Design Considerations for Rate Control of Aggregated TCP Connections

Ashish Mehra, Renu Tewari, and Dilip Kandlur

IBM Thomas J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598
{mehraa,tewarir,kandlur}@watson.ibm.com

Abstract

We study rate control of aggregated TCP connections, i.e., multiple TCP connections treated as a single *aggregate* for purposes of rate control, via traffic conditioning mechanisms such as traffic policing and shaping. This is a likely scenario given the current trends in policy-based service differentiation on the Internet (e.g., Differentiated Services) and content aggregation on the Web (e.g., virtual hosting). Traffic aggregation is increasingly necessary for cost-effective, scalable provisioning and management of network and server resources.

We propose and evaluate a set of mechanisms for fair sharing of an aggregate's allocated bandwidth between connections comprising the aggregate, for traffic conditioning via policing with marking and shaping. We propose logical token buckets (common token bucket with logical partitions) that account for the round-trip times of individual connections to provide fair bandwidth sharing while achieving high aggregate throughput and bandwidth utilization. We propose modifications to TCP's congestion window increase during congestion avoidance to achieve fairness between short and long-lived connections, and introduce the notion of aggregate fairness. We demonstrate that the proposed mechanisms provide a high degree of fairness and bandwidth utilization while limiting an aggregate's bandwidth usage to the desired rate. We also describe the key protocol stack extensions for our AIX-based prototype implementation to enable efficient rate control of TCP connection aggregates.

1 Introduction

The Web and Internet together constitute a critical information, entertainment, and commerce infrastructure that is rapidly evolving from a best-effort service model to one in which *service differentiation* can be provided for users, services, and applications. This service differentiation is in the form of preferential treatment (using priorities, resource reservation, etc.) of one type of user/application traffic over another at the servers, proxies, and network elements that comprise the end-to-end infrastructure. A number of IETF working groups, e.g., Differentiated and Integrated Services, are in the process of developing standards for new technologies that realize and/or support service differentiation on the Internet and Web. Many ISPs, network carriers, and Web sites are enabling some form of service differentiation and this emphasis is likely to become even stronger as the Internet continues its exponential growth.

One of the primary motivations for service differentiation on the Internet is the control it provides over the management of server and network resources. This control allows service providers and carriers to offer a range of customer experiences via new business models and business-to-customer and business-to-business relationships. These relationships are based on resource provisioning for different types of traffic in order to realize a wide variety of service guarantees (such as loss and/or delay bounds, network bandwidth allocation, etc.). The complexity and cost of providing service differentiation is determined to a large extent by the traffic class granularity at which service differentiation is applied. For example, while each end-to-end application/network flow (e.g., all traffic on a given TCP connection) is a good candidate for service differentiation, per-flow resource provisioning and traffic conditioning results in a substantial increase in the complexity and overhead. Besides limiting scalability, it may also negatively impact the service provided to traditional best-effort traffic due to the high overhead and/or under-utilized resources.

Traffic aggregation, i.e., aggregation of individual network flows, is increasingly becoming necessary for cost-effective and scalable management of network resources such as bandwidth and buffers. This emphasis is reflected in a number of standards being defined by the IETF, e.g., Differentiated Services (DS) [1, 2], in which network devices at the edge of

the network aggregate traffic flows onto provisioned "pipes" that traverse a simple and streamlined network core. The edge devices, which include servers and proxies, are mainly responsible for complex quality-of-service (QoS) functions, while the network core only needs to manage a small number of provisioned pipes using simple per-hop QoS and forwarding mechanisms (per-hop behaviors [1]). In this model, network-wide policies and/or service level specifications (SLS) ensure that several per-hop behaviors can be meaningfully combined to realize end-to-end service guarantees.

