Explicit Allocation of Best-Effort Packet Delivery Service

David D. Clark, *Fellow, IEEE*, and Wenjia Fang

***Abstract*— This paper presents the "allocated-capacity" framework for providing different levels of best-effort service in times of network congestion. The "allocated-capacity" framework—extensions to the Internet protocols and algorithms—can allocate bandwidth to different users in a controlled and predictable way during network congestion. The framework supports two complementary ways of controlling the bandwidth allocation: sender-based and receiver-based. In today's heterogeneous and commercial Internet the framework can serve as a basis for charging for usage and for more efficiently utilizing the network resources. We focus on algorithms for essential components of the framework: a differential dropping algorithm for network routers and a tagging algorithm for profile meters at the edge of the network for bulk-data transfers. We present simulation results to illustrate the effectiveness of the combined algorithms in controlling transmission control protocol (TCP) traffic to achieve certain targeted sending rates.**

***Index Terms*— Internet protocol, packet networks, quality of service, rate control, TCP.**

I. Introduction

THIS PAPER describes a new framework—the "allocated-capacity" framework—for providing allocated-capacity service in the Internet. The goal of the mechanism is to allocate the bandwidth of the Internet to different users in a controlled way during periods of congestion. The mechanism applies equally to traditional applications based on transmission control protocol (TCP), such as file transfer, database access, or Web servers, and new applications such as real-time video and audio.

The current Internet assumes the "best-effort" service model. In this model the network allocates bandwidth among all of the instantaneous users as best it can and attempts to serve all of them without making any explicit commitment as to rate or any other service quality. When congestion occurs, the sources of traffic are expected to detect this and slow down, so that they achieve a collective sending rate equal to the capacity of the congestion point. In contrast, the mechanism offered by the "allocated-capacity" framework can provide users with predictable expectations of Internet service. In times of congestion all connections will slow down and reduce their sending rates to the expected rates. Since different users have different allocations, the network offers different levels of best-effort service in times of congestion.

The mechanism in the framework allows users and providers with a wide range of business and administrative goals to make capacity allocation decisions. In the public Internet, where commercial providers offer service for payment, the feedback to customers is most often monetary. Our framework allows the providers to charge different prices to users with different service requirements and, thus, fund the deployment of additional resources. In private networks like corporate or military networks, administrative measures are often used to allocate resources. Our framework provides a means to allocate different resources to different users. Regardless of top-level policy, the same mechanism can be deployed in the underlying infrastructure to allocate bandwidth.

Additionally, the mechanism provides useful information to providers about provisioning requirements. With our mechanism in place, service providers can more easily allocate specific levels of assured capacity to customers and can easily monitor their networks to detect when their customers' needs are not being met.

The rest of the paper is organized as follows. Section II explains the framework in detail. The framework is simple, scalable, and flexible to provide different kinds of service. We also describe two complementary ways of controlling the traffic: sender-based and receiver-based. Section III describes two algorithms: a preferential dropping algorithm, which we propose to be adopted in the center of the network, and a tagging algorithm tailored for bulk-data TCP traffic. As an example, we will use bulk-data TCP transfers with certain throughput expectations to demonstrate the concepts in the framework. Section IV presents results using the above algorithms in simulated environments for bulk-data transfers. The simulations show that the "allocated-capacity" framework is effective in providing different levels of best-effort service with high assurance over the existing Internet. The framework also provides a simple way of identifying nonresponsive users at aggregation points. Section V concludes our work.

A. Related Work

A number of approaches have been proposed for controlling usage and explicit allocation of resources among users in time of overload, both in the Internet and in other packet networks.

MacKie-Mason and Varian proposed dynamic allocation of bandwidth at the packet granularity in their "smart-market" scheme [19]. In this scheme each packet carries a bid—a price that the user is willing to pay for service. At each point of

Manuscript received September 19, 1997; approved by IEEE/ACM Transactions on Networking Editor S. Pink. This work was supported in part by the Advanced Research Projects Agency of the Department of Defense under Contract DABT63-94-C-0072, administrated by Ft. Huachuca, and in part by a Graduate Fellowship from Sun Microsystems, Inc.

D. D. Clark is with the Laboratory for Computer Sciences, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: ddc@lcs.mit.edu).

W. Fang is with the Computer Science Department, Princeton University, Princeton NJ 08540 USA (e-mail: wfang@cs.princeton.edu).

Publisher Item Identifier S 1063-6692(98)05707-0.

congestion, all of the offered packets are ranked by price and a cutoff price is determined, based on current capacity, such that only those packets with a bid above the cutoff are serviced. The others are held in a queue, subjected to increased delay and risk of being dropped. There are a number of drawbacks to this scheme. One is that the linkage between the treatment of each individual packet and the overall transfer rate is not obvious. Also, the "smart market" operates only on a hop-by-hop basis, and it is not obvious how this can be translated into end-to-end performance. Finally, the computation needed for clearing the bids and accounting in each router is likely to be prohibitive.

Gupta *et al.* [15] proposed priority scheduling for allocation of bandwidth among users. This scheme creates service classes of different priorities to serve users with different needs. Higher priority packets always depart the routers first. Thus, the effect of priority queueing is to build up a queue of lower priority packets, which will cause packets in this class to be preferentially dropped due to queue overflow. This scheme might be a useful building block for explicit service discrimination, but it does not have a mechanism for balancing the demands of the various classes.

Weighted fair queueing [3], [6] creates different queues for different connections and ensures that each connection will receive some share of the bandwidth. This mechanism allocates bandwidth among all connections within a router, but does not by itself address how many connections each user has and how they interact. In addition, it is not clear that this scheme is scalable in the center of the network where the routers have a large of amount traffic connections aggregated.

Our approach is based on the idea of tagging packets as *in* or *out* and treating them differently based on the tags. This idea of tagging packets is not a new one. For example, researchers at IBM [1] proposed tagging as part of a flow control scheme. Frame relay has the concept of *in/out* packets as does asynchronous transfer mode (ATM)—the cell loss preference (CLP) bit. Those ideas were proposed in the context of a specific reserved flow or virtual circuit from a source to a destination. In [5] the idea was applied to a packet-switched network where there is no implication that the allocated capacity for any user is reserved along a particular path. Profile meters tag packets based on contracted profiles between Internet service providers (ISP's) and customers. The network preferentially drops *out* packets during periods of congestion. As a consequence, the ISP's can offer different levels of service based on these profiles. Reference [4] also developed a receiver-based scheme for controlling traffic.

Our framework incorporates the above tagging idea, and extends it in the following three aspects: 1) instantiates the framework by designing a set of tagging and dropping algorithms; 2) provides a simple way to identify and isolate nonresponsive connections; and 3) demonstrates the effectiveness of the framework with simulation results.

II. THE "ALLOCATED-CAPACITY" FRAMEWORK

A. Overview

The general approach of this mechanism is to define a service allocation profile for each user and to design a mechanism in the router that favors traffic that is within those service allocation profiles. The core of the idea is very simple—monitor the traffic of each user as it enters the network and tag packets as either *in* or *out* of their service allocation profiles, then at each congested router, preferentially drop packets that are tagged as being *out*.

Inside the network, at the routers, there is no separation of traffic from different users into different flows or queues. The packets of all users are aggregated into one queue, just as they are today. Different users can have very different profiles, which will result in different users having different quantities of *in* packets in the service queue. A router can treat these packets as a single commingled pool. This attribute of the scheme makes it very easy to implement, in contrast to a scheme like RSVP [21] or weighted fair queueing, in which the packets must be explicitly classified at each node.

To implement this scheme, the routers must be augmented to implement a dropping scheme[1] (Section III-A offers the specifics of a preferential dropping algorithm we developed). Additionally, a new function must be implemented to tag the traffic according to its service allocation profile. This algorithm can be implemented as part of an existing network component—host, access device, or router—or in a new component created for the purpose. Conceptually, we will refer to it as a distinct device called a "profile meter."

