Princeton University
Computer Science Department
TR-601-99

The "Expected Capacity" Framework: Simulation Results

Wenjia Fang
Princeton University

January, 1998

Table of Contents

CHAPTER 1

Introduction

Abstract: The current Internet uses the "best-effort" service model, where the network tries to deliver packets at best it can. The "best-effort" model cannot meet the more sophisticated demand of emerging applications, and mechanisms for creating different levels of services are needed. The "Expected Capacity" framework provides schemes for allocating different bandwidth to different users without explicit setup or reservations. This report describes the mechanisms proposed in this framework, and presents comprehensive simulation results for different scenarios.

1.1 Motivation

Perhaps one of the most urgent problems to be solved on the Internet is how to provide Quality of Service (QoS) in a datagram packet-switched network. The Internet today uses a single level service model called "best-effort". In this service model, the network allocates bandwidth among all the instantaneous users at best it can, and attempts to serve all of them without making any explicit commitment as to either bandwidth or delay. The "best-effort" service model allows a high degree of statistical multiplexing and is extremely efficient in utilizing network resources.

Recent years saw the startling success and changes of the Internet. The Internet evolves from a government funded research project to a commercial, complex network with a heterogeneous user base. Rapid growth in number of users and high-bandwidth applications has put a tremendous demand on the current Internet infrastructure. Consequently, congestion occurs often, and users see degradation in application performance. How to provide statistical service assurance in the Internet infrastructure is an important and yet unresolved issue.

Internet's large, diverse users usually have different usage expectations for their applications: some need strict service guarantees whereas others can have a more relaxed requirement. Users can be better served if they can receive different levels of service in times of congestion. (Of course, when there is no congestion, there is no problem.) The current existing mechanisms for providing different services usually require some explicit setup or reservation protocols like RSVP[RSVP93], which are computationally expensive. In this report, we present a new framework — "Expected Capacity" — for allocating different levels of bandwidth to users cheaply and efficiently.

The idea behind the "Expected Capacity" framework is simple. At the network level, the Internet distinguishes between two types of packets — *In* and *Out* packets. When there is no congestion, the two types of packets are treated the same. When there is congestion, *Out* packets are preferentially dropped so that *In* packets can get through. Each user contracts a *profile* from his respective ISP (Internet Service Providers). A profile meter, interpreting the corresponding profile, will tag the user's packets as *In* packets or *Out* packets accordingly. Overall, the framework allocates different bandwidth to users by controlling how many *In* packets each user can send.

In the "Expected Capacity" framework, the mechanism to allocate bandwidth is decoupled from the top-level policies of who gets what amount of bandwidth. This feature will allow the framework to meet a wide range of business and administrative goals. 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 development of future infrastructure. In private networks like corporate or military networks, administrative measures are often used to allocate resources. Such administrative policies can also be implemented with profile meters. In other words, the same mechanism can be deployed in the underlying infrastructure to allocate bandwidth, regardless of top level policy.

It should be noted here that the "Expected Capacity" framework attempts to solve only bandwidth allocation problem, and does not address any delay guarantees. Most Quality of Service (QoS) metric refers to both delay and bandwidth dimensions. In today's high-speed network, queueing delay dominates transmission delay and is the major factor in delay. Queueing delay can be bounded by using buffer allocation and scheduling schemes like CBQ, or queueing management schemes like RED. The mechanism in the "Expected Capacity" framework is orthogonal to mechanisms proposed for managing delay in the network[1].

1. Technically speaking, bandwidth and delay are interrelated. For example, if the network is extremely under-provisioned, where there is not enough bandwidth to satisfy instantaneous demands, the queue naturally forms and sustains and delay increases. Conversely, when the network is over-provisioned, where there is enough bandwidth, then there would be no queues, and no delay. There have been proposals for managing delays in the current IETF int-serv discussions.

1.2 Related Work

A number of approaches have been proposed for control of 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 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 price 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. First is that the linkage between the treatment of each individual packet and the overall transfer rate is not obvious. Second, 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. [Gupta97] 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 [DemKesShe89, 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.

RSVP[RSVP93] is a “receiver-oriented” resource reservation protocol. Receivers periodically send messages to routers to maintain a “reservation” — installed as a soft-state in the routers. The receiver-oriented feature allows RSVP to deal gracefully and efficiently with a large multicast group. However, RSVP is also computationally expensive and does not scale well as the number of reservations grow.

The idea of tagging packets as *In* or *Out* is not a new one. For example, researchers at IBM [Bala90] 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 that in [Clark96], 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. [ClaFan97] is an early and a much more concise version of this report. This report augments [ClaFan97] with in-depth discussions on different tagging and dropping algorithms to be deployed in the framework, proofs of the algorithms and simulation results for many scenarios.

1.3 Organization of the report

There are ten chapters in this report, which can be organized into three parts. Chapters 2 and 3 explain the “Expected Capacity” framework in depth. Chapters 4 to 9 offer extensive simulation results to demonstrate the degree to which “Expected Capacity” can allocate specific amounts of bandwidth to TCP bulk data connections under various network conditions. Each chapter can be read individually. Chapter 10 concludes and offers some insights we learned from this work.

Chapter 2 describes the framework in detail. It defines the terminologies, describes the essential components of the framework, and explains how the framework fits in with the current Internet infrastructure. In particular, the framework supports both sender-based and receiver-based schemes for controlling sending rate. The two schemes can be easily combined and this view is offered at the end of the chapter. This chapter is essential for understanding the framework.

Chapter 3 describes a preferential dropping algorithm to be deployed in the router, and a tagging algorithm for bulk-data TCP transfers. The preferential dropping algorithm (RIO) is based on Random Early Detection (RED) algorithm. It uses two sets of parameters to create differential treatments of *Out* packets during congestion periods. The tagging algorithm is called Time Sliding Window (TSW), which tags packets as *In* according to an estimated sending rate. We include pseudo code for both algorithms, which can be implemented easily. Finally, we offer some experience and insights from designing those algorithms.

From chapter 4 to chapter 9, we describe how the above combined tagging-dropping algorithms would work under different types of network conditions. Chapter 4 offers simulations results in a sender-based, one-way traffic scenario. Simulations and proofs show that the current Internet demonstrates biases against TCP connections with long round trip time[Floyd95]. After adopting the "Expected Capacity" framework, the Internet can allocate bandwidth among TCP connections according to users profiles, instead of round trip times.

Chapter 5 presents simulation results from a receiver-based, one-way traffic scenario. A receiver-based scheme can be implemented if 1) the gateways mark packets instead of dropping them when congestion occurs, and 2) endhost TCP understands the semantics of the Explicit Congestion Notification (ECN) bit. The simulation results also show that in the receiver-based scheme, the "Expected Capacity" framework can allocate bandwidth according to users' profiles.

Chapter 6 studies how the "Expected Capacity" works with a newer version of TCP. TCP-SACK, or TCP with selective acknowledgments can successfully recover multiple packet drops within a TCP window, and avoid unnecessary timeouts. Much of the sophistication in designing the above TSW tagging algorithm was devoted to avoid triggering multiple packet drops in a window. With SACK-TCP, the tagging algorithm can be more aggressive in tagging *Out* packets without triggering unnecessary timeouts, and the system can be more responsive in controlling congestion.

Up to this point, the profile meters — where the tagging algorithms are — have been assumed to be at the traffic ingress point from a user to an ISP. This assumption makes sense for security reasons: the ISP has no reason to trust the end users to tag their own packets. This also means that the profile meters have no knowledge of the transport layer state information. If end-user hosts implement a profile meter in the operating system, then the tagger can access all the state information of TCP and make more accurate decisions as when to tag a packet. When profile meters are implemented at endhosts, the network biases against long round trip TCP connections can be eliminated. However, in order to ensure security, this scheme has to be complemented with a simple checking algorithm on the ingress point from a user to an ISP. Chapter 6 also explores this new design space.

Inevitably, in a heterogeneous network like the current Internet, there will be non-conforming users who do not implement congestion avoidance and control mechanism in their transport layer protocols. Chapter 7 shows that the framework can provide certain “shielding” for users who have usage profiles from those who do not. In addition, the framework offers an simply means to identify non-conforming users.

In the “Expected Capacity” framework, most of the packets from a connection will go through a series of ISPs, and potentially, a series of profile meters, or cascaded profile meters. Chapter 8 explores the issue of how cascaded profile meters, each with its probabilistic tagging scheme, will affect the end-to-end performance of each connection.

Chapter 9 presents a set of simulations of profile meters for aggregated traffic. It answers the question of how bandwidth is distributed among connections sharing the same aggregated profile, and how aggregated profile meters affect the behaviors of individual TCP connections. Chapter 10 summarizes and highlights the conclusions from chapters 4 to 9.

CHAPTER 2 *The "Expected Capacity" framework*

Abstract: This chapter presents the terminology and components in the "Expected Capacity" framework. Two complimentary schemes, sender-controlled and receiver-controlled, are described.

2.1 Overview

The general approach of the "Expected Capacity" framework 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, both *In* and *Out* packets are aggregated into one queue. There is no separation of traffic from different users into different flows or queues. 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 or Weighted Fair Queuing, in which the packets must be explicitly classified at each node.

The routers are augmented with a preferential dropping algorithm. When there is no congestion, routers will treat *In* packets and *Out* packets the same. During periods congestion, the routers will preferentially drop *Out* packets so that *In* packets will have a better chance getting through. There are other schemes being proposed to create preferential treatments of packets, for example, a priority scheme in which packets tagged as "*In*" are put into a separate queue from the “*Out*” packets. We believe that 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 dropping algorithms but assume both *In* and *Out* packets are in the same queue.

At the edge of the network a new function must be added to interpret the service profiles. This function can be implemented as part of an existing network component — for example, access points or end hosts — or in a new component created for the purpose. Conceptually, we will refer to it as a distinct device called a “*profile meter*”. Profile meters will embody a tagging algorithm, which will tag packets going into the network as either *In* or *Out* packets.

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.

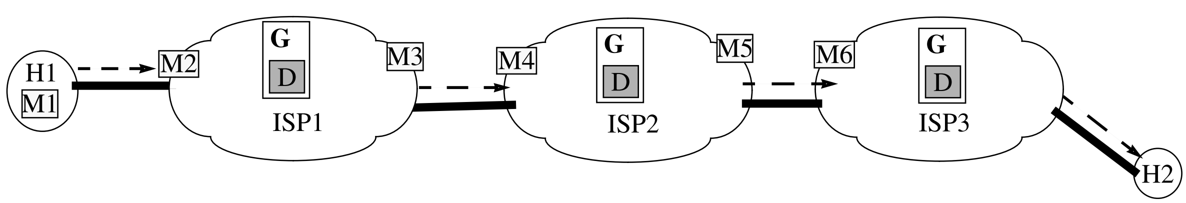


Figure 1. the “Expected Capacity” framework (sender-based)
Host 1, H1 has a sender-based profile, and is sending traffic to host H2 (dotted line). The traffic traverses three ISPs. The gateways, G’s, in the figure are all augmented with preferential dropping algorithms, Ds. There are profile meters, 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 sides 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 tagging algorithms embodied in profile meters may vary depend on the location of the profile meters. At the edge of the network, where traffic patterns of individual connections are visible, the profile meters can provide highest degree of discrimination among the connections. For example, a profile meter integrated into a host implementation of TCP and IP can serve to regulate the relative use of the network by individual flows. In the center of the network, as the traffic merges and aggregates with other traffic, the corresponding profile meters only need to look at large aggregates. 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 over simplify, the preferential dropping scheme adopted in routers in the center of the network will not change over time. Since the characteristics of a service are 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.

Since the framework offers such flexibility, the services provided by this framework are diverse. As a simple example, a service 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.

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 in this framework 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 chapter 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 bandwidth 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

In today’s Internet, it is usually the receiver of a connection, not the sender, is concerned with the level of service it is getting. 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. In the following, we describe a mechanism that implements receiver control of service, which is 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 [Floyd95]. In ECN semantics, congested gateways will turn on the ECN bit in a packet instead of dropping the packet during congestion. TCP receiver copies the ECN bit into the acknowledgment (ack) packet, and TCP sender will gracefully slow down upon receiving an ack with the ECN bit on.

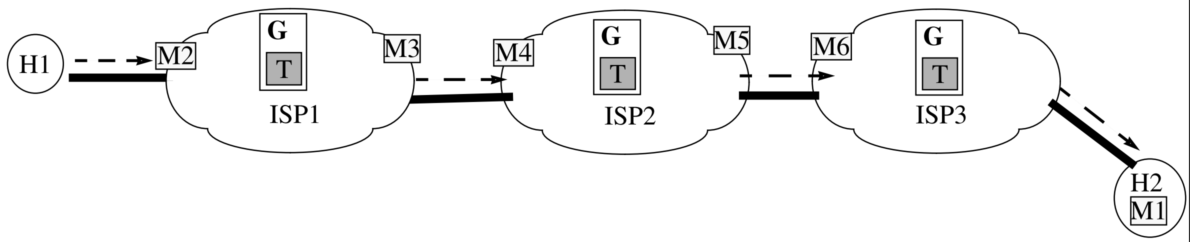


Figure 2. the “Expected Capacity” framework (receiver-based)
Host 1 is sending traffic to host H2 (dotted line). Host H2 has a receiver-based profile. The traffic traverses three ISPs. The gateways, G’s, in the figure are all augmented with preferential tagging algorithms, Ts. There are profile meters, 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.

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.

Figure 2 illustrates a receiver-controlled scheme. All the gateways (G) understand ECN semantics, and can turn on, or tag (T), ECN bit in a packet during congestion. Sender H1 is sending to receiver H2. H2 has contracted a service profile from its internet service provider, ISP3. This service profile is implemented by profile meter M1, from which, the connections from H1 to H2 is debited.

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[1].

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.

1. Though this raises the question of whether dropping packets at the receiver end of the transmission is a desirable behavior. Chapter 5 discusses this issue more.

CHAPTER 3

Dropping and tagging algorithms and their interactions with TCP

Abstract: TCP is the predominate transport-layer protocol in the Internet, therefore, is our focus in this report. We first present the dropping algorithm to be used in the gateways, and then, a specific tagging algorithm to be used in the profile meters to work with TCP traffic.

It is important to realize that the dropping algorithm in the routers, once adopted, is unlikely to change 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 design a dropping algorithm in the routers that will offer enough generality to cooperate with many types of profile meters. Since different profile meters generate different quantities of *In* and *Out* packets. When congestion occurs, the mix of *In/Out* packets at the gateways can be very different. Therefore, the gateways will have to be able to discriminate against *Out* packets, regardless what kind of traffic mix they have during congestion. In the following section, we present a preferential dropping algorithm.

3.1 Differential dropping in the routers: RIO

RIO stands for Random Early Drop (RED) gateways with *In/Out* bit. Like RED algorithm, RIO is also a queue management algorithm. Gateways using RED algorithm are able to [FloJoc93, Flody96a] 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.

3.1.1 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.