Technology trends in the Web server/proxy design space are also moving towards traffic aggregation to reduce site management costs and improve resource utilization. A clear example is co-location of distinct Web sites onto a single powerful server platform, e.g., virtual hosting in the popular Apache web server [3]. Virtual hosting allows multiple web sites to share server resources transparent to the clients, and is actively employed for many Web sites today. Similarly, there is now a trend towards large-scale outsourcing of general Internet applications to "Enterprise Service Providers", motivated once again by substantial cost savings. In these environments, service differentiation becomes essential to partition server resources (e.g., available network bandwidth) between the aggregated traffic belonging to different web sites or applications.

Note that hosts (e.g., Web servers and proxies) are edge devices that already provide some key components required for service differentiation: (i) they maintain per-flow state (e.g., sockets and protocol control blocks), (ii) they are closer to the applications, facilitating efficient control over inbound and outbound traffic, and (iii) they can classify and aggregate traffic based on application or user-level information that is not available in the network. Widespread deployment of the Web (and hence HTTP) has made TCP the dominant transport protocol for the Internet. Thus, we only consider TCP traffic, given that it also provides a special challenge for service differentiation.

In this paper, we focus on design considerations for service differentiation of aggregated TCP connections at Internet servers and proxies. In particular, we study rate control of aggregated TCP connections via traffic conditioning mechanisms such as traffic policing and shaping, a likely scenario given the current trends in policy-based service differentiation on the Web and Internet. Realizing aggregated rate control on heavily-loaded servers and proxies requires: (i) scalable policies and mechanisms for traffic control, (ii) efficient rate control for the aggregate, and (iii) fair allocation of available bandwidth to each connection.

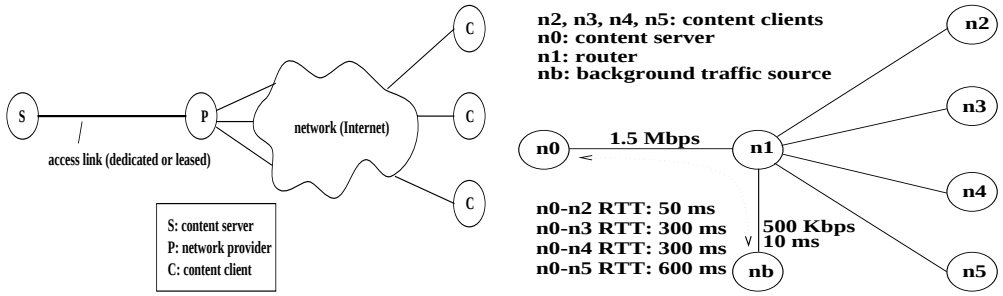
The main contributions of this paper are in proposing and evaluating different approaches for rate control of aggregated TCP connections. We compare per connection rate control and aggregated rate control using a common token bucket to contrast the tradeoffs between fairness and achievable throughput. To balance these tradeoffs, we propose an approach that uses a common token bucket with *logical partitions*. The logical partitioning enables a high degree of fairness, while the use of a common bucket allows dynamic sharing of unused rate, thereby, achieving a high aggregate throughput. Given round-trip time estimates, the fairness mechanisms proposed apply to any bottleneck link traversed by a set of aggregated TCP connections. We also propose modifications to the TCP congestion window increase algorithm during congestion avoidance to achieve fairness amongst a mixture of connections with long and short lifetimes. We define the concept of aggregate fairness to compute the exact modification to the congestion window. Finally, we present the protocol stack extensions in our prototype implementation on AIX 4.2.1, and discuss the implementation overheads for timer-based traffic shaping.

The rest of this paper is organized as follows. The following section outlines the problem specification and presents application scenarios for the problem of aggregated rate control of TCP traffic. Section 3 provides an overview of possible approaches for rate control of individual TCP connections. Section 4 considers rate control of aggregated TCP connections, proposing and evaluating solutions to enable fairness between connections while performing efficient rate control. Possible modifications to TCP congestion window algorithms, and their effectiveness in improving fairness, are explored in Section 5. Key protocol stack extensions developed for our prototype implementation are presented in Section 6. Section 7 discusses related work while Section 8 concludes the paper.