B. Location of Profile Meters in the Network

Fig. 1 illustrates the "allocated-capacity" framework with a sender-based control. All of the routers (G) in the network have adopted a preferential dropping algorithm (D). In the simple sender-based scheme the function that checks whether traffic fits within a profile is implemented by tagging packets at the edge of the network, e.g., the profile meter (M2) is on the access link from H1 to ISP1. The complete story is more complex. A profile describes an expectation of service obtained by a customer from a provider. These relationships exist at many points in the network, ranging from individual users and their campus local area networks (LAN's) to the peering relationships between global ISP's. Any such boundary may be an appropriate place for a profile meter, e.g., M3–M6 in Fig. 1.

Furthermore, the packet tagging associated with this service allocation profile will, in the general case, be performed by devices at both side of a boundary. One such device, located on the sourcing traffic side of a network boundary, is a "policy meter" (M1, M3, and M5 in Fig. 1). This device chooses which packets to tag, based on some administrative policy. Another sort of device, the "checking meter," sits on the arriving traffic side of a network boundary, checks the incoming traffic, and marks packets as *out* if the arriving traffic exceeds the assigned profile, e.g., M2, M4, and M6. In this generalized model a

[1] There are other schemes being proposed to create preferential treatments of packets, including a priority scheme in which packets tagged as *in* are put into a separate queue from the *out* packets, or more elaborate versions. Separate queues for different types of packets will likely cause packet reordering, resulting in performance degradation in TCP or jitter in real-time traffic. In this paper we only focus on using preferential dropping of packets and placing both *in* and *out* packets in the same queue.

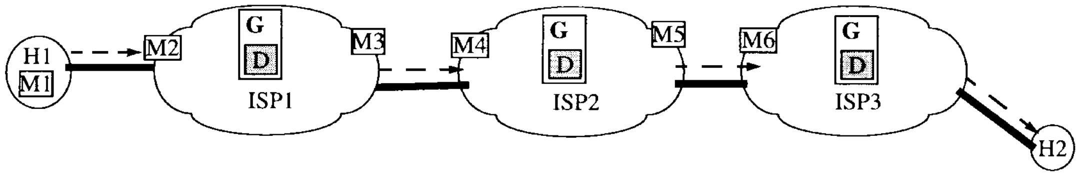


Fig. 1. The "allocated-capacity" framework (sender-based). Host 1 (H1) has a sender-based profile and is sending traffic to host 2 (H2) (dotted line). The traffic traverses three ISP's. The routers G in the figure are all augmented with preferential dropping algorithms D. There are profile meters M at each interface between a customer and an ISP, or between two ISP's. M1 is a profile meter inside a host, M2 is on the access link from H1 to ISP1, and M3–M6 are profile meters on the boundaries of ISP's.

packet will travel through the network, passing a series of cascaded profile meters.

The first meter that the traffic encounters should provide the highest degree of discrimination among the connections. As the traffic merges and aggregates with other traffic in the center of the network, the corresponding profile meters only need to look at large aggregates. A profile meter integrated into a host implementation of TCP and Internet protocol (IP), for example, can serve to regulate the relative use of the network by individual flows. In contrast, subsequent meters at ISP boundaries serve to verify that there is a large enough overall service contract in place at that point to carry all the traffic tagged as *in* at the interior points.

C. A Spectrum of Services

In designing this framework we are serving two potentially conflicting goals. First, we would like to implement a set of simple services which are useful and easy to understand and adopt; second, we do not want to embed the above services into the mechanisms so that the framework cannot adapt to new applications with new service requirements in the future. The decoupling of the service allocation profiles at the edge of the network from the differential dropping in the center of the network allows this flexibility. To oversimplify, the preferential dropping scheme adopted in routers in the center of the network will not change over time. Since the characteristics of a service is defined and captured by its corresponding profile meter, it is only necessary to create the profile meter at the edge of the network to adopt a new service.

The services provided by this framework are diverse. As a simple example, it could be the equivalent of a dedicated link of some specified bandwidth from a source to a destination. Such a model is easy for users to understand. A more elaborate model can be an aggregated commitment to a range of destinations, or anywhere within an ISP, sometimes called a private virtual network. A virtual network is by nature more difficult to offer with high assurance since offering commitments to "anywhere within a virtual network" implies that the ISP has provisioned its resources adequately to support all users sending *in* traffic simultaneously to any destination.

Not all Internet traffic is continuous in its requirement for bandwidth. In fact, most Internet traffic is very bursty. It may thus be that a "virtual-link" service model is not what users really want. It is possible to support bursty traffic by changing the profile meter to implement this new sort of service. The key issue is to ensure, in the center of the network, that there is enough capacity to carry this bursty traffic and, thus, actually meet the commitments implied by the outstanding profiles. This requires a more sophisticated provisioning strategy than the simple "add 'em up" needed for constant bit-rate virtual links. However, in the center of the existing Internet, especially at the backbone routers of major ISP's, there is a sufficiently high degree of aggregation that the bursty nature of individual traffic flows is no longer visible. This suggests that providing bursty service allocation profiles to individual users will not create a substantial provisioning issue in the center of the network, while possibly adding significant value to the service as perceived by the users.

A more sophisticated service allocation profile would be one that attempts to provide a specified and predictable throughput to a TCP stream. This is more complex than a profile that emulates a fixed capacity link, since TCP hunts for the correct operating rate by increasing and decreasing its window size, which causes rate fluctuations to which the profile must conform. The service allocation profile is easy for a user to test by simply running a TCP-based application and observing the throughput. This is an example of a "higher level" profile, because it is less closely related to some existing network components and more closely related to the users' actual demands. In Section III we will describe algorithms to implement such a profile.

In summary, three things must be considered when describing a service allocation profile.

- *Traffic specifications*: What exactly is provided to the customer (for example, 5 Mb/s average throughput)?
- *Geographic scope*: To where is this service provided (examples might be a specific destination, a group of destinations, all nodes on the local provider, or "everywhere")?
- *Probability of assurance*: With what level of assurance is the service provided (or, alternately, what level of performance uncertainty can the user tolerate)?

These things are coupled; it is much easier to provide "a guaranteed 1 Mb/s" to a specific destination than to anywhere in the Internet.

D. Provisioning with Statistical Assurance

The statistical multiplexing nature of the Internet makes efficient use of bandwidth and supports an increasing number of users and new applications. However, it does lead to some uncertainty as to how much of the bandwidth is available at any instant. Our approach to allocating traffic is to follow this philosophy to the degree that the user can tolerate the uncertainty. In other words, we believe that a capacity allocation scheme should provide a range of service assurance. At one extreme, the user may demand an absolute service assurance, even in the face of some network failures. Less demanding users may wish to purchase a service allocation profile that is "usually available" but may still fail with low probability. The presumption is that a higher assurance service will cost substantially more to implement.

We have called these statistically provisioned service allocation profiles "expected capacity" profiles. This term was picked to suggest that the profiles do not describe a strict guarantee but, rather, an expectation that the user can have about the service he will receive during times of congestion. This sort of service will somewhat resemble the Internet of today in that users have some expectation of what network performance that they will receive; the key change is that our mechanism permits different users to have different expectations.

For traffic that requires a higher level of commitment, more explicit actions must be taken. Those actions can be either static, e.g., making a long-term commitment on physical links to a user, or dynamic, e.g., an RSVP-like protocol to set up temporary reservations. It should be noted that traffic requiring this higher level of assurance can still be aggregated with other similar traffic. It is not necessary to separate out each individual flow to ensure that it receives its promised service. For example, there could be two queues in the router, one for traffic that has received a statistical assurance and one for this higher, or "guaranteed," assurance. Within each queue, *in* and *out* tags would be used to distinguish the subset of the traffic that is to receive the preferred treatment.