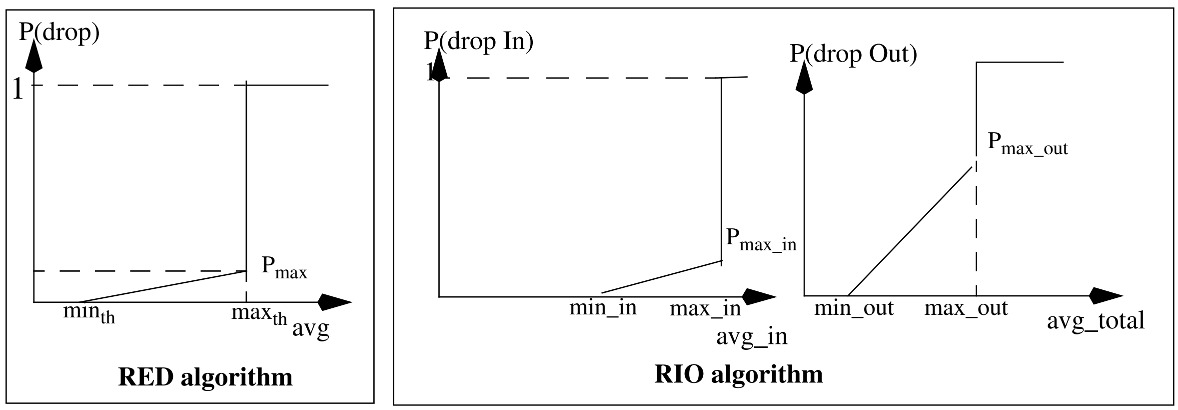


Figure 3. RED and RIO algorithms (Figures not drawn to scale)

A RED gateway is configured with the following parameters: min_{th}, max_{th}, and P_{max}. It works as illustrated in the left figure in figure 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 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.

3.1.2 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*.

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.

```
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 4. RIO algorithm pseudo code

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 in three ways: 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* 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. Figure 4 contains the pseudo code for RIO algorithm.

3.1.3 Choosing RIO parameters

The choice of using *avg_total*, the total average queue size, to determine the probability of dropping an *Out* packets is subtle. Intuitively, the average queue size for *In* packets reflects the amount of traffic that is within contracted profiles. Since the ISP knows how much profiles it has contracted out and monitor usage patterns, it has a reasonable estimate of the expected average *In* queue size, or *avg_in*. In contrast, the average queue size for *Out* packets represents the opportunistic traffic that the gateway sees, and this can be influenced by a variety of reasons: maybe some contracted users are not using the network so the other users can send faster than their contracted profiles, generating a lot of *Out* packets. Or, perhaps some users are simply sending CBR (constant bit rate) traffic exceeding their profiles. In any case, there is no reasonable estimate of the average *Out* queue size to represent the proper amount of *Out* packets. If we had used the average *Out* packet queue, or *avg_out*, 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. When the *avg_in* is small, the gateway will allow more *Out* packets. However, this might allow too many *Out* packets so that the gateway loses control of total queue length, defeating the purpose of keeping the total queue length short. An extreme case of this, for example, is that when there is no *In* packets in the gateway. With this scheme, a gateway will not control the number of *Out* packets into the gateway, and therefore degrade into a drop-tail gateway.

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. When the traffic mix is mostly of *In* packets, *avg_total* is very close to *avg_in*, so that the gateway will allow limited amount of *Out* packets; when the traffic mix is mostly of *Out* packets, the total queue length can control the average short queue length while maintaining high throughput.

Conceivably, 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.

We recommend choosing the RIO parameters along the following principles:

- **Make *min_in* high enough to fully utilize network resources**
  So that the network won't start dropping any *In* packets until *avg_in* reaches *min_in*.
- **Make the (*min_in*, *max_in*) range wide enough to accomodate bursts**
  TCP packets come in bursts, so the range of (*min_in*, *max_in*) should be large enough to accomodate at least one RTT's worth of packets without dropping.
- **Control discrimination against *Out* packets by adjusting (*min_out*, *max_out*, P_{max_out}).**
  Follow the recommendations in 3.1.2.

3.2 Profile meters for TCP bulk data transfers

In this following two sections, we present a profile meter tailored specifically to TCP bulk-data transfers. We chose TCP because it is the predominant transport layer protocol. 97% of the Internet traffic is TCP traffic [End97]. The specific service profile is "an average TCP throughput of X from this host to anywhere", where X is the targeted rate. This service profile is easy to understand and representative. Users can test it by running a TCP connection and measure the achieved throughput. However, the corresponding tagging algorithm in the profile meter is not trivial, due to the following reasons. 1) TCP implements congestion avoidance and control mechanisms. It hunts for available bottleneck link bandwidth by increasing and decreasing its window size, so the rate is not easily controllable. 2) Due to TCP's window open-up algorithm, the network has a strong bias against TCP connections with long round trip times (RTTs). Consequently, to achieve the same throughput regardless of destinations is not easy.

In this section, we will discuss TCP and the network bias against long RTT TCP connections. We will discuss the implications of such TCP idiosyncrasies and bias, and present a tagging scheme that works well with TCP bulk data transfers in the following section.

3.2.1 TCP rate adjustment

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

TCP takes packet drops as congestion signals from the network and reacts to them by reducing sending rate. The mechanisms used by TCP to deal with congestion are based on [Jacob88]. 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 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, or Reno-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[Hoe96]. 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 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.

Figure 5 illustrates the ideal rate oscillations of a TCP connection that operating only in the "Fast Recovery" phase. The sender receives a packet drop exactly when its sending rate has reached 1.33 R_T, where R_T is this TCP's target rate. The sender will reduce its sending rate to 0.66 R_T after detecting the drop. Its window size, and the corresponding rate will go up to 1.33R_T again gradually, until another drop pushes down its sending rate again. This sawtooth-like pattern repeats. Overtime, this TCP will achieve R_T average. In the "Fast-Recovery" phase, TCP is sensitive to transient network congestion by slowing down, but also resilient enough to recover loss packets without suffering much in performance.

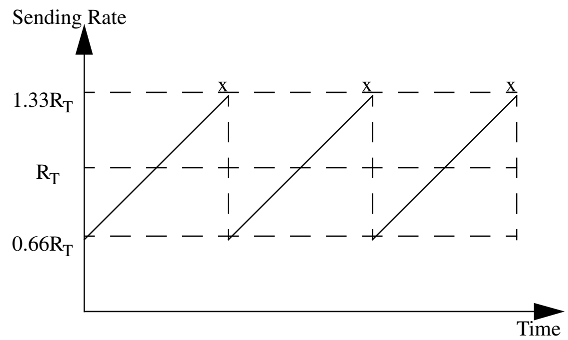


Figure 5. Ideal TCP oscillations
R_T is target rate, and X indicates a packet drop. If TCP would operate perfectly between $0.66R_T$ and $1.33R_T$, the time average is R_T.

3.2.2 Network bias against long RTT connections

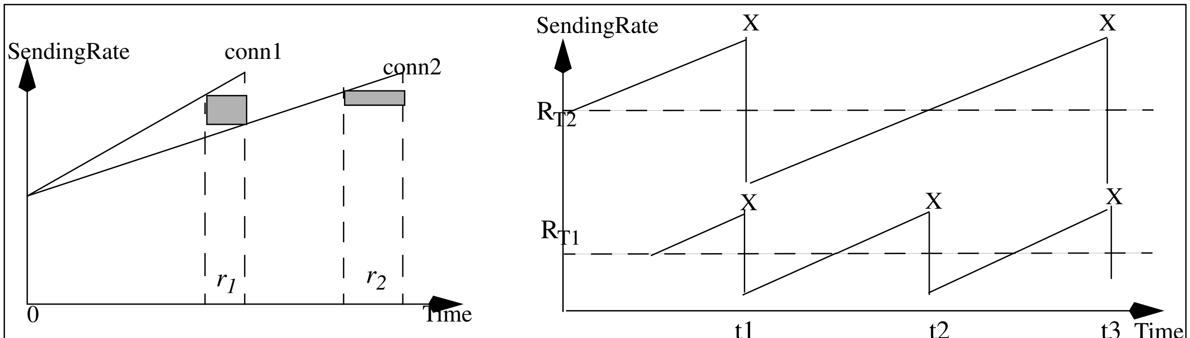
Recall that TCP oscillates around the equilibrium operating point by increasing congestion window, or *cwnd* by one packet[1] each round trip time. Let r_i denotes connection i's average round trip time including queuing delays. Therefore, in one round trip time, r_i, TCP increases its congestion window by one packet. In other words, TCP increases its sending rate by $1/r_i$ pkts/sec each r_i time, or it increases its sending rate by $1/(r_i)^2$ each second.

Consider two connections with different round trip time, r_1 and r_2, with $r_2 >> r_1$. Let both connections be sending at the same rate, and both receive a single packet drop. Both sources will reduce their sending rate by one half, but it takes connection 2 much longer to recover to the previous sending rate again, since the pace at which connection 2 increases its sending rate (or the acceleration) by opening up its *cwnd* is much slower.

The left half of figure 6 illustrates the point. Connection 1 (conn1) and connection 2 (conn2) both reduce their sending rate at time 0 after receiving a packet drop each. Conn1 has a short RTT, r_1, shown as the distance bounded by two dotted lines on X axis. Similarly, conn2 has a long RTT, r_2. During each respective round trip time, each connection opens up its window by one packet. The number of bytes in a packet, depicted as the area of the shaded regions, is the same for both connections. However, the rate increase, or the height of the shaded

1. Real implementations of TCP increase window by segments, instead of packets; we use packets for simplicity in explanation.

regions, is different. Therefore, it takes conn2 much longer to reach the same sending rate again. Section 4.2 demonstrates this bias with simulation results.



Each shaded box is size of a packet, since the RTTs (widths) for the two connections are different, the height, or the rate increase are different for the two connections.

Network bias against connections with long RTT

Each X is a packet drop. Two TCP connections have different target rates, but same RTT, so that the slopes of the two lines are the same but it takes the high-target-rate connection longer time to recover from a drop.

Network bias against connections with high target rates

Figure 6. Network biases due to TCP window algorithms

Let *tcp period* (T) denote the time it takes a TCP connection to recover to its target sending rate (or window size) after a packet drop. Then, if two TCP connections, conn1 and conn2, with round trip times r1 and r2, respectively, and $r_2 = n \bullet r_1$. All else being equal, then the periods for two connections, T1 and T2, follow the following relation:

$$T_2 = n^2 \bullet T_1$$

3.3 Designing profile meters for TCP

3.3.1 Profile meters for TCP

TCP has a rather sophisticated congestion control and avoidance algorithm, which makes designing the tagger to work with TCP particularly challenging. The challenges come from the facts that 1) TCP uses window algorithm to pace out packets, rather than a timer to pace out packets. (As a result, TCP tends to send out a cluster of packets back-to-back. This makes it hard to estimate the "average sending rate" of a TCP connection); 2) Reno-TCP doesn't deal very well with multiple packet drops in a window, and usually goes into a Slow-Start phase when that happens; 3) TCP's window algorithm has biases against long RTT connections and high target rates.

Much of the difficulties in designing a profile meter for TCP traffic is to accomodate such TCP idiosyncrasies. We considered using a simple token bucket with two variables: bucket depth and replenish rate. However, after some initial experiments, we decided that this has a limited design space and is not suitable for TCP. We also experimented with at least ten other algorithms for profile meters, and eventually honed in on one design, the Time Sliding Window.

A profile meter for TCP has to answer the following three questions: 1) How fast is a TCP sending? 2) When should packets be tagged as *Out* packets? 3) How should the *Out* packets be spaced? We will discuss the three aspects of a profile meter in turn.

To determine how fast a TCP is sending, a rate estimator is needed. A rate estimator should be sensitive to the changes in sending rate when TCP speeds up or slows down, and it should smooth out the burstiness when TCP sends out packets in a cluster. In order to be sensitive to changes in sending rate, the estimator would have to "forget" the past history, and take into account the new sending rate; so some kind of decay function is needed. For example, if the profile meter uses the last 20 seconds as the period to estimate the average sending rate of TCP, it would take a lot packets before the estimator realizes that the sending rate has increased. On the other hand, in order to smooth out burstiness, a rate estimator would have to "remember" the past history, at least in the order of an RTT.

The profile meter can start tagging packets as *Out* whenever the estimator reports that the sending rate has exceeded the *target rate*, R_T. This is perfectly fine for aggregated traffic in the center of the network. However, when the tagger can be installed next to the host, the answer can be different. For example, when tagging one TCP connection, if the profile meter has a very good estimate of the TCP sending rate, then the profile meter can start tagging only when TCP exceeds 1.33 R_T. This would ensure that TCP can reach its full speed of $1.33R_T$ before being clamped down with a packet drop.

The answer to the third question has a lot to do with the fact that current implementations of TCP, or TCP-Reno, fail to recover multiple packet drops within a TCP window. Multiple packet drops will invariably trigger TCP-Reno to go into Slow-Start phase [FalFlo96]. Therefore, when the profile meter starts tagging packets as *Out* packets, it should try to space out the *Out* packets as much as possible as to avoid triggering Slow-Starts. Section 3.4 deals with this problem in detail.

3.3.2 Time Sliding Window (TSW) profile meter

TSW has two parts: a rate estimator that estimates the sending rate over certain period of time, and a tagging scheme (abbreviated as tagger in the following context) that tags packets as *Out* based on the rate reported by the rate estimator. They are two independent components. The rate estimator takes into account the burstiness [1] and smooths out its estimate to approximate the actual sending rate at the source TCP. With the estimated rate, the TSW tagger determines whether the sending host has exceeded its target rate, in which case, the tagger will tag packets as *Out* packets probabilistically to reduce the likelihood of packets within a window get dropped together in gateways.

- **TSW rate estimator**

Since the tagger has to make a tagging decision upon each packet arrival, an obvious design of the tagger seems to be using a low-pass filter of instantaneous sending rate (the packet size divided by inter packet arrival time) to smooth out the burstiness of TCP, and decay the past history over time. However, this design suffers an unfortunate flaw: the decaying of estimated rate is according to packet arrival, instead of time. If the source is sending fast, the decaying is also fast, and vice versa. As a consequence, a TCP sending at a high rate will forget its history faster than a TCP sending at a low rate. The sending rate is not decaying, for example, when the TCP has nothing to send for a while.

TSW rate estimator escapes this misfortune by estimating the sending rate upon each packet arrival, but decaying the past history over time. Since the decaying is according to time, rather than packet arrivals, the decay factor is the same regardless how faster the source is sending.

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;                    (1)
New_bytes    = Bytes_in_TSW + pkt_size;                  (2)
Avg_rate     = New_bytes / (now - T_front + Win_length); (3)
T_front      = now;                                      (4)
```

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

Figure 7. TSW rate estimator algorithm

1. TCP traffic tends to be bursty due to a number of reasons. Window-based algorithm in TCP results in clustering effect [ZhaSheCla91]. This effect can also be exacerbated by "Compressed Acks" or delayed acks.

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 never changed. Figure 7 presents the algorithm for the rate estimator.