2 Problem Specification

Consider the scenario depicted in Figure 1(a). A collection of servers (a server farm) for content hosting are connected to the Internet through a network provider. The connectivity between the servers and the network provider is in the form of a dedicated leased line (e.g., T1 or T3), or a virtual leased line with the server site contracting with the network provider for a certain amount of link bandwidth. We refer to the available bandwidth between the server farm and the network provider as the access bandwidth. Clients with varying levels of connectivity (in terms of bandwidth and delay) connect to one or more servers to download content across the Internet. The content serviced by the servers is transported across the access link either via an access device concentrator or an intervening proxy. Referring to Figure 1, the content server (S) generates traffic in response to requests from one or more clients (C).

We assume that each server in the server farm is assigned a certain fraction of the available access bandwidth (e.g., a virtual leased line), and that the server has sufficient processing power to fully utilize the available access bandwidth. This is highly likely with modern high-performance servers even for access speeds of 100 Mb/s or more [4], especially when each



(a) Problem scenario (b) Simulation topology

Figure 1: (a): Problem scenario depicting a content server S that connects to the network (and clients C) via a network access provider P across a dedicated or leased access link. (b): simulation topology.

server is allocated only a fraction of the access link bandwidth (e.g., T3 virtual circuits). For example, the network provider may establish an SLS with the content server, restricting the server to a specified maximum bandwidth usage on the access link, and/or charge the server for any excess traffic. This allows the network provider to limit the resources consumed by the server's traffic before it is injected into the external network. To comply with the SLS, the server must perform appropriate *traffic conditioning* (i.e., marking, policing, dropping, shaping) on the *total* transmitted traffic.

Traffic conditioning specification is a fundamental component of an SLS or policy, and specifies (i) a traffic profile and (ii) actions such as policing, marking, dropping, or shaping. The traffic profile describes the temporal properties of a connection's traffic using a leaky bucket based specification (e.g., peak rate, average rate, and burst size). The traffic profile is used to determine whether a particular packet is in-profile (compliant) or out-of-profile (non compliant). We consider two common traffic conditioning actions: policing with marking and shaping; our results also apply to traffic conditioning functions such as policing with dropping. With policing and marking, in-profile packets are sent marked and out-of-profile are sent on a best-effort basis. With shaping, out-of-profile packets are delayed (i.e., buffered) until they become compliant with the traffic profile.

2.1 TCP Connection Aggregates and Rate Control

Since the access link provides the primary connectivity to the external network (e.g., the Internet), the access bandwidth (but not necessarily the access link) available to a content server is often a precious (and bottleneck) resource that must be allocated and managed properly. To manage the access bandwidth and distinguish between individual connections, the server may partition the access bandwidth via one or more policies (or SLSes) configured by a system administrator. These policies define one or more service classes (e.g., a DS codepoint [1], peak or average transmission rates) and specify appropriate traffic conditioning functions for each class. Such policies allow the server to differentiate between connections based on a number of criterion, e.g., the server content accessed, the server applications or services invoked, and their connectivity to the network. The server assigns traffic to a particular service class and performs the specified traffic conditioning function. The rate limits for a given policy may be specified by some global network-wide policy database, or derived independently from the SLS between the server and the network provider.

Table 1 lists some example policies specifying traffic classes and traffic conditioning functions. As illustrated in Table 1, each policy controls traffic on multiple traffic flows (such as TCP connections or UDP sessions) with each service class. We refer to the set of flows controlled by a policy as a flow *aggregate*. In the rest of the paper we focus on policy-based rate control of TCP connection aggregates. One of our goals is to explore the design tradeoffs in rate control of TCP connection aggregates, and its pros and cons relative to rate control of individual TCP connections.

