Explicit Allocation of Best Effort Packet Delivery Service

David D. Clark — Laboratory for Computer Sciences, Massachusetts Institute of Technology

Wenjia Fang — Computer Science Department, Princeton University

{ddc, wfang}@lcs.mit.edu

Abstract: *This paper presents the "Expected Capacity" framework for providing different levels of best-effort service in times of network congestion. The "Expected 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 complimentary 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 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 TCP traffic to achieve certain targeted sending rates.

1.0 Introduction

This paper describes a new framework — the "Expected 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 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 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 “Expected 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 expectations, 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 2 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 3 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 4 presents results using the above algorithms in simulated environments for bulk-data transfers. The simulations show that the “Expected 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 non-responsive users at aggregation points. Section 5 concludes our work.

1.1 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 packet granularity in their "smart market" scheme [MacVar]. In this scheme, each packet carries a bid, a price that the user is willing to pay for service. At each point of congestion, all 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. 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. Third, in order to get packets through, the user either has to know the bid for each transfer ahead of time, or he will have to bid dynamically after receiving some feedback from the market. In a real network, neither is realistic. In addition, the computation needed for clearing the bids and accounting in each gateway is likely to be prohibitive.

Gupta et. al. [Gupta] 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 gateways 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 Queuing [Demers, Clark92] creates different queues for different connections, and assures each connection will receive some share of the bandwidth. This mechanism allocates bandwidth among all connections within a gateway, but does not by itself address how many connections each user has, and how they interact. In addition, it is not clear this scheme is scalable in the center of the network where the gateways have a large of amount traffic connections aggregated.

The idea of tagging packets as *In* or *Out* is not a new one. For example, researchers at IBM [Bala] proposed tagging as part of a flow control scheme. Frame Relay has the concept of *In/Out* packets as does ATM: the Cell Loss Preference, or CLP bit. Those ideas were proposed in the context of a specific reserved flow or virtual circuit from a source to a destination. [Clark97] applied the idea to a packet switched network where there is no implication that the expected capacity for any user is reserved along a particular path. Profile meters tag packets based on contracted profiles between ISPs and customers. The network preferentially drops *Out* packets during periods of congestion. As a consequence, the ISPs can offer different levels of service based on these profiles. [Clark97] 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 non-responsive connections; 3) demonstrates the effectiveness of the framework with simulation results.

2.0 The "Expected Capacity" framework

2.1 Overview

The general approach of this mechanism is to define a *service profile* for each user, and to design a mechanism in the router that favors traffic that is within those service profiles. The core of the idea is very simple – monitor the traffic of each user as it enters the network, and tag packets as being "*In*" or "*Out*" of their service profiles. Then at each router, if congestion occurs, 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 [Zhang93] or Weighted Fair Queuing, 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 3.1 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 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".

2.2 Location of profile meters in the network

Figure 1 illustrates the "Expected Capacity" framework with a sender-based control. All the gateways (*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 LANs to the peering relationships between global ISP's. Any such boundary may be an appropriate place for a profile meter, e.g., M3 to M6 in figure 1.

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 of it. 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 our model, we use preferential drops. In this paper, we only focus on using preferential drops of packets, and placing both "*In*" and "*Out*" packets in the same queue.

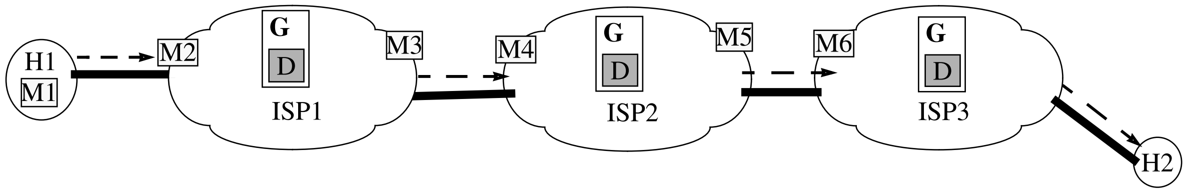


Figure 1. the "Expected Capacity" framework (sender-based)
Host 1, H1 has a sender-based profile, and is sending traffic to host H2 (dotted line). The traffic traverses three ISPs. The gateways, G's, in the figure are all augmented with preferential dropping algorithms, Ds. There are profile meters,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

Furthermore, the packet tagging associated with this service 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 figure 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 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 IP, for example, can serve to regulate the relative use of the network by individual flows. Whereas 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.