Fundamentally, statistical assurance is a matter of provisioning. In our scenario an ISP can track the amount of traffic tagged as *in* crossing various links over time, and provide enough capacity to carry this subset of the traffic, even at times of congestion. This is how the Internet is managed today, but the addition of tags gives the ISP a better handle on how much of the traffic at any instant is "valued" traffic and how much is discretionary or opportunistic traffic for which a more relaxed attitude can be tolerated.

E. Receiver-Controlled Scheme

The tagging scheme described above implements a model in which the sender, by selecting one or another service allocation profile, determines what service will govern each traffic flow. However, in today's Internet, the receiver of the traffic, not the sender, is often more the appropriate entity to make such decisions. We describe a mechanism that implements receiver control of service, which is similar in approach and complementary to the sender-controlled tagging scheme.

The receiver-based scheme in the "allocated-capacity" framework is the dual of the sender-based scheme. It relies on a newly proposed change to TCP called the explicit congestion notification (ECN) bit [11]. In ECN semantics, congested routers will turn on the ECN bit in a packet instead of dropping the packet. The TCP receiver copies the ECN bit into the acknowledgment (ACK) packet, and the sender TCP will gracefully slow down upon receiving an ack with the ECN bit on.

In the receiver-based expected capacity scheme, routers will not be modified; they will turn on the ECN bit in a packet when there is congestion. A profile meter, installed at the receiver, can check whether a stream of received packets is inside of the profile. Each arriving packet will debit the receiver's service allocation profile. If there is enough profile to cover all arriving packets, the meter will turn off the ECN bits in those packets which had encountered congestion since the receiver is entitled to receive at this rate. If the receiver's profile is exceeded, packets with their ECN bits on will be left unchanged at the profile meter. If packets arrive at the TCP receiver with ECN bits still on, it means that the receiver has not contracted for sufficient capacity to cover all of the packets that encountered congestion, and the sender will be notified to slow down.

1) Difference Between Sender-Based Control and Receiver-Based Control: There are a number of interesting asymmetries between the sender and the receiver versions of this tag and profile scheme, which arise from the fact that the data packets flow from the sender to the receiver. In the sender scheme the packet first passes through the meter, where it is tagged, and then through any point of congestion. In contrast, in the receiver-controlled scheme the packet first passes through any points of congestion, where it is tagged, and then through the receiver's meter. The receiver scheme, since routers only set the ECN bit if congestion is actually detected, can convey to the end point dynamic information about the current congestion levels. In the sender scheme, in contrast, profile meters must tag the packets as *in* or *out* without knowing if congestion is actually present. Thus, we could construct a service, based on the receiver scheme, to bill the user for actual usage during congestion.

On the other hand, the receiver scheme is more indirect in its ability to respond to congestion. Since in the sender scheme a packet carries the explicit assertion of whether it is *in* or *out* of profile, the treatment of the packet is evident when it reaches a point of congestion. In the receiver scheme the data packet itself carries no such profile indication, so, at the point of congestion, the router must set the ECN bit, and still attempts to forward the packet, trusting that the sender will correctly adjust its transmission rate. Of course, if the profile meter at the receiver's side employs a dropping algorithm, which will drop any packets that has exceeded the profile, the sender will slow down if it is a properly behaved TCP.

Another difference between the two schemes is that in the sender scheme, the sending application can set the *in/out* bit selectively to control which packets are favored during the congestion. In the receiver scheme all packets sent to the receiver pass through and debit the profile meter before the receiver host gets them. Thus, in order for the receiver host to distinguish those packets that should receive preferred service, it would be necessary for it to install some sort of packet filter in the profile meter.

2) Combining Sender-Based and Receiver-Based Schemes: The sender-based scheme can be combined with the receiver-based scheme [4]. One extra bit in the packet header will indicate whether this packet is a sender-pay packet or a receiver-pay packet. The receiver-pay scheme is the dual of the sender-pay scheme; for example, in the receiver-pay case

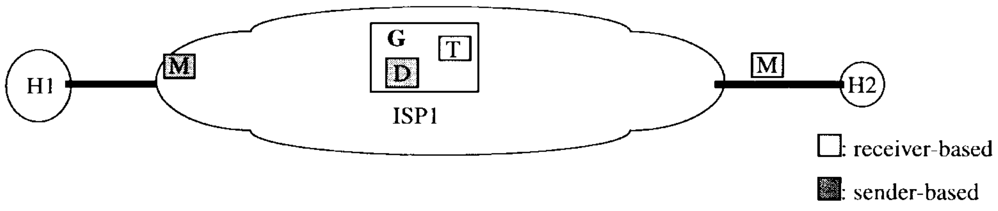


Fig. 2. Simplified "allocated-capacity" framework. The simplified framework to illustrate the different levels of service provided for bulk-data TCP transfers. Both sender-based and receiver-based are shown. The darkly shaded boxes, profile meter M, and dropper D are for sender-based control. The lightly-shaded boxes, tagger T in the router and profile meter M at the receiver side, are for receiver-based control. G are all the routers in ISP1. There are no cascaded profile meters.

the routers turn on the ECN bit of a packet whenever such a packet would have been dropped had it been an *out* packet in the sender-pay case. For sender-pay packets, the routers always know which packets can be preferentially dropped by looking at the *in/out* bit, whereas for the receiver-pay packets, the routers only mark the packets and rely on the end hosts to back off. Therefore, there could potentially be theft-of-service by malicious users in the receiver-pay scheme. In a related work [2] addresses the robustness and security issues of a framework in which the two schemes can be combined.

III. "Allocated Capacity" for Bulk-Data Transfers

It is important to realize that the dropping algorithm in the routers, once adopted, is unlikely to change again over time; however, the service allocation profiles and corresponding profile meters will evolve as users have more sophisticated needs for new applications. Therefore, we need to find a dropping algorithm that will offer enough generality to cooperate with many types of profile meters. In this section we propose a preferential dropping algorithm to create discrimination in the center of the network. Additionally, we present a tagging algorithm tailored for bulk-data TCP transfers. For the sake of simplicity, we assume a simplified network with only one ISP between any two connecting hosts, as illustrated in Fig. 2. There is no cascading of profile meters. The service allocation profiles that we use are easy for the users to understand—they can provide a specific average throughput to anywhere within this network, with round trip times (RTT's) ranging from 20 to 100 ms (which is roughly comparable to metropolitan connections and cross-US connections, respectively). We will call the expected throughput the *target rate*, or R_T. Different levels of service refer to the different target rates specified in service allocation profiles. The profile meters are on the access link from the host to its immediate ISP. The network is well provisioned since the sum of all service allocation profiles sold to customers does not exceed the link speed. We first concern ourselves with the kind of assurance different TCP connections can achieve using the combined tagging and dropping algorithms. We then explore both sender-based and receiver-based schemes, and, finally, we study the results when traffic from a nonresponsive source persistently congests the router.

A. TCP Rate Adjustment in the Current Internet

In today's Internet the end-host transport-layer TCP has congestion control and avoidance mechanisms which adjust TCP's sending rate in response to congestion. Additionally, the router queue management algorithm drops packets as a mechanism for congestion feedback. We take advantage of both mechanisms to adjust TCP's sending rate. We will discuss them in turn.

The mechanisms used by TCP to deal with congestion are based on [18]. TCP has two modes of dealing with congestion. The first mode, "fast recovery," is triggered by the loss of very few packets, typically one. In this mode the TCP cuts its sending window size in half and, following a successful retransmission, increases its window size by one packet each RTT. Since the achieved transmission rate for any window size is roughly proportional to that window size, cutting the window size in half has the effect of reducing the achieved sending rate by up to half. The second mode is called "slow start," and typically occurs when the retransmission timer goes off. In the slow-start mode TCP first reduces its window size to one, and then opens its window exponentially until the window reaches its slow-start threshold, or *ssthresh*, after which TCP opens up its window linearly. *ssthresh* reflects what TCP perceives to be the optimal operating point and is cut in half each time TCP detects a packet loss. In the current implementations of TCP, when many packet losses occur within one RTT, TCP cannot use fast recovery to recover packets and has to rely on the retransmission timer. When the retransmission timer goes off, TCP enters the slow-start mode. This has a more drastic effect on the TCP performance. First, the retransmission timer is often crude, measured in a granularity as coarse as 500 ms, and TCP does not send data during this period. Second, in slow start the sending TCP sets its window size to one packet when it starts again, also with a much reduced *ssthresh.* This essentially reduces the sending rate to zero. Therefore, the rate adjustments currently implemented by TCP are both imprecise and, on occasion, drastic. Given this, there is a concern that TCP's rate adjustment mechanism cannot be used with enough precision to achieve a specific overall throughput, especially if slow start is triggered.