2.2 Scalable and Fair Sharing of Aggregate Bandwidth

With each policy exercising rate control on a large number of connections simultaneously, the aggregation mechanisms must scale with the number of connections. In practice this requires reduction or amortization of traffic conditioning overheads. Since short-lived TCP connections are common, the aggregation mechanisms must also efficiently support a dynamically changing set of connections controlled by a single policy.

The aggregation mechanisms must ensure fair sharing of the assigned bandwidth (referred to as available access bandwidth for the rest of the paper) between the individual connections comprising an aggregate. In the absence of fairness an aggressive TCP client can consume an unfairly large proportion

Filter	Traffic Profile	Traffic Conditioning
<128.34.16.4, *, *, *>	(100 Kbps, 100 Kbps, 10 KB)	police and mark
<128.34.16.8, 80, *, *>	(1 Mbps, 2 Mbps, 20 KB)	shape
<128.34.16.4, *, 141.213.8.108, *>	(100 Kbps, 300 Kbps, 30 KB)	shape

Table 1: Example policies requiring rate control of traffic aggregates.

of the assigned bandwidth at the expense of well-behaved TCP clients. Even though TCP congestion control mechanisms are designed for global fairness, TCP connections can be aggressive for a variety of reasons, the primary being smaller round-trip times [5] and/or a high rate of connection requests per second originating from a given client.

Definition of Fair Share: For an aggregate comprising n TCP connections, we consider fair share to be an equal allocation of (leaky bucket) tokens among individual connections. More precisely, the sharing of tokens is $max\text{-}min$ or bottleneck fair, i.e., if any connection uses an amount less than its fair (i.e., equal) share, the unused amount is shared equally among the remaining connections. Thus, each connection is granted a share of $s_i = min(S, \hat{s_i})$, where S is the fair (i.e., equal) share and $\hat{s_i}$ is the share requested.

To measure the fairness in the throughput achieved by each TCP connection, a commonly used metric is the 'fairness" index, which is given by $\frac{(\Sigma R_i)^2}{n\Sigma R_i^2}$, where R_i is the observed throughput of the i^{th} TCP connection on the shared link [6]. This metric measures the deviation from an equal share; a maximum fairness index of 1 indicates an equal share. Since a TCP connection may not use its equal share due to some other bottleneck link in its path, we modify the fairness index to handle max-min fairness. If the max-min fair share is s_i, the fairness index is given by, $\frac{(\Sigma R_i/s_i)^2}{n\Sigma (R_i/s_i)^2}$.

By virtue of fair bandwidth allocation at the source, at each bottle link traversed by an aggregate's connections, bandwidth allocation to individual connections is fair, irrespective of the round-trip times of the individual connections. The aggregation mechanisms must also ensure high access bandwidth utilization while maintaining a high degree of fairness, especially for a mix of long-lived and short-lived connections.

We explore several approaches to provide fair bandwidth sharing for TCP connection aggregates while applying rate control on the total access bandwidth consumed by the aggregate. These approaches, which exploit knowledge of round-trip times of all the connections constituting the aggregate, are readily realizable at the server since it is the endpoint for each connection constituting an aggregate.

3 TCP and Rate Control

In this section, we first provide an operational overview of data transfer on individual TCP connections and the factors that affect fairness between connections. We then describe how traffic conditioning functions such as policing and shaping can be used to control the rate of individual TCP connections.

3.1 Regular TCP

TCP traffic is governed by window-based flow control clocked primarily by the receipt of acknowledgments (ACKs) from the receiver. In regular TCP, the sender sends a minimum of the congestion window (`cwnd`) and the receiver's advertised window. During the slow-start phase the `cwnd` grows exponentially from 1 segment, doubling every round-trip time, until a threshold, `ssthresh`, is reached. In the congestion avoidance phase the window grows linearly, increasing by 1 segment after every round-trip time. TCP reacts to congestion by dynamically adjusting the window size. When congestion is detected by inferring a packet loss (e.g., on receiving multiple duplicate ACKs), the congestion window is halved, in what is called the multiplicative decrease; a loss inferred due to a retransmission timeout results in slow-start.