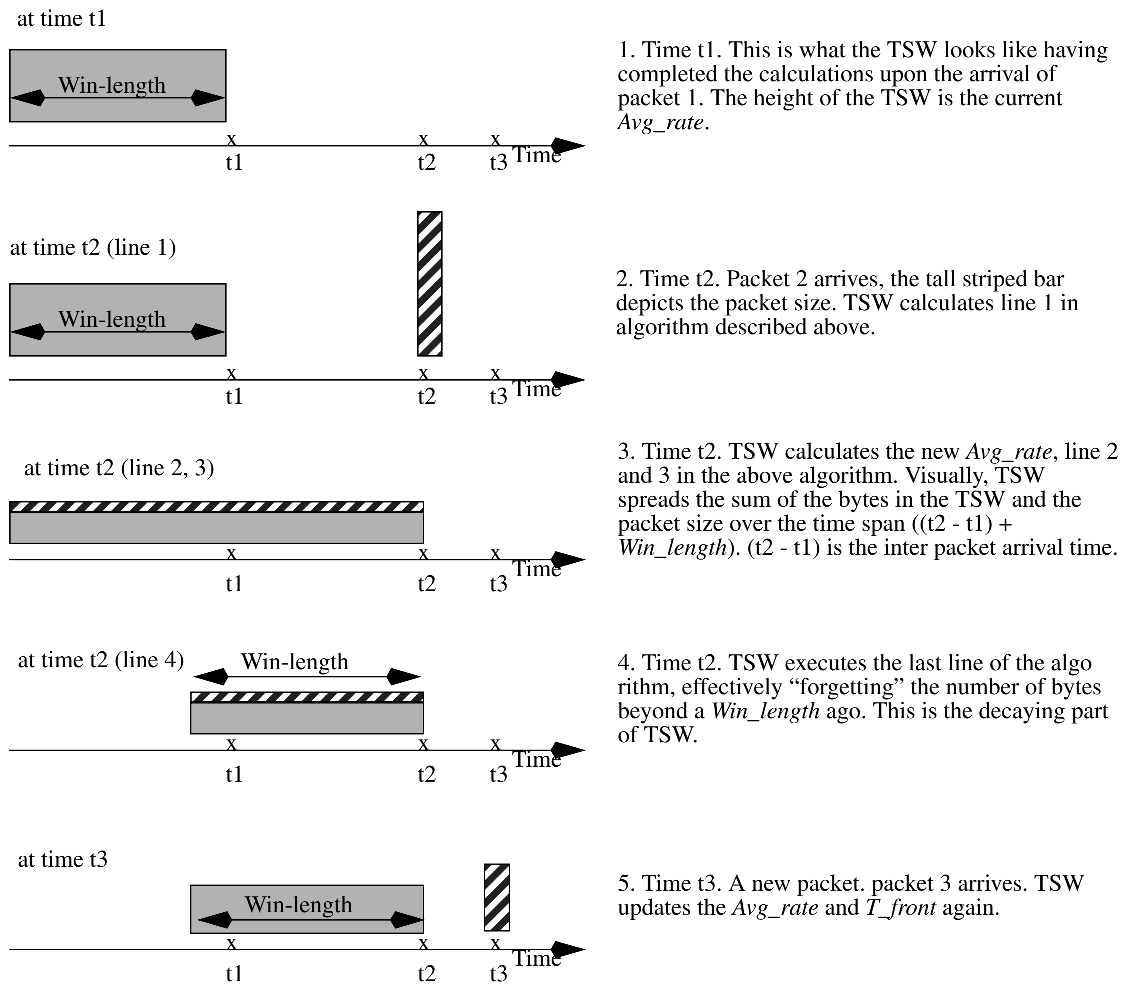


Figure 8. The operations of TSW rate estimator

We will illustrate how TSW calculates *Avg_rate* in figure 8, There are three packet arrivals, depicted as crosses on the time line, at time t1, t2 and t3, respectively. The lightly shaded box depicts the TSW, with its width along the time axis as the *Win_length*, and its height as the *Avg_rate*. The right edge of the box is the *T_front*. Therefore, the bytes contained in the TSW is the area covered by the shaded box. The tall striped bars at time t2 and t3 refers the packet size of the packets arrived at time t2 and t3. Similarly, the area covered by the striped bars depicts the size of the packet. So that packet arrived at time t3 is smaller than the packet arrived at time t2.s.

- **Decaying function embedded in the TSW rate estimator**

In the following, we present a proof for the decaying function embedded in TSW rate estimator. Similar proofs can be constructed for non constant arrival rate, but they are much more complex and are not included here.

Let w denote the size of each packet. Let δ denote the arriving interval between packets. So that the sending rate is w/δ. Let L denote the *win_length* of a Time Sliding Window.

Let R_0 denote the estimated rate by TSW at time t_0. And subsequently, R_i denote the estimated rate by TSW upon receiving *i*th packet. Then,

$$R_1 = R_0\left\langle\frac{L}{L+\delta}\right\rangle + \frac{w}{L+\delta}$$ (1) after receiving the first packet

$$R_2 = R_1\left\langle\frac{L}{L+\delta}\right\rangle + \frac{w}{L+\delta}$$ (2) after receiving the second packet

$$= \left(R_0\left\langle\frac{L}{L+\delta}\right\rangle + \frac{w}{L+\delta}\right)\left\langle\frac{L}{L+\delta}\right\rangle + \frac{w}{L+\delta}$$

$$= R_0\left\langle\frac{L}{L+\delta}\right\rangle^2 + \frac{w}{L+\delta}\left\langle\frac{L}{L+\delta}\right\rangle + \frac{w}{L+\delta}$$

....

$$R_n = R_0\left\langle\frac{L}{L+\delta}\right\rangle^n + \frac{w}{L+\delta}\bullet\left(\frac{L}{L+\delta}\right)^{n-1} + \frac{w}{L+\delta}\bullet\left(\frac{L}{L+\delta}\right)^{n-2} + \ldots + \frac{w}{L+\delta}$$ after receiving the nth packet

The last n-1 terms are of exponential series, so the last equation can be reduced to be

$$R_n = R_0\left\langle\frac{L}{L+\delta}\right\rangle^n + \frac{w}{L+\delta}\bullet\sum_{i=0}^{n-1}\left(\frac{L}{L+\delta}\right)^i$$, recall that $\sum_{i=0}^{n} q^i$, where q < 1, is $\frac{1-q^n}{1-q}$, or $\frac{1}{1-q}$, when n goes to ∞.

Additionally, since $n = \frac{L}{\delta}$, the first term becomes $\left(\frac{L}{L + \frac{L}{n}}\right)^n = \left(\frac{1}{1 + \frac{1}{n}}\right)^n$.

Therefore, $R_n = R_0\left(\frac{1}{1 + \frac{1}{n}}\right)^n + \frac{w}{L + \delta} \bullet \left(\frac{1 - \left(\frac{L}{L + \delta}\right)^n}{1 - \left(\frac{L}{L + \delta}\right)}\right)$.

When $n \to \infty$, $\lim_{n \to \infty} R_n = R_0 \bullet \frac{1}{e} + \frac{w}{L + \delta} \bullet \frac{1}{\left(\frac{L + \delta - L}{L + \delta}\right)}$

or $\lim_{n \to \infty} R_n = R_0 \bullet \frac{1}{e} + \frac{w}{\delta}$

Therefore, after *win_length* or L time later, we forget the past sending R_0 by a factor of *e*. Intuitively, TSW remembers *Win_length* worth of past history.

- **TSW tagging algorithm**

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* 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 – in 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, unless specified otherwise.

```
Estimate sending rate R
    if R < R_T
        mark packet as In;
    else
        calculate P_drop =(R - R_T)/R_T ;
        with P_drop , drop the packet.
```

Figure 9. TSW tagging algorithm

3.4 The probabilistic function in RIO and TSW

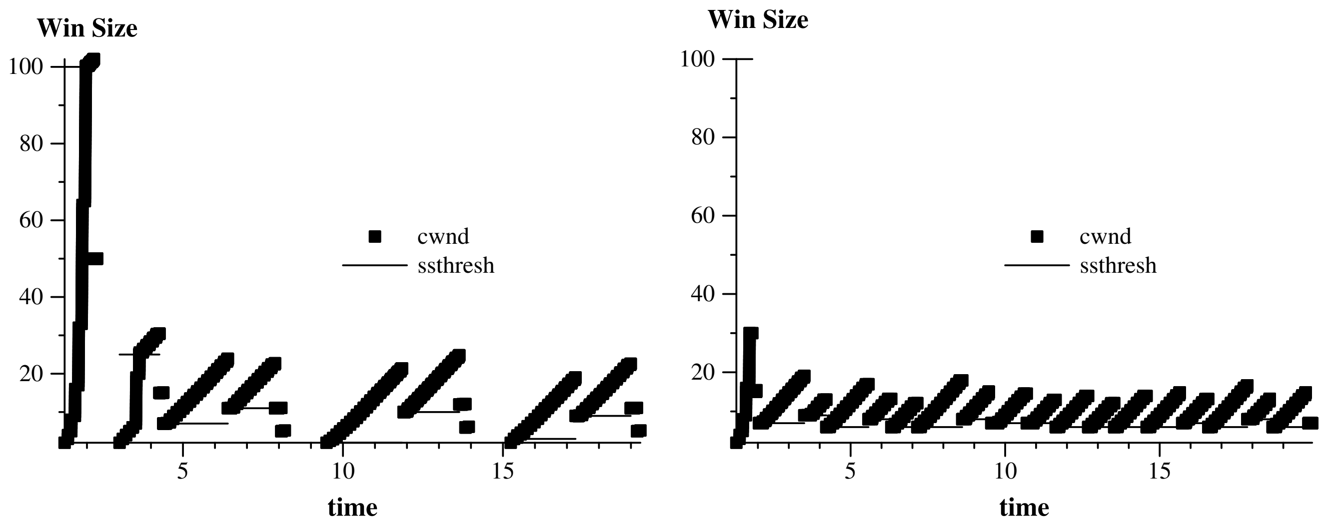
In both RIO and TSW, we use a probabilistic function to spread out the dropping and tagging of an *Out* packet, respectively. This is to keep TCP operating in the Fast-Retransmit phase. Current implementation of TCP fails to recover multiple packets in a TCP window, and go into Slow-Start phase. Therefore, in order to keep TCP in the more controllable Fast-Retransmit phase, at most two packets can be dropped within a TCP window. The subtlety in accomplishing this lies in two factors: when dropping packets, the gateway tries to drop randomly, with fairly regular intervals; when tagging packets, the tagger tries to space *Out* packets evenly across time. The combined effect is to decrease the likelihood of consecutive *Out* packets from a connection being dropped by the gateway.

In both RIO and TSW algorithms, we use the a method discussed in detail in [FloJoc93], which has the property of evenly spread out tagging and dropping.

3.5 Combined effect on TCP

Figure 10 illustrates the results of such combined algorithm. The left graph shows the TCP window oscillation for a particular connection operating under RED gateway; the right graph shows the same connection, with a usage profile and a tagger, operating under RIO gateway. The horizontal thin line indicates the values of *ssthresh* at any time. Each dot on the graph is the TCP congestion window size, *cwnd* so that a heavy, solid line formed by dark black dots of *cwnd* shows a typical TCP window swing as it probes the network bandwidth. A broken line and a sudden drop usually indicates a packet drop and a thereafter *cwnd* retraction. Consecutive packet drops usually leads TCP into a retransmission time out and a period of silence. For instance, in left graph, from time 8.4 to time 9.2, TCP is completely silent due to the two consecutive drops in roughly time 8.1. A retransmission time out is followed by a Slow-Start, but TCP is operating with a much smaller *ssthresh*. This kind of sending pattern - a silent period followed by a long period of low sending rate -- is extremely undesirable for bulk data transfer, and it has been one of our goals to eliminate them in designing a tagging algorithm.

In comparison, the TCP window swing for connection 9 in RIO-TSW is very much like the ideal sawtooth. The congestion window roughly operates around 14 packets, and occasionally a single packet drop will cause TCP to retract its window to half of the previous value. TCP can soon recover the lost packet and remain in the congestion avoidance phase, linearly increasing its window size.



Congestion Window 9 (Reno_10_flows)

Congestion Window 9 (Sack_10_flows)

Figure 10. TCP window oscillations

CHAPTER 4

Simulation Results: sender controlled scheme

Abstract: This chapter presents simulation results in the "Expected Capacity" framework using a sender-controlled scheme. In particular, sections 4.1 discusses the general simulation setup which is referred to by other chapters, and section 4.2 illustrates network bias against long-RTT TCP connections using simulations.

4.1 Simulation setup

We developed modules for RIO gateways and TSW profile meters using the *ns* [Ns] network simulator from LBL. Simulations scripts are written in Tcl. Each script defines a topology, describes network devices and their parameters, and sets network events. Please see (archived code here) for details.

In this chapter, we present simulation results in the "Expected Capacity" framework using a sender-controlled scheme. The configurations of the simulations are listed as following. They are applicable to simulations in other chapters as well.

- *TCP connections*
  Unless otherwise specified, all TCP connections are configured as follows: they are Reno-TCP, implementing Slow-Start, Fast-Retransmit and Fast-Recovery algorithms. The packet size is 1000 bytes and the tcp timer granularity (*tcp-tick* variable) is 0.1 second. The receiver's advertised window is usually configured large enough so that it is never a limit on the TCP sender's window. The TCP connections start at a slightly different time from each other, but all within the first 5 seconds of simulations. Each simulation simulates 20 seconds of network condition unless otherwise specified.

- *Gateways*

Gateways are RIO gateways. We use tables to present the throughput of each connection. The parameters of RIO are described in table headers.

- *Profile meters*
  Profile meters are located on the access link. In other words, after data traffic leaves the sending hosts, it reaches its profile meter before arriving at a gateway. In the simulator, a profile meter is implemented as a subclass of a drop-tail link (drop-tail.cc).

- *Service profiles*
  Service profiles are "*average TCP throughput of R_T from this host to anywhere*[1]", where R_T is the target-rate. The round trip times used in simulations range from 20ms to 150ms.

- *Metric*
  As the evaluating metric, we use the average throughput each host's TCP achieves. Average throughput is calculated at the receiver's side (tcp-sink.cc) after a TCP connection has reached a stable state (after first 5 seconds). Average throughput is defined as the total amount of data received over total transfer time. Total amount of data is the data packets TCP layer presents to upper layer applications, excluding all unnecessary retransmissions[2]. The closer the average throughput is to the respective target-rate, the better a scheme works.

- *Presentations*
  Simulation results are presented in a number of ways. Most data results are presented in tables to save space. In addition, we use three types of graphs throughout this report. They are:

  Throughput graphs. They are graphs of achieved throughput vs. TCP's round trip time. Each data point on the graph is a throughput for a particular TCP connection with certain RTT. Each data point is calculated by averaging the results from 5 simulation runs. This graph is to illustrate the effectiveness of the "Expected Capacity" framework in different network scenarios.

  TCP window oscillation graphs: They are graphs of TCP congestion window (*cwnd*) vs. time in a typical simulation run. They are to illustrate how different versions of TCP (TCP-Reno, TCP-Sack) behave in the "Expected Capacity" framework.

1. The RTTs of connections in this report range from 30ms to 150ms, which roughly correspond to RTTs of connections within continental U.S.
2. This is sometimes referred to as "goodput". However, throughout the report, the retransmission rate is low, and the "goodput" and throughput differ little. In this paper, we use "throughput" to refer to what other people might call "goodput".

Gateway average queue length graphs: They are graphs of gateways parameters (avg_in, avg_out, avg_total) during simulations. They are to illustrate the traffic mix of *In* and *Out* packets in different scenarios and how RIO gateways respond to the traffic mix.

