_h\backslash\{\{h,h\}\}$
    $\backslash\backslash$ `Reinitialize the set of links of` h
    $links(h) := \emptyset$
    $\backslash\backslash$ `Flush the queues of` h
    foreach $l \in L_h$ **do:** $mqueue(h,l) := \emptyset$
    $\backslash\backslash$ `Send PRUNE pkt to gateway router of` h
    $p := comp\text{-}prune\text{-}pkt(h)$
    $enqueue(p, mqueue(h,l))$

output $\texttt{mleave-ack}_h$
pre $h \in leaving \wedge (\forall\, l \in L_h, mqueue(h,l) = \emptyset)$
eff $status(h) :=$ `idle`

input $\texttt{msend}_h(p)$
eff **if** $h \in up$ **then**
    foreach $l \in links(h)\backslash\{\{h,h\}\}$ **do:**
      $enqueue(p, mqueue(h,l))$

output $\texttt{mrecv}_h(p)$
pre $p = head(mqueue(h,\{h,h\}))$
eff $\backslash\backslash$ `Dequeue` p `from delivery queue at` h
  $dequeue(mqueue(h,\{h,h\}))$

output $\texttt{mdrop}_{nl}(p)$
choose $n' \in N$ **such that** $l = \{n,n'\}$
pre $n \neq n' \wedge n' \in members \cup routers$
  $\wedge l \in links(n') \wedge p = head(mqueue(n,l))$
eff $\backslash\backslash$ `Dequeue` p `from queue at` n
  $dequeue(mqueue(n,l))$

input $\texttt{usend}_h(p)$
eff **if** $h \in up$ **then** $upkts \cup= \{p\}$

output $\texttt{urecv}_n(p)$
where $n \in H$
pre $n \in up \wedge n = dest(p) \wedge p \in upkts$
eff $upkts \backslash= \{p\}$

output $\texttt{urecv}_n(p)$
where $n \in R$
pre $n = dest(p) \wedge p \in upkts$
eff $upkts \backslash= \{p\}$
  $\backslash\backslash$ `Subcast` p `down turning-point link of` p
  $s := source(p)$
  if $tp\text{-}link(p) \in links(n)\backslash\{upstream\text{-}link(r,s)\}$ **then**
    $p' := comp\text{-}repl\text{-}pkt(p)$
    $enqueue(p', mqueue(n, tp\text{-}link(p)))$

output $\texttt{udrop}(p)$
pre $p \in upkts$
eff $upkts \backslash= \{p\}$

time-passage $\nu(t)$
pre $\forall\, h \in H, mqueue(h,\{h,h\}) = \emptyset$
eff $now := now + t$

$\texttt{mjoin-ack}_h$ action sets the $status(h)$ variable to `member`, thus recording the fact that the client at h has become a member of the IP multicast group. Moreover, it nondeterministically extends the IP multicast tree to include h. So as to simplify our model of the IP multicast communication service, we model this extension abstractly and atomically. In particular, the $\texttt{mjoin-ack}_h$ action instantaneously extends the IP multicast tree by adding an appropriate set of routers to the IP multicast tree and nondeterministically updating the state of each IP multicast group member and each IP multicast tree router so as to satisfy the IP multicast tree state constraint specified in Figure 7.10.

The input action $\texttt{mleave}_h$ models the request of the client at h to leave the IP multicast group. The $\texttt{mleave}_h$ action is effective only while the host is either a member of or in the process of joining the IP multicast group. When effective, the $\texttt{mleave}_h$ action sets the $status(h)$ variable to `leaving`, thus recording the fact that the host h has initiated the process of leaving the IP multicast group. Leave requests overrule join requests; that is, when an $\texttt{mleave}_h$ action is performed while the host h is in the process of joining the IP multicast group, its effects are to abort the process of joining and to initiate the process of leaving the IP multicast group. If the client is either idle with respect to or already in the process of leaving the IP multicast group, then the $\texttt{mleave}_h$ action is superfluous.

Moreover, the $\texttt{mleave}_h$ action reinitializes the IP multicast links of h and flushes the IP multicast packet queues of h. Finally, the $\texttt{mleave}_h$ action composes a prune packet and enqueues it on the IP multicast transmission queue of h leading to its former gateway router. This packet is intended to prune the IP multicast tree and to flush the replier state of any router that leads to h — since h has initiated the process of leaving the group, h can no longer function as a replier.

The output action $\texttt{mleave-ack}_h$ acknowledges the leave request of the client at h. The $\texttt{mleave-ack}_h$ action is enabled only when the host is in the process of leaving the IP multicast group and no packets are enqueued for transmission at any of the IP multicast transmission queues of h. This latter condition prevents the acknowledgment of a leave request prior to transmitting the aforementioned prune packet to the former gateway router of h and, thus, initiating the process of pruning the IP multicast tree and flushing stale replier state. The effects of the $\texttt{mleave-ack}_h$ action are to set the $status(h)$ variable to $\texttt{idle}$, thus recording the fact that the client at h has become idle with respect to the IP multicast group.

The input action $\texttt{msend}_h(p)$ models the IP multicast transmission of the packet p by the client at h. The $\texttt{msend}_h(p)$ action is effective only if the client is a member of the IP multicast group. Here, our model of the IP multicast service departs from our earlier models of the IP multicast service where host need not be members of the IP multicast group prior to sending packets to the IP multicast group. Requiring that a host be a member of the group off-loads the issue of extending the IP multicast tree to include h to the process of joining the IP multicast group. Our decision to model the IP multicast service in this fashion does not affect our modeling of the reliable multicast service since the reliable multicast processes send packets using the IP multicast service only while being members of the IP multicast group.

The effects of the $\texttt{msend}_h(p)$ action are to enqueue the packet p onto the IP multicast transmission queue of h leading to its gateway router. Recall that the set of IP multicast tree links of h includes only two links: the link $\{h, h\}$, which is used to deliver packets to the client of the IP multicast service at the host h, and a link $\{h, r\}$, for some $r \in routers$, which corresponds to the link connecting the host h to its gateway router r. When h is not a member of the IP multicast group, the set of IP multicast links of h is empty and, thus, $\texttt{msend}_h(p)$ doesn't affect the state of LMS-IP.

The output action $\texttt{mrecv}_h(p)$ models the delivery of the packet p to the IP multicast client at h. The action $\texttt{mrecv}_h(p)$ is enabled when p is at the front of the delivery queue at h, *i.e.*, $p = head(mqueue(h, \{h, h\}))$. The effects of $\texttt{mrecv}_h(p)$ are to remove p from the delivery queue at h.

The output action $\texttt{mdrop}_{nl}(p)$ models the unsuccessful transmission, *i.e.*, the loss, of the packet p from the node n along the link l. Letting $n' \in N$, such that $l = \{n, n'\}$, the action $\texttt{mdrop}_{nl}(p)$ is enabled when n and n' are different nodes, n' is either a router of the IP multicast tree or a member of the IP multicast group, the link l is an IP multicast tree link of n', and the packet p is at the front of the IP multicast transmission queue of h for l. The effects of $\texttt{mdrop}_{nl}(p)$ are to remove p from the IP multicast transmission queue of h for l.

The input action $\texttt{usend}_h(p)$ models the unicast transmission of the packet p by the client at h. The $\texttt{usend}_h(p)$ action is effective only when the client is operational. In such a case, the $\texttt{usend}_h(p)$ action adds p to the set of unicast packets $upkts$ whose delivery is pending.

The output action $\texttt{urecv}_n(p)$ models the delivery of the unicast packet p to the node n. When n is a host node, the occurrence of $\texttt{urecv}_n(p)$ models the delivery of the unicast packet p to the client at the host n. In particular, if n is a host node, is operational, and is the destination of p and p is a pending unicast packet, *i.e.*, $p \in upkts$, then the effects of $\texttt{urecv}_n(p)$ are to remove p from the set of pending unicast packets $upkts$.

When n is a router node, the occurrence of $\texttt{urecv}_n(p)$ models the delivery of the unicast packet

p to the router n. In this case, the router n is responsible for sub-casting the packet p down the turning-point link *tp-link*(p) of p. In particular, if *tp-link*(p) is an IP multicast link of n other than the upstream link of n for the source s to which p pertains, then $\texttt{urecv}_n(p)$ composes an IP multicast reply packet p' corresponding to p and enqueues p' onto the IP multicast transmission queue of n for *tp-link*(p).

The output action $\texttt{udrop}(p)$ models the loss of the unicast packet p. The $\texttt{udrop}(p)$ action is enabled when p is a unicast packet whose delivery is pending, *i.e.*, $p \in \mathit{upkts}$. The effects of $\texttt{udrop}(p)$ are to remove p from the set *upkts*.