During periods of non-congestion the window size increases linearly. However, this rate increase is not uniform for TCP connections with different round-trip times (`rtt`). As previously shown [6, 5], when two TCP connections share a congested link, the short `rtt` connection ramps up much faster than the long `rtt` connection. The bias against long `rtt` connections is of the order of rtt^{α} where $\alpha < 2$ [5]. Various schemes like the Constant Rate scheme [6] or the Increase-by-K scheme [7] attempt to reduce this inherent unfairness among TCP connections. We revisit these schemes in Sections 4 and 5 in the context of rate control of TCP connection aggregates.

The rate of a TCP connection can be controlled either by policing with marking or by shaping. We describe each of these below in the context of aggregate rate control.

3.2 Policing and Marking

Policing of TCP connections is typically accomplished via a leaky-bucket based token allocation scheme. Each connec-

tion's traffic specification is a 4-tuple $< b_i, r_i, r_p, L_m >$. The depth of the token bucket b_i is the burst size, i.e., the maximum number of back-to-back packets that can be transmitted at the peak rate. The average rate r_i, is the rate at which tokens fill the token bucket. The traffic specification also defines a peak rate r_p, such that the minimum duration between successive packet transmissions is $1/r_p$. L_m is the maximum size of a packet. When a connection needs to send a packet, the token bucket is checked for available tokens; if a token exists the packet is sent as marked (compliant or in-profile), otherwise it is sent as unmarked. The marking is done using IP TOS bits or a suitable differentiated services (DS) codepoint [1].

We assume that at a backbone router, in the path between the TCP source and sink, marked packets are given higher priority or have a lower loss probability compared to unmarked packets. The enhanced random early detection (ERED) [8] is one such scheme that assumes a single first-in-first-out queue at the router with different discard probabilities for marked and unmarked packets. As in the original RED scheme [9], packets are dropped randomly when the queue length exceeds a given threshold. Other schemes like FRED, buffer-based provisioning [10], etc., have been proposed for fairness. However, such schemes are more useful at access routers that perform flow classification and policing, and not at backbone routers which only consider packet marking.

Figure 2 shows congestion window growth as a function of time for a typical TCP connection. While the number of marked packets sent is limited by the target rate, the unmarked packets are controlled by TCP's window. However, loss of an unmarked packet halves the congestion window, thus affecting the number of marked packets transmitted. For a given rate r_i and round trip time rtt_i, the compliant window size for marked packets (also called rate window) is limited to $r_i * rtt_i$.

3.3 Shaping

With shaping, transmission of a non-compliant packet is delayed until it becomes compliant with the traffic specification. TCP traffic can be shaped by three different approaches [11]: (i) *window-based*, which limits the growth of TCP's congestion window, (ii) *timer-based*, which uses a timer-based trigger to send the delayed non-complaint packets, and (iii) *ack-based pacing*, which paces acknowledgments to indirectly limit the growth of the congestion window to the desire value. We briefly compare and contrast each of these approaches.

Window-based: This approach limits the increase in the TCP congestion window (`cwnd`) to an upper bound computed using the desired average rate. For a TCP connection with round-trip time estimate rtt_i and desired average rate r_i, the congestion window size is limited by the rate window to $r_i * rtt_i$. This limits the average rate to r_i, provided the round-trip time estimates are accurate. A key limitation of this approach is that it does not provide any control over the peak transmission rate and burstiness, and does not work for non-TCP traffic (e.g., UDP sessions). While it works well for controlling the average rate of individual TCP connections, it cannot ensure fairness when applied to aggregate rate control. We do not consider window-based rate control in the remainder of the paper.

