On Scalable and Efficient Distributed Failure Detectors

Indranil Gupta
Department of Computer Science
Cornell University
Ithaca, NY 14853, USA
gupta@cs.cornell.edu

Tushar D. Chandra
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY, USA
tushar@us.ibm.com

Germán S. Goldszmidt
IBM T.J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY, USA
gsg@us.ibm.com

ABSTRACT

Process groups in distributed applications and services rely on failure detectors to detect process failures *completely*, and as *quickly, accurately*, and *scalably* as possible, even in the face of unreliable message deliveries. In this paper, we look at quantifying the optimal scalability, in terms of network load, (in messages per second, with messages having a size limit) of distributed, complete failure detectors as a function of *application-specified* requirements. These requirements are 1) quick failure detection by *some* non-faulty process, and 2) accuracy of failure detection. We assume a crash-recovery (non-Byzantine) failure model, and a network model that is probabilistically unreliable (w.r.t. message deliveries and process failures). First, we characterize, under certain independence assumptions, the optimum worst-case network load imposed by any failure detector that achieves an application's requirements. We then discuss why traditional heartbeating schemes are inherently unscalable according to the optimal load. We also present a randomized, distributed, failure detector algorithm that imposes an equal expected load per group member. This protocol satisfies the application defined constraints of completeness and accuracy, and speed of detection on an average. It imposes a network load that differs from the optimal by a sub-optimality factor that is much lower than that for traditional distributed heartbeating schemes. Moreover, this sub-optimality factor does not vary with group size (for large groups).

Keywords

Distributed systems, Failure detectors, Efficiency, Accuracy, Scalability.

1. INTRODUCTION

Failure detectors are a central component in fault-tolerant distributed systems based on process groups running over unreliable, asynchronous networks eg., group membership protocols [3], supercomputers, computer clusters [13], etc. The ability of the failure detector to detect process failures *completely* and *efficiently*, in the presence of unreliable messaging as well as arbitrary process crashes and recoveries, can have a major impact on the performance of these systems. "Completeness" is the guarantee that the failure of a group member is eventually detected by every non-faulty group member. "Efficiency" means that failures are detected *quickly*, as well as *accurately* (i.e., without too many mistakes).

The first work to address these properties of failure detectors was by Chandra and Toueg [5]. The authors showed why it is impossible for a failure detector algorithm to deterministically achieve both completeness and accuracy over an asynchronous unreliable network. This result has lead to a flurry of theoretical research on other ways of classifying failure detectors, but more importantly, has served as a guide to designers of failure detector algorithms for real systems. For example, most distributed applications have opted to circumvent the impossibility result by relying on failure detector algorithms that guarantee completeness deterministically while achieving efficiency only probabilistically [1, 2, 4, 6, 7, 8, 14].

The recent emergence of applications for large scale distributed systems has created a need for failure detector algorithms that minimize the network load (in bytes per second, or equivalently, messages per second with a limit on maximum message size) used, as well as the load imposed on participating processes [7, 14]. Failure detectors for such settings thus seek to achieve good *scalability* in addition to efficiency, while still (deterministically) guaranteeing completeness.

Recently, Chen et al. [6] proposed a comprehensive set of metrics to measure the Quality of Service (QoS) of complete and efficient failure detectors. This paper presented three primary metrics to quantify the performance of a failure detector at *one* process detecting crash-recovery failures of a *single* other process over an unreliable network. The authors proposed failure detection time, and recurrence time and duration times of mistaken detection as the primary metrics for complete and efficient failure detectors. However, the paper neither deal with the optimal relation among these metrics, nor focussed on distributed or scalable failure detectors.

In this paper, we first address the question of quantifying the optimum worst-case network load (in messages per second, with a limit on messages sizes) needed by a complete distributed failure detector protocol to satisfy the efficiency requirements *as specified by the application*. We are concerned with distributed failure detectors working in a *group* of uniquely identifiable processes, which are subject to failures and recoveries, and communicate over an unreliable network. We deal with complete failure detectors that satisfy application-defined *efficiency* constraints of 1) (quickness) detection of any group member failure by *some* non-faulty member within a time bound, and 2) (accuracy) probability (within this time bound) of no other non-faulty member detecting a given non-faulty member as having failed.

The first (quickness) requirement merits further discussion. Many systems, such as multi-domain server farm clusters [7, 13] and virtual synchrony implementations [3] rely on a single or a few central computers to aggregate failure detection information from across the system. These computers are then responsible for disseminating that information across the entire system. In such systems, efficient detection of a failure depends on the time the failure is *first* detected by a non-faulty member. Even in the absence of a central server, notification of a failure is typically communicated, by the first member to detect it, to the entire group via a (possibly unreliable) broadcast [3]. Thus, although achieving completeness is important, efficient detection of a failure is more often related with the time to the *first* detection, by another non-faulty member, of the failure.

We derive the optimal worst-case network load (in messages per second, with a limit on maximum message size) imposed on the network by a complete failure detector satisfying the above application-defined constraints. We then discuss why the traditional and popular distributed heartbeating failure detection schemes (eg., [7, 14]) do not achieve these optimal scalability limits. Finally, we present a randomized distributed failure detector that can be configured to meet the application-defined constraints of completeness and accuracy, and expected speed of detection. With reasonable assumptions on the network unreliability (member and message failure rates of up to 15%), the worst-case network load imposed by this protocol has a sub-optimality factor that is much lower than that of traditional distributed heartbeat schemes. This sub-optimality factor does not depend on group size (in large groups), but only on the application-specified efficiency constraints and the network unreliability probabilities. Furthermore, the average load imposed per member is independent of the group size.

In arriving at these results, we will assume that message loss and member failures can each be characterized by probabilistic distributions, independent across messages and failures. While the practicality of these assumptions in real networks will probably be subject to criticism, these assumptions are necessary in order to take this first step towards quantifying and achieving scalable and efficient failure detectors. Besides, we believe that these independence assumptions are partially justified because of 1) the randomized nature of the new failure detector algorithm, and 2) the large temporal separation between protocol periods, typically $O(seconds)$ in practice (mitigating much of the correlation among message loss probability distributions).