In the current Internet, routers deal with congestion by dropping packets. Each time a packet is dropped, it causes a rate adjustment in one of the sending TCP's. Between drops, all the TCP's with data to send will increase their rate in an attempt to fill the network links fully. So infrequent packet drops, which might seem to be preferred mode of operation, actually provide fewer opportunities to adjust the rates among the various senders. As long as the packet drops trigger only the fast-recovery behavior, rather than the slow-start behavior,

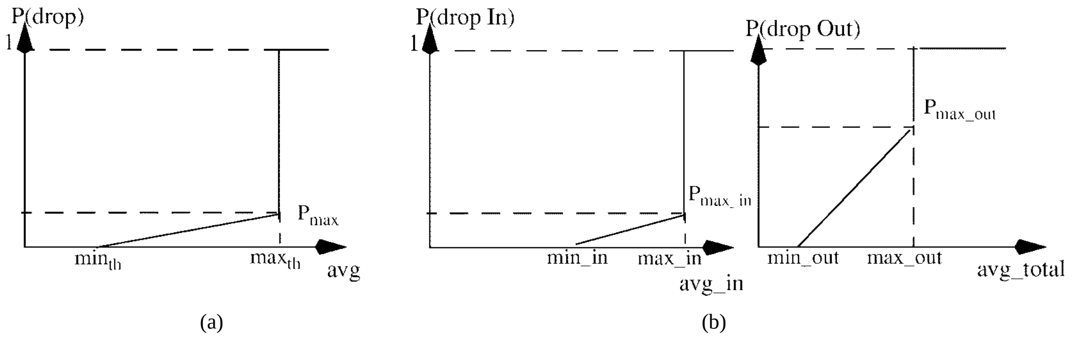


Fig. 3. (a) RED and (b) RIO algorithms (figures not drawn to scale).

TCP stays in a phase with quantifiable rate adjustments and is much more controllable.

B. Differential Dropping in the Routers: RIO

RIO stands for random early drop (RED) routers with *in/out* bit. RED routers [10] keep the overall throughput high while maintaining a small average queue length, and tolerate transient congestion. When the average queue has exceeded a certain threshold, RED routers drop packets at random so that TCP connections back off at different times. This avoids the global synchronization effect of all connections and maintains high throughput in the routers. RIO retains all these attractive attributes. In addition, it discriminates against *out* packets in times of congestion. At a high level, RIO uses twin RED algorithms for dropping packets, one for *in*s and one for *out*s. By choosing the parameters for respective algorithms differently, RIO is able to preferentially drop *out* packets. We will briefly describe the RED algorithm before presenting RIO.

1) RED Algorithm: An RED router operates as follows. It computes the average queue size and when the average queue size exceeds a certain threshold, it drops each arriving packet with a certain probability, where the exact probability is a function of the average queue size. The average queue size is calculated using a low-pass filter from instantaneous queue size, which allows transient bursts in the router. Persistent congestion in the router is reflected by a high average queue size and a high dropping probability. The resulting high dropping probability will discard packets early and, thus, detect and control congestion.

A RED router is configured with the following parameters: $\min_{th}$, $\max_{th}$, and $P_{\max}$. It works as illustrated in the leftmost figure in Fig. 3—the x-axis is avg, the average queue size, which is calculated using a low-pass filter of instantaneous queue size upon each packet arrival. The y-axis is the probability of dropping an arriving packet. There are three phases in RED, defined by the average queue size in the range of [0, $\min_{th}$), [$\min_{th}$, $\max_{th}$), and [$\max_{th}$, ∞), respectively. The three phases are normal operation, congestion avoidance, and congestion control, respectively. During the normal operation phase, when the average queue size is below $\min_{th}$, the router does not drop any packets. When the average queue size is between the two thresholds, the router is operating in the congestion avoidance phase, and each packet drop serves the purpose of notifying the end-host transport layer to reduce its sending rate. Therefore, the dropping probability is a fraction of $P_{\max}$, and is usually small. When the average queue size is above $\max_{th}$, the router drops every arriving packet, hoping to maintain a short queue size.

2) Twin Algorithms in RIO: RIO uses the same mechanism as in RED but is configured with two sets of parameters, one for *in* packets and one for *out* packets. Upon each packet arrival at the router, the router checks whether the packet is tagged as *in* or *out*. If it is an *in* packet, the router calculates $\mathrm{avg\_in}$, the average queue for the *in* packets; if it is an *out* packet, the router calculates $\mathrm{avg\_total}$, the average total queue size for all (both *in* and *out*) arriving packets. The probability of dropping an *in* packet depends on $\mathrm{avg\_in}$, and the probability of dropping an *out* packet depends on $\mathrm{avg\_total}$.

As illustrated in Fig. 3, there are three parameters for each of the twin algorithms. The three parameters $\min\_\mathrm{in}$, $\max\_\mathrm{in}$, and $P_{\max\_\mathrm{in}}$ define the normal operation [0, $\min\_\mathrm{in}$), congestion avoidance [$\min\_\mathrm{in}$, $\max\_\mathrm{in}$), and congestion control [$\max\_\mathrm{in}$, ∞) phases for *in* packets. Similarly, $\min\_\mathrm{out}$, $\max\_\mathrm{out}$, and $P_{\max\_\mathrm{out}}$ defines the corresponding phases for *out* packets.

The discrimination against *out* packets in RIO is created by carefully choosing the parameters ($\min\_\mathrm{in}$, $\max\_\mathrm{in}$, $P_{\max\_\mathrm{in}}$), and ($\min\_\mathrm{out}$, $\max\_\mathrm{out}$, $P_{\max\_\mathrm{out}}$). As illustrated in two right figures in Fig. 3, a RIO router is more aggressive in dropping *out* packets on three counts. First, it drops *out* packets much earlier than it drops *in* packets; this is done by choosing $\min\_\mathrm{out}$ smaller than $\min\_\mathrm{in}$. Second, in the congestion avoidance phase it drops *out* packets with a higher probability by setting $P_{\max\_\mathrm{out}}$ higher than $P_{\max\_\mathrm{in}}$. Third, it goes into congestion control phase for the *out* packets much earlier than for the *in* packets by choosing $\max\_\mathrm{out}$ much smaller than $\max\_\mathrm{in}$. In essence, RIO drops *out* packets first when it detects incipient congestion, and drops all *out* packets if the congestion persists. Only as a last resort, occurring when the router is flooded with *in* packets, it drops *in* packets in the hope of controlling congestion. In a well-provisioned network this should never happen. When a router is consistently operating in a congestion control phase by dropping *in* packets, this is a clear indication that the network is underprovisioned. Appendix A contains the pseudocode for RIO algorithm.