The time-passage action $\nu(t)$, for $t \in \mathbb{R}^{\geq 0}$, models the passage of t time units. The action $\nu(t)$ is enabled while all host delivery queues are empty. Its effects are to increment the variable *now* by t time units.

Figures 7.12 and 7.13 specify the internal action $\texttt{mprop}_{nl}(p)$, which models the successful transmission of the packet p from the node n along the link l. The action $\texttt{mprop}_{nl}(p)$ is enabled only if l does not correspond to the delivery queue of n, *i.e.*, $l \neq \{n, n\}$, and p is at the front of the transmission queue of n pertaining to l. The effects of $\texttt{mprop}_{nl}(p)$ depend on the type of the link l and the type of the packet p. Irrespective however of the type of l and p, the action $\texttt{mprop}_{nl}(p)$ dequeues p from the transmission queue of n pertaining to l.

We first consider the case where the node n is a router and the link l connects n to a host h (Figure 7.12). In this case, if h is a member of the reliable multicast group, n is the gateway router of h, and p is either an original transmission, a request, a reply, or a replier cost solicitation packet, then $\texttt{mprop}_{nl}(p)$ enqueues p to the delivery queue of h, *i.e.*, the transmission queue of h pertaining to the link $\{h, h\}$.

Next, we consider the case where the node n in either a host or a router and the link l connects n to a router r. In this case, the effects of $\texttt{mprop}_{nl}(p)$ depend on the type of the packet p. First, consider the case where p is a `DATA` packet (Figure 7.12). If l is an IP multicast tree link of r, then the packet p is enqueued on all IP multicast transmission queues of r other than the one pertaining to l. Since p is a `DATA` packet, *i.e.*, an original transmission of s, $\texttt{mprop}_{nl}(p)$ records that l is the upstream link of r for s by assigning l to the state variable *upstream*(r, s). Moreover, if either r has not set its replier state for s, or the upstream link affords less replier cost than the current replier link pertaining to s, then $\texttt{mprop}_{nl}(p)$ sets the replier link of r for s to be the upstream link l. This is achieved by assigning the tuple $\langle l, c, \mathit{now} + \texttt{REPL-TIMEOUT}\rangle$ to the state variable *repl-state*(r, s), where c is the replier cost of the upstream link l. The replier cost c of l is equal to 0, if r is adjacent to s, *i.e.*, $n = s$, and equal to ∞, otherwise. By assigning a replier cost of ∞ to the upstream link l when r is not adjacent to the source host of p, we effectively give priority for becoming a replier link to downstream links — downstream links will presumably have finite replier costs. Conversely, when r is adjacent to the source s of p, we give priority to the upstream link l by assigning to it a replier cost of 0.

Second, consider the case where p is a `SOLICIT` packet (again, Figure 7.12). In this case, if the link l is an IP multicast tree link of r and, moreover, is the upstream link of r for s, then $\texttt{mprop}_{nl}(p)$ attempts to send a refresh packet upstream so as to advertise its replier cost upstream. If the replier state is stale, then $\texttt{mprop}_{nl}(p)$ reinitializes it. Similarly to above, if the router r is adjacent to the source s to which p pertains, then $\texttt{mprop}_{nl}(p)$ resets the replier link to the upstream link $\{r, s\}$ of r for s and the replier cost to 0. Otherwise, $\texttt{mprop}_{nl}(p)$ resets the replier to the upstream link $\{r, s\}$ of r for s with a replier cost of ∞ and propagates p on all downstream links.

Alternatively, if the replier state of r is current, then $\texttt{mprop}_{nl}(p)$ composes a refresh packet including the replier cost of r for s and enqueues it on the transmission queue of the upstream link of r for s. This refresh packet is the response to the replier cost solicitation packet p.

Figure 7.12 The LMS-IP automaton — Discrete Transitions, Cont'd

Discrete Transitions:

```
internal mprop_nl(p)
where l ∈ L_nH
choose h ∈ H where l = {n, h}
pre n ≠ h ∧ p = head(mqueue(n, l))
eff \\ Dequeue p for the queue at n
    dequeue(mqueue(n, l))
    \\ Propagate p to delivery queue at h
    if h ∈ members ∧ l ∈ links(h) then
      if type(p) ∈ {DATA, RQST, REPL, SOLICIT} then
        enqueue(p, mqueue(h, {h, h}))

internal mprop_nl(p)
where l ∈ L_nR ∧ type(p) = DATA
choose r ∈ R where l = {n, r}
pre n ≠ r ∧ p = head(mqueue(n, l))
eff \\ Dequeue p from the queue at n
    dequeue(mqueue(n, l))
    \\ Propagate p to router r
    if l ∈ links(r) then
      foreach l' ∈ links(r)\{l} do:
        enqueue(p, mqueue(r, l'))
      s := source(p)
      upstream(r, s) := l
      if n = s then c := 0 else c := ∞
      if repl-state(r, s) =⊥ ∨c < repl-cost(r, s) then
        repl-state(r, s) := ⟨l, c, now + REPL-TIMEOUT⟩

internal mprop_nl(p)
where l ∈ L_nR ∧ type(p) = SOLICIT
choose r ∈ R where l = {n, r}
pre n ≠ r ∧ p = head(mqueue(n, l))
eff \\ Dequeue p from the queue at n
    dequeue(mqueue(n, l))
    s := source(p)
    \\ Propagate p to router r
    if l ∈ links(r) ∧ upstream-link(r, s) ≠⊥
      ∧l = upstream-link(r, s)
    then
      if repl-timeout(r, s) < now then
        if upstream-link(r, s) = {r, s} then
          \\ Reset replier state
          repl-state(r, s) :=
            ⟨upstream-link(r, s), 0, now + REPL-TIMEOUT⟩
        else
          \\ Flush replier state
          repl-state(r, s) :=
            ⟨upstream-link(r, s), ∞, now + REPL-TIMEOUT⟩
          \\ Propagate p downstream
          foreach l' ∈ links(r)\{upstream-link(r, s)} do:
            enqueue(p, mqueue(r, l'))
      else
        \\ Send REFRESH pkt upstream
        p' := comp-refresh-pkt(r, s, repl-cost(r, s))
        enqueue(p', mqueue(r, upstream-link(r, s)))

internal mprop_nl(p)
where l ∈ L_nR ∧ type(p) = RQST
choose r ∈ R where l = {n, r}
pre n ≠ r ∧ p = head(mqueue(n, l))
eff \\ Dequeue p from the queue at n
    dequeue(mqueue(n, l))
    s := source(p)
    \\ Propagate p to router r
    if l ∈ links(r) ∧ upstream-link(r, s) ≠⊥ then
      \\ Handle stale replier state
      if repl-timeout(r, s) < now then
        if upstream-link(r, s) = {r, s} then
          \\ Reset replier state
          repl-state(r, s) :=
            ⟨upstream-link(r, s), 0, now + REPL-TIMEOUT⟩
        else
          \\ Flush replier state
          repl-state(r, s) :=
            ⟨upstream-link(r, s), ∞, now + REPL-TIMEOUT⟩
          \\ Send FLUSH control pkt upstream
          p' := comp-flush-pkt(r, s)
          enqueue(p', mqueue(r, upstream-link(r, s)))
          \\ Solicit downstream replier costs
          p' := comp-solicit-pkt(r, s)
          foreach l' ∈ links(r)\{upstream-link(r, s)} do:
            enqueue(p', mqueue(r, l'))
        \\ Propagate p upstream
        enqueue(p, mqueue(r, upstream-link(r, s)))
      else
        if l = repl-link(r, s) then
          \\ Propagate p upstream
          enqueue(p, mqueue(r, upstream-link(r, s)))
        else
          if l ≠ upstream-link(r, s) then
            tp-router(p) := r; tp-link(p) := l
          \\ Propagate p down replier link
          enqueue(p, mqueue(r, repl-link(r, s)))

internal mprop_nl(p)
where l ∈ L_nR ∧ type(p) = REPL
choose r ∈ R where l = {n, r}
pre n ≠ r ∧ p = head(mqueue(n, l))
eff \\ Dequeue p from the queue at n
    dequeue(mqueue(n, l))
    s := source(p)
    \\ Propagate p to router r
    if l ∈ links(r) ∧ upstream-link(r, s) ≠⊥ then
      if l = upstream-link(r, s) then
        foreach l' ∈ links(r)\{l} do:
          enqueue(p, mqueue(r, l'))
```

Third, consider the case where p is a `RQST` packet (again, Figure 7.12). If l is an IP multicast tree link of r and the upstream link of r for s is set, then $\texttt{mprop}_{nl}(p)$ attempts to appropriately forward p. If the replier state is stale and the router r is adjacent to the source s of the packet being requested, then $\texttt{mprop}_{nl}(p)$ resets the replier state of r pertaining to s to the tuple $\langle \mathit{upstream}(r, s), 0, \mathit{now} + \texttt{REPL-TIMEOUT}\rangle$. If the replier state is stale and the router r is not adjacent to s, then $\texttt{mprop}_{nl}(p)$ i) flushes the replier state of r for s by setting it to the tuple $\langle \mathit{upstream}(r, s), \infty, \mathit{now} + \texttt{REPL-TIMEOUT}\rangle$, ii) sends notice upstream that its replier state has been flushed, and iii) solicits replier costs from all downstream links of r with respect to s. Once the replier state has been reset, $\texttt{mprop}_{nl}(p)$ forwards the request packet p on the upstream link of r for s.