2.3 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 profiles at the edge of the network from the differential dropping in the center of the network allows this flexibility. To oversimply, 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 ISPs, 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 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 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 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 3, we will describe algorithms to implement such a profile.

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

- *Traffic specifications: what exactly is provided to the customer? (For example, 5Mbps average throughput).*
- *Geographic scope: to where this service is 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 one megabit per second" to a specific destination than to anywhere in the Internet.

2.4 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 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 profiles the “expected” 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 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., 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 received 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.

2.5 Receiver-controlled scheme

The tagging scheme described above implements a model in which the sender, by selecting one or another service 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 “Expected 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. In ECN semantics, congested gateways 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 scheme, gateways will not be modified; they will turn <u>on</u> 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 profile. If there is enough profile to cover all arriving packets, the meter will turn <u>off</u> 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 <u>on</u> will be left unchanged at the profile meter. If packets arrive at the TCP receiver with ECN bits still <u>on</u>, it means that the receiver has not contracted sufficient capacity to cover all the packets that encountered congestion, and the sender will be notified to slow down.

2.5.1 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 user for actual usage during congestion.

On the other hand, the receiver scheme is more indirect in its ability to respond to congestion. Since a packet carries the explicit assertion of whether it is *In* or *Out* of profile in the sender scheme, 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 gateway must set the ECN bit, and still attempts to forward the packet, trusting 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.

3.0 "Expected Capacity" for bulk data transfers

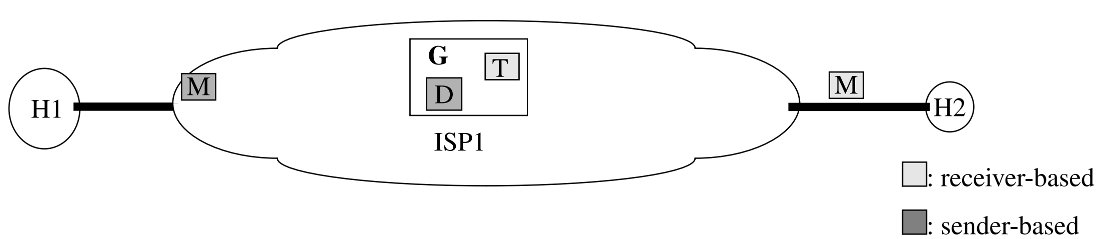


Figure 2. Simplified "Expected Capacity" framework

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 gateways in ISP1. There are no cascaded profile meters.

It is important to realize that the dropping algorithm in the routers, once adopted, is unlikely to change again over time; however, the service 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 co-operate 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 figure 2. There is no cascading of profile meters. The service profiles 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 ranging from 20ms to 100ms (which is roughly comparable to metropolitan connections and cross-U.S. 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 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 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 non-responsive source persistently congests the gateway.

3.1 TCP rate adjustment in the current Internet

In today's Internet, rate adjustments are accomplished by both the end host transport layer TCP and the queue management algorithm in the gateways. We will discuss both in turn.

The mechanisms used by TCP to deal with congestion are based on [Jacobson]. 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 retranmission, increases its window size by one packet each round trip time. 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 some amount up to half. The second mode is called “Slow-Start”, and typically occurs when a large number of packets are lost. Current implementations of TCP fail to recover multiple packet losses within a window, and have to rely on the retransmission timer to recover. When multiple packet losses occur, current TCP implementations usually remain silent until the retransmission timer goes off, and then enters “Slow-Start” mode. This has a more drastic effect on the TCP performance. First, the retransmission timer is crude, usually measured in a granularity of 500ms, 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, with a much reduced Slow-Start threshold, *ssthresh*. Since *ssthresh* is what TCP perceives as the optimal operating point, and TCP opens up window linearly after its window size reaches *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, gateways deal with congestion by dropping packets. Each time a packet is dropped, it causes a rate adjustment in one of the sending TCPs. 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, TCP stays in a phase with quantifiable rate adjustments and is much more controllable.

3.2 Differential dropping in the routers: RIO

RIO stands for Random Early Drop (RED) gateways with *In/Out* bit. RED gateways [Floyd93] keep the overall throughput high while maintaining a small average queue length, and tolerate transient congestion without causing global synchronization. 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 discriminate against *Out* packets. We will briefly describe the RED algorithm before presenting RIO.