- *Topology*

  We use the topologies shown in figure 11. In the topology, ten hosts are connected to ten other hosts, sharing a common link between routers A and B. Each host (*i*) has a TCP connection to its peer host (*i+10*). There are altogether 10 TCP connections. The ten connections are of different RTTs. They can be divided into five groups. The two connections in each group have the same RTTs, but different target-rates. For example, connections 0 and 1 both have 40ms round trip time, but have 5Mbps and 1Mbps as target-rates, respectively. Each of the five groups has a different RTT than other groups. The RTTs for the five groups are 20ms, 40ms, 50ms, 70ms and 100ms, respectively.

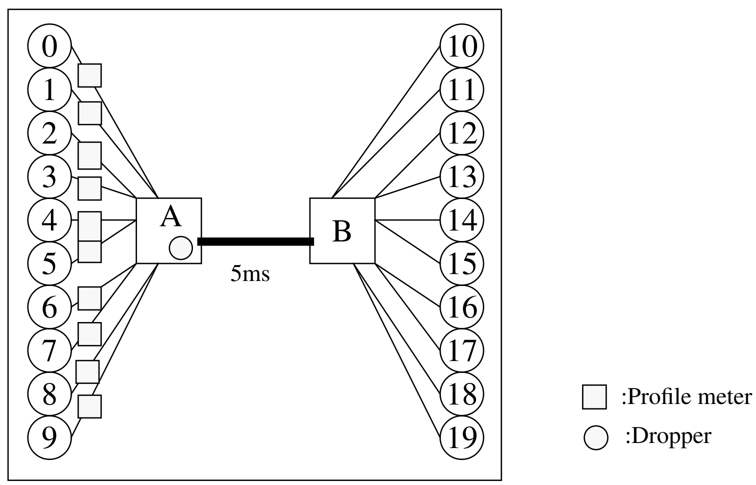


Figure 11. Topology used in a sender-control scheme

4.2 Network bias against long RTT connections

In this section, we will illustrate the network bias against long RTT TCP connections using simulations. As discussed in section 3.1.2, this bias is due to TCP window open up algorithm. During each RTT, TCP opens up its window size by exactly one packet (in bytes). The longer the RTT (seconds), the slower TCP increases its sending rate (bytes/seconds). Therefore, when a TCP connection reduces its sending rate as a result of a packet loss, it takes the long RTT connection longer time to recover to its original sending rate than a short RTT connection.

In this simulation, bottleneck link A-B has a link speed of 6Mbps. Ten TCP connections with different RTTs start sending during the first five seconds of simulations. The average throughputs from a particular simulation run are listed in column 3 of Table 1. The respective RTTs are listed along with the throughput in column 2.This simulation reflects current situation in the Internet, with most hosts implement TCP-Reno. We make two observations from this simulation. First, the network bandwidth is distributed according to the RTTs of TCP connection and there is a strong bias against long RTT connections. Second, in TCP-Reno, the congestion window oscillations 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 (22%). Figure 12 confirms this observation. Figure 12 is a TCP congestion window oscillation graph for one of the connections (connection #8). Each diamond corresponds to the connection's *cwnd* value when a packet is sent, and the corresponding *ssthresh* value is plotted as a short line. *ssthresh* is reset when a retransmission occurs. Therefore, each sudden drop of *cwnd* in the graph means that TCP has detected a packet loss and has retransmitted a packet. When multiple packets from the same TCP window are lost, TCP usually fails to recover all of them, and goes into the Slow-Start phase. In the graph, this corresponds to a silent time period without any diamond points for example, between time 8 second and 9 second. In figure 12, this silent period is quite common.

TABLE 1. TCP's bias against long RTT connections. Topology 1. Link A-B capacity = 6Mbps. RED gateway.

Conn #	RTT(ms)	Achieved Bw by Reno-TCP (Mbps)	Achieved Bw by Modified TCP (Mbps)
0	20	0.829482	0.623805
1	20	1.0932	0.620590
2	40	0.541590	0.600449
3	40	0.694729	0.655939
4	50	0.690377	0.674704
5	50	0.538202	0.527582
6	70	0.430481	0.503961
7	70	0.384579	0.620259
8	100	0.389476	0.671800
9	100	0.419572	0.511367
Total		6.011688	6.010456

Further, we want to verify that this bias is caused precisely by the fact that TCP opens up a constant number of packet (1) each round trip time. If the algorithm were to open up the window according to RTT^2, then the TCP connection will get the same bandwidth regardless the round trip time. This hypothesis can be verified using the *ns* simulator. Such window opening up algorithm was already implemented[1] in the *ns* package. We also need to refine TCP's timer granularity[2]. The results are listed in column 4 of Table. It is clear that the bias against long RTT is eliminated, as the 10 connections sharing a 6Mbps link achieves roughly 0.6Mbps each. The variance in throughput is due to the Slow-Start effects some connections suffer from, a result of TCP-Reno's failure to recover from multiple packet losses.

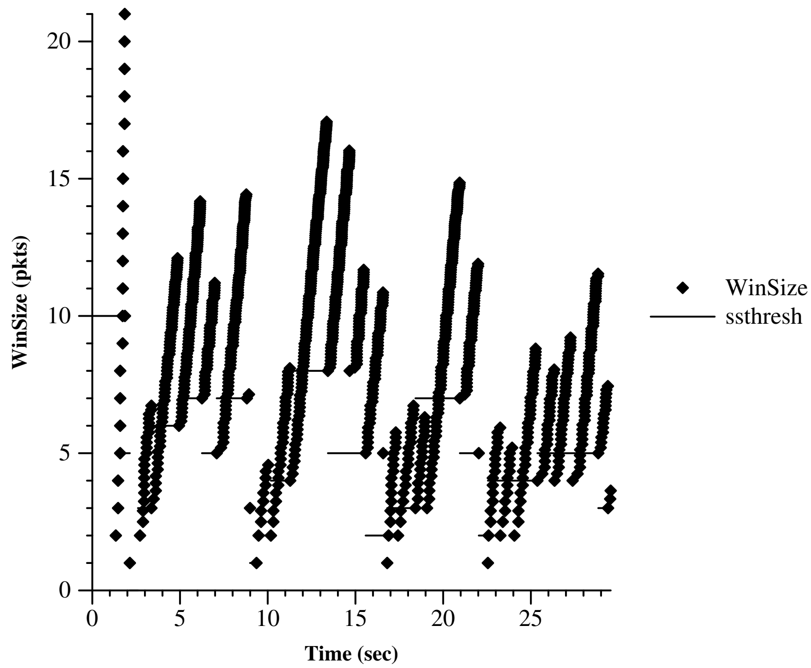


Figure 12: Reno-TCP Window Oscillations (connection 8)

4.3 Taggers at the edge of the network: single TCP connections

In the following simulations, we study bandwidth allocation in the "Expected Capacity" framework. In this set of simulations, bottleneck link A-B has a speed of 33Mbps. Each of the 10 hosts has a contracted service profile and a respective target-rate, R_T. Each host has one TCP connection. We run two simulations: one with 10 connections in the "Expected Capacity" framework using their respective profile meters, and another in the current Internet scenario with no profile meters or dropping algorithms.

1. see ns_default.tc l file, change ns_tcp(window-option) to be 2, the RTT^2 option, instead of 1, the standard algorithm.
2. see ns_default.tc l file, change ns_tcp(tcp_tic k) to be 0.01, instead of 0.1.

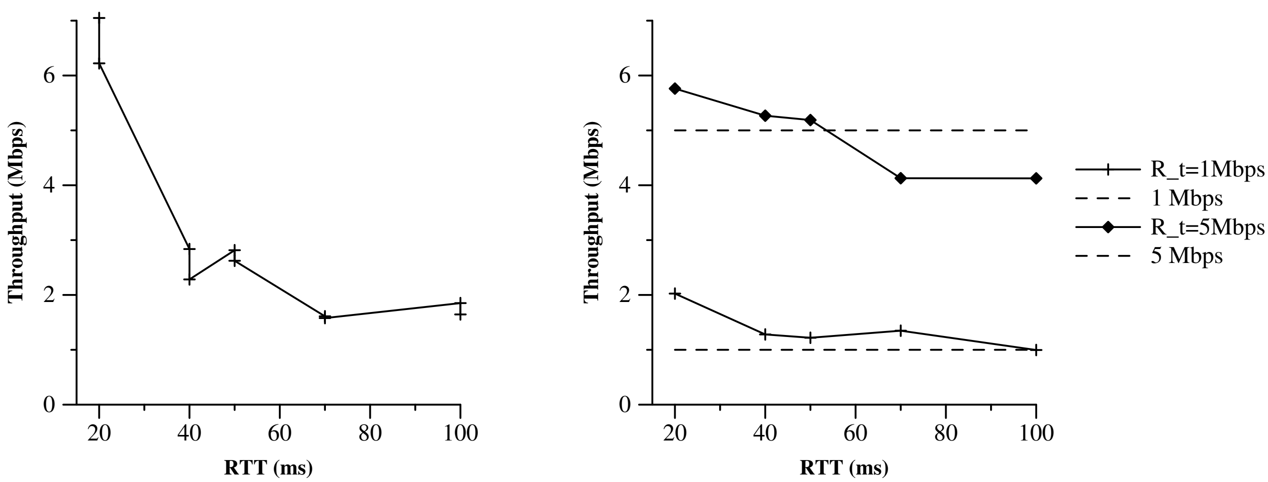
Table 2 compares the results from these two simulations. The RTTs of 10 connections are listed in column 2. Column 3 lists the throughput these 10 TCP (TCP-Reno) connections can achieve in today's Internet. The network bias against long RTT connections is evident. Similar to simulations in the previous section, TCP window oscillations are drastic and unpredictable. Column 4 lists the target rates, or R_T, for the 10 connections; and column 5 lists the throughput achieved by the TCP after having adopted a profile meter. The total link throughputs in both simulation runs are comparable: 30.51Mbps vs. 31.6Mbps, or 92.5% vs. 96% link usage.

Comparing the two simulation runs, we conclude 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 the Slow-Start phase in TCP 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 eliminated). Most importantly, now the system can provide quite different expected capacities to different connections with reasonable assurance.

TABLE 2. 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)	Non-EC (Mbps)	R_T (Mbps)	EC (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

Figure 13 are the throughput graphs of the above two scenarios. For each graph, we run five simulations of the exact configuration. Since each simulation uses a randomly generated seed, we get a slightly different result each time. Each data point on the throughput graph is an average of 5 simulations runs. Both graphs have the X axis as the TCP's round trip time, measured in ms; and the Y axis as the achieved throughput of a connection, measured in Mbps. The left graph plots the throughput in the current Internet scenario. Since every two TCP connections have the same RTT, there are two points for the same RTT. The achieved bandwidth decreases as the RTT goes up. There is a large variation in throughput for a given RTT, showing the unpredictably in the current Internet environment. The right graph plots the achieved throughput with their respective target-rates. The target rates are 1 Mbps and 5 Mbps, in dotted lines. A visual interpretation of how well a scheme works is how close the achieved throughputs are to the dotted lines. It is clear that the network bias is still visible, as shown by the slowly decreasing throughput as RTT increases. However, such bias has been much mitigated since the slop is not as deep as that in the left graph.



Achieved Throughput in the Current Internet   Achieved Throughput in the "Expected Capacity" Framework

Figure 13: Achieved throughput (TCP-Reno in a sender-controlled scheme)

Another interesting graph depicts the amount of *In* packets and *Out* packets in a RIO gateway queue. It gives an indication of what kind of mixed packets a gateway receives. Figure 14 is the gateway queue graph for a simulation run. It shows the *avg_in*, *avg_out*, and *avg_total* parameters of RIO gateway (gateway A). We observe the following: 1) The *avg_total* (avg_q in the figure) fluctuates between 7 and 40 packets. This implies that, on average, a packet waits between 1ms and 9ms[1] in the queue. 2) The gateway is dominated with *In* packets. This is because all hosts are sending simultaneously, using their respective profile meters. There are very few *Out* packet in the gateway (the avg_out is close to 0, barely distinguishable from the X axis). This is due to the gate-

1. (40 packets * 8 bits/bytes * 1000 bytes/packet) / 33Mbps

way's configuration. In this simulation, the parameters for dropping *Out* packets are (10, 30, 0.5). In other words, when *avg_total* is operating between 10 and 30 packets, the gateway is dropping with a certain probability (calibrated from 0.5); when the *avg_total* is above 30 packets, the gateway is dropping all *Out* packets. From the graph, avg_total is mostly between 15 and 30 packets, implying a high dropping probability for *Out* packets. Consequently, there are very few *Out* packets in the gateway.

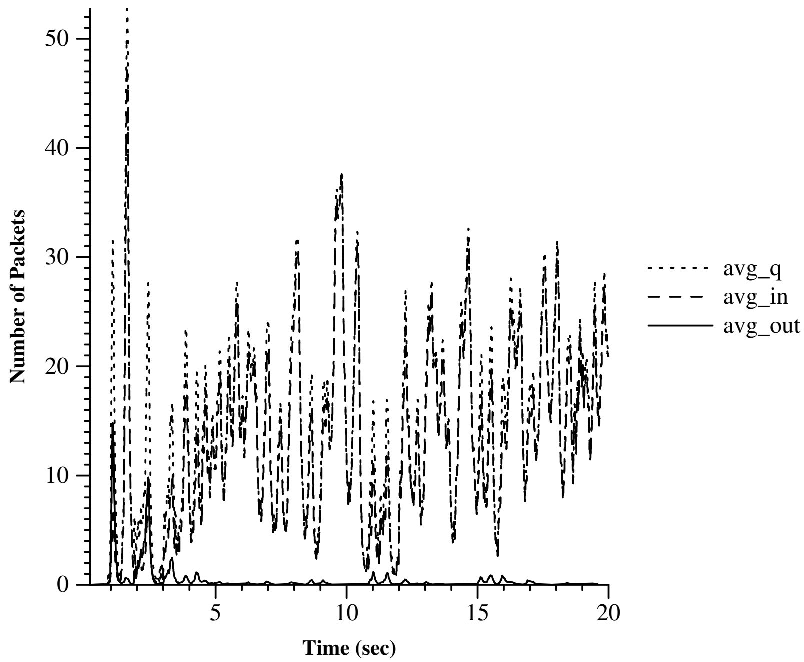


Figure 14: RIO gateway queue length

4.4 Sender-controlled scheme and network bias against long-RTT connections

It is important to realize that in all the above "Expected Framework" scenarios, the network bias against long RTT connections is only ameliorated, but not eliminated. This is because the profile meters are on the access path. Since the profile meters are outside the hosts, they have no knowledge of the instantaneous RTTs of TCP connections. Therefore, in order to achieve the "average target-rates to all destinations" goal, without knowing the RTT to any specific destination, each profile meter uses a presumed RTT to calculate its *win-length* variable. When the RTT to a destination is longer than this presumed RTT[1], there is still discrimination against connections with longer RTTs. Conversely, when a connection with an RTT shorter than the presumed RTT, it can still obtain more than its target-rate, due to its more responsive and opportunistic window algorithm. There-

1. The presumed RTT in this simulation is 60ms.

fore, there is a limit on how well the scheme of using a “presumed RTT” works. Put it 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.

CHAPTER 5

Simulation Results: receiver controlled scheme

Abstract: We present a receiver-controlled scheme in the “Expected Capacity” framework Details of the algorithm are presented and simulations are analyzed. The Explicit Congestion Notification (ECN) mechanism in TCP and RED gateways provides the basis for an elegant solution to solve the receiver-controlled problem, but unresolved issues remain.