If the replier state is not stale and l is the replier link of r for s, then $\texttt{mprop}_{nl}(p)$ forwards the request packet p on the upstream link of r for s. If the replier state is not stale and l is not the replier link of r for s, then $\texttt{mprop}_{nl}(p)$ forwards the request packet p down the replier link of r for s. In this case, if l is not the upstream link of r for s, then, prior to forwarding p, $\texttt{mprop}_{nl}(p)$ sets the turning point router and link fields of p to r and l, respectively — since p is received from a downstream link of r for s and forwarded on the replier link of r for s, this constitutes the turning point of p.

Fourth, consider the case where p is a `REPL` packet (again, Figure 7.12). In this case, l is an IP multicast tree link of r, the upstream link of r for s is set, and l, in particular, is the upstream link of r for s, then $\texttt{mprop}_{nl}(p)$ enqueues the packet p on all transmission queues of r other than the one pertaining to l. Thus, reply packets are only forwarded downstream with respect to s.

Fifth, consider the case where p is a `REFRESH` packet (Figure 7.13). In this case, if l is an IP multicast link of r and the upstream link of r for s is set, then $\texttt{mprop}_{nl}(p)$ attempts to appropriately forward the refresh packet p. The $\texttt{mprop}_{nl}(p)$ action determines whether it should update the replier state of r for s, updates it according to the information contained in p, and propagates p on the upstream link of r for s. The replier state is updated and p is propagated upstream when either: i) the node n is the source s, ii) the link l is the replier link of r for s, or iii) the replier link of r for s does not connect r to s, the link l is not the upstream link of r for s, and the replier cost advertised by the refresh packet p is less than the current replier cost of r for s. In the first scenario, the refresh packet simply refreshes the replier state of r to be the link leading to the source s. In the second scenario, the refresh packet is advertising a new replier cost for the current replier link. Thus, $\texttt{mprop}_{nl}(p)$ refreshes the replier state by setting the replier cost to the cost advertised by p irrespective of whether its is lower than the current replier cost. The third scenario corresponds to the case where the refresh packet p is advertising a lower replier cost from a link that is neither the replier nor the upstream link of r for s.

Sixth, consider the case where p is a `FLUSH` packet (again, Figure 7.13). In this case, if l is an IP multicast link of r, the upstream link of r for s is set, and l, in particular, is the replier link of r for s, then $\texttt{mprop}_{nl}(p)$ appropriately flushes the replier state of r for s. If the router r is adjacent to the source s to which the flush packet p pertains, then the replier link is reset to the upstream link $\{r, s\}$ of r for s and the replier cost is set to 0. Otherwise, the replier link is reset to the upstream link $\{r, s\}$ of r for s and the replier cost is set to ∞. Moreover, $\texttt{mprop}_{nl}(p)$ propagates p on the upstream link of r for s so as to alert the ancestors of r with respect to s to the fact that r is no longer a valid replier since it has just flushed its replier state. Finally, $\texttt{mprop}_{nl}(p)$ composes a replier cost solicitation packet and forwards it on all links of r except the upstream link of r for s and the ex-replier link l. This packet solicits replier costs from all candidate downstream links of r for s.

Finally, consider the case where p is a `PRUNE` packet (again, Figure 7.13). In this case, if l is an IP multicast link of r and the upstream link of r for s is set, then $\texttt{mprop}_{nl}(p)$ attempts to appropriately

Figure 7.13 The LMS-IP automaton — Discrete Transitions, Cont'd

Discrete Transitions:

```
internal mprop_nl(p)
where l ∈ L_nR ∧ type(p) = REFRESH
choose r ∈ R where l = {n, r}
pre n ≠ r ∧ p = head(mqueue(n, l))
eff  \\ Dequeue p from the queue at n
     dequeue(mqueue(n, l))
     s := source(p)
     \\ Propagate p to router r
     if l ∈ links(r) ∧ upstream-link(r, s) ≠⊥ then
       c := cost(p)
       if n = s ∨ l = repl-link(r, s)
          ∨(repl-link(r, s) ≠ {r, s}
            ∧l ≠ upstream-link(r, s) ∧ c < repl-cost(r, s))
       then
         repl-state(r, s) := ⟨l, c, now + REPL-TIMEOUT⟩
         \\ Propagate p upstream
         enqueue(p, mqueue(r, upstream-link(r, s)))

internal mprop_nl(p)
where l ∈ L_nR ∧ type(p) = FLUSH
choose r ∈ R where l = {n, r}
pre n ≠ r ∧ p = head(mqueue(n, l))
eff  \\ Dequeue p from the queue at n
     dequeue(mqueue(n, l))
     s := source(p)
     \\ Propagate p to router r
     if l ∈ links(r) ∧ upstream-link(r, s) ≠⊥ then
       \\ Propagate p to router r
       if l = repl-link(r, s) then
         if upstream-link(r, s) = {r, s} then
           \\ Reset replier state
           repl-state(r, s) :=
             ⟨upstream-link(r, s), 0, now + REPL-TIMEOUT⟩
         else
           \\ Flush replier state
           repl-state(r, s) :=
             ⟨upstream-link(r, s), ∞, now + REPL-TIMEOUT⟩
           \\ Propagate p upstream
           enqueue(p, mqueue(r, upstream-link(r, s)))
           \\ Solicit downstream replier costs;
           \\ except from upstream and ex-replier link
           p' := comp-solicit-pkt(r, s)
           foreach l' ∈ links(r)\{upstream-link(r, s), l} do:
             enqueue(p', mqueue(r, l'))

internal mprop_nl(p)
where l ∈ L_nR ∧ type(p) = PRUNE
choose r ∈ R where l = {n, r}
pre n ≠ r ∧ p = head(mqueue(n, l))
eff  \\ Dequeue p from the queue at n
     dequeue(mqueue(n, l))
     s := source(p)
     \\ Propagate p to router r
     if l ∈ links(r) ∧ upstream-link(r, s) ≠⊥ then
       \\ Prune r if part of chain leading to n
       if |links(r)\{l}| ≤ 1 then
         \\ Flush the queues of r
         foreach l' ∈ L_r do: mqueue(r, l') := ∅
         \\ Propagate p upstream
         foreach l' ∈ links(r)\{l} do:
           enqueue(p, mqueue(r, l'))
         \\ Reset router's replier state
         foreach s ∈ sources(r) do:
           upstream-link(r, s) :=⊥; repl-state(r, s) :=⊥
         \\ Reinitialize links of r
         links(r) := ∅
         \\ Remove r from router set
         routers \= {r}
       else
         \\ Remove l from router's links
         links(r) \= {l}
         \\ Flush the l queue of r
         mqueue(r, l) := ∅
         \\ Reset replier state of r
         foreach s ∈ sources(r) do:
           if repl-link(r, s) = l then
             \\ Flush replier state
             repl-state(r, s) :=
               ⟨upstream-link(r, s), ∞, now + REPL-TIMEOUT⟩
             \\ Send FLUSH control pkt upstream
             p' := comp-flush-pkt(r, s)
             enqueue(p', mqueue(r, upstream-link(r, s)))
             \\ Solicit downstream replier costs
             p'' := comp-solicit-pkt(r, s)
             foreach
                 l' ∈ links(r)\{l, upstream-link(r, s)}
               do:
               enqueue(p'', mqueue(r, l'))
```

prune the IP multicast tree of the router r. The effects of $\mathtt{mprop}_{nl}(p)$, however, depend on whether r is part of a chain of routers whose sole purpose is to extend the IP multicast tree to include the node n. If r has only two IP multicast tree links (including l), then r is indeed part of such a chain. In this case, the action $\mathtt{mprop}_{nl}(p)$ prunes the router r from the IP multicast tree by flushing all IP multicast transmission queues of r and propagating p on all its IP multicast tree links other than l (since r is part of a chain, there is only one such link and this link is the upstream link of r for s). Moreover, $\mathtt{mprop}_{nl}(p)$ reinitializes the upstream link and replier state of r for all sources, reinitializes the links of r, and removes r from the set of routers *routers* that are part of the IP multicast tree.