3.2.1 RED algorithm

A RED gateway 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 gateway. Persistent congestion in the gateway is reflected by a high average queue size and a high dropping probability. The resulting high dropping probability will discard packets early, detect and control congestion.

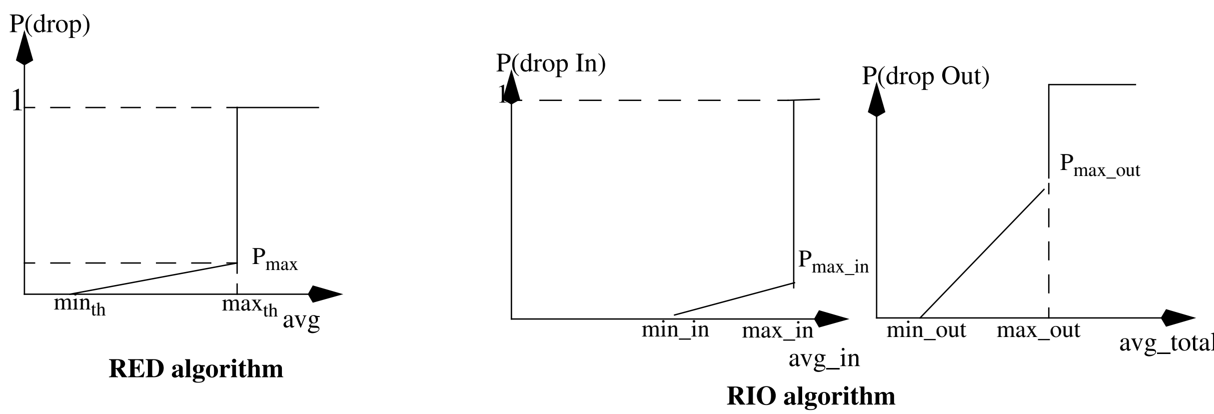


Figure 3. RED and RIO algorithms (Figures not drawn to scale)

A RED gateway is configured with the following parameters: min_{th}, max_{th}, and P_{max}. It works as illustrated in the leftmost figure in figure 3: the *x* axis is *avg*, the average queue size, which is calculated using a low-pass filter of instanenous 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 $[max_{th}, \infty)$, 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 gateway does not drop any packets. When the average queue size is between the two thresholds, the gateway 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}, which is usually caused by sustained large queue size, the gateway drops every arriving packet hoping to maintain a short queue size.

3.2.2 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 gateway, the gateway checks whether the packet is tagged as *In* or *Out*. If it is an *In* packet, the gateway calculates *avg_in*, the average queue for the *In* packets; the gateway calculates *avg_total*, average total queue size for *all (both In and Out)* arriving packets. The probability of dropping an *In* packet depends on *avg_in*, and the probability of dropping an *Out* packet depends on *avg_total*.

As illustrated in figure 3, there are three parameters for each of the twin algorithms. The three parameters, *min_in*, *max_in* and P_{max_in}, define the normal operation [0, *min_in*), congestion avoidance [*min_in*, *max_in*), and congestion control [*max_in*, ∞) phases for *In* packets. Similarly, *min_out*, *max_out*, and P_{max_out} defines the corresponding phases for *Out* packets.

The discrimination against *Out* packets in RIO is created by carefully choosing the parameters (*min_in, max_in,* P_{max_in}), and (*min_out, max_out,* P_{max_out}). As illustrated in two right figures in figure 3, a RIO gateway is more aggressive in dropping *Out* packets on three accounts: first, it drops *Out* packets much earlier than it drops *In* packets, this is done by choosing *min_out* smaller than *min_in*. Second, in the congestion avoidance phase, it drops *Out* packets with a higher probability, by setting P_{max_out} higher than P_{max_in}. Third, it goes into congestion control phase for the *Out* packets much earlier than for the *In* packets, by choosing *max_out* much smaller than *max_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 gateway 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 gateway is consistently operating in a congestion control phase by dropping *In* packets, this is a clear indication that the network is under provisioned. Appendix A contains the pseudo code for RIO algorithm.