5.1 Receiver-controlled scheme

In the “Expected Capacity” framework, the receiver-controlled scheme is the dual of the sender-controlled scheme. It requires two things from the current Internet: TCP implementations which understand Explicit Congestion Notification (ECN) bit semantics; and RED gateways which mark, instead of drop, packets when congestion occurs. In the receiver-controlled scheme, profile meters are installed on the access link from the receiver to the network. They interpret the service profiles and reset the ECN bit of a packet before the packet reaches receivers. Gateways that understand the RED semantics do not need to be modified to accomodate the receiver-controlled scheme.

5.1.1 RED for receiver-controlled scheme

The algorithms of RED for receiver-based scheme is the same algorithm as in figure 4, only substituting every “drop” with “mark”.

5.1.2 TSW profile meter for receiver-based scheme

Figure 15 contains the pseudo code for the TSW profile meter in the receiver-controlled scheme. Receiver-controlled profile meter will turn off ECN bit in a packet if the traffic has not yet exceeded the corresponding service profile.

```
Estimate sending rate R;
causedCong = whether ECN bit is set;
tagged = mark_packet(pkt, R);
if ( causedCong) {
        if ( ! tagged)
            turn ECN bit off;
        else
            /* leave ECN bit on, do nothing */
}
/* else, if it didn't cause congestion. let the packet through */

subroutine:   mark_packet (pkt, R)
              {
              if R < R_T, return 0;
              if R > R_T, with P_mark = (R - R_T)/ R_T, return 1;
              }
```

Figure 15. Pseudo of TSW profile meter for receiver-controlled scheme

5.2 Simulation setup

The configurations of the simulations are close to that described in section 4.1. The differences are listed as follows:

- *Gateways*
  Gateways are RED gateways that marks, instead of drops packets when congested.
- *Profile meters*
  Profile meters are located on access link from the receivers to the network. In other words, packets leaves the sender, passes the bottleneck link, and reaches the profile meters before it reaches its receiver. In the simulator, a profile meter is implemented as a subclass of a drop-tail link (drop-tail.cc).
- *Topology*
  We use a similar network topology as in the sender-controlled scheme. The only difference is that the profile meters are installed at the receivers' side, instead of sender's side. Figure 16 illustrates the receiver-controlled scheme with profile meters. All other configurations are the same.

5.3 Simulation results - single TCP connections

In this simulation, there are 10 TCP connections from hosts 0-9, each sending to their respective receivers. The bottleneck (router A to router B) link speed is 33Mbps. We have two separate simulations runs for comparison. One reflects the current Internet situation, another reflects the scenario in the "Expected Capacity" framework with profile meters installed on the receiver's access links.

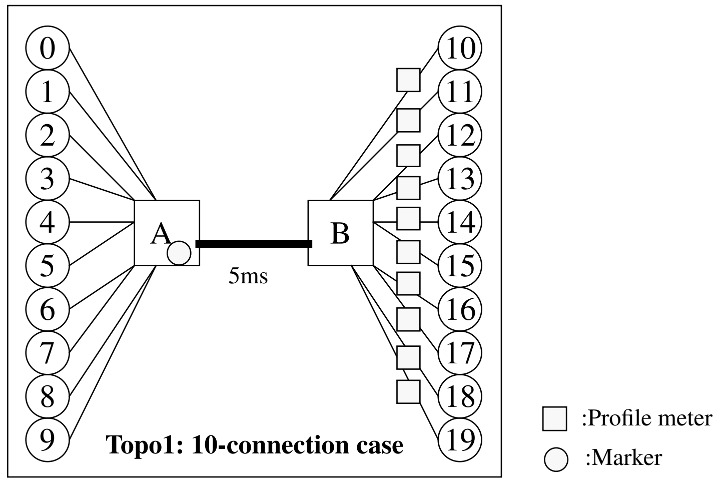


Figure 16. Topology used in a receiver-control scheme

TABLE 3. **Receiver-based. 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. Used TCP-Reno with ECN semantics.**

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

Table 4 lists the results from two simulation runs. Column 2 lists the round trip time of each TCP connection and column 4 lists their respective target-rates for the second simulation run. As shown in column 3, in today's network, short RTT connections have a clear advantage over long RTT connections. Whereas in the "Expected Capacity" framework (column 5), the network can allocate bandwidth according to the target rates R_T in times of network 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 gateways only mark packets, there are no packet drops. Consequently, TCP does not retransmits and 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.

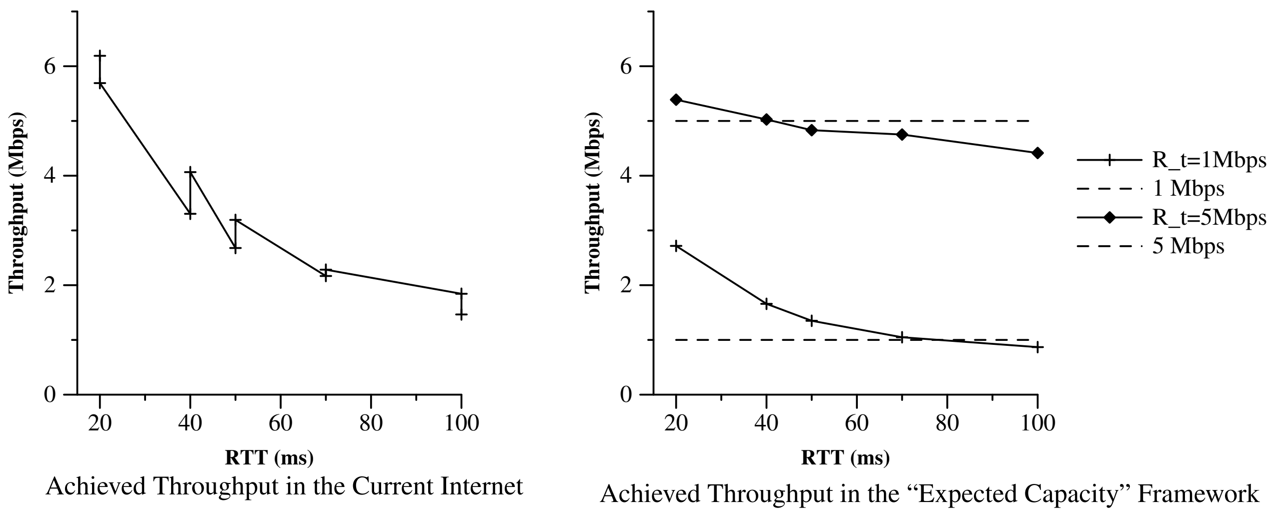


Figure 17: Achieved throughput in a receiver-controlled scheme

Figure 17 is the throughput graphs of the above mentioned scenarios, used averages from five simulations runs. Both graphs have X axis as the TCP's round trip time, measured in ms; and Y axis as the achieved throughput of a connection, measured in Mbps. The left graph plots the throughput in the current Internet. Since every two TCP connections have the same RTT, there are two points for the same RTT. The decaying curve shows that in the current Internet, network bias against long RTT connections is evident, and the achieved throughput is not predictable, since there is a large variation in throughput for a given RTT. The right graph plots the achieved throughput with their respective target-rates. The target rates are 1 Mbps and 5 Mbps, in dotted lines. In this case, most connections achieve a throughput (bullets) close to their contracted rate (dotted line). A visual interpretation of how well a scheme works is how close the achieved throughput is to the dotted lines. Network bias against long RTT connections is still visible, as shown by the decreasing throughput as RTT increases. However, such bias has been much ameliorated since the slop is not as deep as that in the left graph.

It is also worthwhile to compare the right graph in figure 17 with the right graph in figure 13, which is send-based scheme simulation with the same configuration. We notice that using the receiver-controlled scheme, the "Expected Capacity" framework can allocate bandwidth more effectively: the achieved throughput of each connection deviates less from its respective target-rate. This is due to the fact that in the receiver-controlled scheme, rate adjustments are more only through Fast-Retransmit phase, therefore, more controllable than the sender-controlled scheme.

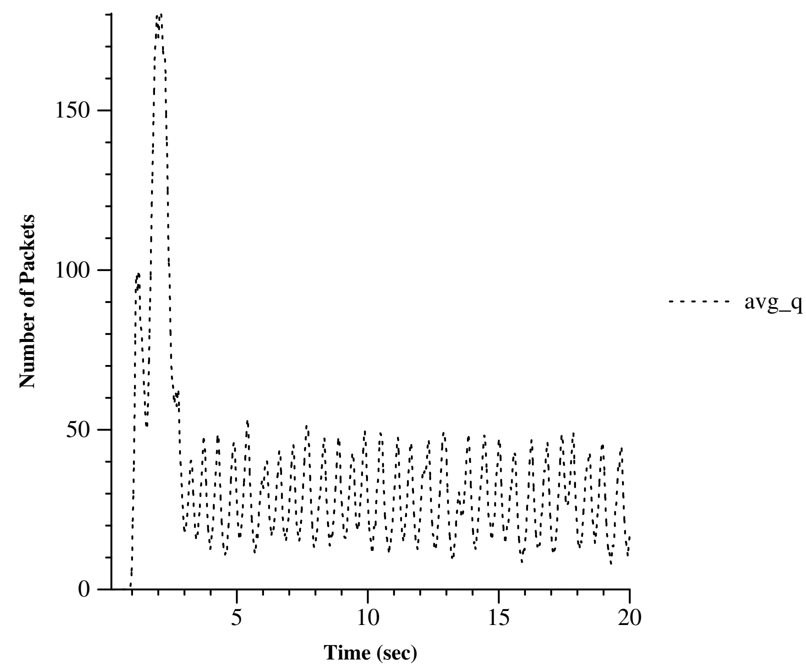


Figure 18: Average queue length in RED gateway (with ECN semantics)

Figure 18 is the average queue graph of the RED gateway in the simulation. There are two interesting observations about this graph. First, there is an initial big surge in queue length that results in an average queue length (avg_q) to be near 180 packets. This occurred during the first 3 seconds, during which period all connections just started. This initial surge warrants the following explanation. TCP starts off by probing the available network bandwidth in the Slow-Start phase, during which, it sends a surge of packets until one gets dropped. However, in the receiver-controlled scheme, the gateways do not drop any packets (unless they run out of buffers), so they would have to accomodate this surge of packets. The queue length only reduces when the sender has received acknowledgment packets from the receivers with ECN bit on. This reduction in queue length can only occur one round trip time later, and in the mean time, the gateways experience a big burst of packets.

The second observation is that after first three seconds, the avg_q oscillates between 20 to 50 packets. In other words, a packet is expected to wait between 5 ms to 12 ms[1] in the queue. Since the gateway starts turn on the ECN bit in a packet when the average queue length exceeds 15 packets, most packets leaving the gateway have their ECN bit on. Most of them will have the ECN bit turned off by the profile meters according to the profiles because the network is well provisioned. Only those packets which exceeded the receiver's profiles will still have their ECN bit on when arriving at the receiver. Therefore, in the next round trip, only a few selected connections will slow down. Consequently, the queue length reduces. Comparing to the sender-controlled scheme, however, the receiver-controlled scheme cannot manage the average queue length as effectively.

5.4 Receiver-controlled scheme and network bias

Similar to the sender-controlled scheme, in all the above "Expected Framework" scenarios, network bias against long RTT connections is only mitigated, but not eliminated. This is because the profile meters are on the access path. Since the profile meters are outside the hosts, they have no knowledge of the instantaneous RTT of TCP connections. Therefore, in order to achieve the "average target-rates to all destinations" goal, without knowing the RTT to any specific destination, each profile meter uses some presumed RTT to calculate its *win-length* variable. When the RTT to a destination is longer than this presumed RTT[2], there is still discrimination against connections with longer RTTs. Conversely, when a connection with an RTT shorter than the presumed RTT, it can still obtain more than its target-rate, due to its more responsive and opportunistic window algorithm. Therefore, there is a limit on how well the scheme of using a "presumed RTT" works.

A solution to the problem is presented in the next chapter.

5.5 Concerns about receiver-controlled scheme

There are two issues about the receiver-controlled scheme which can be potentially detrimental to the network. First, using receiver-controlled scheme might result in persistent long queues in the gateways. This can happen when the downstream ISP has contracted too much service profiles. Most packets with ECN bit turned on by

1. (avg_q (pkts) * 8000 (bits/bytes)) / 33Mbps.
2. The presumed RTT in this simulation is 60ms.

the gateways will have their ECN bit turned off by the corresponding profile meters, therefore, the congestion feedback loop via acknowledgment packets to the original sender is broken. The sender will continue to overwhelm the gateways with packets until the gateway runs out buffer. Up to that point, the gateway has long, persistent queue length, causing long delays. When the gateways run out of buffer, they degenerate into a drop-tail gateway, and will likely to cause global synchronization. But during the time of building up the queue, packets will experience long delay in the congested gateway. A gateway which supports receiver-based scheme can have an additional RED algorithm to manage queues with higher (min, max) values to prevent persistent long queues, or drop-tail like global synchronization.

Second, a denial-of-service attack can be easily designed to take advantage of the receiver-controlled scheme. For example, a malicious user has learned that some host, say, a web server, offers receiver-paid and receiver-controlled scheme in order to attract customers (This is the network equivalent of 1-800 numbers). But instead of running a regular TCP connection with ECN semantics, the user sends a CBR stream, ignoring all the acknowledgment packets with ECN bits on. The CBR stream not only will cause significant congestion (and queue) at the gateway, but also will consume a portion of the service profile of the receiver. The portion of the profile could be large enough to prevent other regular users from talking to the receiver.

A similar scenario can be constructed, which is not only detrimental to a particular receiver, but also to the network and possibly cause network collapse: a host sets its receiver-controlled bit on, so that packets will be delivered by network routers to the receiving host, except that the presumed receiver is not willing to pay for such packets. As a result, network delivers packets which are ultimately being dropped. This is a waste of network resources, and might lead to congestion collapse.

Unlike the similar case with a sender-controlled scheme (detailed in chap 7), where the “Expected Capacity” framework has an elegant technical solution to defend malicious users, a denial-of-service attack in a receiver-controlled scheme would have to resort means other than technical ones. In absence of any network level defense mechanisms against network abuse, network would have to rely on higher level protocols or administrative means to stop such attack. Some current work has started looking into this issue. [FloFal97, Rena, Rob’s work].

CHAPTER 6 Looking into the future: TCP-Sack and taggers in the hosts

Abstract: We present simulations results for the "Expected Capacity" using TCP-Sack and putting profile meters in the operating systems of the end hosts. Both modifications have improved results.

6.1 TCP-Sack

TCP-Sack[Flody96b, FalFlo96], or TCP with Selective Acknowledgment, has a very different approach for handling acknowledgments from the previous versions of TCP, TCP-Reno or TCP-Tahoe. Previous versions of TCP acknowledges the highest sequence number the receiver ever receives. When a packet is lost, either due to a congested gateway or a lossy link, the receiver will send duplicate acknowledgments (duplicate acks) upon receiving successive packets after the lost packet. The sender infers from the duplicate acks that a packet had been lost, and retransmits the presumably lost packet, usually, the packet immediately after the one that the duplicated acks acknowledge. Each such retransmission also results in a reduction in congestion window *cwnd*, and subsequently, a reduction in sending rate.