The rest of the paper is organized as follows. Section 2 briefly summarizes previous work in this area. In Section 3, we formally describe the process group model assumed in this paper. Section 4 presents a discussion of how an application can *specify* efficiency requirements to a failure detector, and quantifies the optimal worst-case network load a failure detector must impose, in order to meet these requirements. Section 5 presents the new randomized failure detector protocol. We conclude in section 6.

2. PREVIOUS WORK

Chandra and Toueg [5] were the first to formally address the completeness and accuracy properties of failure detectors. Subsequent work has focused on different properties and classifications of failure detectors. This area of literature has treated failure detectors as *oracles* used to solve the Distributed Consensus/Agreement problem [9], which is unsolvable in the general asynchronous network model. These classifications of failure detectors are primarily based on the weakness of the model required to implement them, in order to solve the Distributed Consensus/Agreement problem [11].

Proposals for implementable failure detectors have sometimes assumed network models with weak unreliability semantics eg., timed-asynchronous model [8], quasi-synchronous model [2], partial synchrony model [12], etc. These proposals have treated failure detectors only as a tool to efficiently reach agreement, ignoring their efficiency from an application designer's viewpoint. For example, most failure detectors such as [12] provide *eventual* guarantees, while applications are typically concerned about *real* timing constraints.

In most real-life distributed systems, the failure detection service is implemented via variants of the "Heartbeat mechanism" [1, 2, 4, 6, 7, 8, 14], which have been popular as they guarantee the completeness property. However, all existing heartbeat approaches have shortcomings. Centralized heartbeat schemes create hot-spots that prevent them from scaling. Distributed heartbeat schemes offer different levels of accuracy and scalability depending on the exact heartbeat dissemination mechanism used, but we show that they are inherently not as efficient and scalable as claimed.

Probabilistic network models have been used to analyze heartbeat failure detectors in [4, 6], but only with a *single* process detecting failures of a *single* other process. [6] was the first paper to propose metrics for non-distributed heartbeat failure detectors in the crash-recovery model. These metrics were not inclusive of scalability concerns.

Our work differs from all this prior work in that it is the first to approach the design of failure detectors from a distributed application developer's viewpoint. We quantify the performance of a failure detector protocol as the network load it requires to impose on the network, in order to satisfy the application-defined constraints of completeness, and quick and accurate detection[1]. We also present an efficient and scalable distributed failure detector. The new failure detector incurs a constant expected load per process, thus

[1]We will state these application-defined requirements formally in Section 4.

avoiding the hot-spot problem of centralized heartbeating schemes.

3. MODEL

We consider a large *group* of n ($\gg 1$) members[2]. This set of potential group members is fixed *a priori*. Group members have unique identifiers. Each group member maintains a list, called a *view*, containing the identities of all other group members (faulty or otherwise). Our protocol specification and analysis assumes that this maximal group membership is always the same at all members, but our results can be extended to a model with dynamically changing membership and members with incomplete views, using methodologies similar to [10].

Members may suffer crash (non-Byzantine) failures, and recover subsequently. Unlike other papers on failure detectors (eg., [14]) that consider a member as faulty if they are perturbed and sleep for a time greater than some pre-specified duration, our notion of failure considers that a member is faulty if and only if it has really crashed. Perturbations at members that might lead to message losses are accounted for in the message loss rate p_{ml} (which we will define shortly).

Whenever a member recovers from a failure, it does so into a new *incarnation* that is distinguishable from all its earlier incarnations. At each member, an integer in non-volatile storage, that is incremented every time the member recovers, suffices to serve as the member's incarnation number. The members in our group model thus have crash-recovery semantics with incarnation numbers distinguishing different failures and recoveries. When a member M_i crashes (fails), it does so in its current incarnation (say its l'th incarnation). We say that such a failure is "detected" at exactly the first instant of time that some other non-faulty member detects either 1) failure of M_i in incarnation greater than or equal to l, or 2) recovery of M_i in an incarnation strictly greater than l.

We characterize the member failure probability by a parameter p_f. p_f is the probability that a random group member is faulty at a random time. Member crashes are assumed to be independent across members.

We assume no synchronization of clocks across group members. We only require that each individual member's clock drift rate (from some fixed clock rate) remains constant.

Members communicate using unicast (point-to-point) messaging on an asynchronous, fault-prone network. Since we are interested in characterizing the network bandwidth utilized, we will assume that maximal message sizes are a constant, containing at most a few bytes of data (assuming a bound on the size of message identifiers and headers, as is typical in IP packets).

Each message sent out on the network fails to be delivered at its recipient (due to network congestion, buffer overflow at the sender or receiver due to member perturbations, etc.) with probability $p_{ml} \in (0, 1)$. The worst-case message propagation delay (from sender to receiver through the network) for any delivered message is assumed to be so small compared to the application-specified detection time (typically $O($ several seconds $)$) that henceforth, for all practical purposes, we can assume that each message is either delivered immediately at the recipient with probability $(1 - p_{ml})$, or never reaches the recipient.[3]

This message loss distribution is also assumed to be independent across messages. Message delivery losses could, in fact, be correlated in such a network. However, if application-specified failure detection times are much larger than message propagation and congestion repair times in the network, messages exchanged by the failure detector will have considerable temporal separation. This reduces the correlation among the loss distributions of different messages. Randomized selection of message destinations in the new failure detector also weakens such message loss correlation.

In the rest of the paper, we use the shorthands q_f and q_{ml} instead of $(1 - p_f)$ and $(1 - p_{ml})$ respectively.

4. SCALABLE AND EFFICIENT FAILURE DETECTORS

The first formal characterization of the properties of failure detectors was offered in [5], which laid down the following properties for distributed failure detectors in process groups:

- **{Strong/Weak} Completeness**: crash-failure of any group member is detected by {all/some} non-faulty members[4],
- **Strong Accuracy**: no non-faulty group member [5] is declared as failed by any other non-faulty group member.