If r is not part of a chain and has multiple IP multicast tree links other than l, then $\mathtt{mprop}_{nl}(p)$ removes the link l from the set of IP multicast tree links of r and flushes the IP multicast transmission queue of r for l. Moreover, if the replier link for any source $s \in sources(r)$ is l, then $\mathtt{mprop}_{nl}(p)$ flushes the replier state for s, composes and forwards a flush packet on the upstream link of r for s, and composes and forwards a replier cost solicitation packet on all links of r apart from l and the upstream link of r for s.

7.3 LMS Correctness

In this section, we state the correctness of our model of the LMS protocol against the reliable multicast service specification of Chapter 3.

As in the case of the SRM and CESRM protocols, our model of the LMS protocol involves the LMS processes at each host and the underlying IP multicast service; that is, the automaton $\prod_{h \in H} \text{LMS}_h \times \text{LMS-IP}$, where $\text{LMS}_h = \text{LMS-}\textsc{mem}_h \times \text{LMS-IP}\textsc{buff}_h \times \text{LMS-}\textsc{rec}_h$. We define the automaton LMS to be the composition $\prod_{h \in H} \text{LMS}_h \times \text{LMS-IP}$ after hiding all output actions that are not output actions of the specification $\text{RM}(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$; that is, $\text{LMS} = \mathit{hide}_\Phi(\prod_{h \in H} \text{LMS}_h \times \text{LMS-IP})$, with $\Phi = \mathit{out}(\prod_{h \in H} \text{LMS}_h \times \text{LMS-IP}) \backslash \mathit{out}(\text{RM}(\Delta))$. Furthermore, we let LMS_I and $\text{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, denote the implementation and the specification of the reliable multicast service each composed with all the client automata; that is, $\text{LMS}_I = \text{LMS} \times \textsc{rmClients}$ and $\text{RM}_S(\Delta) = \text{RM}(\Delta) \times \textsc{rmClients}$.

The correctness analyses of both SRM and CESRM in Sections 4.4.4 and 6.3, respectively, show that SRM_I and CESRM_I, respectively, are faithful implementations of $\text{RM}_S(\infty)$. However, the reliable multicast specification $\text{RM}_S(\infty)$ enforces no timeliness guarantee as to the delivery of the packets transmitted using the reliable multicast service. Thus, the correctness proofs of SRM_I and CESRM_I effectively state that both SRM_I and CESRM_I may deliver the appropriate packets to the appropriate members of the reliable multicast group.

The functionality that dictates which packets are delivered to each member of the reliable multicast group is identical in all reliable multicast protocol implementations SRM_I, CESRM_I, and LMS_I. Moreover, this functionality is independent of the functionality governing how losses are recovered in each of the protocols. Thus, we claim that the correctness proof of each of the protocols is practically identical, with minor proof modifications. For purposes of brevity, instead of repeating the correctness proof for LMS_I, we simply state it.

Lemma 7.1 *R is a timed forward simulation relation from LMS_I to $\text{RM}_S(\infty)$.*

Proof: The proof is similar to that of Lemmas 4.11 and 6.11. Once again, the proof involves introducing the history variables of Section 4.4.2, defining the relation R from LMS_I to $\text{RM}_S(\Delta)$, for any $\Delta \in \mathbb{R}^{\geq 0} \cup \infty$, similar to that of Definitions 4.1 and 6.1, and showing that R is a timed forward simulation relation from LMS_I to $\text{RM}_S(\infty)$. ❐

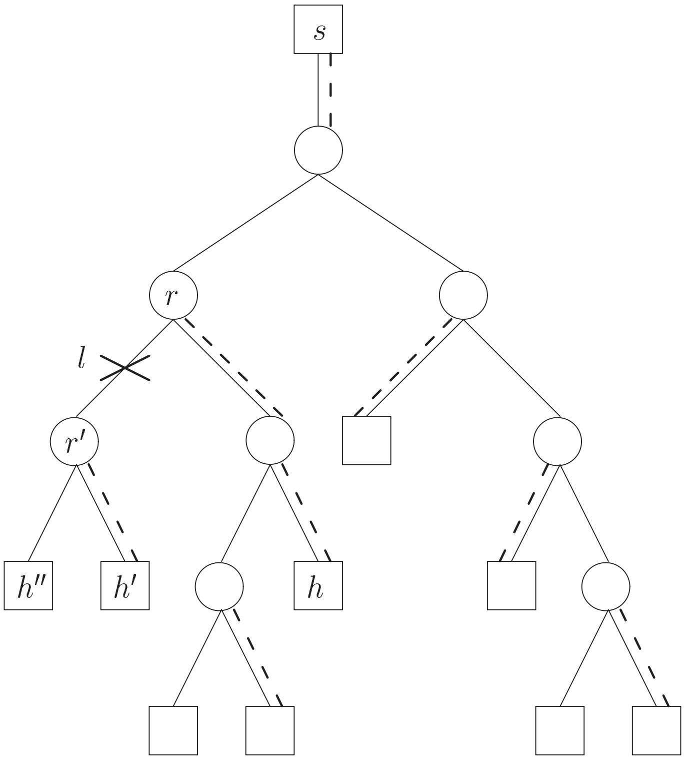
Theorem 7.2 $\text{LMS}_I \leq \text{RM}_S(\infty)$

Proof: Follows directly from Lemma 7.1. ❐

7.4 LMS Informal Timeliness Analysis

In this section, we informally comment on the timeliness of LMS. We begin by stating the worst-case recovery latency afforded by LMS when the recovery process proceeds smoothly; that is, when its not inhibited by, for example, unstable repliers, host crashes and host leaves. Then, we estimate the recovery latency of LMS in scenarios in which recovery packets are dropped, the replier state is

Figure 7.14 Example of LMS lossy transmission scenario. The dashed lines correspond to the replier links of the routers.



unstable, and hosts either crash or leave the reliable multicast group. We conclude by summarizing the conclusions of our simple recovery latency analysis of LMS and compare its performance to that of SRM and CESRM.

Throughout this section, we consider the transmission scenario depicted in Figure 7.14 involving the transmission of a packet p by s and the loss of p on the link l. Moreover, we let $\texttt{DET-BOUND} \in \mathbb{R}^{\geq 0}$ be an upper bound on the time it takes for reliable multicast group member to detect the loss of a proper packet, $\overline{d} \in \mathbb{R}^{\geq 0}$ be an upper bound on both the IP unicast and the IP multicast transmission latencies, and $\overline{RTT} = 2\,\overline{d}$ be an upper bound on the inter-host round-trip-time.

7.4.1 Ideal Recovery

In this section, we consider the ideal scenario in which the recovery of p proceeds smoothly. For simplicity, we presume that, throughout the recovery of p, the IP multicast topology and replier hierarchy remain stable (unchanged), no recovery packets are dropped, no repliers either crash or leave the reliable multicast group, and h' considers p to be a proper packet.

According to the replier state depicted in Figure 7.14, the designated requestor and replier that are responsible for recovering p are the hosts h' and h, respectively. Thus, upon detecting the loss of p, h' multicasts a request for p. This request is routed according to the replier state of the IP multicast tree to h. Upon receiving this request, h unicasts p to the router r and, in turn, r forwards p on l. The worst-case recovery latency incurred during such a recovery scenario is given by:

$$\texttt{DET-BOUND} + 3\overline{d}. \tag{7.1}$$

This worst-case recovery latency includes the time required for h' to detect the loss of p, the latency of multicasting the request from h' to h, the latency of unicasting the reply from h to r, and the latency of multicasting the reply from r to any of its descendant hosts.

7.4.2 Improper Packet Recovery

In our model of LMS, we presume that hosts take the initiative to detect and to recover from the loss of proper packets only. However, since hosts may act as designated requestors on behalf of other hosts, they may need to initiate and carry out the recovery of packets which they consider to be improper. Since the recovery of an improper packet is initiated by the designated requestor upon receiving a request for the given packet, the packet's recovery incurs some additional delays. We proceed by giving an example of such a scenario and estimating the recovery latency afforded in such cases.

In the transmission scenario depicted in Figure 7.14, suppose that h' considers p to be improper and h'' considers p to be proper. In this case, h' does not initiate the recovery of p until it receives a request for p from h''. Thus, the worst-case recovery latency of p is given by:

$$\texttt{DET-BOUND} + 4\overline{d}. \tag{7.2}$$

It follows that the recovery latency is increased by $\overline{d}$ time units. This additional delay corresponds to the time it may take for the request of h'' for p to reach h' and, in effect, instruct h' to initiate the recovery of p. It is plausible for such delays to accumulate when the reliable multicast group is large and there are several repliers that must successively be instructed to initiate the recovery of a packet.