Timer-based: In this approach non-compliant packets are delayed until they are compliant as per the traffic profile. A system timer initiates packet transmission once a packet is compliant, and controls both the average and peak rates for TCP as well as UDP traffic. For this reason we consider timer-based shaping in the rest of this paper. Fine-grained timers, however, incur an excessive overhead of timer interrupts and related processing. As we discuss later in Section 6, aggregate rate shaping can be realized using coarse grain shaping timers by exploiting triggers other than timer interrupts.

External ACK pacing: In this approach, ACKs arriving at a source are paced to regulate the sending behavior of the source. ACK regulation is done external to the source, e.g., at a front-end switch [12]. We do not consider ack-based pacing given our focus on source-based rate control mechanisms.

Rate based pacing of individual TCP connections has been studied in other contexts (e.g., TCP over ATM networks [12]). However, traffic conditioning specifications are defined for behavior aggregates and not individual connections. In directly applying traffic conditioning schemes designed for individual connections, various issues arise, namely, (i) fairness, (ii) bandwidth utilization, and (iii) scalability due to implementation overheads. We now consider approaches for aggregate traffic conditioning of TCP connections that address these issues.

4 Rate Control of TCP Connection Aggregates

As discussed earlier, current trends indicate that in environments such as virtual hosting, traffic conditioning specifications will be defined over aggregates of individual connections or micro-flows. Such aggregates comprise a group of connections that match a given filter, e.g., a particular source or destination address, or a source or destination port. For example, in a web hosting environment, a filter of the form $<$ `a.b.c.d, 80, ANY_ADDR, ANY_PORT` $>$ would aggregate all connections originating at port 80 of the source $a.b.c.d$. We assume that individual connections are classified into aggregates and traffic conditioning done at the traffic source or the access router. This is in accordance with the differentiated-services proposal which moves complex rate control mechanisms to the network edges. This is especially useful for TCP connections as end-to-end information about round-trip delays and loss rate is available more accurately at the source.

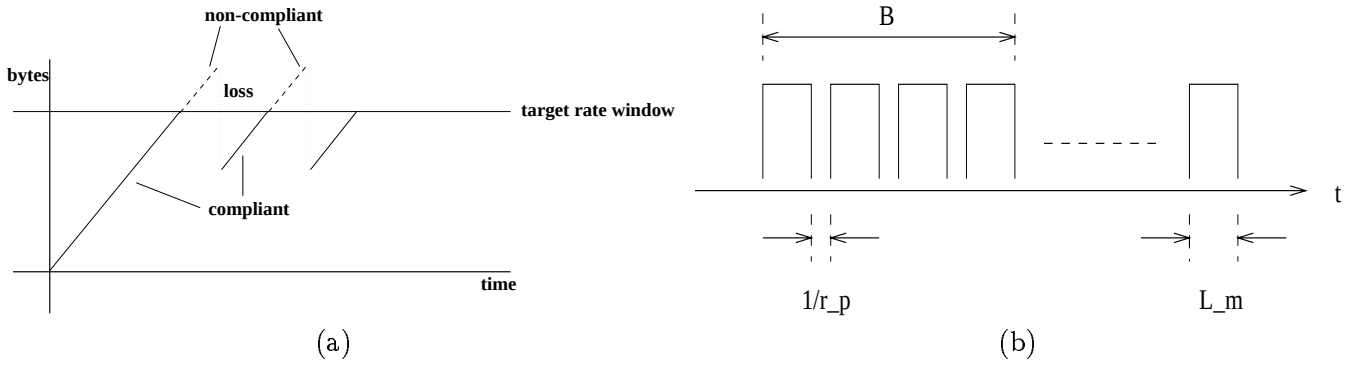


Figure 2: (a) Behavior of TCP with rate-based policing and marking. (b) Leaky bucket parameters