[5] also showed that a perfect failure detector i.e., one which satisfies both Strong Completeness and Strong Accuracy, is sufficient to solve distributed Consensus, but is impossible to implement in a fault-prone network.

Subsequent work on designing efficient failure detectors has attempted to trade off the Completeness and Accuracy properties in several ways. However, the completeness properties required by most distributed applications have lead to the popular use of failure detectors that guarantee Strong Completeness always, even if eventually [1, 2, 4, 5, 6, 7, 8, 14]. This of course means that such failure detectors cannot guarantee Strong Accuracy always, but only with a probability less than 1. For example, all-to-all (distributed)

[2] All of which are either processes, or servers, or network adaptors etc.

[3] This assumption is made for simplicity. In fact, the optimality results of section 4 hold if p_{ml} is assumed to be the probability of message delivery within $\mathcal{T}$ time units after its send. The randomized protocol of section 5 and its analysis can be extended to hold if p_{ml} is the probability of message delivery within a sixth of the protocol period.

[4] Recollect that in our model, since members recover with unique incarnations, detection of a member's failure or recovery also implies detection of failure of all it's previous incarnations.

[5] in its current incarnation

heartbeating schemes have been popular because they guarantee Strong Completeness (since a faulty member will stop sending heartbeats), while providing varying degrees of accuracy.

We have explained in Section 1 why in many distributed applications, although the failure of a group member must eventually be known to all non-faulty members, it is important to have the failure detected quickly by *some* non-faulty member (and not necessarily all non-faulty members). In other words, the quickness of failure detectors depends on the time from a member failure to *Weak Completeness* with respect to that failure, although Strong Completeness is a necessary property.

The requirements imposed by an application (or its designer) on a failure detector protocol can thus be formally specified and parameterized as follows:

1. COMPLETENESS: satisfy eventual Strong Completeness for member failures.
2. EFFICIENCY:
   (a) SPEED: every member failure is detected by *some* non-faulty group member within $\mathcal{T}$ time units after its occurrence ($\mathcal{T} \gg$ worst-case message round trip time).
   (b) ACCURACY: at any time instant, for every non-faulty member M_i not yet detected as failed, the probability that no other non-faulty group member will (mistakenly) detect M_i as faulty within the next $\mathcal{T}$ time units is at least $(1 - \mathcal{PM}(\mathcal{T}))$.

$\mathcal{T}$ and $\mathcal{PM}(\mathcal{T})$ are thus parameters specified by the application (or its designer). For example, an application designer might specify $\mathcal{T} = 3$ seconds, and $\mathcal{PM}(3 \text{ seconds}) = 10^{-8}$.

To measure the scalability of a failure detector algorithm, we use the *worst-case* network load it imposes - this is denoted as L. Since several messages may be transmitted simultaneously even from one group member, we define:

Definition 1. The *worst-case network load* L of a failure detector protocol is the maximum number of messages transmitted by any run of the protocol within any time interval of length $\mathcal{T}$, divided by $\mathcal{T}$.

We also require that the failure detector impose a uniform expected send and receive load at each member due to this traffic.

The goal of a near-optimal failure detector algorithm is thus to satisfy the above requirements (COMPLETENESS, EFFICIENCY) while guaranteeing:

- *Scale*: the worst-case network load L imposed by the algorithm is close to the optimal possible, with equal expected load per member.

That brings us to the question - what is the optimal worst-case network load, call it L^*, that is needed to satisfy the above application-defined requirements - COMPLETENESS, SPEED ($\mathcal{T}$), ACCURACY ($\mathcal{PM}(\mathcal{T})$) ? We are able to answer this question in the network model discussed earlier when the group size n is very large ($\gg 1$), and $\mathcal{PM}(\mathcal{T})$ is very small ($\ll p_{ml}$).

THEOREM 1. *Any distributed failure detector algorithm for a group of size n ($\gg 1$) that deterministically satisfies the* COMPLETENESS, SPEED, ACCURACY *requirements above, for given values of $\mathcal{T}$ and $\mathcal{PM}(\mathcal{T})$ ($\ll p_{ml}$), imposes a minimal worst-case network load (messages per time unit, as defined above) of:*

$$L^* = n \cdot \frac{log(\mathcal{PM}(\mathcal{T}))}{log(p_{ml}) \cdot \mathcal{T}}$$

Furthermore, there is a failure detector that achieves this minimal worst-case bound while satisfying the COMPLETENESS, SPEED, ACCURACY *requirements.*

L^* is thus the optimal worst-case network load required to satisfy the COMPLETENESS, SPEED, ACCURACY *requirements.*

PROOF. We prove the first part of the theorem by showing that each non-faulty group member could transmit up to $\frac{log(\mathcal{PM}(\mathcal{T}))}{log(p_{ml})}$ messages in a time interval of length $\mathcal{T}$.

Consider a group member M_i at a random point in time t. Let M_i not be detected as failed yet by any other group member, and stay non-faulty until at least time $t + \mathcal{T}$. Let m be the maximum number of messages sent by M_i, in the time interval $[t, t + \mathcal{T}]$, in any possible run of the failure detector protocol starting from time t.

Now, at time t, the event that "all messages sent by M_i in the time interval $[t, t+\mathcal{T}]$ are lost" happens with probability at least p_{ml}^m. Occurrence of this event entails that it is indistinguishable to the set of the rest of the non-faulty group members (i.e., members other than M_i) as to whether M_i is faulty or not. By the SPEED requirement, this event would then imply that M_i is detected as failed by some non-faulty group member between t and $t + \mathcal{T}$.

Thus, the probability that at time t, a given non-faulty member M_i that is not yet detected as faulty, is detected as failed by some other non-faulty group member within the next $\mathcal{T}$ time units, is at least p_{ml}^m. By the ACCURACY requirement, we have $p_{ml}^m \leq \mathcal{PM}(\mathcal{T})$, which implies that $m \geq \frac{log(\mathcal{PM}(\mathcal{T}))}{log(p_{ml})}$.