Although this extraneous delay may seem artificial since it results from our treatment of proper and improper packets, we argue that, even in the case of LMS, it is preferable for hosts to take the initiative to detect and recover from the loss of proper packets only. By adopting this behavior, hosts initiate the recovery of improper packets on a need basis and, thus, avoid incurring the overhead of recovering from packets whose delivery is not required.

7.4.3 Lossy Recovery

In LMS, a particular pair of repliers is responsible for carrying out the recovery of a particular loss. Since the recovery of a packet relies on the successful transmission of the request and the reply for the given packet, a single loss may cause the failure of any single recovery attempt. Suppose that the transmission and the recovery of the packet p (whose transmission is depicted in Figure 7.14) incurs at most $k \in \mathbb{N}^+$ packet drops. Presuming that each attempt of h' to recover p fails solely due to packet drops, at most $k - 1$ recovery attempts of h' may fail — the transmission and the recovery of p incurs at most k packet drops and the first such drop corresponds to the loss of p on l.

Presuming that requests are periodically transmitted by h' every `RQST-TIMEOUT` time units (which is one of the schemes proposed by Papadopoulos *et al.* [32, 34]), the worst-case recovery latency of p is given by:

$$\texttt{DET-BOUND} + (k-1)\ \texttt{RQST-TIMEOUT} + 3\overline{d}. \tag{7.3}$$

Presuming that requests are transmitted by h' at exponentially increasing intervals, the worst-case recovery latency is given by:

$$\texttt{DET-BOUND} + (2^k - 1)\ \texttt{RQST-TIMEOUT} + 3\overline{d}. \tag{7.4}$$

Of course, this simplistic analysis presumes that the recovery attempts proceed smoothly; that is, that h' considers p to be a proper packet, that the replier state of all the routers involved in forwarding the requests of h' to h remains unchanged during the recovery of p, that h and h' neither crash nor leave the IP multicast group, and that all hosts that share the loss of p remain descendants of r' so as to receive h's retransmissions of p.

7.4.4 Unstable Replier State

In this section, we describe a subtle scenario that demonstrates that unstable (rapidly changing) replier state may delay the recovery of a packet. In particular, we demonstrate that if the replier state of r' changes rapidly, it is possible to temporarily trap the requests for p within the subtree rooted at r' and, thus, delay the recovery of p. We proceed by describing an example of such a scenario.

Suppose that h' does not consider p to be a proper packet and thus does not initiate the recovery of p until it receives a request from h''. Suppose that h'' detects the loss of p and multicasts a request for p. This request is routed by r' to h'. Moreover, suppose that immediately after forwarding the request from h'' to h', r' receives a refresh packet from h'' advertising a lower replier cost. Then, r' switches replier links and its new replier is now h''. Upon receiving the request of h'' for p, h' initiates the recovery of p by multicasting a request for p. However, upon receiving this request, r' forwards it to h'' since its replier link now leads to h''. Unless the replier state of r' remains stable for a sufficiently long enough time, r' may keep forwarding all requests for p downstream. The recovery of p may thus be delayed.

For our simple example, it may be highly unlikely for the replier state of r' to oscillate among its two downstream links fast enough to delay the recovery of p by a large amount of time. However, replier state may be more susceptible to such instability in large IP multicast trees where routers have a large number of downstream links and descendant hosts.

7.4.5 Replier Crashes/Leaves

As explained in Section 7.1.1, perhaps the most important weakness of LMS is its lack of robustness to scenarios in which hosts either crash or leave the reliable multicast group. According to the replier state depicted in the transmission scenario depicted in Figure 7.14, the designated requestor and the designated replier for the loss of p on l are the hosts h' and h, respectively. Thus, the recovery of p relies on h' multicasting a requests for p and on h replying to this request for p. However, if prior to carrying out the recovery of p either h or h' crashes or leaves the reliable multicast group, then the recovery of p may be substantially prolonged.

We first consider the scenario in which either h or h' crashes prior to carrying out the recovery of p. In particular, consider the scenario in which h' crashes prior to requesting the retransmission of p. Until the replier state of r' changes, any requests received by r' would be forwarded to h' and, thus, fail to recover p. However, it may take up to REPL-TIMEOUT time units for the replier state at r' to become stale and be refreshed. Presuming that the recovery of p proceeds smoothly thereafter, a rough upper bound on the recovery latency of p is given by:

$$\texttt{DET-BOUND} + \overline{d} + \texttt{REPL-TIMEOUT} + \texttt{RQST-TIMEOUT} + 3\overline{d}. \tag{7.5}$$

This recovery latency is exhibited by the following recovery scenario. The host h' detects the loss of p after `DET-BOUND` time units, sends a refresh packet, and then crashes. This refresh packet is received by r' after $\overline{d}$ time units, at which point it the replier state of r' for s is refreshed. The replier state of r' for s becomes stale `REPL-TIMEOUT` time units thereafter. Presuming that the recovery of p proceeds smoothly once the replier state of r' becomes stale and that h' transmits requests for p periodically with a period of `RQST-TIMEOUT` time units, it may take up to `RQST-TIMEOUT` time units for h'' to transmit another request for p. Once this request is transmitted, the recovery of p takes $3\overline{d}$ time units to complete — $\overline{d}$ time units for the request to be received by h, $\overline{d}$ time units for the unicast reply to be received by the turning point router r, and $\overline{d}$ time units for the reply subcast on the turning point link to be received by all the descendants of the turning point link.

The recovery of p may also be prolonged when either the designated requestor or the designated replier for the loss of p on l leaves the reliable multicast group. When a host issues a request to leave the IP multicast group, the IP multicast service uses a prune packet to prune the IP multicast tree and to flush the replier state pertaining to the given host. Provided it is not dropped, this prune packet appropriately prunes the branch of the IP multicast tree leading to the given host and flushes the replier state of any router whose replier link leads to the given host. Presuming that the recovery of the packet proceeds smoothly thereafter, a rough upper bound on the recovery latency of p is given by:

$$\texttt{DET-BOUND} + \overline{d} + \texttt{RQST-TIMEOUT} + 3\overline{d}. \tag{7.6}$$

This recovery latency is exhibited by the following recovery scenario. The host h' detects the loss of p after `DET-BOUND` time units and leaves the reliable multicast group prior to transmitting a request for p. The prune packet of h' flushes the replier state of r' within $\overline{d}$ time units. Thereafter, r' forwards all requests for p upstream. Subsequently, presuming that h'' periodically transmits requests for p with a period of `RQST-TIMEOUT` time units, it may take h'' up to `RQST-TIMEOUT` time units to transmit another request for p. Once this request is transmitted the recovery of p takes $3\overline{d}$ time units to complete.

Thus, when either the designated requestor or the designated replier leaves the IP multicast group, it is possible for some recovery attempts to fail. The recovery during such leaves may be prolonged further when the prune packet is dropped prior to flushing the replier state of the appropriate repliers. In fact, the recovery delay in such scenarios is equivalent to that of scenarios where the same designated requestor/replier crashes, given in (7.5). It follows that, in a highly lossy environment, even graceful leaves may substantially prolong packet recovery.

7.4.6 Comparison to SRM

Since SRM does not rely on particular members of the reliable multicast group to carry our the recovery of each loss, SRM's recovery scheme is not as susceptible as is LMS to either crashes or leaves. In particular, irrespective of whether hosts crash or leave the reliable multicast group while a packet is being recovered, a rough upper bound on the average recovery latency of a successful first-round recovery of SRM is given by:

$$\texttt{DET-BOUND} + (C_1 + C_2/2)\overline{d} + \overline{d} + (D_1 + D_2/2)\overline{d} + \overline{d}. \tag{7.7}$$

This recovery latency is afforded by the recovery scenario in which both the request and reply are scheduled for transmission at the midpoint of the request and reply scheduling intervals, respectively. This is a rough upper bound for two reasons. First, $\overline{d}$ is an upper bound on the inter-host transmission latencies and their estimates. Second, since multiple requests may be scheduled per loss, the request that instigates a packet's recovery is either sent or received with higher probability in the first half of the request interval. This is similarly true for replies.

Given the typical SRM scheduling parameter values used by Floyd *et al.* [12,13] of $C_1 = C_2 = 2$ and $D_1 = D_2 = 1$, the rough upper bound on the average recovery latency of a successful first-round recovery of SRM is $6.5\overline{d}$, or $3.25\overline{RTT}$.