The choice of using *avg_total*, the total average queue size, to determine the probability of dropping an *Out* packets is subtle. Unlike *In* packets, which the network can properly provision for, the *Out* packets represents the opportunistic traffic, and there is no valid indication of what amount of *Out* packet is proper. If we had used the average *Out* packet queue to control the dropping of *Out* packets, the choice for the corresponding three parameters is difficult, and has no direct intuitive correlation with those parameters for *In* packets. We could have used *avg_in*, the average queue for the *In* packets, to see how much "free space" the gateways has 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 gateways has a good control on the number of *Out* packets and total queue length. By using the *avg_total*, total average queue size, gateways can maintain short queue length and high throughput no matter what kind of traffic mix it has. 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.

3.3 Profile meters for bulk data transfers: TSW tagger

The profile meter we designed for bulk-data transfers is called Time Sliding Window (TSW) tagger. TSW taggers 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[1] 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.

1. The burstiness of TCP traffic is a well know phenomenon. Articulated in [Zhang91], it is caused by the fact that TCP paces out packets using window algorithm, and possible "ack compression" in two-way traffic.

A rate estimator is used to smooth out the burstiness of TCP traffic as well as to be sensitive to instanenous sending rate. TSW estimates the sending rate upon each packet arrival, and decays, or forgets, the past history over time[1]. 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_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_front* are updated each time a packet arrives, but *Win_length* is pre-configured when the profile meter is installed.

TSW rate estimator works as follows:

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.

Figure 4. TSW algorithm

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 $0.66R_T$ to $1.33R_T$ – and tag packets as *Out*

1. Though a low-past filter of instantaneous sending rate (packet size divided by inter packet arrival time) seems to be an obvious choice for the rate estimator, it suffers a flaw: it decays the sending rate over packet arrival, 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.

with $P = (avgrate - R_T)/(avgrate)$, 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 $1.33R_T$, 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 aggregated TCP traffic or other type of traffic. In our simulations, we use the second approach.

3.4 Difficulties in designing RIO-TSW

Our service 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 ISPs. 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 round trip times (RTTs). In the Fast-Recovery phase, TCP increases *cwnd* by one packet[1] each round trip time. Let *r* denote round trip time. Each round trip time, TCP increases its sending rate by $1/r$ pkts/sec, 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 connection has a higher average sending rate than the long RTT connection. This explains why a service profile with specific source destination pair is comparatively easier to implement, because its RTT is known[2]. As we discussed in section 3.1, 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

1. Real implementation of TCP increases window measured in bytes, instead of packets, we use packets for simplicity in explanation.
2. Of course, the actual RTT depends on the queueing delay caused by congestion during the transmission, but ISPs usually have a crude estimate.

all possible RTTs 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 RTTs. In section 4.2, we will demonstrate this network bias against long RTT connections with simulations, and explain how the "Expected Capacity" framework can alleviate such a bias.

The second challenge is to avoid TCP's retransmission time-outs. 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 time-out. 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 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[1]. When this is not feasible, our framework can provide better assurances by keeping the TCPs 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, 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

1. 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 4.2 explores this topic.

TCP (TCP-SACK or TCP with selective acknowledgment) has similar properties, and can work well in our framework.

4.0 Simulations results

4.1 Simulation setup

We use the *ns* [Ns] network simulator from LBL for our simulations. We use a simple topology with ten hosts connecting to their respective destinations via one common ISP. Figure 4 shows the topology. The ten connections, each with a profile meter, share a common bottleneck of 33Mbps, whereas the total contracted profiles are 30Mbps. The connections have different RTTs, ranging from 20ms to 100ms. They are grouped into five pairs. Each pair has a connection with target rate R_T of 1Mbps, and another connection with R_T of 5Mbps. We experiment with both sender-based and receiver-base schemes, with different versions of TCP, and with how to deal with non-responsive flows. All TCP connections run for 20 seconds 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.