A failure detector that satisfies the COMPLETENESS, SPEED, ACCURACY requirements and meets the L^* bound works as follows. It uses a highly available, non-faulty server as a group leader[6]. Every other group member sends $[\frac{log(\mathcal{PM}(\mathcal{T}))}{log(p_{ml})}]$ "I am alive" messages to this server every $\mathcal{T}$ time units. The

[6]The set of central computers, that collect failure information and disseminate it to the system, can be designated as the server.

server declares a member as failed when it does not receive any "I am alive" message from it for $\mathcal{T}$ time units[7]. □

Corollary: The optimal bound of Theorem 1 applies to the crash-stop model as well.

Proof: By exactly the same arguments as in the proof of Theorem 1. □

Definition 2. The *sub-optimality factor* of a failure detector algorithm that imposes a worst-case network load L, while satisfying the COMPLETENESS and EFFICIENCY requirements, is defined as $\frac{L}{L^*}$.

In the traditional distributed Heartbeating failure detection algorithms, every group member periodically transmits a "heartbeat" message (with an incremented counter) to every other group member. A member M_i is declared as failed by a non-faulty member M_j when M_j does not receive heartbeats from M_i for some consecutive heartbeat periods (this duration being the detection time $\mathcal{T}$).

Distributed heartbeating schemes have been the most popular implementation of failure detectors because they guarantee COMPLETENESS - a failed member will not send any more heartbeat messages. However, the accuracy and scalability guarantees of heartbeating algorithms differ, depending entirely on the actual mechanism used to disseminate heartbeats.

In the simplest implementation, each member M_i transmits a few "I am alive" messages to each group member it knows of, every $\mathcal{T}$ time units. The worst-case number of messages transmitted by each member per unit time is $\theta(n)$, and the worst-case total network load L is $\theta(n^2)$. The sub-optimality factor (i.e., $\frac{L}{L^*}$) varies as $\theta(n)$, for any values of p_{ml}, p_f and $\mathcal{PM}(\mathcal{T})$.

The Gossip-style failure detection service, proposed by van Renesse et al. [14], uses a mechanism where every t_{gossip} time units, each member gossips a $\theta(n)$ list of the latest heartbeat counters (for all group members) to a few other randomly selected group members. The authors show that under this scheme, a new heartbeat count typically takes an average time of $\theta[log(n) \cdot t_{gossip}]$ to reach an arbitrary other group member. The SPEED requirement thus leads us to choose $t_{gossip} = \theta[\frac{\mathcal{T}}{log(n)}]$. The worst-case network load imposed by the Gossip-style heartbeat scheme is thus $\theta[\frac{n^2}{t_{gossip}}] = \theta[\frac{n^2 \cdot log(n)}{\mathcal{T}}]$. The sub-optimality factor varies as $\theta[n \cdot log(n)]$, for any values of p_{ml}, p_f and $\mathcal{PM}(\mathcal{T})$.

In fact, distributed heartbeating schemes do not meet the optimality bound of Theorem 1 because they inherently attempt to communicate a failure notification to *all* group members. As we have seen above, this is an overkill for systems that can rely on a centralized coordinated set of servers to disseminate failure information. These systems require only *some* other non-faulty member to detect a given failure.

Other heartbeating schemes, such as Centralized heartbeating (as discussed in the proof of Theorem 1) and heartbeating along a logical ring of group members [7], can be configured to meet the optimal load L^*, but have problems such as creating hot-spots (centralized heartbeating) or unpredictable failure detection times in the presence of multiple simultaneous faults at larger group sizes (heartbeating in a ring).

5. A RANDOMIZED DISTRIBUTED FAILURE DETECTOR PROTOCOL

In the preceding sections, we have characterized the optimal worst-case load imposed by a distributed failure detector that satisfies the COMPLETENESS, SPEED and ACCURACY requirements, for application specified values of $\mathcal{T}$ and $\mathcal{PM}(\mathcal{T})$ (Theorem 1). We have then studied why traditional heartbeating schemes are inherently not scalable.

In this section, we relax the SPEED condition to detect a failure within an *expected* (rather than exact, as before) time bound of $\mathcal{T}$ time units after the failure. We then present a randomized distributed failure detector algorithm that guarantees COMPLETENESS with probability 1, detection of any member failure within an expected time $\mathcal{T}$ from the failure, and an ACCURACY probability of $(1 - \mathcal{PM}(\mathcal{T}))$. The protocol imposes an equal expected load per group member, and a worst-case (and average case) network load L that differs from the optimal L^* of Theorem 1 by a sub-optimality factor (i.e., $\frac{L}{L^*}$) that is independent of group size n ($\gg 1$). In such large groups, at reasonable values of member and message delivery failure rates p_f and p_{ml}, this sub-optimality factor is much lower than the sub-optimality factors of the traditional distributed heartbeating schemes discussed in the previous section.

5.1 New Failure Detector Algorithm

The failure detector algorithm uses two parameters: protocol period T' (in time units) and integer k, which is the size of failure detection subgroups. We will show how the values of these parameters can be configured from the required values of $\mathcal{T}$ and $\mathcal{PM}(\mathcal{T})$, and the network parameters p_f, p_{ml}. Parameters T' and k are assumed to be known *a priori* at all group members. Note that this does not need clocks to be synchronized across members, but only requires each member to have a steady clock rate to be able to measure T'.

The algorithm is formally described in Figure 1. At each non-faulty member M_i, steps (1-3) are executed once every T' time units (which we call a protocol *period*), while steps (4,5,6) are executed whenever necessary. The data contained in each message is shown in parentheses after the message. If sequence numbers are allowed to wrap around, the maximal message size is bounded from above.