The choice of using $\mathrm{avg\_total}$, the total average queue size to determine the probability of dropping an *out* packet, is

```
Initially:

        Win_length = a constant;
        Avg_rate   = connection's target rate, R_T;
        T_front    = 0;

Upon each packet arrival, TSW updates its state variables as follows:

        Bytes_in_TSW = Avg_rate * Win_length;
        New_bytes    = Bytes_in_TSW + pkt_size;
        Avg_rate     = New_bytes / (now - T_front + Win_length);
        T_front      = now;

Whereas, now is the time of current packet arrival; and pkt_size is the packet size of the arriving packet.
```

Fig. 4. TSW algorithm.

subtle. Unlike *in* packets, which the network can properly provision for, the *out* packets represent opportunistic traffic, and there is no valid indication of what amount of *out* packets is proper. If we had used the average *out* packet queue to control the dropping of *out* packets, this would not cover the case where the total queue is growing due to arriving *in* packets. We could have used avg_in, the average queue for the *in* packets, to see how much "free space" the routers have for *out* packets, i.e., drop fewer *out* packets when avg_in is small and drop more *out* packets when avg_in is large. But this only works when the number of *in* packets in the queue is large, so the routers have good control on the number of *out* packets and total queue length. By using the avg_total, total average queue size, routers can maintain short queue length and high throughput no matter what kind of traffic mix they have. It is conceivable that one could achieve a more responsive control of *out* packets by changing the dropping parameters to depend on both the average *in* queue size avg_in and the average total queue size avg_total, but we have not explored this idea.

C. Profile Meters for Bulk-Data Transfers: TSW Tagger

The profile meter that we designed for bulk-data transfers is called the time-sliding window (TSW) tagger. The TSW tagger has two distinct parts—a rate estimator and a tagging algorithm. TSW refers to the rate estimator algorithm. TSW provides a smooth estimate of the TCP sending rate[2] over a period of time. With the estimated rate avg_rate, the tagging algorithm can tag packets as *out* packets once the traffic exceeds a certain threshold.

A rate estimator is used to smooth out the burstiness of TCP traffic as well as to be sensitive to instantaneous sending rates. TSW estimates the sending rate upon each packet arrival and decays, or forgets, the past history over time.[3] The design of TSW is also extremely simple. TSW maintains three state variables—Win_length, which is measured in units of time, Avg_rate, the rate estimate upon each packet arrival, and $T\text{\_front}$, which is the time of last packet arrival. TSW is used to estimate the rate upon each packet arrival, so state variables Avg_rate and $T\text{\_front}$ are updated each time a packet arrives, but Win_length is preconfigured when the profile meter is installed.

The TSW rate estimator works as shown in Fig. 4.

We do not include a proof of the decaying function embedded in TSW. Intuitively, TSW remembers Win_length worth of past history, and decays the estimated sending rate by a factor of e over Win_length period of time.

In terms of tagging algorithm there are two different approaches. Ideally, a profile meter can keep a TCP connection oscillating between 0.66 R_T, the target rate, and 1.33 R_T so that, on average, the connection can achieve R_T. The first approach is that the meter could remember a relatively long past history—in the order of a TCP sawtooth from 0.66 to 1.33 R_T—and tag packets as *out* with $P = (\text{avgrate} - R_T)/R_T$, when the avg_rate exceeds R_T. All packets are tagged as *in* when the avg_rate is below R_T. The second approach is for the profile meter to remember a relatively short history—on the order of an RTT—and look for the peak of a TCP sawtooth when TCP exceeds 1.33 R_T, at which point, the tagger starts tagging packets as *out*. When the profile meters are next to the host, where TCP sawtooths are quite visible, the second approach is more effective. On the other hand, the first approach is more general and can be applied not only to individual TCP connections but also to aggregated TCP traffic or other type of traffic. In our simulations we use the second approach.

D. Difficulties in Designing RIO-TSW

Our service allocation profile "certain target throughput to anywhere (within ISP)," albeit simple, is in fact difficult to accomplish if the profile meters are on the access link from the hosts to their ISP's. There are two reasons for this. First of all, with the TCP algorithm for opening up windows, there is a strong network bias in favor of connections with short RTT's. In the fast-recovery phase TCP increases cwnd by one packet[4] each RTT. Let r denote RTT. Each RTT, TCP increases its sending rate by $1/r$ packets/s, or it increases its sending rate by $1/(r)^2$ each second. For example, when a connection has an RTT five times that of another connection, the increase in sending rate for this connection is 1/25 of the other connection. Therefore, when both connections receive drops simultaneously, it takes the long RTT connection much longer to recover to its sending rate before the drop than the short one. During this period of recovery, the short RTT

[2] The burstiness of TCP traffic is a well-known phenomenon. Articulated in [20], it is caused by the fact that TCP paces out packets using its window algorithm and possible "ACK compression" in two-way traffic.

[3] Though a low-pass filter of instantaneous sending rate (packet size divided by interpacket arrival time) seems to be an obvious choice for the rate estimator, it suffers a flaw: it decays the sending rate over packet arrivals, not over time. Consequently, a fast TCP is decaying its past history faster than a slow TCP, and when TCP is not sending, the past history is not decayed. TSW is designed to avoid this.

[4] Real implementation of TC increases window measured in bytes, instead of packets; we use packets for simplicity in explanation.

connection has a higher average sending rate than the long RTT connection. This explains why a service allocation profile with specific source–destination pair is comparatively easier to implement because its RTT is known.[5] As we discussed in Section III-A, the profile meters are on the access link from the host to the ISP, and do not know the RTT of TCP connections at the host. Therefore, the profile meter has to assume some fixed RTT value and use it for all possible RTT's, as the host TCP is promised to send to "anywhere" with certain rate. Consequently, when the host is sending to a closer destination, it can usually achieve better throughput than the target rate R_T, whereas it will fall slightly under expectations for longer RTT connections. Our goal is to explore what kind of assurance we can have for what range of RTT's. In Section IV-B we will demonstrate this network bias against long RTT connections with simulations, and explain how the "allocated-capacity" framework can alleviate such a bias.

The second challenge is to avoid TCP's retransmission timeouts. As we have discussed earlier, the fast-recovery phase of TCP provides much more controllable and quantitative adjustments of rates than the slow-start phase following a retransmission timeout. In the current version of TCP, fast-recovery phase is maintained so long as TCP only suffers one or two packet drops within one RTT. Once a TCP connection is sending above its target rate R_T, the profile meter starts to tag packets as *out* packets. If a cluster of packets are tagged as *out*, the likelihood of them getting dropped together is also high. This will drive TCP into a slow-start phase and create undesirable consequences. Our solution is to introduce a probabilistic function while tagging packets as *out*. The probabilistic function will space out packets that are tagged as *out*, reducing the probability of going into slow starts.

Both of the above difficulties are consequences of the fact that the profile meter is separated from the host's TCP and, hence, has no knowledge of RTT or any other internal state information of the TCP. If, instead, the profile meter could be integrated with the host's TCP code, then we can precisely avoid the above difficulties.[6] When this is not feasible, our framework can provide better assurances by keeping the TCP's strictly in the fast-recovery phase. For example, in the receiver-based scheme, there will be no packet drops. Instead of *inferring* from a packet drop that congestion had occurred, the sending TCP can receive explicit congestion notification, reduce its window size appropriately, and operate in the fast-recovery phase most of the time. A new version of TCP [TCP with selective ACK (TCP-SACK)] has similar properties and can work well in our framework.

IV. Simulations Results

A. Simulation Setup

We use the ns [17] network simulator from Lawrence Berkeley National Laboratory (LBNL) for our simulations.

[5]Of course, the actual RTT depends on the queueing delay caused by congestion during the transmission, but ISP's usually have a crude estimate.

[6]This profile meter, of course, should be augmented with a "checking" profile meter on the access link to make sure that the host isn't cheating. Section IV-B explores this topic.

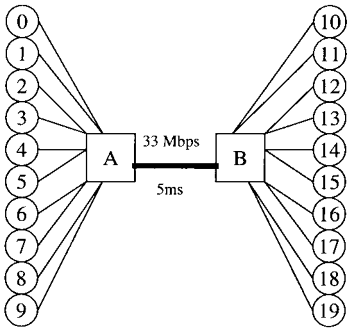


Fig. 5. Ten-connection case.