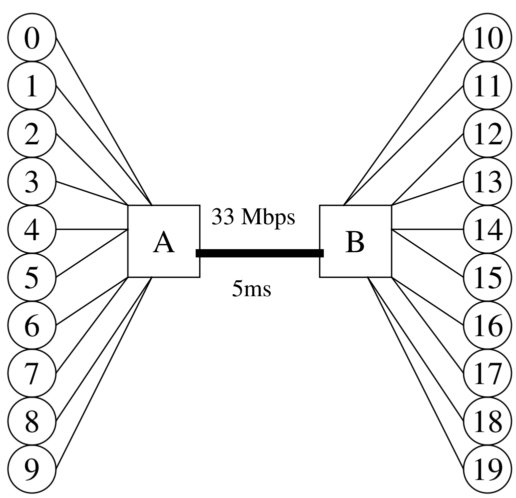


Figure 5. Ten-connection case

4.2 Comparison with TCP in today's Internet (sender-based, TCP-Reno)

Table 1 compares the throughput of the 10 connections in the current Internet environment and in the "Expected Capacity" framework. The RTTs of 10 connections, ranging from 20ms to 100ms, are listed in column 2. Column 3 lists the throughput these 10 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 TCP-Reno can achieve is usually unpredictable, e.g., connections 2 & 3 have the same RTT, but differ significantly in their throughput (24%). The last two columns list relevant information for the "Expected Capacity" framework: column 4 lists the target rates, or R_T, for the 10 connections, and column 5 are the throughput achieved by the TCP after having adopted a profile meter. The total link throughputs in both cases are comparable: 30.51Mbps vs. 31.6Mbps, or 92.5% vs. 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 RTTs of the connection, and strongly in favor of connections with short RTTs. The throughput achieved by TCP is subject to circumstances of gateway congestion and can be very unpredictable, especially when Slow-Start phase is triggered. In contrast, the "Expected Capacity" framework can allocate network capacity according to the service profiles the users have contracted for. For TCPs with the same service profile, e.g. connections 1 and 9, the disadvantage 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.

TABLE 1. Comparison of RED and RIO-TSW for ten-connection scenario. Link Bw =33Mbps. Parameters for RED (10, 30, 0.02); parameters for RIO (40,70, 0.02) for *In*s and (10, 30, 0.2) for *Out*s. Used TCP-Reno

Conn #	RTT (ms)	RED gateways (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

4.3 Receiver based (TCP-Reno with ECN)

TABLE 2. Receiver-based. TCP-Reno with ECN semantics. Bw = 33Mbps. The gateway is RED with parameters (15, 40, 0.02), except it doesn't drop packets, but only sets ECN bits when congested.

Conn #	RTT (ms)	RED gateways (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

In the same format, table 2 lists the results from the receiver-based scheme. The configuration of the system is comparable to that of the sender-based scheme. The advantage short RTT connections have in today's network is pronounced in column 3. Results in column 5 – the throughput achieved by the 10 connections respectively – lead us to similar conclusions that the "Expected 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: 32.88Mbps vs. 30.51Mbps (table 1). 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: 32.88Mbps (99.6% link utilization) for using RED gateways, and 32Mbps (97%) for using combined RIO gateways and profile meters. The Explicit Congestion Notification (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 *win-length* variable, discriminating against connections with longer RTTs than the presumed value. Conversely, it gives an advantage to connections with shorter RTTs than the presumed value. Such bias limits the chances of predictability for long RTT connections in our framework. 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 profile.

4.4 A step into the future: working with TCP-SACK and profile meters in the host

TCP with selective acknowledgment, or TCP-SACK [Floyd96b], has very different semantics in its acknowledgment 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 “Expected 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 round trip time. 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 RTTs.

Table 3 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 “Expected Capacity” framework vs. using RED gateways 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 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 RTTs to change *win-length* variable dynamically. We deliberately make the range of RTTs bigger to make our point: the RTTs are from 16ms to 150ms, approximating communications within a city and across continental United States, respectively. TCP in the hosts (TCP-SACK) tag the outgoing packets by using the same TSW algorithm we mentioned before. For comparison, column 8 lists throughput of TCP-SACK for this range of RTTs when the profile meters are outside the host, therefore not knowing the respective RTTs. Results in column 10 are both better and more predictable, as we had expected.

TABLE 3. 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=33Mbps. Used TCP-SACK.

Conn #	**RTT (ms)**	**RED (Mbps)**	**R_T (Mbps)**	**In Expected 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

4.5 Dealing with non-responsive connections

Non-responsive connections are those connections that do not have any congestion avoidance mechanisms and do not slow down when its packets are dropped at the gateways. In the current Internet, in the presence of non-responsive connections, TCP – in fact, any transport layer protocol that implements congestion avoidance mechanisms – is at a disadvantage. While TCP backs off upon detecting congestion, non-responsive connections will get their packets through while continuing to cause congestion. The currently proposed ways of dealing with non-responsive connection includes using Fair Queueing mechanism to isolate different connections from each other, and using some kind of "penalty box" to identify and isolate non-responsive connections, as recently proposed by [Floyd97].