Figure 2 illustrates the protocol steps initiated by a member M_i, during one protocol period of length T' time units. At the start of this protocol period at M_i, a random member

[7] This implementation, which is essentially a centralized heartbeat mechanism, is undesirable as it requires a highly available server and has bad load balancing (does not satisfy the *Scale* property).

```
Integer pr; /* Local period number */

Every T' time units at M_i:

0.   pr := pr + 1
1.   Select random member M_j from view
     Send a ping(M_i, M_j, pr) message to M_j
     Wait for the worst-case message round-trip time for
         an ack(M_i, M_j, pr) message
2.   If have not received an ack(M_i, M_j, pr) message yet
         Select k members randomly from view
         Send each of them a ping-req(M_i, M_j, pr) message
         Wait for an ack(M_i, M_j, pr) message until
             the end of period pr
3.       If have not received an ack(M_i, M_j, pr) message yet
             Declare M_j as failed

Anytime at M_i:

4.   On receipt of a ping-req(M_m, M_j, pr) (M_j ≠ M_i)
         Send a ping(M_i, M_j, M_m, pr) message to M_j
         On receipt of an ack(M_i, M_j, M_m, pr) message from M_j
         Send an ack(M_m, M_j, pr) message to received to M_m

Anytime at M_i:

5.   On receipt of a ping(M_m, M_i, M_l, pr) message from
     member M_m
         Reply with an ack(M_m, M_i, M_l, pr) message to M_m

Anytime at M_i:

6.   On receipt of a ping(M_m, M_i, pr) message from member M_m
         Reply with an ack(M_m, M_i, pr) message to M_m
```

Figure 1: Protocol steps at a group member M_i. Data in each message is shown in parentheses after the message. Each message also contains the current incarnation number of the sender.

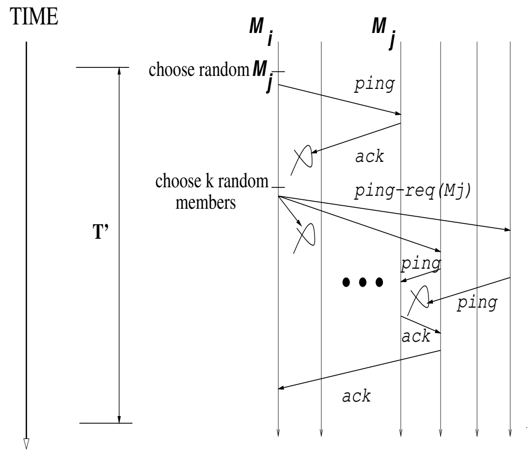


Figure 2: Example protocol period at M_i. This shows all the possible messages that a protocol period may initiate. Some message contents excluded for simplicity.

is selected, in this case M_j, and a ping message sent to it. If M_i does not receive a replying ack from M_j within some time-out (determined by the message round-trip time, which is $\ll \mathcal{T}$), it selects k members at random and sends to each a ping-req message. Each of the non-faulty members among these k which receives the ping-req message subsequently pings M_j and forwards the ack received from M_j, if any, back to M_i. In the example of Figure 2, one of the k members manages to complete this cycle of events as M_j is up, and M_i does not suspect M_j as faulty at the end of this protocol period.

In the above protocol, member M_i uses a randomly selected subgroup of k members to out-source ping-req messages, rather than sending out k repeat ping messages to the target M_j. The effect of using the randomly selected subgroup is to distribute the decision on failure detection across a subgroup of $(k+1)$ members. Although we do not analyze it in this paper, it can be shown that the new protocol's properties are preserved even in the presence of some degree of variation of message delivery loss probabilities across group members. Sending k repeat ping messages may not satisfy this property. Our analysis in Section 5.2 shows that the cost (in terms of sub-optimality factor of network load) of using a $(k+1)$-sized subgroup is not too significant.

5.2 Analysis

In this section, we calculate, for the above protocol, the expected detection time of a member failure, as well as the probability of an inaccurate detection of a non-faulty member by some other (at least one) non-faulty member. This will lead to calculation of the values of T' and k, for the above protocol, as a function of parameters specifying application-specified requirements and network unreliability, i.e., $\mathcal{T}$, $\mathcal{PM}(\mathcal{T})$, p_f, p_{ml}.

For any group member M_j, faulty or otherwise,

$$\begin{aligned} \texttt{Pr} \quad & [\text{at least one non-faulty member chooses to} \\ & \quad \text{ping } M_j \text{ (directly) in a time interval } T'] \\ = \quad & 1 - (1 - \frac{1}{n} \cdot q_f)^{n-1} \\ \simeq \quad & 1 - e^{-q_f} (\text{since } n \gg 1) \end{aligned}$$

Thus, the expected time between a failure of member M_j and its detection by some non-faulty member is

$$\mathtt{E}[\mathcal{T}] = T' \cdot \frac{1}{1 - e^{-q_f}} = T' \cdot \frac{e^{q_f}}{e^{q_f} - 1} \qquad (1)$$

This gives us a configurable value for T' as a function of $\mathcal{T}, p_f$.

Now, denote

$$C(p_f) = \frac{e^{q_f}}{e^{q_f} - 1}$$

At any given time instant, a non-faulty member M_j will be detected as faulty by another non-faulty member M_l within the next $\mathcal{T}$ time units if M_l chooses to ping M_j within the next $\mathcal{T}$ time units and does not receive any acks, directly or indirectly from transitive ping-req's, from M_j. Then, $\mathcal{PM}(\mathcal{T})$, the probability of inaccurate failure detection of member M_j within the next $\mathcal{T}$ time units, is simply the probability that there is at least one such member M_l in the group.