We now roughly estimate the average recovery latency of LMS when either the designated requestor or the designated replier crashes. By presuming that the recovery proceeds smoothly once the replier state leading to the crashed host becomes stale and, potentially, gets updated, the average recovery latency of LMS is roughly:

$$\texttt{DET-BOUND} + \overline{d} + \texttt{REPL-TIMEOUT}/2 + \texttt{RQST-TIMEOUT}/2 + 3\overline{d}. \tag{7.8}$$

The amount of time `REPL-TIMEOUT` that a router's replier state remains current, prior to becoming stale, is on the order of several round-trip times. For our analysis in this section, we adopt a valuation of $3\ \overline{RTT}$ for the parameter `REPL-TIMEOUT`. Moreover, since hosts must allocate enough time for a particular request to instigate a packet's recovery prior to transmitting another request, it follows that $\texttt{RQST-TIMEOUT} > \overline{RTT}$. Choosing `RQST-TIMEOUT` close to the worst-case round-trip-time would result in recovering a packet sooner, but would potentially also introduce the transmission of extraneous (superfluous) requests. Thus, for our analysis in this section, we adopt a valuation of $2\ \overline{RTT}$ for the parameter `RQST-TIMEOUT`. Even for these modest valuations of the parameters `REPL-TIMEOUT` and `RQST-TIMEOUT`, the recovery latency afforded by LMS is roughly equal to $\texttt{DET-BOUND} + 9\ \overline{d} = \texttt{DET-BOUND} + 4.5\ \overline{RTT}$, which is worse than that afforded by SRM. Choosing higher values for the parameters `REPL-TIMEOUT` and `RQST-TIMEOUT` would further increase the recovery latency afforded by LMS.

SRM is also relatively robust to recovery packet drops. In SRM, when a packet suffers a loss, all the hosts that reside in the subtree of the IP multicast tree affected by the loss schedule requests for the given packet. Depending on how effective the suppression of requests is, one or more of these requests get multicast. Similarly, each host that receives such a request and has received the requested packet schedules the transmission of a reply for the given packet. Again, depending on how effective the suppression of replies is, one or more of these replies get multicast. Thus, even if SRM suffers losses during a packet's first recovery round, the packet may still be recovered by the first recovery round as a result of the transmission of duplicate requests and replies. SRM effectively trades off the additional overhead of transmitting duplicate requests and replies for robustness against recovery packet losses.

In contrast, LMS relies on a particular request and a particular reply to recover a particular loss. If either this request or this reply is dropped, then the particular recovery attempt fails. In effect, LMS's recovery scheme introduces specific points of failure and is thus less robust to losses in recovery packets.

Although SRM may perform comparably and, often even better, than LMS in highly dynamic and faulty environments, SRM's performance remains the same even when the topology is static and the recovery is lossless and fault-free; that is, a rough upper bound on SRM's average recovery latency afforded by successful first-round recoveries is $\texttt{DET-BOUND} + 3.25\overline{RTT}$. Conversely, in such cases, LMS affords a worst-case recovery latency of $\texttt{DET-BOUND} + 3\overline{d} = \texttt{DET-BOUND} + 1.5\overline{RTT}$, which is substantially better.

7.4.7 Comparison to CESRM

CESRM bridges the performance gap between SRM and LMS. When the topology is static, CESRM's caching-based expedited recovery scheme effectively establishes a hierarchy of repliers similar to the one established by LMS — CESRM's hierarchy is dictated by the locality in the IP multicast losses as opposed to replier state maintained by the IP multicast routers in LMS. When the topology is static, we expect CESRM's expedited recovery scheme to successfully recover a large percentage of the losses. Recall that successful expedited recoveries in CESRM incur a worst-case recovery latency of $\texttt{DET-BOUND} + 2\overline{d} = \texttt{DET-BOUND} + \overline{RTT}$. Thus, the worst-case recovery latency of all losses recovered by CESRM's expedited recovery scheme is comparable to (if not better than) than that afforded by a smooth recovery in LMS.

In highly dynamic and faulty environments, CESRM's caching-based expedited recovery scheme may fail to recover a large percentage of the losses. In such cases, CESRM falls back onto SRM's recovery scheme, which is highly robust to losses and failures.

In conclusion, CESRM provides an attractive alternative to LMS. CESRM's expedited recovery scheme promptly recovers from a large percentage of losses in static environments, while CESRM's fall-back recovery scheme, which mimics that of SRM, ensures CESRM's robust to highly dynamic and faulty environments.

7.4.8 Summary

Our simple analysis of the recovery latency afforded by LMS in a variety of scenarios has confirmed that, while LMS promptly recovers packets in static environments, it is not particularly robust to highly dynamic and faulty environments. This weakness can be mitigated by requiring routers to refresh their replier state more frequently, *i.e.*, reducing the value of the parameter `REPL-TIMEOUT`, and by having hosts transmit requests at a higher frequency, *i.e.*, reducing the value of the parameter `RQST-TIMEOUT`. Tuning LMS in this fashion, however, introduces additional overhead and, potentially, the transmission of extraneous requests for packets. When the parameters `REPL-TIMEOUT` and `RQST-TIMEOUT` of LMS must be chosen to reduce this overhead, then LMS may loose its performance advantage to SRM and, in particular, CESRM.

Chapter 8

Conclusions

In this thesis, we conduct an extensive case study on formally modeling, analyzing, and designing retransmission-based reliable multicast protocols. We begin by presenting an abstract model of the reliable multicast service that several reliable multicast protocols [12, 13, 32–34] strive to provide. This model precisely specifies i) what it means to be a member of the reliable multicast group, ii) which packets are guaranteed delivery to which members of the group, and iii) how long it takes for a packet to be delivered to the appropriate members of the reliable multicast group.

We proceed by modeling the Scalable Reliable Multicast (SRM) protocol [12, 13] and proving that this model is a faithful implementation of our reliable multicast service model. Under some timeliness assumptions and presuming a fixed number of per-recovery packet drops, we also show that our model of SRM guarantees the timely delivery of packets. This timeliness guarantee is shown by bounding the number of recovery rounds that may fail prior to recovering a packet. Our timeliness analysis of SRM reveals that the careless selection of SRM's scheduling parameters may introduce superfluous recovery traffic and may undermine the loss recovery process. This is an important observation that has, to date, been overlooked.

We then design, model, and analyze the Caching-Enhanced Scalable Reliable Multicast (CESRM) protocol. The design of CESRM is motivated by our observation that losses in IP multicast transmissions exhibit locality — the property that losses suffered by a receiver at proximate times often occur on the same link of the IP multicast tree. This observation stems from our analysis of the effectiveness of a simple caching-based expedited recovery scheme. In this scheme, receivers cache information about the recovery of recently recovered packets and use this information to estimate the links responsible for subsequent losses. The effectiveness of this scheme when applied to the IP multicast transmission traces of Yajnik *et al.* [41] reveals that, indeed, IP multicast losses exhibit substantial locality and that caching can be very effective.

CESRM augments SRM with a caching-based expedited recovery scheme that exploits packet loss locality in IP multicast transmissions by attempting to recover from losses in the manner in which recent losses were recovered. Since CESRM uses SRM's recovery scheme as a fall-back, when an expedited recovery fails to recover a loss, either due to additional losses or because the replier has also shared the loss, then the packet is recovered, in due time, through SRM's recovery scheme. Thus, CESRM inherits SRM's robustness to dynamic environments while, thanks to its caching-based expedited recovery scheme, drastically reducing SRM's average recovery latency in static environments. We show that CESRM is a faithful implementation of our reliable multicast service model. Furthermore, we analytically show that the worst-case recovery latency for successful expedited recoveries in CESRM is roughly 1 round-trip time (RTT) where as that of successful first-round recoveries in SRM (and, similarly, in CESRM) is 4 RTT (for typical scheduling parameter settings). Finally, we evaluate the performance of CESRM using trace-driven simulations. By using traces to drive our simulations, the simulated IP multicast transmissions exhibit the packet loss locality of actual IP multicast transmissions. Our simulations reveal that CESRM reduces the average recovery latency of SRM by roughly 50% and incurs less overhead in terms of recovery traffic.

Finally, we model the Light-weight Multicast Services (LMS) router-assisted reliable multicast protocol [32–34]. This protocol enhances the functionality of IP multicast routers so as to intelligently forward recovery traffic and achieve localized loss recovery. Again, we show that LMS is a faithful implementation of our reliable multicast service model. Furthermore, through careful reasoning, we show that, although LMS promptly recovers from packets in static membership and topology environments, it may not perform well in dynamic environments. Thus, our analyses of CESRM and LMS demonstrate that CESRM is a preferable reliable multicast protocol to both SRM and LMS; CESRM inherits SRM's robustness to dynamic environments and, thanks to its caching-based expedited recovery scheme, drastically reduces the average recovery latency of SRM in static environments.