Since the information as which packet was lost is incomplete in early version of TCP, the sender can only deduce which packets had been lost. This leads to either unnecessary retransmissions of packets that did not get lost, or successive reduction of congestion window (*cwnd*), and eventually, a *cwnd* so small that the sender can no longer recover the lost packets. The sending TCP will have to wait for a timeout to recover the lost packets, and start from Slow-Start again, with a much reduced *ssthresh*. This has a much more drastic effect on TCP performance, and the "Expected Capacity" framework tries to avoid it by adding probabilistic functions in both the tagging and dropping schemes.

TCP-SACK uses a completely different semantics: the acknowledgments now contain complete information as which packets have been received and which haven't. This complete information allows the TCP sender to retransmit packets selectively, and avoid unnecessary retransmits. The TCP sender can also recover multiple packet losses within a window, but only reducing the congestion window once. Therefore, TCP-sack can stay in the Fast-Retransmit phase most of time, making better use of the network bandwidth.

As the TCP-SACK is able to maintain Fast-Retransmit phase, the design for the profile meters and droppers could be simpler. In the tagging algorithm designed for TCP-Reno, profile meters have to carefully avoid tagging two consecutive packets as *Out*, as to avoid them being dropped together by a gateway when congestion occurs. For example, TSW incorporates a probabilistic tagging algorithm to achieve such goal. Using TCP-SACK allows the tagging algorithm to be precise about tagging *Out* packets when the connection exceeds the target rate. Thus, the amount of *Out* packets arriving at gateways is a more precise indication that some connections have exceeded their respective profiles. The gateways can be more discriminating when dropping *Out* packets since TCP-SACK can recover from successive packet drops. As a result, the overall system is more responsive to congestion without suffering potential performance degradation.

6.2 Simulation results

We use the same topology as in the sender-controlled scheme (section 4.1). We repeat those simulations with each host using TCP-Sack: the first scenario is to simulate the current Internet; the second case is to simulate the "Expected Capacity" framework. The profile meters are installed on the access links.

Table 6 lists the simulation results. The two primary observations from section 4.3 (sender-based, TCP-Reno) still hold here: in the current Internet, network bias against long-RTT TCP connections is evident; and in the "Expected Capacity" framework, the allocation of bandwidth is roughly according to R_T, the target-rates.

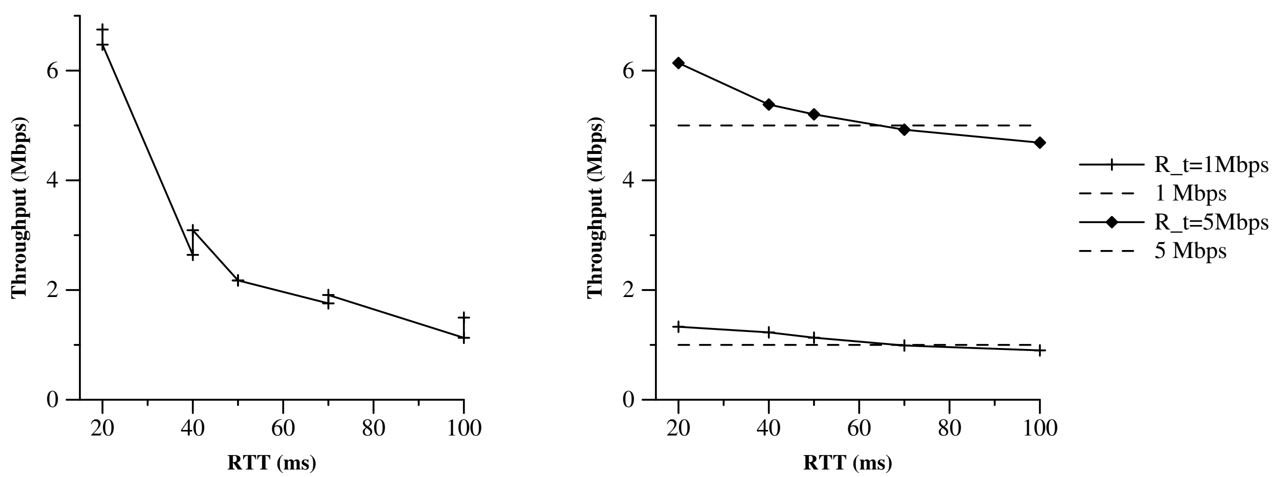
TABLE 4. With TCP-Sack to recover multiple packet losses. Parameters for RED are (10, 30, 0.02); and parameters for RIO are (40, 70, 0.02) and (10, 30, 0.5). Bw=33Mbps. Used TCP-SACK.

Conn #	RTT (ms)	without EC (Mbps)	R_T (Mbps)	In EC (Mbps)
0	20	6.74918	1	1.33071
1	20	6.47331	5	6.13875
2	40	2.64113	1	1.2283
3	40	3.09084	5	5.38146
4	50	2.17542	1	1.13145

TABLE 4. 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)	without EC (Mbps)	R_T (Mbps)	In EC (Mbps)
5	50	2.65581	5	5.20269
6	70	1.75715	1	0.988921
7	70	1.90942	5	4.92265
8	100	1.12921	1	0.899915
9	100	1.49762	5	4.68653
total		30.0791		31.9114

The RIO gateway configurations are slightly different in this set of simulations from those in section 4.3. The parameters for *Out* packets is (10, 30, 0.5), instead of (10, 30, 0.2) as in section 4.3. This means that the gateways drop *Out* packets harsher (with probability calibrated on 0.5 vs. 0.2) when the average total queue length exceeds threshold of 10. Since the end host TCP-Sack can recover from multiple packet drops in a window without going into Slow-Start phase, there is no apparent danger for gateways to drop multiple packets consecutively together. This allows the gateway to distinguish *In* and *Out* packets more clearly. As a result, the achieved throughputs in the "Expected Capacity" framework is better, deviating less from the expected target-rates. This is shown in column 5 in table 6.



Achieved Throughput in the Current Internet Achieved Throughput in the "Expected Capacity" Framework

Figure 19: Achieved throughput in a sender-controlled scheme using TCP-Sack

Figure 19 are the average throughput graphs for the above simulations. Compared to figure 13, the graphs for similar simulations with TCP-Reno, the graphs in figure 19 show smaller variances. In the left graph, the achieved throughput is more consistent and predictable. In the right graph, achieved throughputs are close to the respective target-rates. This reflects that TCP-SACK works better than other versions of TCPs in the "Expected Capacity" framework.

6.3 Location of the profile meters

In all the previous simulations, the profile meters are located on the access link between the hosts and the immediate gateways. Since the profile meters are separate from the hosts, they have no knowledge about important state information available to TCP, for example, the RTTs. One of the limiting factors in achieving the contracted profile — "same throughput to anywhere" — is the network bias against long round trip time. Since the profile meters do not know the instantaneous RTTs of TCP, the profile meter can not know the appropriate amount of *In* packets to fill in the pipe[1]. To solve this problem, TSW profile meters assumes some *average* round trip time and calculates the acceptable number of packets tagged as *In* using this assumed RTT. Since this *average* RTT does not reflect the instantaneous RTT of a particular TCP connection, the profile meters are limited in terms of how well they can actually perform to fulfill the goal of this type of service profile. (Had the service profiles be another other kind, for example, "a certain throughput from A to B", then this TSW profile meter is perfectly sufficient.)

There are a number of solutions to the above problem of "not knowing the instantaneous RTT". For one, a profile meter on the access link can *estimate* the connection's RTT. However, this requires keeping state information about particular connections from a host, and duplicating some of the functionality that is already in the TCP implementations. (TCP estimates its round trip time and uses the estimate to set the retransmission time-out timer.) This approach, of course, preserves the separation of the profile meters from the hosts, so that the trust is never compromised so long the profile meters are controlled by the ISP. We find this approach cumbersome and unnecessary.

A second approach is to change TCP's window open up algorithm to be "fair", regardless of the distance this connection is sending to. The suggested algorithm (section 4.2) shows that TCP connections through the same bottleneck link can achieve the same throughput regardless the RTTs. However, whether this window open up algorithm for TCP is "fair" is quite controversial. There is no reason to believe that this algorithm is more fair than the existing one. In addition, changing an exiting, well-accepted protocol in a major way is practically impossible.

1. Imagine a pipe of RTT long and target-rate wide.

A third approach is to separate functionality of trust from the functionality of “bandwidth guarantees” in the profile meters. The idea is to push profile meters to the endhost’s operating system. A simple solution is a module in an operating system that has read access to current TCP state variables, specifically, only the estimated RTT variable. Since TCP module in a host already estimates the instantaneous round trip time, this profile meter module can simply take advantage of it. This accomplishes the bandwidth guarantee functionality. In addition, a simple profile meter can still sit on the access link on the ISP side just to monitor the incoming In packets. This profile meter can be invoked once in a while, to make sure that the endhost is not cheating by replace the correct in-host profile meter module.

In our simulations, we use the third approach. The profile meters are able to access the instantaneous RTT from the TCP module, and uses this to calculate the *win-len* variable in TSW. This works well in allocating the precise target-rate to different connections regardless of RTT.

6.4 Simulations for tagging in the hosts

6.4.1 Simulation setup

We use the topology in the sender-controlled scheme (section 4.1). We deliberately make the range of RTTs for the 10 TCP connections longer to demonstrate that this scheme works well for our service profiles regardless of the distance. The RTTs range from 16ms to 150ms.

6.4.2 Simulation results

Table 7 lists the results of simulation. We compare two scenarios: one with profile meters separate from the end hosts, and one with the profile meters inside the hosts, using TCP’s estimate of RTTs to change *win-length* variable dynamically. Profile meters in the hosts (TCP-Sack) tag the outgoing packets by using the same TSW algorithm. Column 3 lists throughput of connections when the profile meters are outside the host, therefore not knowing the respective RTTs. Column 5 lists the corresponding throughput when the profile meters are inside the hosts. It is evident that if the profile meters have knowledge of the RTTs, allocation of bandwidth in the “Expected Capacity” framework can be very accurate. For example, when a TCP connection has a RTT of

150ms, using a profile meter inside the host, TCP can achieve 0.9748Mbps (for a 1Mbps service profile), vs. 0.622Mbps using a profile meter outside the host.

TABLE 5. RIO gateway parameters are (40, 70, 0.02) for Ins and (10, 30, 0.5) for Outs. Used TCP-Sack.

Conn #	RTT (ms)	Tagger outside host (Mbps)	R_T (Mbps)	Tagger in host (Mbps)
0	16	1.5981	1	1.69298
1	16	6.3969	5	5.95957
2	50	1.18279	1	1.207
3	60	5.21947	5	5.32791
4	90	0.856795	1	1.03357
5	100	4.62787	5	5.15628
6	120	0.674382	1	1.03703
7	130	4.71473	5	5.17554
8	150	0.622031	1	0.974896
9	160	4.94274	5	4.96005
total		30.83581		32.524

Figure 20 are the average throughput from the above two scenarios. Each data point in the graphs are an aver-

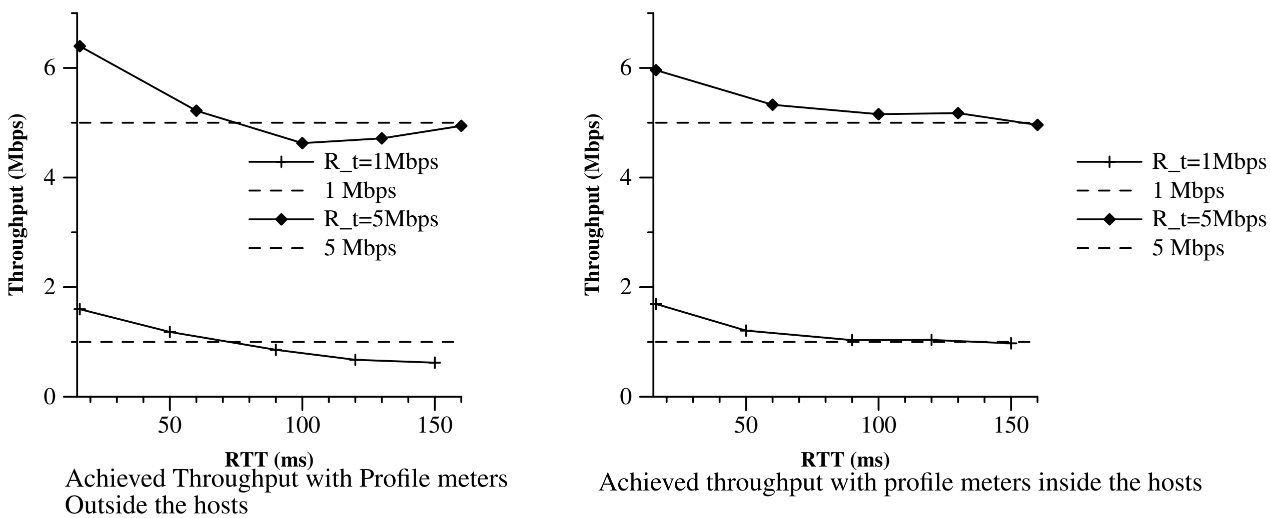


Figure 20: Comparison of effects of Profile meters at different locations

age from five simulations runs. It is evident that when using profile meters inside the hosts (right graph), the "Expected Capacity" can meet the requirements of the service profiles very well. The allocated bandwidth during network congestion is precisely according to the target-rates, with very small variations. The short-RTT connections are more opportunistic, so whenever there is available bandwidth in network, the small-RTT connections will always be the first ones to get such bandwidth. The total link bandwidth is 33Mbps, greater than the sum of all profiles (30Mbps). This explains why small-RTT connections perform slightly better than long-RTT connections. In contrast, when the profile meters are outside the host (left graph), the network bias is more pronounced.

CHAPTER 7 In the presence of non-responsive connections

Abstract: This chapter shows simulation results from the "Expected Capacity" with some non-conforming users. The "Expected Capacity" provides can protect users who have service profiles from those who do not. It also provides a simple mechanism to easily identify the non-conforming users.

7.1 Non-responsive users in the "Expected Capacity" framework

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, or 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 can 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 for a sufficient long period, 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.

7.2 Simulation setup

We use a similar topology as in section 4.2. As shown in figure 21, ten hosts are connected to ten peer hosts, sharing a common link between routers A and B. Each host (*i*) has a TCP connection to its peer host (*i+10*). There are altogether 10 TCP connections. The ten connections are of different Round Trip Time (RTTs). They can be divided into five groups. The two connections in each group have the same RTTs, but different target-rates. The five groups each has a different RTT than other groups. The RTTs for the five groups are 20ms, 40ms, 50ms, 70ms and 100ms, respectively.

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. The achieved throughput of the CBR connection is calculated by creating a receiver sink and calculated the received packets over time.