A random group member M_l is non-faulty with probability

q_f, and the probability of such a member choosing to ping M_j within a time interval $\mathcal{T}$ is $\frac{1}{n} \cdot C(p_f)$. Given this, the probability that such a M_l receives back no acks, direct or indirect, according to the protocol of section 5.1 equals

$$((1 - q_{ml}^2) \cdot (1 - q_f \cdot q_{ml}^4)^k)$$

Therefore,

$$\mathcal{PM}(\mathcal{T}) = 1 - [1 - \frac{q_f}{n} \cdot C(p_f) \cdot (1 - q_{ml}^2) \cdot (1 - q_f \cdot q_{ml}^4)^k]^{n-1}$$

$$\simeq \quad 1 - e^{-q_f \cdot (1-q_{ml}^2) \cdot (1-q_f \cdot q_{ml}^4)^k \cdot C(p_f)} \quad \text{(since } n \gg 1\text{)}$$

$$\simeq \quad q_f \cdot (1 - q_{ml}^2) \cdot (1 - q_f \cdot q_{ml}^4)^k \cdot C(p_f) \quad \text{(since } \mathcal{PM}(\mathcal{T}) \ll 1\text{)}$$

This gives us

$$k = \frac{log[\frac{\mathcal{PM}(\mathcal{T})}{(q_f \cdot (1-q_{ml}^2) \cdot \frac{e^{q_f}}{e^{q_f}-1})}]}{log(1 - q_f \cdot q_{ml}^4)} \tag{2}$$

Thus, the new randomized failure detector protocol can be configured using equations (1) and (2) to satisfy the SPEED and ACCURACY requirements with parameters $\mathrm{E}[\mathcal{T}]$, $\mathcal{PM}(\mathcal{T})$. Moreover, given a member M_j that has failed (and stays failed), every other non-faulty member M_i will eventually choose to ping M_j in some protocol period, and discover M_j as having failed. Hence,

THEOREM 2. *This randomized failure detector protocol:*
(a) satisfies eventual Strong Completeness, i.e., the COMPLETENESS *requirement,*
(b) can be configured via equations (1) and (2) to meet the requirements of (expected) SPEED, *and* ACCURACY, *and*
(c) has a uniform expected send/receive load at all group members.

PROOF. From the above discussion and equations (1), (2). □

Finally, we upper-bound the worst-case and expected network load (L, $\mathrm{E}[L]$ respectively) imposed by this failure detector protocol.

The worst-case network load occurs when, every T' time units, each member initiates steps (1-6) in the algorithm of Figure 1. Steps (1,6) involve at most 2 messages, while steps (2-5) involve at most 4 messages per ping-req target member. Therefore, the worst-case network load imposed by this protocol (in messages/time unit) is

$$L \quad = \quad n \cdot [2 + 4 \cdot k] \cdot \frac{1}{T'}$$

Then, from Theorem 1 and equations (1),(2),

$$\frac{L}{L^*} \quad = \quad [2 + 4 \cdot \frac{log[\frac{\mathcal{PM}(\mathcal{T})}{(q_f \cdot (1-q_{ml}^2) \cdot \frac{e^{q_f}}{e^{q_f}-1})}]}{log(1 - q_f \cdot q_{ml}^4)}] \times [\frac{e^{q_f}}{e^{q_f}-1} \cdot \frac{log(p_{ml})}{log(\mathcal{PM}(\mathcal{T}))}] \tag{3}$$

L thus differs from the optimal L^* by a factor that is independent of the group size n. Furthermore, (3) can be written as a linear function of $\frac{1}{-log(\mathcal{PM}(\mathcal{T}))}$ as:

$$\frac{L}{L^*} \quad = \quad g(p_f, p_{ml}) + \frac{1}{-log(\mathcal{PM}(\mathcal{T}))} \cdot f(p_f, p_{ml}) \tag{4a}$$

where $g(p_f, p_{ml})$ is:

$$[4 \cdot \frac{log(p_{ml})}{log(1 - q_f \cdot q_{ml}^4)} \cdot \frac{e^{q_f}}{e^{q_f}-1}] \tag{4b}$$

and $f(p_f, p_{ml})$ is:

$$[\{2 - 4 \cdot \frac{log(q_f \cdot (1 - q_{ml}^2) \cdot \frac{e^{q_f}}{e^{q_f}-1})}{log(1 - q_f \cdot q_{ml}^4)}\} \times (-log(p_{ml})) \cdot \frac{e^{q_f}}{e^{q_f}-1}] \tag{4c}$$

THEOREM 3. *The sub-optimality factor $\frac{L}{L^*}$ of the protocol of Figure 1, is independent of group size n ($\gg 1$). Furthermore,*

1. *if $f(p_f, p_{ml}) < 0$,*
   (a) $\frac{L}{L^*}$ is monotonically increasing with $-log(\mathcal{PM}(\mathcal{T}))$, and
   (b) As $\mathcal{PM}(\mathcal{T}) \to 0^+$, $\frac{L}{L^*} \to g(p_f, p_{ml})^-$
2. *if $f(p_f, p_{ml}) > 0$,*
   (a) $\frac{L}{L^*}$ is monotonically decreasing with $-log(\mathcal{PM}(\mathcal{T}))$, and
   (b) As $\mathcal{PM}(\mathcal{T}) \to 0^+$, $\frac{L}{L^*} \to g(p_f, p_{ml})^+$

PROOF. From equations (4a) through (4c). □

We next calculate the *average* network load imposed by the new failure detector algorithm. Every T' time units, each non-faulty member (numbering $(n \cdot q_f)$ on an average) executes steps (1-3), in the algorithm of Figure 1. Steps (1,6) involve at most 2 messages, while steps (2-5) (which are executed only if no ack is received from the target of the ping of step (1) - this happens with probability $(1 - q_f \cdot q_{ml}^2)$) involve at most 4 messages per non-faulty ping-req target member. Therefore, the average network load imposed by this protocol (in messages/time unit) is

$$\mathrm{E}[L] \quad \leq \quad n \cdot q_f \cdot [2 + (1 - q_f \cdot q_{ml}^2) \cdot 4 \cdot k] \cdot \frac{1}{T'}$$

Then, from Theorem 1 and equations (1),(2),

$$\frac{\mathtt{E}[L]}{L^*} \le q_f \cdot [2 + 4 \cdot (1 - q_f \cdot q_{ml}^2) \cdot \frac{log[\frac{\mathcal{PM}(\mathcal{T})}{(q_f \cdot (1-q_{ml}^2) \cdot \frac{e^{q_f}}{e^{q_f}-1})}]}{log(1 - q_f \cdot q_{ml}^4)}] \times [\frac{e^{q_f}}{e^{q_f}-1} \cdot \frac{log(p_{ml})}{log(\mathcal{PM}(\mathcal{T}))}] \tag{5}$$

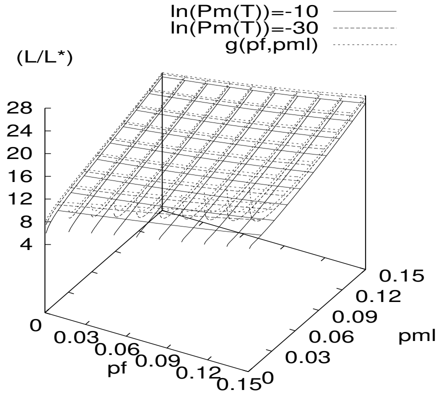
Even $\mathtt{E}[L]$ can be upper-bounded from the optimal L^* by a factor that is independent of the group size n.