8.1 Contributions

Our case study on formally modeling, analyzing, and designing reliable multicast protocols makes several contributions of distinct nature.

First, a byproduct of using a formal modeling and analysis approach are the formal specifications of both the reliable multicast service and the reliable multicast protocols. The specification of the reliable multicast service formalizes the notion of eventual delivery in the multicast setting. Moreover, by parameterizing our specification by a worst-case packet delivery bound, particular instantiations of our specification formalize the notion of a timely reliable multicast communication service. This timely specification may be used to prove both the correctness and the timeliness of particular reliable multicast protocols. Furthermore, the specifications of SRM, CESRM, and LMS precisely and completely describe the behavior of the respective protocols. In so doing, they also abstractly specify the underlying communication primitives each of the protocols uses. The abstract specification of these underlying communication primitives is also an important contribution. This is especially true in the case of LMS where the IP multicast communication service includes the behavior of the extended router functionality introduced by LMS.

Second, we demonstrate how simulation relations can be used to prove both protocol correctness and performance. The use of a simulation proof to show the correctness of an implementation with respect to a more abstract specification is standard practice. We use a similar approach to show that a particular reliable multicast protocol guarantees the timely delivery of multicast packets. This is achieved by instantiating our abstract model of the reliable multicast service with the worst-case packet delivery latency. Instantiating this model using a worst-case packet delivery latency of infinity specifies an eventual delivery guarantee. Thus, showing that a protocol implements such a timely reliable multicast service constitutes a timeliness claim about the protocol. We also demonstrate how to state and show conditional timeliness guarantees. This is particularly useful when a reliable multicast protocol guarantees timely delivery only under particular assumptions. Conditioning the simulation proof on these assumptions leads to a conditional performance claim about either a protocol's correctness or timeliness.

Third, our timeliness analysis of SRM reveals that choosing SRM's scheduling parameters arbitrarily may result in either superfluous recovery traffic or the failure of particular recovery rounds due to scheduling issues rather than losses. Our analysis gives rise to several constraints on SRM's scheduling parameters. These constraints constitute guidelines for choosing SRM's scheduling parameters so that scheduling issues do not induce superfluous traffic and recovery round failure.

To our knowledge, these constraints, or even similar ones, have not been expressed to date. This demonstrates that a formal approach to protocol modeling and analysis may often help in better understanding and, potentially, redesigning a protocol's behavior.

Fourth, we present a methodology for estimating the potential effectiveness of caching in multicast loss recovery. This methodology analyzes the performance of a caching-based loss location estimation scheme that estimates the links responsible for the losses suffered by each reliable multicast group member. By applying this methodology to the IP multicast transmission traces of Yajnik *et al.* [41] we observe that indeed packet loss locality in IP multicast transmissions can be exploited through a caching-based scheme very effectively.

Fifth, motivated by the expected effectiveness of caching in multicast loss recovery, we demonstrate such a caching scheme by designing the Caching-Enhanced Scalable Reliable Multicast (CESRM) protocol. CESRM employs a caching-based expedited recovery scheme that opportunistically attempts to recover from losses in the manner in which recent losses were recovered. By using SRM as a fall-back loss recovery scheme, CESRM may only reduce the average recovery latency incurred by SRM. In fact, trace-driven simulations reveal that, under realistic packet loss locality conditions, CESRM reduces the average recovery times of SRM by an average of roughly 50%, reduces the total number of retransmissions, and incurs comparable control packet traffic to that of SRM.

Sixth, CESRM also demonstrates the effectiveness of the system design paradigm in which an opportunistic and highly efficient scheme for performing a task is complemented by a more robust but less efficient scheme to handle the cases where the opportunistic scheme fails. The opportunistic scheme is usually based on a particular assumption about the behavior of the system at hand. When this assumption indeed holds, the opportunistic scheme succeeds in performing the task at hand. When the assumption does not hold, the task is performed by the fall-back scheme, albeit not as efficiently. In CESRM, this assumption corresponds to the assumption that packet losses in IP multicast transmissions exhibit locality and, thus, that the replier to which CESRM's expedited request is sent is indeed capable of retransmitting the given packet. This design paradigm is prevalent in many computer systems, *e.g.*, the traditional caching schemes used in processor memory designs. CESRM demonstrates that the same paradigm can very effectively be used in wide-area network protocols.

Finally, through careful reasoning, we expose several scenarios in which packet loss recovery in LMS may be prolonged and even inhibited due to changes in either the reliable multicast group membership or the replier hierarchy. With the proliferation of host mobility and wireless connections, a protocol's performance in dynamic environments becomes increasingly important. This suggests that future protocol designs should put substantial emphasis on their performance in highly dynamic and faulty environments. Moreover, it indicates that CESRM is a preferable reliable multicast protocol to both SRM and LMS; CESRM inherits SRM's robustness to dynamic environments and, thanks to its caching-based expedited recovery scheme, takes advantage of packet loss locality and affords good recovery latency in static environments.

8.2 Future Work

Similarly to other router-assisted reliable multicast protocols, LMS uses the enhanced IP multicast router functionality to introduce a recovery hierarchy. This hierarchy is very effective in achieving localized recovery and, thus, reducing recovery exposure. However, it may not fare well in highly dynamic environments where reliable multicast group members may either leave or crash unexpectedly. In such cases, the replier state maintained by the IP multicast routers becomes stale and must be updated. Such updates may prolong and even inhibit packet loss recovery.

CESRM's caching-based expedited recovery scheme establishes a similar hierarchy of repliers. This hierarchy is dictated by the packet loss locality exhibited by the losses suffered during the IP multicast transmission. Moreover, it evolves to match the changing reliable multicast group membership resulting from member leaves and crashes. Although this adaptation may not be immediate, the recovery of packets is undisturbed because when expedited recoveries fail, losses are recovered by SRM's recovery scheme, which is robust to failures and membership changes.

As future work, we propose designing a router-assisted CESRM protocol in which expedited recoveries are carried out locally. This can be achieved by augmenting IP multicast routers to: i) annotate reply packets with their *turning point routers*, *i.e.*, the routers at which reply packets are received from and forwarded on downstream links with respect to the source of the original packet, and ii) allow the subcasting of expedited reply packets downstream. This functionality is nearly identical to that of LMS [32, 34], with the exception that LMS requires routers to maintain replier state.

CESRM may exploit this extra router functionality as follows. Recovery tuples may be augmented to include the turning point router involved in the recoveries of the respective packets. By annotating each expedited request with the pertinent recovery tuple, including the pertinent turning point router, the resulting expedited reply may be unicast to the particular turning point router, which may subsequently subcast the reply downstream. Since IP multicast routers need not maintain replier state, such a scheme offers a lighter-weight local recovery scheme than that of LMS. Moreover, by employing SRM as a fall-back recovery scheme, this scheme is also robust to highly dynamic and faulty environments.

References

[1] Bolot, J.-C., Crépin, H., and Vega Garcia, A. Analysis of Audio Packet Loss in the Internet. In *Proc. 5th International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV'95)* (Durham, NH, Apr. 1995), vol. 1018 of *Lecture Notes in Computer Science*, pp. 154–165.

[2] Cáceres, R., Duffield, N. G., Horowitz, J., and Towsley, D. F. Multicast-Based Inference of Network-Internal Loss Characteristics. *IEEE Transactions on Information Theory 45*, 7 (Nov. 1999), 2462–2480.

[3] Cáceres, R., Duffield, N. G., Moon, S. B., and Towsley, D. Inference of Internal Loss Rates in the MBone. In *Proc. Global Telecommunications Conference (IEEE/GLOBECOM'99), Global Internet: Application and Technology* (Rio de Janeiro, Brazil, Dec. 1999), vol. 3, pp. 1853–1858.

[4] Cain, B., Deering, S., Fenner, B., Kouvelas, I., and Thyagarajan, A. Internet Group Management Protocol, Version 3. Internet-Draft, Internet Engineering Task Force, Nov. 2000. Proposed Update to RFC 2236.

[5] Carle, G., and Biersack, E. W. Survey on Error Recovery for IP-based Audio-Visual Multicast Applications. *IEEE Network Magazine 11*, 6 (Nov./Dec. 1997), 24–36.

[6] Casner, S. Frequently Asked Questions (FAQ) on the Multicast Backbone (MBONE). Technical Memo, Information Sciences Institute (ISI), Dec. 1994. URL=ftp://ftp.isi.edu/mbone/faq.txt.