TABLE I
Comparison of RED and RIO-TSW for Ten-Connection Scenario. Link $BW = 33$ Mb/s, Parameters for RED (10, 30, 0.02), Parameters for RIO (40, 70, 0.02) for *ins* and (10, 30, 0.2) for *outs*, Used TCP-RENO

Conn #	RTT (ms)	RED routers (Mbps)	R_T (Mbps)	with RIO-TSW (Mbps)
0	20	7.04873	1	2.27289
1	20	6.22214	5	5.7619
2	40	2.83662	1	1.28011
3	40	2.28316	5	5.26757
4	50	2.62307	1	1.21957
5	50	2.81556	5	5.18823
6	70	1.61073	1	1.34831
7	70	1.57837	5	4.12794
8	100	1.64488	1	0.996326
9	100	1.85132	5	4.12563
total		30.51458		31.588476

We use a simple topology with ten hosts connecting to their respective destinations via one common ISP. Fig. 5 shows the topology. The ten connections, each with a profile meter, share a common bottleneck of 33 Mb/s, whereas the total contracted profiles are 30 Mb/s. The connections have different RTT's, ranging from 20 to 100 ms. They are grouped into five pairs. Each pair has a connection with target rate R_T of 1 Mb/s, and another connection with R_T of 5 Mb/s. We experiment with both sender-based and receiver-based schemes, with different versions of TCP, and with how to deal with nonresponsive flows. All TCP connections run for 20 s unless otherwise noted. Due to limited space, we abbreviate results as the average throughput achieved by the TCP receiver after TCP has reached stable state, and present them in tables.

B. Comparison with TCP in Today's Internet (Sender-Based, TCP-Reno)

Table I compares the throughput of the ten connections in the current Internet environment and in the "allocated-capacity" framework. The RTT's of ten connections, ranging from 20 to 100 ms, are listed in column 2. Column 3 lists the throughput that these ten TCP (TCP-Reno) connections can achieve in today's Internet from a particular simulation run. The network bias against long RTT connections is evident. Graphs of TCP window oscillations (not shown) are usually drastic and unpredictable. As a result, the throughput that TCP-

Reno can achieve is usually unpredictable, e.g., connections 2 and 3 have the same RTT but differ significantly in their throughput (24%). The last two columns list relevant information for the "allocated-capacity" framework: column 4 lists the target rates, or R_T, for the ten connections, and column 5 is the throughput achieved by the TCP after having adopted a profile meter. The total link throughputs in both cases are comparable: 30.51 versus 31.6 Mb/s, or 92.5 versus 96% link usage.

In comparing the two cases we observe the following—all other things being equal, the network bandwidth in the current Internet is distributed according to the RTT's of the connection and is strongly in favor of connections with short RTT's. The throughput achieved by TCP is subject to circumstances of router congestion and can be very unpredictable, especially when slow-start phase is triggered. In contrast, the "allocated-capacity" framework can allocate network capacity according to the service allocation profiles for which the users have contracted. For TCP's with the same service allocation profile, e.g., connections 1 and 9, the disadvantage that long RTT connections have is still visible, but it has been significantly mitigated (though not corrected). Most importantly, now the system can provide quite different expected capacities to different connections with reasonable assurance.

C. Receiver-Based (TCP-Reno with ECN)

In the same format, Table II lists the results from the receiver-based scheme. The configuration of the system is comparable to that of the sender-based scheme. The advantage that short RTT connections have in today's network is pronounced in column 3. Results in column 5—the throughput achieved by the ten connections, respectively—lead us to similar conclusions that the "allocated-capacity" framework can allocate bandwidth according to the target rates R_T in the times of congestion, and there is a strong discrimination against connections with small R_T. The overall performance is slightly better than that of the sender-based scheme: 32.88 versus 30.51 Mb/s (Table I). In the receiver-based scheme, since there are no packet drops, there are no retransmits, and TCP operates mostly in the fast-recovery phase, adjusting its windows gracefully. This attribute also makes the system more predictable in allocating bandwidth. In the receiver-based schemes the link usage is high: 32.88 Mb/s (99.6% link utilization) for using RED routers, and 32 Mb/s (97%) for using combined RIO routers and profile meters. The ECN scheme provides an elegant way of controlling the sending rate of TCP.

It is important to realize that in both sender-based and receiver-based schemes, network bias against long RTT connections is not totally eliminated because the profile meters are on the access path. The profile meters, not knowing the instantaneous RTT of TCP, have to use a presumed RTT to calculate its $\mathrm{Win\_length}$ variable, discriminating against connections with longer RTT's than the presumed value. Conversely, it gives an advantage to connections with shorter RTT's than the presumed value. Such bias limits the chances of predictability for long RTT connections in our framework.

TABLE II
RECEIVER-BASED, TCP-RENO WITH ECN SEMANTICS, $BW = 33$ Mb/s, RED ROUTER WITH PARAMETERS (15, 40, 0.02), EXCEPT IT DOES NOT DROP PACKETS, BUT ONLY SETS ECN BITS WHEN CONGESTED

Conn #	RTT (ms)	RED routers (Mbps)	R_T (Mbps)	with RIO-TSW (Mbps)
0	20	6.18994	1	2.71758
1	20	5.69154	5	5.38844
2	40	3.30375	1	1.65922
3	40	4.06525	5	5.02699
4	50	2.67894	1	1.3496
5	50	3.19256	5	4.83154
6	70	2.16772	1	1.04867
7	70	2.28335	5	4.7517
8	100	1.84308	1	0.868532
9	100	1.46514	5	4.41489
total		32.88127		32.057

To put in another way, if the profile is to "anywhere," then there is a certain range of "anywhere" that is feasible with high assurance by our designed service allocation profile.

D. A Step into the Future: Working with TCP-SACK and Profile Meters in the Host

TCP-SACK [13] has very different semantics in its ACK packets. The sending TCP has precise information on the received or lost packets and can make correct decisions about retransmissions and window adjustments. It can recover multiple packet loss in a window and remain in the fast-recovery phase. Similar to the receiver-based scheme, when the host is using TCP-SACK, the "allocated-capacity" framework can provide different levels of services with high assurance, since no unexpected slow starts can cause large variations in throughput.

The limitation in our TSW profile meter is that it does not know the instantaneous RTT of the TCP connections and therefore has to presume one. This limitation can be eliminated if the profile meter is implemented in TCP itself, which has a good estimate of the instantaneous RTT. In this case an additional "checking" profile meter will have to be installed at the connecting ISP to ensure that the host is not cheating by sending more *in* packets than it is promised. The corresponding "checking meter" on the access link from the host to ISP can be designed simply. Inside the host the "policy meter," knowing the RTT, can insure differential best-effort service to connections with a much longer range of RTT's.

Table III lists the results of simulation of the above two cases. The results are presented in one table to save space. The left half of the table shows the results of "allocated-capacity" framework versus using RED routers only when the host TCP has been upgraded to TCP-SACK. Though the achieved throughput is only slightly better than those in the TCP-Reno case, the predictability is much higher. In other words, the results with TCP-SACK are much more consistent and with small variation. The TCP window graphs in simulations show perfect sawtooth behaviors, and no slow starts (Appendix B includes two TCP window graphs for

TABLE III
WITH TCP-SACK TO RECOVER MULTIPLE PACKET LOSSES. PARAMETERS FOR RED ARE (10, 30, 0.02) AND PARAMETERS FOR RIO ARE (40, 70, 0.02) AND (10, 30, 0.5). $Bw = 33$ Mb/s, USED TCP-SACK

Conn #	RTT (ms)	RED (Mbps)	R_T (Mbps)	In Allocated Capacity (Mbps)		RTT (ms)	Tagger outside host (Mbps)	R_T (Mbps)	Tagger in host (Mbps)
0	20	6.74918	1	1.33071		16	1.5981	1	1.69298
1	20	6.47331	5	6.13875		16	6.3969	5	5.95957
2	40	2.64113	1	1.2283		50	1.18279	1	1.207
3	40	3.09084	5	5.38146		60	5.21947	5	5.32791
4	50	2.17542	1	1.13145		90	0.856795	1	1.03357
5	50	2.65581	5	5.20269		100	4.62787	5	5.15628
6	70	1.75715	1	0.988921		120	0.674382	1	1.03703
7	70	1.90942	5	4.92265		130	4.71473	5	5.17554
8	100	1.12921	1	0.899915		150	0.622031	1	0.974896
9	100	1.49762	5	4.68653		160	4.94274	5	4.96005
total		30.0791		31.9114			30.83581		32.524

TCP-Reno and TCP-SACK, respectively). The right half of the table lists the results from using a "policy meter" inside the host, using the TCP's estimate of RTT's to change the Win_length variable dynamically. We deliberately make the range of RTT's bigger to make our point: the RTT's are from 16 to 150 ms, approximating communications within a city and across continental U.S., respectively. TCP in the hosts (TCP-SACK) tag the outgoing packets by using the same TSW algorithm that we mentioned before. For comparison, column 8 lists throughput of TCP-SACK for this range of RTT's when the profile meters are outside the host, therefore not knowing the respective RTT's. Results in column 10 are both better and more predictable, as we had expected.