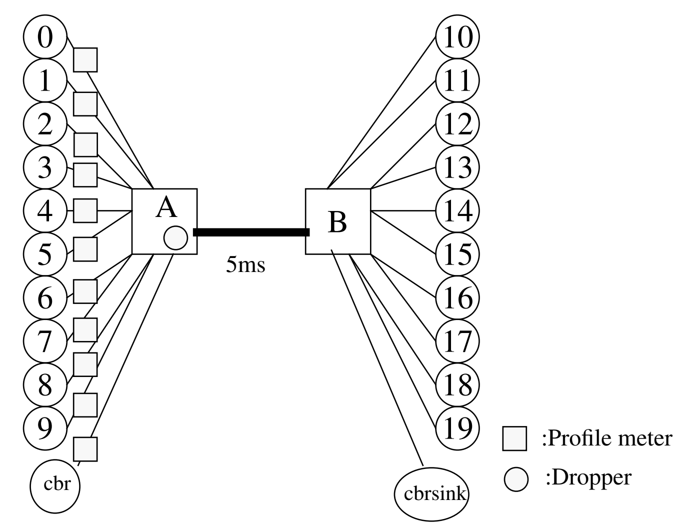


Figure 21: Topology with a non-responsive user

7.3 Simulation results

Table 6 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 cause consistent congestion in the gateway, making all responsive TCP connections to back off. Column 3 illustrates this effect: the CBR gets almost all its packets through at the expense of other TCP connections' performance degradation. Compared to column 3 in table 3, where the CBR is absent, connections 0 to 9 in this simulation all suffer some performance degradation. In 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 1.78Mbps or 29.6% of its 6Mbps sending rate in our framework, vs. 5.85Mbps or 97% of its sending in today's Internet.

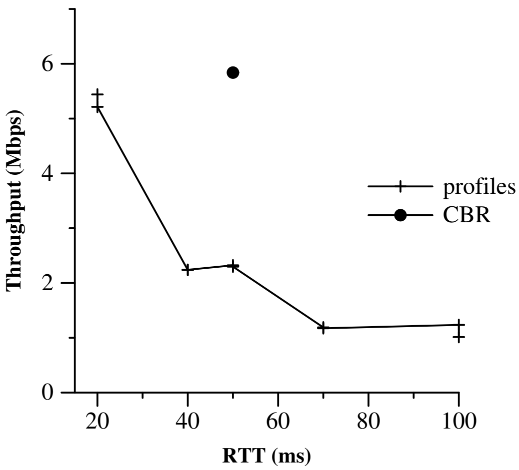
Otherwise, the observations from previous chapters still hold. The discrimination against long-RTT connections is present in both current Internet and the "Expected Capacity" framework. In the "Expected Capacity" framework, such bias is mitigated.

In addition, unlike the TCP-Sack simulations, where TCP connections with the same RTTs have achieved quite comparable throughput: the throughput for the same RTT connections here is quite unpredictable. This is due to the fact that when gateways are flooded with *Out* packets, they have consistent long queues. As a result, the gateway drops most of the *Out* packets. The discard of a significant amount of *Out* packets affects the performance of regular TCP connections with profiles, as they might recover from multiple packet drops within a window.

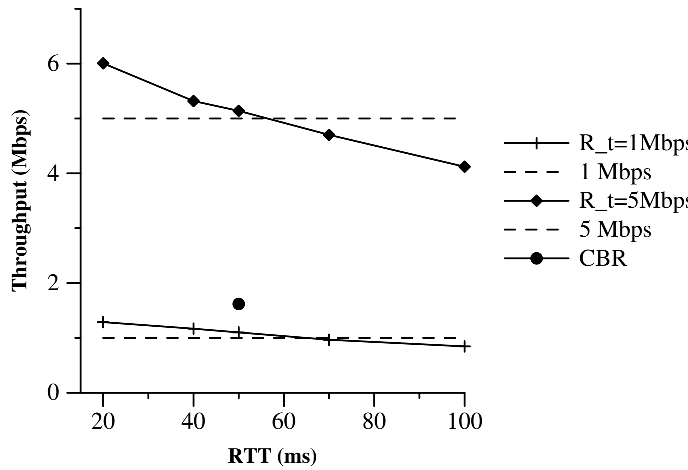
TABLE 6. 10-connection case with a non-responsive connection (CBR). BW= 33Mpbs, CBR is sending at 6Mbps. RIO parameters: (40, 70, 0.02) for Ins and (10, 30, 0.5) for Outs. Used TCP-SACK

Conn #	RTT (ms)	Today's Internet (Mbps)	R_T (Mbps)	with RIO-TSW (Mbps)
0	20	5.40752	1	1.21924
1	20	5.36329	5	5.87978
2	40	1.98478	1	1.1372
3	40	2.56938	5	5.15273
4	50	2.443	1	0.989687
5	50	2.54567	5	4.99475
6	70	1.28912	1	0.837817
7	70	1.53377	5	4.86143
8	100	1.1074	1	0.720575
9	100	1.50127	5	4.72234
CBR	50	5.85338	0	1.78195
total		31.59858		32.297499

Figure 22 is the average throughput graph for the non-conforming user scenario. The left graph shows the distribution of bandwidth in the presence of a CBR connection in the current Internet. The right graph shows the distribution of bandwidth in the "Expected Capacity" framework when all other TCP connections are protected by their respective profiles. The most significant change from the left graph to the right graph is that the CBR connection gets a much smaller share of total bottleneck bandwidth, as shown in the down shift of an unconnected dot. Nevertheless, the “Expected Capacity” framework cannot completely eliminate the interference from the CBR connections, and other TCP connection’s throughput suffer as a result. This can be observed by comparing this with the right graph of figure 13.



Achieved throughputs in the presence of a non-conforming connection. Current Internet



Achieved throughputs in the presence of a non-conforming connection. The Expected Capacity

Figure 22: Comparison of performance in the presence of a non-conforming connection

CHAPTER 8 Cascaded profile meters

Abstract: This chapter focuses on connections' end-to-end performance after traversing cascaded profile meters. For a properly provisioned network where profile meters are properly configured, simulations show that cascaded profiles has little negative effect on end-to-end performance.

8.1 Cascade profile meters

When the "Expected Capacity" framework has been fully deployed, a packet traverse through the network to reach its destination will encounter a series of profile meters, each located at different AD boundaries. Figure 1 in Chapter 1 illustrates this scenario. The profile meters on those boundaries will interpret the corresponding profiles. In a well-provisioned network, an ISP should contract a profile from its downstream ISP (data flows from source to destination, or from upstream to downstream.) equal to or greater than the sum of all upstream profiles it has contracted out. This would ensure that all *In* packets from upstream will remain *In* packets throughout. However, often this is not the case. Occasionally, when an ISP is under-provisioned, it contracts a profile less than the sum of all upstream meters, some *In* packets will be turned into *Out* packets at downstream profile meters. Additionally, some *In* packets can be turned into *Out* packets at downstream profile meters even when the downstream profile supposedly equal to the sum of all upstream profiles. This maybe the case when the tagging algorithm used is not linearly addible, whereas the profiles are. For example, when the tagging algorithm used is a probabilistic one like TSW rather than a simple token-bucket. The effect of probabilistic tagging algorithm in cascaded profile meter on end-to-end TCP performance is not clear.

In this chapter, we will explore this topic. We show results from a set of simulations to illustrate the effect of cascade profile meters on end-to-end performance, in which the tagging algorithm is probabilistic. We study the case in a well-provisioned network and in an under-provisioned network.

8.2 Simulation results

8.2.1 Simulation setup

The simulation setup is listed as following:

- *Topology*
  There are two topologies used for this set of simulations (Figure 23), one as the controlled case where there are individual profile meters for each of the five connections, another as the compared case where the same five connections go through a series of profile meters and droppers. In the compared case, each connection first traverses its individual profile meter, a dropper (router A), an aggregated profile meter at router B, and then, a dropper and a congested bottleneck at router C.

  In both cases, the individual access link from a host to the first router is 10Mbps. Each individual profile meter is set to have target-rate as 1Mbps. The RTTs of different connections are different, ranging from 30ms to 70ms. The corresponding connections in two cases have the same RTT.

  In the compared case, links from router A to router B, and from B to C, are 10Mbps each. Link C-D has a speed of 5.6Mbps, and is the only bottleneck. For the aggregated profile meter at router B, we experimented with different target-rates.

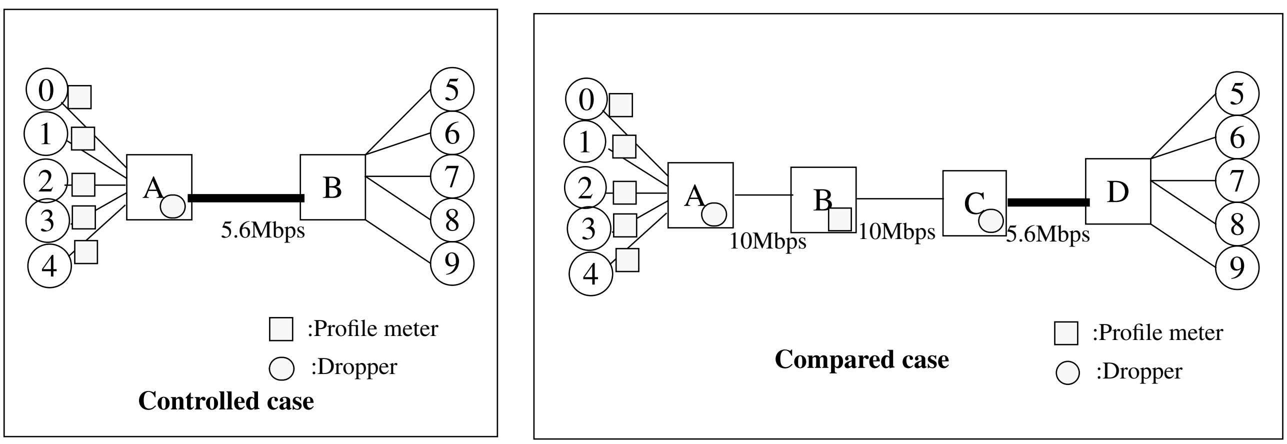


Figure 23: Compare cascaded profiles with single profile

- *RIO parameters*
  In the controlled case, RIO (router A) parameters: (7,14, 0.02) for *In*s and (5, 10, 0.5) for *Out*s. In the compared case, RIO (router C) parameters: (7,18,0.02) for *In*s, and (5, 15, 0.5). TCP packet size is 500 bytes.

- *Cascaded profile meter*
  The cascaded profile meter at router B use TSW algorithm. The algorithm differs slightly from the one presented in chapter 3, in the following way: it estimates the average rate from *In* packets only, and when this estimated rate exceeds its target-rate, it tags *In* packets into *Out* packets. In other words, the cascaded profile meter leave *Out* packets from the upstream intact, but may turn *In* packets into *Out* packets. The cascaded profile meter tags packets probabilistically. In the simulation, the cascaded profile meter has a target-rate of 5Mbps.

8.2.2 Simulation results

The throughput results from a particular simulation run are listed in Table 7. There is still the network bias against long RTT connections, as discussed in previous chapters. In addition, the simulation results show that the cascaded profile meter's impact on end-to-end throughput is rather negligible. To confirm this, we traced

TABLE 7. Cascaded profile meters with controlled and compared cases. Bottleneck link speed= 5.6Mpbs. The additional profile meter has target-rate of 5Mbps. Used TCP-SACK.

Conn #	RTT(ms)	Target-Rate (Mbps)	Achieved Rate (controlled case) Mbps	Achieved Rate (compared case) Mbps	In packets tagged	Out packets tagged
0	30	1	1.30024	1.22604	2549	437
1	40	1	1.09194	1.17093	2450	323
2	50	1	1.09401	1.01913	2413	480
3	60	1	1.06269	1.107	2362	255
4	70	1	0.953347	0.932333	2084	193
Total			5.502227	5.455433	11858	

the number of *In/Out* packets tagged at each individual profile meters and the number of *In* packets turned into *Out* packets at the cascaded profile meter. The number of *In/Out* packets tagged at by individual profile meters are listed in columns 6 and 7 in Table 7. Just for comparison, out of the total 11857[1] *In* packets that go through the cascaded profile meter, 158 are turned into *Out* packets. 158 *Out* packets distributed among five connections is significantly fewer than the number of *Out* packets tagged by individual profile meters (column 7). Therefore, the impact of an additional profile meter is minimum.

We also did simulations using the same topology in which the additional downstream profile meter has a *target-rate* of 4Mbps, which is significantly less than the sum of all upstream individual profiles. However, since the limiting factor is rather the bottleneck link speed of 5.6Mbps, each connection still achieves throughput similar

1. The sum of all *In* packets from upstream profile meters (11858) less one, which was dropped at link A-B.

to those listed in Table 7. In this particular case, the additional downstream profile meter has no constraint effect on the throughput. This case is similar to a well-provisioned network where the contracted profile is less than the bottleneck link speed, in which case, a connection can go beyond its profile. The result of this simulation is shown column 4 of Table 8.

However, one should realize that if there are additional congestion in the downstream, then the under-contracted profile will have an impact on individual connection's end-to-end performance. In this case, the link speed becomes a limiting factor, and the case is reduced to a under-provisioned case where neither connection can achieve its target profile, since some *In* packets have to be dropped due to congestion. The result is shown in column 5 of Table 8. The bandwidth of link C-D is 4Mbps, and the profile is 4Mbps.

TABLE 8. Cascaded profile meter case. When the downstream profile meter is not sufficient, or the network is under-provisioned.

Conn #	RTT(ms)	Target-Rate (Mbps)	The link C-D is 5.6Mbps but the cascaded profile is 4Mbps	When the network is under-provisioned. Link C-D = 4Mbps
0	30	1	1.20459	0.874846
1	40	1	1.17409	0.794778
2	50	1	1.00739	0.740870
3	60	1	1.0069	0.835148
4	70	1	0.954569	0.734181
Total			5.347539	3.979823

8.3 Conclusion

We have found that even with probabilistic profile meters using TSW algorithm, the cascaded profile meters can be installed without affecting the end-to-end performance of TCP connections significantly. This implies that between AD boundaries, additional profile meters can be installed so long as the downstream profiles are greater than the upstream profiles an ISP has contracted out.

CHAPTER 9 Aggregated Profile Meters

Abstract: Cases where profile meters for aggregated traffic are studied. We study 1) how bandwidth is allocated among connections sharing the same profile, 2) how well aggregated profile meters vs. individual profile meters regulate individual connections.

9.1 Introduction

In this chapter, we are concerned with how the "Expected Capacity" framework allocates bandwidth when traffic is aggregated from a number of TCP sources. Aggregated traffic has different characteristics from that of an individual TCP connection. Usually, aggregated traffic is more stable, and does not have the pronounced sawtooth swings an individual TCP connection has. Aggregated profile meters are profile meters located at routers where traffic has already been mixed.

In particular, we are interested in study the distribution of bandwidth among connections, and how aggregated traffic profile meters can control the individual traffic rate. We study the two questions in two separate sender-based simulations.

9.2 Taggers in the center of the network: aggregated traffic

9.2.1 Simulation setup

We use topology presented in figure 24. The simulation setup is similar to that in section 4.1 except that the profile meters are for a number of hosts instead of an individual host (figure 11). For example, one such profile meter is for aggregated traffic from hosts 0, 1, 2, 3 and 4, and the other is for packets from hosts 5, 6, 7, 8 and 9. The topology is used to study behaviors of individual connections when there is an aggregated profile meter for a number of TCP connections.