[7] Chayat, R., and Rom, R. Applying Deterministic Feedback Suppression to Reliable Multicasting Protocols. In *Proc. 10th IEEE International Conference on Computer Communications and Networks (IEEE/ICCCN'01)* (Scottsdale, AZ, Oct. 2001), pp. 81–88.

[8] Deering, S. E. Host Extensions for IP Multicasting. Request for Comments (RFC 1112), Internet Engineering Task Force, Aug. 1989. Obsoletes RFCs 988 and 1054.

[9] Deering, S. E., and Cheriton, D. R. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems (TOCS) 8*, 2 (May 1990), 85–110.

[10] Diot, C., Dabbous, W., and Crowcroft, J. Multipoint Communication: a Survey of Protocols, Functions, and Mechanisms. *IEEE Journal on Selected Areas in Communications 15*, 3 (Apr. 1997), 277–290.

[11] Fenner, W. C. Internet Group Management Protocol, Version 2. Request for Comments (RFC 2236), Internet Engineering Task Force, Nov. 1997. Updates RFC 1112.

[12] Floyd, S., Jacobson, V., McCanne, S., Liu, C.-G., and Zhang, L. A Reliable Multicast Framework For Light-Weight Sessions And Application Level Framing. In *Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (ACM/SIGCOMM'95)* (Aug. 1995), ACM Press, New York, pp. 342–356.

[13] Floyd, S., Jacobson, V., McCanne, S., Liu, C.-G., and Zhang, L. A Reliable Multicast Framework For Light-Weight Sessions And Application Level Framing. *IEEE/ACM Transactions on Networking 5*, 6 (Dec. 1997), 784–803.

[14] Garland, S. J., Lynch, N. A., and Vaziri, M. IOA: A Language For Specifying, Programming, and Validating Distributed Systems. Draft of User and Reference Manual, Lab. for Computer Science, MIT, 1997.

[15] Handley, M. An Examination of MBone Performance. Research Report RR-97-450, University of Southern California (USC)/Information Sciences Institute (ISI), Jan. 1997.

[16] Holbrook, H. W., Singhal, S. K., and Cheriton, D. R. Log-Based Receiver-Reliable Multicast For Distributed Interactive Simulation. In *Proc. Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, ACM Special Interest Group on Data Communication (ACM/SIGCOMM'95)* (1995), ACM Press, New York, pp. 328–341.

[17] Levine, B. N., and Garcia-Luna-Aceves, J. J. A Comparison of Known Classes of Reliable Multicast Protocols. In *Proc. 4th IEEE International Conference on Network Protocols (IEEE/ICNP'96)* (Columbus, Ohio, Oct./Nov. 1996), pp. 112–121.

[18] Levine, B. N., and Garcia-Luna-Aceves, J. J. A Comparison of Reliable Multicast Protocols. *ACM Multimedia Systems Journal 6*, 5 (Aug. 1998), 334–348.

[19] Li, D., and Cheriton, D. R. OTERS (On-Tree Efficient Recovery using Subcasting): A Reliable Multicast Protocol. In *Proc. 6th IEEE International Conference on Network Protocols (IEEE/ICNP'98)* (Austin, Texas, 1998), pp. 237–245.

[20] Lin, J. C., and Paul, S. RMTP: Reliable Multicast Transport Protocol. In *Proc. 15th Annual Joint Conference of the IEEE Computer and Communications Societies, Networking the Next Generation (IEEE/INFOCOM'96)* (San Francisco, CA, Mar. 1996), vol. 3, pp. 1414–1424.

[21] Liu, C.-G., Estrin, D., Shenker, S., and Zhang, L. Local Error Recovery in SRM: Comparison of Two Approaches. Technical Report TR97-648, University of Southern California, 1997.

[22] Liu, C.-G., Estrin, D., Shenker, S., and Zhang, L. Local Error Recovery in SRM: Comparison of Two Approaches. *IEEE/ACM Transactions on Networking 6*, 6 (Dec. 1998), 686–692.

[23] Livadas, C., and Keidar, I. The Case for Exploiting Packet Loss Locality in Multicast Loss Recovery. Technical Report MIT/LCS/TR-867, Lab. for Computer Science, MIT, Cambridge, MA, Oct. 2002.

[24] Livadas, C., Keidar, I., and Lynch, N. A. Designing a Caching-Based Reliable Multicast Protocol. In *Proc. International Conference on Dependable Systems and Networks (IEEE/DSN'01), Fast Abstracts Supplement* (Göteborg, Sweden, July 2001), IEEE Computer Society, pp. B44–B45.

[25] Lynch, N. A. *Distributed Algorithms*. Morgan Kaufmann Publishers, Inc., San Francisco, CA, 1996.

[26] Lynch, N. A., and Vaandrager, F. Forward and Backward Simulations — Part II: Timing-Based Systems. Technical Memo MIT/LCS/TM-487.c, Lab. for Computer Science, MIT, Cambridge, MA, Apr. 1995.

[27] Lynch, N. A., and Vaandrager, F. Forward and Backward Simulations — Part II: Timing-Based Systems. *Information and Computation 128*, 1 (July 1996), 1–25. Preliminary version appeared as Ref. 26.

[28] Markopoulou, A., and Tobagi, F. A. Hierarchical Reliable Multicast: Performance Analysis and Placement of Proxies. In *Proc. 2nd International Workshop on Networked Group Communication (NGC'2000)* (Stanford University, Palo Alto, CA, Nov. 2000).

[29] Nonnenmacher, J., and Biersack, E. W. Performance Modelling of Reliable Multicast Transmission. In *Proc. 16th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE/INFOCOM'97)* (Kobe, Japan, Apr. 1997), vol. 2, pp. 471–479.

[30] Nonnenmacher, J., Biersack, E. W., and Towsley, D. Parity-Based Loss Recovery for Reliable Multicast Transmission. *IEEE/ACM Transactions on Networking 6*, 4 (Aug. 1998), 349–361.

[31] Obraczka, K. Multicast Transport Protocols: A Survey and Taxonomy. *IEEE Communications Magazine 36*, 1 (Jan. 1998), 94–102.

[32] Papadopoulos, C. *Error Control for Continuous Media and Large Scale Multicast Applications*. Doctor of Philosophy Thesis, Washington University in St. Louis, St. Louis, Missouri, 1999.

[33] Papadopoulos, C., and Laliotis, E. Incremental Deployment of a Router-Assisted Reliable Multicast Scheme. In *Proc. 2nd International Workshop on Networked Group Communication (NGC'2000)* (Stanford University, Palo Alto, CA, Nov. 2000).

[34] Papadopoulos, C., Parulkar, G., and Varghese, G. An Error Control Scheme For Large-Scale Multicast Applications. In *Proc. 17th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE/INFOCOM'98)* (San Francisco, CA, Mar. 1998), vol. 3, pp. 1188–1196.

[35] Paul, S., Sabnani, K. K., Lin, J. C., and Bhattacharyya, S. Reliable Multicast Transport Protocol (RMTP). *IEEE Journal on Selected Areas in Communications 15*, 3 (Apr. 1997), 407–421.

[36] Perkins, C., Hodson, O., and Hardman, V. A Survey of Packet-Loss Recovery Techniques for Streaming Audio. *IEEE Network Magazine 12*, 5 (Sept./Oct. 1998), 40–48.

[37] Semeria, C., and Maufer, T. Introduction to IP Multicast Routing. Internet-Draft (Informational), Internet Engineering Task Force, July 1997. Also, Technical Memo, Networking Solutions Center, 3Com Corporation.

[38] Sharma, P., Estrin, D., Floyd, S., and Zhang, L. Scalable Session Messages in SRM. Technical Report TR98-670, University of Southern California, Aug. 1997.

[39] Sharma, P., Estrin, D., Floyd, S., and Zhang, L. Scalable Session Messages in SRM using Self-Configuration. Technical Report TR98-670 (Updated Version), University of Southern California, Feb. 1998.

[40] Towsley, D., Kurose, J., and Pingali, S. A Comparison of Sender-Initiated and Receiver-Initiated Reliable Mutlicast Protocols. *IEEE Journal on Selected Areas in Communications 15*, 3 (Apr. 1997), 398–406.

[41] Yajnik, M., Kurose, J., and Towsley, D. Packet Loss Correlation in the MBone Multicast Network. In *Proc. Global Telocommunications Conference (IEEE/GLOBECOM'96), Communications: The Key to Global Prosperity* (London, England, Nov. 1996), pp. 94–99.

[42] Yajnik, M., Moon, S. B., Kurose, J., and Towsley, D. Measurement and Modeling of the Temporal Dependence in Packet Loss. In *Proc. 18th Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE/INFOCOM'99)* (New York, NY, Mar. 1999), vol. 1, pp. 345–352.