E. Dealing with Nonresponsive Connections

Nonresponsive connections are those connections that do not have any congestion avoidance mechanisms and do not slow down when their packets are dropped at the routers. In the current Internet, in the presence of nonresponsive connections, TCP—in fact, any transport-layer protocol that implements congestion avoidance mechanisms—is at a disadvantage. While TCP backs off upon detecting congestion, nonresponsive connections will get their packets through while continuing to cause congestion. The currently proposed ways of dealing with nonresponsive connections includes using a fair-queueing mechanism to isolate different connections from each other, and using some kind of "penalty box" to identify and isolate nonresponsive connections, as recently proposed by [14].

The "allocated-capacity" framework provides simple mechanisms to shield users as well as ISP's from nonresponsive sources in two ways. First, when a user has a service allocation profile, the *in* packets he sends are far less likely to be dropped in times of congestion, which implicitly shields him from the *out* packets. Second, when a disproportional number of *out* packets being dropped are from the same source, the router can take that as an accurate indication that this source is nonresponsive. In addition, instead of examining the history of all dropped packets, as proposed in [14], to identify nonresponsive connections, in the "allocated-capacity" framework, the routers only need to look at the history of *out* packet drops, in which nonresponsive connections are overrepresented.

TABLE IV
TEN-CONNECTION CASE WITH NONRESPONSIVE CONNECTION (CBR). $BW = 33$ Mb/s, CBR IS SENDING AT 6 Mb/s. RIO PARAMETERS: (40, 70, 0.02) FOR *ins* AND (10, 30, 0.5) FOR *outs*. USED TCP-SACK

Conn #	RTT (ms)	Today's Internet (Mbps)	R_T (Mbps)	with RIO-TSW (Mbps)
0	20	5.40752	1	1.44003
1	20	5.36329	5	5.21102
2	40	1.98478	1	1.07188
3	40	2.56938	5	5.01237
4	50	2.443	1	1.23827
5	50	2.54567	5	4.76236
6	70	1.28912	1	1.0332
7	70	1.53377	5	4.47648
8	100	1.1074	1	0.817773
9	100	1.50127	5	4.3094
CBR	50	5.85338	0	2.61297
total		31.59858		31.9857

In simulation we use a constant-bit-rate (CBR) source to model nonresponsive sources. We add a CBR connection to the above scenario, with a sending rate of 6 Mb/s, or roughly 20% of the total bandwidth. Table IV lists the throughputs in both the current Internet and the "allocated-capacity" framework, with the hosts using TCP-SACK.

In today's Internet this nonresponsive connection will inflict consistent congestion in the router, causing all responsive TCP connections to back off. Column 3 illustrates this effect—the CBR gets almost all of its packets through at the expense of the TCP's performance. Connections 0–9 all suffer performance loss, compared to column 3 in Table III, where the CBR is absent. With the "allocated-capacity" framework, connections with service allocation profiles are protected from the CBR: the link bandwidth is allocated according to the contracted service allocation profile while packets from CBR are severely dropped. The CBR connection receives 2.61 Mb/s or 43.5%

of its 6-Mb/s sending rate in our framework, versus 5.85 or 97% of its sending rate in today's Internet.

F. Extensive Simulations and Future Work

We have done extensive simulations on both sender-based and receiver-based in one-way and two-way traffic. We have also simulated scenarios where there is a mix of bursty traffic and bulk-data transfer traffic, and cascading traffic profiles. The result of the simulations can be found in [8]. It should be noted that the case we presented—"average throughput to everywhere"—is a harder case than "average throughput for a source destination pair," so that the "allocated-capacity" framework can provide for a simpler service allocation profile with high assurance. Our future work includes implementing and testing our algorithms in a real testbed. From simulations alone, we conclude that with the "allocated-capacity" framework, TCP's, especially newer version of TCP's, can achieve different throughput with high assurance.

V. Conclusions

Key to the success of the Internet is its high degree of traffic aggregation among a large number of users, each of whom has a very low duty cycle. Because of the very high degree of statistical sharing, the Internet makes no commitment about the capacity that any user will actually receive. It does not make separate capacity commitments to each user.

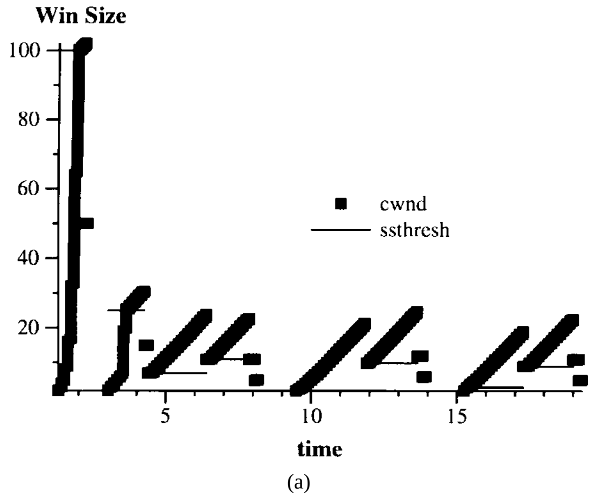
We conclude that while the mechanisms in the Internet seem to work today, a valuable service enhancement would be a means to distinguish and separately serve users with very different transfer objects, so that each could be better satisfied. This paper suggests that instead of allocating capacity to users by explicit reservations, we should take a much simpler step—using service allocation profiles to separate demands into those within the profiles (*in*s) and those outside the profiles (*out*s), and dealing with the delivery of *in* packets as a matter of provisioning. We argue that the users not only want differential services but also higher predictability than what the current Internet can provide. The proposed "allocated-capacity" framework provides mechanisms for allocating different levels of services with high predictability. Since the service allocation profiles represent how resources are allocated when they are in demand, they are a rational basis for cost allocation. With the current Internet facing the imminent "tragedy of the commons," a basis for cost allocation can alleviate congestion, utilize the existing resources more efficiently, and fund further growth of Internet infrastructure.

The mechanism proposed here, which is the discrimination between packets marked as *in* and *out* for congestion pushback at times of overload, represents an example of the separation of mechanism and policy. It is capable of implementing a wide range of policies for allocation of capacity among users. It allows providers to design widely different service and pricing models, without having to build these models into all of the packet switches and routers of the network. The mechanisms that must be agreed upon and implemented globally are the format of the control flags in packets and the differential treatments of *out* packets in the system. In contrast, the service allocation profiles will change and adapt to needs of future applications and business models of ISP's, and will only affect the edge of the network. This design thus pushes most of the complexity to the edge of the network, making it scalable and flexible.

```
For each packet arrival
    if it is an In packet
        calculate the average In queue size avg_in;
    calculate the average queue size avg_total;

If it is an In packet
    if min_in < avg_in < max_in
        calculate probability $P_{in}$;
        with probability $P_{in}$, drop this packet;
    else if max_in < avg_in
        drop this packet.
if this is an Out packet
    if min_out <avg_total < max_out
        calculate probability $P_{out}$;
        with probability $P_{out}$, drop this packet;
    else if max_out <avg_total
        drop this packet.
```

Fig. 6. RIO algorithm.