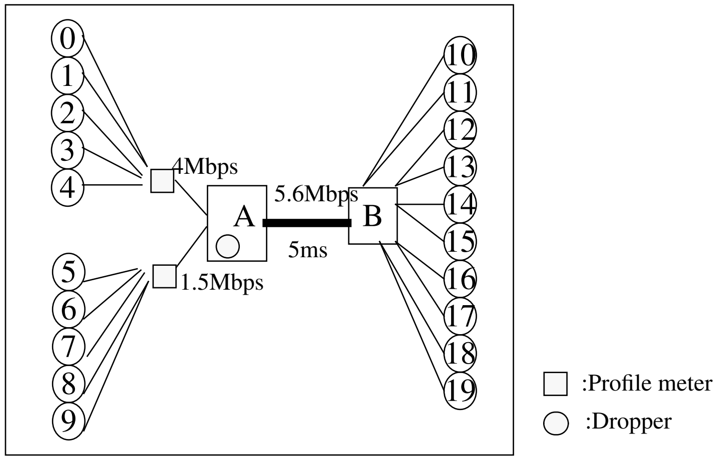


Figure 24: Connections with aggregated traffic meter

The bottleneck link A-B has a link speed of 5.6Mbps. The two aggregated profile meters are configured with target-rates of 4Mbps and 1.5Mbps, respectively. Hosts 0 - 4 share the aggregated profile meter 1 (4Mbps), and hosts 5-9 share the aggregated profile meter 2 (1.5Mbps). We compare two cases. In the first case, hosts 0-4 each has a TCP connection, sharing aggregated profile meter 1; simultaneously, only host 6's TCP is active and using profile meter 2. The results are listed in column 4 of Table 9. For comparison, in the second case, all hosts have active TCP connections. The first five TCPs share profile meter 1, and the second five TCPs profile meter 2. The results are listed in column 5 of Table 9.

We make the following observations. First, it is clear that among the connections which share the same profile meter, the distribution of the bandwidth is primarily according to the RTT of each connection. For example, in scenario 1, as the RTT increases, the achieved throughput of connections 0-4 decreases. The effect of a profile meter is to "protect" the contracted share of bandwidth during congestion, as connection 6 achieves 1.416Mbps (given a 1.5Mbps service profile). Had connection 6 not have such a profile, it would have achieved a throughput comparable to that of connection 2 (1.01318Mbps), since they have the same RTT of 40ms. Among the connections that are sharing the same profile meter, the network bias against long RTT TCP connections is evident. For example, in scenario 2, the bandwidth distribution among connections 0 to 4, and among connections

5-9 is strictly according to the RTT. However, since the two aggregated profile meters have different target-rates, the throughput of each connection is constrained by the target-rate of the profile meter it is using.

TABLE 9. Aggregated taggers case. Topology 2. Bottleneck link A-B = 5.6Mbps. Gateway parameters are (12, 30, 0.02) for Ins and (2, 15, 0.5) for Outs, respectively.

Conn #	RTT (ms)	Target rate (Mbps)	Scenario 1 individual rate (Mbps)	Scenario 2 individual rate (Mbps)
0	20		1.14091	1.19158
1	40		1.0318	0.891620
2	50		0.960004	0.787156
3	70		0.643260	0.575159
4	100		0.392509	0.564507
Subtotal		4	4.168483	4.01
5	20		N/A	0.443117
6	40		1.41626	0.347406
7	50		N/A	0.320943
8	70		N/A	0.235642
9	100		N/A	0.232686
Subtotal		1.5	1.41626	1.57979
Link Total		5.6	*5.584743*	*5.58979*

9.3 Effect of aggregated profile meters on individual connections

9.3.1 Simulation setup

In this set of simulations, we study how different profile meters -- individual or aggregated profiles meters-- affect the performance of individual TCP connections. We compare the performance of five connections sharing one aggregated profile meters with five connections with individual profile meters.

- *Topology*

  The topology is shown in figure 25. Connections 0-4 traverse through two kinds of profile meters, one individual profile meter and one aggregated profile meter. We did two simulations, one controlled case, and another case for comparison. The difference is that in the controlled case, connections 5-9 share one aggregated profile meter, and in the compared case, connections 5-9 each has an individual profile meter.

  In both cases, each access link (from host to the first router, either A, C, or E) is an ethernet link, with 10Mbps. The speed of links between routers are shown in the figure. Connections 0-5 each has an individual profile of 1Mbps, and the aggregated profile for them is 5Mbps, enforced by the profile meter at router B. The bottleneck in both cases is the link between routers D and E, In the controlled case, router C has an aggregated profile meter of 5Mbps, shared among five connections 5-9. In the compared case, each connection in connections 5-9 has an individual profile meter of 1Mbps, same as that of connections 0-4.

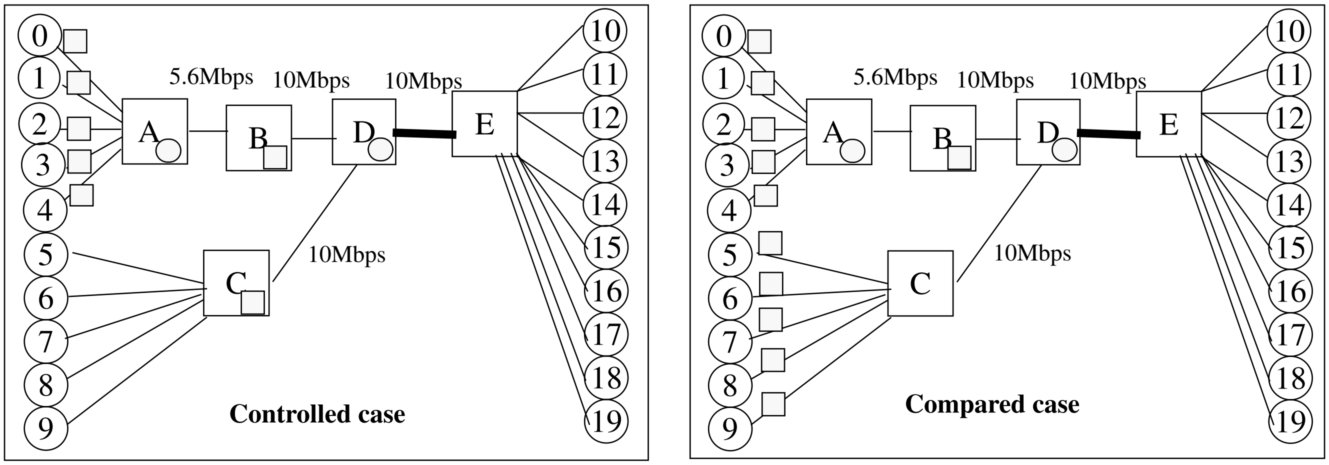


Figure 25: Compare aggregated profiles with individual profiles

- *RIO parameters*

  In the controlled case, RIO (router A) parameters: (7,14, 0.02) for *In*s and (5, 10, 0.5) for *Out*s. In the compared case, RIO (router C) parameters: (7,18,0.02) for *In*s, and (5, 15, 0.5) for *Out*s. TCP packet size is 500 bytes. TCP-Sack is used.

- *Cascaded profile meter*

  The cascaded profile meter is implemented as stated in 8.2.1.1.

9.3.2 Simulation results

Performance of connections 0-4 are fairly consistent in both controlled and compared cases. Each achieves close to 1Mbps target-rate with slight deviations due to network bias against long RTT connections. The more interesting results come from the bottom half of the table, where throughputs of connections 5-9 differ considerably when the profile meters are different. When the profile meter is for aggregated traffic (the controlled case), it allows the network bias against long RTT connections among the connections which share the profile to persist. In contrast, when the profile meters are for each individual connection, such a bias is ameliorated.

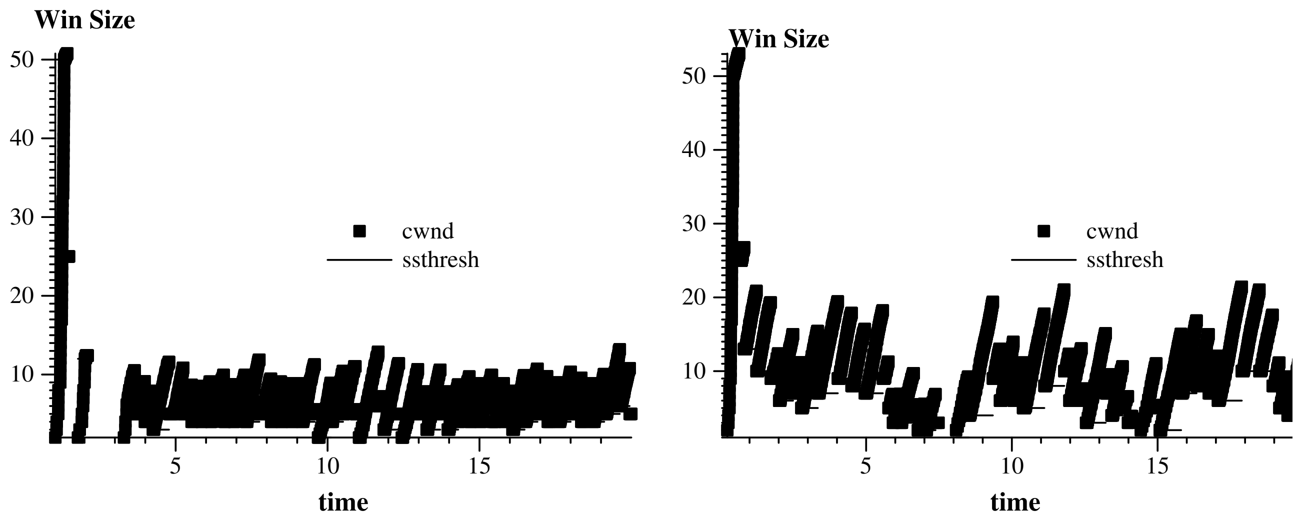
There are two things worth noting. First, a careful trace of the *In* and *Out* packets for both cases lead us to draw the same conclusion from section 8.2.1.2, that the additional cascaded profile does not have a significant impact on the traffic performance. Second, in the controlled case, when connections 5-9 are sharing one single profile meter together, the combined RIO-TSW rate adjustment mechanism is not particularly effective. TCP window graphs show less frequent, but more drastic window swings. In contrast, connections 0-4 have frequent drops at regular intervals, and regulate the TCP window swings very well. This can be seen in the TCP window oscilla-

TABLE 10. Cascaded profile meters with controlled and compared cases. Bottleneck link speed= 10Mpbs. Used TCP-SACK

Conn #	RTT	Target-Rate (Mbps)	Achieved Rate (Control) (Mbps)	Achieved Rate (Compare) (Mbps)
0	30	1	1.02679	1.09818
1	40	1	0.943344	1.02829
2	50	1	0.949632	0.979580
3	60	1	0.929313	1.00836
4	70	1	0.901795	0.950515
subtotal			4.750874	5.064925
5	30	1	1.46652	1.06329
6	40	1	1.1965	1.00333
7	50	1	0.936831	0.980113
8	60	1	0.831452	0.876147
9	70	1	0.752801	0.855813
subotal			5.184104	4.778693
Total			9.934978	9.843618

tion graphs for connections 1 and 6 in figure 26.

The reason is when a connection has an individual profile meter, the tagging mechanism is more sensitive to changes in the connection, for example, a TCP burst, since TCP tends to send a cluster of packets. In contrast, when a connection is aggregated before going through a profile meter, a burst from one connection has already mixed with bursts from other connections, and the aggregated traffic result in a much less bursty connection. Therefore, the profile meter for the aggregated traffic is not as sensitive to the rate change of any particular connection. Therefore, a connection with individual profile meter is likely to have more packets tagged as *Out* when it exceeds the target-rate, than when it goes through an aggregated profile meter. Consequently, this connection is likely to receive more drops than one that goes through an aggregated profile meter. So long as TCP operates in the Fast-Retransmit mode, frequent adjustments tend to regulate this connection's performance better. Therefore, we see the connections with individual profile meters have better and more consistent throughputs.



Congestion Window 1

Individual profile meter regulates individual connection with frequent, regular rate adjustments.

Congestion Window 6

Aggregated profile meter results in irregular, less frequent rate adjustments.

Figure 26: TCP window oscillations for connections using individual and aggregated profile meters.

CHAPTER 10

Conclusions

This report is an extended version of [ClaFan97]. We present the algorithms used in the “Expected Capacity” framework, and analyzed simulation in detail. We conclude that the “Expected Capacity” framework is capable in allocating “best-effort” bandwidth according to some pre-defined policy at the network boundary. Therefore, 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.

We summarize and highlight the major observations from all simulations in the following:

- The current Internet allocates bandwidth in an unpredictable manner. Connections with short RTTs, due to TCP congestion window open up algorithm, are more opportunistic and tend to get more bandwidth than connections with long RTTs. This type of bias is very visible.
- In the “Expected Capacity” framework, bandwidth can be allocated according to the contracted profiles, so long as the network is well-provisioned, in both sender-based and receiver-based schemes.
- Unless the profile meters take into account of RTTs of connections, the network bias against long RTT connections mentioned above can be mitigated, but not corrected. A possible solution to correct such bias is to install profile meters in the operating systems of the hosts. A simple module to use the TCP estimate of RTT will be able correct such bias. No modification to TCP is needed.

- The Explicit Congestion Notification (ECN) mechanism provides an elegant way to do receiver-controlled bandwidth allocation in the "Expected Capacity" framework. The receiver-controlled scheme works better than sender-based scheme with TCP-Reno. However, there are a few potentially dangerous attacks can be easily designed to take advantage of the receiver-controlled scheme. Further research is needed to solve these problems.
- Different versions of TCP can achieve slightly different performance under "Expected Capacity" framework. TCP-SACK, or any version that can cope with multiple packet drops within a window, works better than TCP-Reno in the framework.
- Cascaded profile meters have little impact on the end-to-end connection performance, so long as the network is sufficiently provisioned. Whether the tagging algorithm deployed in cascaded profile meters is a probabilistic one or a simple additive token-bucket does not make too much difference.
- An individual profile meter can regulate traffic of an host better than an aggregated profile meter. When traffic are mixed and go through aggregated profile meters, the aggregated profile meter loses its ability to closely regulate individual connections. The distribution of bandwidth among the connections which are sharing the same profile is primarily according to the RTTs of connections, as in the current Internet: the shorter the connection, the higher bandwidth it gets.

Acknowledgment:

The author would like to acknowledge Sun Microsystems Inc.'s generous financial support to make a visiting position to M.I.T. possible. The author also like to thank Dr. David Clark for guidance and insightful intutions throughout this work. Many ideas are not possible without the inspiring discussions with the inhabitants of NE43-537. The author especially like to thank Rob Cheng, Dina Katabi, Rena Yang for their support.

References:

[Bala90] 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

[ClaFan97] Clark. D., Fang, W., *Explicit Allocation of Best Effort Packet Delivery Service*, To appear in IEEE Communications for Networks. http://www.cs.princeton.edu/~wfang/Research/tsw-format.ps.Z

[DemKesShe89] Demers, A., Keshave, S., and Shenker, S. 1989, *Analysis and Simulations of a Fair Queueing Algorithm*. Proceedings of ACM SIGCOMM '89

[End97] end-to-end meeting in MIT-LCS. August, 1997.

[FalFlo96] 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

[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

[FloFal97] 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.

[Gupta97] 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

[Hoe96] 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

[ZhaSheCla91] 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.

[RSVP93] Zhang, L., Deering, S., Estrin, D., Shenker, S., and Zappala, D. *RSVP: A New Resource ReSerVation Protocol. IEEE Network*, September 1993.