The "Expected Capacity" framework provides simple mechanisms to shield users as well as ISPs from non-responsive sources in two ways. First, when a user has a service 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 gateway can take that as an accurate indication that this source is non-responsive. In addition, instead of examining the history of all dropped packets as proposed in [Floyd97] to identify non-responsive connections, in the "Expected Capacity" framework, the gateways only need to look at the history of *Out* packet drops, in which non-responsive connections are over-represented.

TABLE 4. 10-connection case with a non-responsive connection (CBR). BW= 33Mpbs, CBR is sending at 6Mbps. 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 non-responsive sources. We add a CBR connection to the above scenario, with a sending rate of 6Mpbs, or roughly 20% of the total bandwidth. Table 4 lists the throughputs in both the current Internet and the "Expected Capacity" framework, with the hosts using TCP-SACK.

In today's Internet, this non-responsive connection will inflict consistent congestion in the gateway, causing all responsive TCP connections to back off. Column 3 illustrates this effect: the CBR gets almost all its packets through at the expense of the TCP's performance. Connections 0 to 9 all suffer performance loss, compared to column 3 in table 3, where the CBR is absent. With the "Expected Capacity" framework, connections with service profiles are protected from the CBR: the link bandwidth is allocated according to the contracted service profile while packets from CBR are severely dropped. The CBR connection receives 2.61Mbps or 43.5% of its 6Mbps sending rate in our framework, vs. 5.85 or 97% of its sending in today's Internet.

4.6 Extensive simulations and future work

We have done extensive simulations on both sender-based, receiver-based in one-way and two-way traffic. We 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 [Fang97b]. 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 "Expected Capacity" framework can provide for a simpler service 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 "Expected Capacity" framework, TCPs, especially newer version of TCPs, can achieve different throughput with high assurance.

5.0 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 profiles to separate demands into those within the profiles (*In*s) and those outside the profiles (*Out*s), and reducing 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 “Expected Capacity” framework provides mechanisms for allocating different levels of services with high predictability. Since the service 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 the packet switches and gateways of the network. The mechanism that must be agreed upon and implemented globally is the format of the control flags in packets, and differential treatments of *Out* packets in the system. In contrast, the service profiles will change and adapt to needs of future applications and business models of ISPs, 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.

References:

[Bala] Bala, K., Cidon, I., and Schraby, K. 1990. *Congestion Control for High-speed Packet Switched Networks*. Proceedings of Infocom 1990. pp 520-526.

[Clark92] Clark. D., Shenker, S., and Zhang, L. 1992. *Supporting Real-Time Applications in an Integrated Services Packet Network: Architecture and Mechanism.* In Proceedings of ACM SIGCOMM '92, Baltimore.

[Clark96] Clark, D. 1996. *Combining Sender and Receiver Payment Schemes in the Internet*. Presented at Telecommunications Policy Research Conference, Solomon, MD

[Clark97] Clark, D., *Internet Cost Allocation and Pricing*, Internet Economics, McKnight, L. and Bailey, J. editors. MIT, 1997, pp215-253

[Demers] Demers, A., Keshave, S., and Shenker, S. 1989, *Analysis and Simulations of a Fair Queueing Algorithm*. Proceedings of ACM SIGCOMM '89

[Fall] Fall, K., and Floyd, S. *Simulation-based Comparisons of Tahoe, Reno, and SACK TCP.* Computer Communications Review, V. 26 N. 3, July 1996, pp. 5-21

[Fang97b] Fang, W. *Simulation Results of bulk-data transfer in the "Expected Capacity" framework*, MIT-Princeton Technical Report. To be published.

[Floyd92] Floyd, S., and Jacobson, V., *On Traffic Phase Effects in Packet-Switched Gateways*, Internetworking: Research and Experience, V.3 N.3, September 1992, p. 115-156

[Floyd93] Floyd, S., and Jacobson, V., 1993. *Random Early Detection Gateways for Congestion Avoidance*, IEEE/ACM Transactions on Networking, August 1993

[Floyd95] Floyd, S. *TCP and Explicit Congestion Notification*, Computer Communication Review, Vol 24:5, October, 1995

[Floyd96a] Floyd, S., *Ns Simulator Tests for Random Early Detection (RED) Gateways*, available via http://www-nrg.ee.lbl.gov/nrg-papers.html.