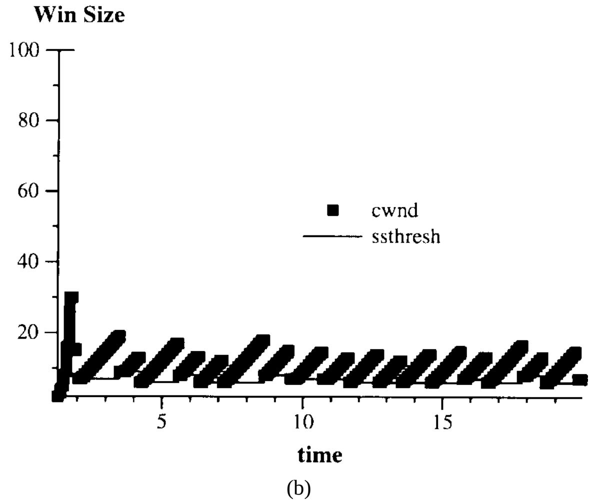


Fig. 7. Win size. (a) Congestion window 9 (Reno_10_flows) and (b) congestion window 9 (Sack_10_flows).

Appendix A
RIO Algorithm

The RIO algorithm is shown in Fig. 6.

Appendix B
Congestion Window Oscillations for TCP-RENO and TCP-SACK in "Allocated-Capacity" Framework

Fig. 7(a) and (b) shows that the TCP congestion window $cwnd$ changes over time for connection #9 ($RTT = 100$ ms). The connection uses TCP-Reno in Fig. 7(a) and uses TCP-SACK in Fig. 7(b). TCP-SACK can recover multiple drops gracefully and keep the window oscillating in a perfect sawtooth fashion, whereas TCP-Reno's window swings are more drastic. See Fig. 7.

Acknowledgment

The authors would like to thank J. Wroclawski, P. Zayas, and R. Cheng for their valuable input and feedback during the course of this work. W. Fang would especially like to thank members of the Advanced Network Architecture Group at MIT for many insightful discussions. This material does not reflect the position or policy of the U.S. Government, and no official endorsement should be inferred.

References

[1] K. Bala, I. Cidon, and K. Schraby, "Congestion control for high-speed packet switched networks," in *Proc. INFOCOM'90*, San Francisco, CA, 1990, pp. 520–526.
[2] R. Cheng, "Receiver and sender payment of differential services in the Internet," Massachussets Inst. Technol., Cambridge, Tech. Rep., 1998, to be published.
[3] D. Clark, S. Shenker, and L. Zhang, "Supporting real-time applications in an integrated services packet network: Architecture and mechanism," in *Proc. ACM SIGCOMM'92* Baltimore, MD, 1992, pp. 14–26.
[4] D. Clark, "Combining sender and receiver payment schemes in the Internet," in *Interconnection and the Internet: Selected Papers from 1996 Telecom Policy Research Conf.*, G. L. Rosston and D. Waterman, Eds. Mahwah, NJ: Elrbaum, 1996, pp. 95–112.
[5] ——, "Internet cost allocation and pricing," in *Internet Economics*, L. McKnight and J. Bailey, Eds. Cambridge, MA: MIT Press, 1997, pp. 215–253.
[6] A. Demers, S. Keshave, and S. Shenker, "Analysis and simulations of a fair queueing algorithm," in *Proc. ACM SIGCOMM'89*, Austin, TX, 1989, pp. 1–12.
[7] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno, and SACK TCP," *Comput. Commun. Rev.*, vol. 26, no. 3, pp. 5–21, July 1996.
[8] W. Fang. "Simulation results of bulk-data transfer in the 'expected capacity' framework," Princeton Univ., Princeton, NJ, Tech. Rep. [Online]. Available FTP: mercury.lcs.mit.edu Directory: /pub/wfang
[9] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched routers," *Internetworking: Res. Exp.*, vol. 3, no. 3, pp. 115–156, Sept. 1992.
[10] ——, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Networking*, vol. 3, pp. 397–413, Aug. 1993.
[11] S. Floyd, "TCP and explicit congestion notification," *Comput. Commun. Rev.*, vol. 24, no. 5, pp. 10–23, Oct. 1994.
[12] —— (Oct. 1996). "Ns simulator tests for random early detection (RED) gateways." [Online]. Available: http://www-nrg.ee.lbl.gov/ns
[13] ——. (Mar. 1996). "Issues of TCP with SACK," Lawrence Berkeley Nat. Lab., Tech. Rep. [Online]. Avalable FTP: ftp://ftp.ee.lbl.gov Directory: /papers/issues.sa.ps.Z
[14] S. Floyd and K. Fall, "Router mechanisms to support end-to-end congestion control," [Online]. Available www: http://www-nrg.ee.lbl.gov/nrg-papers.html
[15] A. Gupta, D. Stahl, and A. Whinston, "Priority pricing of integrated services networks," in *Internet Economics*, L. McKnight and J. Bailey, Eds. Cambridge, MA: MIT Press, 1997, pp. 253–279.
[16] J. Hoe, "Improving the start-up behaviors of a congestion control scheme for TCP," in *Proc. ACM SIGCOMM'96*, Stanford, CA, 1996, pp. 270–280.
[17] Ns [Online]. Available www: http://www-nrg.ee.lbl.gov/ns/
[18] V. Jacobson, "Congestion avoidance and control," in *Proc. ACM SIGCOMM'88*, Stanford, CA, 1988, pp. 314–329.
[19] J. MacKie-Mason and H. Varian, "Economic FAQ's about the Internet," in *Internet Economics*, L. McKnight and J. Bailey, Eds. Cambridge, MA: MIT Press, pp. 27–63.
[20] L. Zhang, S. Shenker, and D. Clark, "Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic," in *Proc. ACM SIGCOMM'91*, Zurich, Switzerland, Sept. 1991, pp. 133–148.
[21] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource ReSerVation protocol," *IEEE Network*, vol. 7, pp. 8–18, Sept. 1993.

David D. Clark (F'98) received the B.S.E.E. degree from Swarthmore College, Swarthmore, PA, in 1966, and the Ph.D. degree from Massachusetts Institute of Technology, Cambridge, in 1973.

Since 1973 he has been with the Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, where he is currently a Senior Research Scientist in charge of the Advanced Network Architecture Group. After receiving his Ph.D., he worked on the early stages of the ARPAnet and on the development of token ring local area network technology. Since the mid-1970's he has been involved in the development of the Internet—from 1981 to 1989 he acted as Chief Protocol Architect in this development, and he chaired the Internet Activities Board. His current research area is protocols and architectures for very large and very high-speed networks. Recent activities include extensions to the Internet to support real-time traffic, explicit allocation of service, pricing, and new network technologies. He was a major author of two studies by the Computer Science and Telecommunications Board of the National Research Council on information infrastructure.

Dr. Clark is a member of the Association for Computing Machinery (ACM) and the National Academy of Engineering. He is Chairman of the Computer Science and Telecommunications Board of the National Research Council. He received the ACM SIGCOMM Award, the IEEE Award in International Communications, and the IEEE Hamming Award for his work on the Internet.

Wenjia Fang was born in Beijing, China. She received the B.S. degree in computer science and engineering from the University of Pennsylvania, Philadelphia, in 1994, and is currently working toward the Ph.D. degree in computer science at Princeton University, Princeton, NJ. She has done research at Bell Laboratories, NEC Laboratories, and Sun Microsystems, and she attended TsingHua University prior to coming to the United States.

She is currently a Visiting Scholar working on network architecture issues with the Advanced Network Architecture Group, Laboratory for Computer Science, Massachussets Institute of Technology, Cambridge.