Do the values of $\frac{L}{L^*}$ and $\frac{\mathtt{E}[L]}{L^*}$ go very high compared to the ideal value of 1.0 ? The answer is a 'No' when values of p_f, p_{ml} are low, yet reasonable. Figure 3(a) shows the variation of $\frac{L}{L^*}$ as in equation (3), at low but reasonable values of p_f, p_{ml}, and $\mathcal{PM}(\mathcal{T})$. This plot shows that the sub-optimality factor of the network load imposed by the new failure detector rises as p_{ml} and p_f increase, or $\mathcal{PM}(\mathcal{T})$ decreases, but is bounded above by the function $g(p_f, p_{ml})$, at all values of $\mathcal{PM}(\mathcal{T})$. This happens because $f(p_f, p_{ml}) < 0$ at such low values of p_f and p_{ml}, as seen from Figure 3(b) - Theorem 3.1 thus applies here. From figure 3(a), the function $g(p_f, p_{ml})$ (bottom-most surface), does not attain too high values (staying below 26 for the values shown). Thus the performance of the new failure detector algorithm is good for reasonable assumptions on the network unreliability.

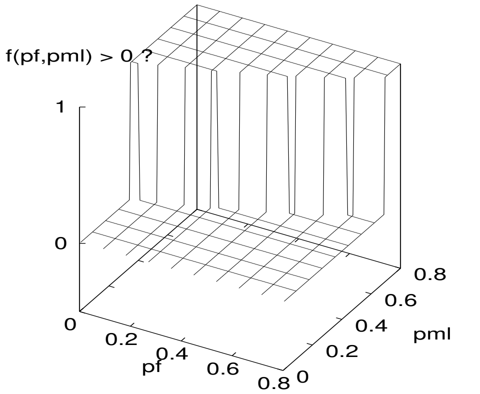
Figure 3(c) shows that the upper bound on $\frac{\mathtt{E}[L]}{L^*}$ stays very low (below 8) for values of p_f and p_{ml} up to 15%. Moreover, as $\mathcal{PM}(\mathcal{T})$ is decreased, the bound on $\frac{\mathtt{E}[L]}{L^*}$ actually *decreases*. This curve reveals the advantage of using randomization in the failure detector. Unlike traditional distributed heartbeating algorithms, the average case network load behavior of the new protocol is much lower than the worst-case network load behavior.

Figure 3 reveals that for values of p_f and p_{ml} below 15%, the $\frac{L}{L^*}$ for the new randomized failure detector stays below 26, and $\frac{\mathtt{E}[L]}{L^*}$ stays below 8. Further, as is evident from equations (3) and (5), the variation of these sub-optimality factors does not depend on the group size (at large group sizes). Compare this with the sub-optimality factors of distributed heartbeating schemes discussed in Section 4, which are typically at least $\theta[n]$.

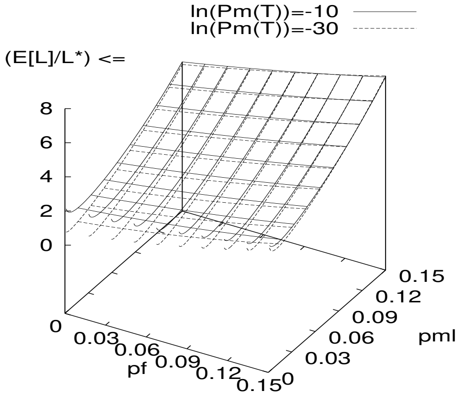
In reality, message loss rates and process failure rates could vary from time to time. The parameters p_f and p_{ml}, needed to configure protocol parameters T' and k, may be difficult to estimate. However, Figure 3 shows that assuming reasonable bounds on these message loss rates/failure rates and using these bounds to configure the failure detector suffices. In other words, configuring protocol parameters with $p_f, p_{ml} = 15\%$ will ensure that the failure detector preserves the application specified constraints ($\mathcal{T}$, $\mathcal{PM}(\mathcal{T})$) while imposing a network load that differs from the optimal worst-case load L^* by a factor of at most 26 in the worst-case, and 8 in the average case, as long as the message loss/process failure rates do not exceed 15% (this load is lower when loss or failure rates are lower).



(a) Variation of $\frac{L}{L^*}$ (according to equation (3)) versus p_{ml}, p_f, at different values of $\mathcal{PM}(\mathcal{T})$. For low values of p_{ml} and p_f, $g(p_f, p_{ml})$ is an upper bound on $\frac{L}{L^*}$.



(b) Values of p_f, p_{ml} for which $f(p_f, p_{ml})$ is positive or negative.



(c) Variation of $\frac{\mathtt{E}[L]}{L^*}$ versus p_{ml}, p_f (according to equation (5)).

Figure 3: Performance of new failure detector algorithm

5.3 Future Work and Optimizations

At Cornell University, we are currently testing performance of a scalable distributed membership service that uses the new randomized failure detection algorithm.

Extending the above protocol to the crash-stop model inherent to dynamic groups involves several protocol extensions. Every group member join, leave or failure detection entails a broadcast to the non-faulty group members in order to update their view. Further, this broadcast may not be reliable.

Implementing this protocol over a group spanning several subnets requires that the load on the connecting routers or gateways be low. The protocol currently imposes an $O(n)$ load (in bytes per second) on such routers during every protocol period. Reducing this load inevitably leads to compromising some of the EFFICIENCY properties of the protocol, as pings are sent less frequently across subnets.