[Floyd96b] Floyd, S., *Issues of TCP with SACK*, Technical report, Mar, 1996. ftp://ftp.ee.lbl.gov/papers/issues.sa.ps.Z

[Floyd97] Floyd, S., and Fall, K., *Router Mechanisms to Support End-to-End Congestion Control*, available via http://www-nrg.ee.lbl.gov/nrg-papers.html.

[Gupta] Gupta, A., Stahl, D., and Whinston, A. 1997. *Priority Pricing of Integrated Services Networks*. Internet Economics, ed. Lee McKnight and Joseph Bailey. pp 253-279

[Hoe] Hoe, J. *Improving the Start-up Behaviors of a Congestion Control Scheme for TCP*. Proceedings ACM SIGCOMM, '96. Stanford, CA.

[Ns] Ns. Available via http://www-nrg.ee.lbl.gov/ns/.

[Jacob88] Jacobson, V. 1988. *Congestion Avoidance and Control*. Proceedings of ACM SIGCOMM '88, Stanford, CA.

[MacVar] MacKie-Mason, J. and Varian H. *Economic FAQs about the Internet.* Internet Economics, ed. Lee McKnight and Joseph Bailey. pp 27-63

[Zhang91] Zhang, L., Shenker, S., and Clark, D. Observations on the dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. Proceedings of ACM SIGCOMM '91.

[Zhang93] Zhang, L., Deering, S., Estrin, D., Shenker, S., and Zappala, D. RSVP: A New Resource ReSerVation Protocol. IEEE Network, September 1993

Appendix A: RIO algorithm.

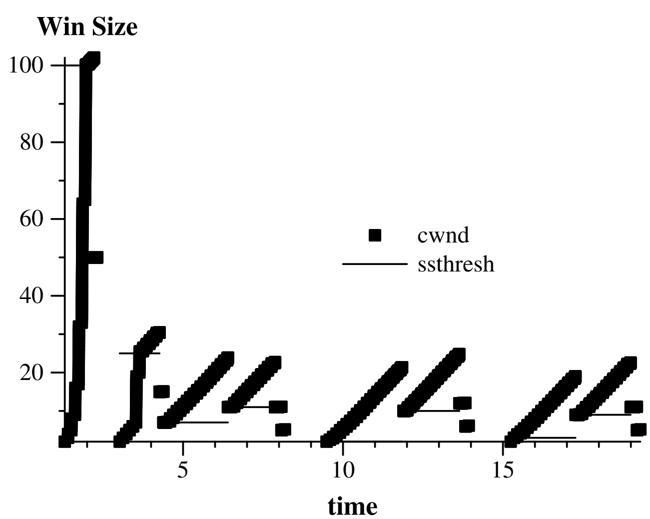
```
For each packet arrival
    if it is an In packet
        calculate the average In queue size avg_in ;
    calculate the average queue size avg_total ;

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.
```

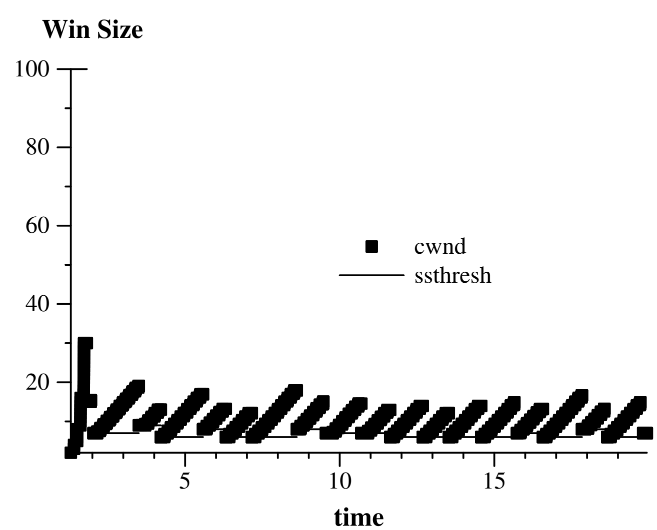
Figure 6. RIO algorithm

Appendix B: Congestion window oscillations for TCP-Reno and TCP-SACK in "Expected Capacity" framework.

The following graphs show the TCP congestion window, *cwnd*, changes over time for connection #9 (RTT=100ms). The connection uses TCP-Reno in the left graph, and uses TCP-SACK in the right graph. 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.



Congestion Window 9 (Reno_10_flows)



Congestion Window 9 (Sack_10_flows)