The protocol can also be optimized to trade off worse SCALE properties for better ACCURACY properties. One such optimization is to follow a failure detection (by an individual non-faulty member through the described protocol) by multicast of a *suspicion* of that failure, waiting for some time before turning this suspicion into a declaration of a member failure. With such a suspicion multicast in place, protocol periods at different non-faulty group members, targeting this suspected member, can be correlated to improve the ACCURACY properties. This would also reduce the effect of correlated message failures on the frequency of mistaken failure declarations.

A disadvantage of the protocol is that since messages are restricted to contain at most a few bytes of data, large message headers mean higher overheads per message. The protocol also precludes optimizations involving piggy-backed messages, primarily due to the random selection of ping targets.

The discussion in this paper also points us to several new and interesting questions.

Is it possible to design a failure detector algorithm that, for an asynchronous network setting, satisfies COMPLETENESS, EFFICIENCY, *Scale* requirements, and the SPEED requirement (section 4) with a *deterministic* bound on time to detection of a failure ($\mathcal{T}$), rather than as an average case as we have done in this paper ?[8] Notice that this is not difficult to achieve in a synchronous network setting (by modifying the new failure detector algorithm to choose ping targets in a deterministic and globally known manner during every protocol period).

We also leave as an open problem the specification and realization of optimality load conditions for a failure detector with the SPEED timing parameter $\mathcal{T}$ set as the time to achieve *Strong Completeness* for any group member failure (rather than just Weak Completeness).

[8] Heartbeating along a logical ring among group members (eg., [7]) seems to provide a solution to this question. However, as pointed out before, ring heartbeating has unpredictable failure detection times in the presence of multiple simultaneous failures.

Of course, it would be ideal to extend all such results to models that assume some degree of correlation among message losses, and perhaps even member failures.

6. CONCLUDING COMMENTS

In this paper, we have looked at designing complete, scalable, distributed failure detectors from timing and accuracy parameters specified by the distributed application. We have restricted ourselves to a simple, probabilistically lossy, network model. Under certain independence assumptions, we have first quantified the optimal worst-case network load (messages per second, with a limit on maximal message size) required by a complete failure detector algorithm in a process group over such a network, derived from application-specified constraints of 1) detection time of a group member failure by *some* non-faulty group member, and 2) probability (within the detection time period) of no other non-faulty member detecting a given non-faulty member as having failed. We have then shown why the popular distributed heartbeating failure detection schemes inherently do not satisfy this optimal scalability limit.

Finally, we have proposed a randomized failure detector algorithm that imposes an equal expected load on all group members. This failure detector can be configured to satisfy the application-specified requirements of completeness and accuracy, and speed of failure detection (on average). Our analysis of the protocol shows that it imposes a worst-case network load that differs from the optimal by a sub-optimality factor greater than 1. For very stringent accuracy requirements ($\mathcal{PM}(\mathcal{T})$ as low as e^{-30}), reasonable message loss probabilities and process failure rates in the network (up to 15% each), the sub-optimality factor is not as large as that of traditional distributed heartbeating protocols. Further, this sub-optimality factor does not vary with group size, when groups are large.

We are currently involved in implementing and testing the behavior of this protocol in dynamic group membership scenarios. This involves several extensions and optimizations to the described protocol.

Acknowledgments

We thank all the members of the Océano group for their feedback. We are also immensely grateful to the anonymous reviewers and Michael Kalantar for their suggestions towards improving the quality of the paper.

7. REFERENCES

[1] M. K. Aguilera, W. Chen, and S. Toueg. Heartbeat: a timeout-free failure detector for quiescent reliable communication. In *Proceedings of 11th International Workshop on Distributed Algorithms (WDAG'97)*, pages 126–140, September 1997.

[2] C. Almeida and P. Verissimo. Timing failure detection and real-time group communication in real-time systems. In *Proceedings of 8th Euromicro Workshop on Real-Time Systems*, June 1996.

[3] K. P. Birman. The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37–53, December 1993.

[4] R. Bollo, J.-P. L. Narzul, M. Raynal, and F. Tronel. Probabilistic analysis of a group failure detection protocol. In *Proceedings of 4th International Workshop on Object-Oriented Real-Time Dependable Systems*, 1998.

[5] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.

[6] W. Chen, S. Toueg, and M. K. Aguilera. On the quality of service of failure detectors. In *Proceedings of 30th International Conference on Dependable Systems and Networks (ICDSN/FTCS-30)*, June 2000.

[7] S. A. Fakhouri, G. S. Goldszmidt, I. Gupta, M. Kalantar, and J. A. Pershing. Gulfstream - a system for dynamic topology management in multi-domain server farms. Technical Report RC 21954, IBM T.J. Watson Research Center, February 2001.

[8] C. Fetzer and F. Cristian. Fail-awareness in timed asynchronous systems. In *Proceedings of 15th Annual ACM Symposium on Principles of Distributed Computing (PODC'96)*, pages 314–321a, May 1996.

[9] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed Consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, April 1985.

[10] I. Gupta, R. van Renesse, and K. P. Birman. A probabilistically correct leader election protocol for large groups. In *Proceedings of 14th International Symposium on Distributed Computing (DISC 2000), LNCS-1914*, pages 89–103, October 2000.

[11] J. M. Helary and M. Hurfin. Solving Agreement problems with failure detectors; a survey. *Annals of Telecommunications*, 52(9-10):447–464, September-October 1997.

[12] M. Larrea, A. Fernandez, and S. Arevalo. Optimal implementation of the weakest failure detector for solving Consensus. In *Proceedings of 19th Annual ACM-SIGOPS Symposium on Principles of Distributed Computing (PODC 2000)*, July 2000.

[13] G. Pfister. *In search of Clusters, the Ongoing Battle in Lowly Parallel Computing*. Prentice Hall, 1998.

[14] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-style failure detection service. In *Proceedings of International Conference and Distributed Systems Platforms and Open Distributed Processing (IFIP)*, 1998.