Similar to the per connection traffic specification, a leaky bucket-based specification consisting of the tuple $< B, r_m, r_p, L_m >$ is provided for the aggregate. The token filling rate, r_m, represents the average throughput of the aggregate in bytes/second. The token bucket depth, B, limits the maximum burst size of the aggregate. The peak rate, r_p, limits the minimum inter-packet spacing to be $1/r_p$ secs, and L_m is the maximum size of a packet that can be transmitted. Given the aggregate's traffic specification, the question we address is: how should the individual TCP connections be rate controlled to comply with the specification? One straight-forward approach is to partition the token bucket equally amongst the connections and perform independent per connection rate control. A second approach is to share a common token bucket among individual connections. We also propose a third approach, which is to have a common token bucket but assign a logical partition for each connection.

To compare these approaches we consider:

(i) how are the tokens of the aggregate shared among individual TCP connections such that each connection gets a "fair" share of the aggregate rate?

(ii) how is the link utilization affected by interactions with TCP's window based flow control?

(iii) what is the implementation overhead of rate control?

Before discussing the design and evaluation of the three approaches, we describe the simulator used for experiments and define how we measure fairness.

4.1 Evaluation Methodology

We modified ns version 2.1b4 [13] to support rate control for TCP connection aggregates. A new `qosmgr` agent maintains traffic aggregates and performs traffic conditionining functions. Each aggregate is defined by a policy comprising a filter, the traffic conditioning specification, and a list of TCP connections and their states. The filter is defined by the tuple < `src_addr, dst_addr, src_port, dst_port` > . A value of `ANY_ADDR` for the address fields, or `ANY_PORT` for the port fields, matches all values. The traffic envelope specifications consist of the average transfer rate, peak rate, burst size, and the maximum packet size for the aggregate.

The `qosmgr` agent is configured to perform a combination of policing with marking, and shaping on the connections within an aggregate. With only policing and marking enabled, compliant packets are marked by setting the priority field in the packet header, while non-compliant and best-effort packets are sent unmarked. The marked packets can be configured to suffer lower loss rates compared to unmarked packets. We add a new queue discipline of ERED [8] (an extension of RED) that assigns a lower drop probability for a marked packet when queue length is greater than the threshold. By default a marked packet is has a 1.5 times lower drop probability.

When shaping is enabled, we use a timer-based mechanism which associates a shaping timer with each aggregate. Packets are transmitted only if the aggregate is compliant, otherwise transmission is delayed until the next time the shaping timer expires. The check for compliance is made when a received ACK triggers packet transmission or when the application sends data, i.e, whenever TCP's output routine is invoked. We assume a default shaping timeout of 15 ms. The *fairness policy* controls which connection in an aggregate is allowed to transmit next and the number of packets it can transmit. We extended NewRenoTCP to invoke the `qosmgr`'s traffic conditioning functions before packet transmission.

We use the simple network topology shown in Figure 1(b) for all the experiments. For the equal bandwidth case the link bandwidth between the client and network provider's router is assumed to 500kbps. For the unequal bandwidth case, the bandwidths are set to 1000 kbps, 500 kpbs, and 100 kbps. The `rtt` values are set to 50 ms, 300 ms, and 600 ms. The topology is chosen such that the access link between the server and the network provider (represented by the sole router) is the bottle-

neck link. Client connectivity to the network is represented by direct links to the network provider with appropriate bandwidth and delay properties. We use a synthetic workload to emulate short and long HTTP and FTP-style transfers from the server to the clients. The evaluation focuses on the effects of the proposed fairness policies and TCP modifications. The primary metrics used are the aggregate and per-connection end-to-end throughput and the fairness in bandwidth allocation.

4.2 Mechanisms for Rate Control

For rate control of the aggregate we consider policing with marking and rate shaping as the two traffic conditioning mechanisms. We assume that the intermediate routers can distinguish between marked (i.e., compliant) and unmarked packets and provide better service (lower loss or delay) to the marked packets. As discussed in Section 3, we use timer-based mechanisms for shaping; Section 6 discusses timer overheads and design implications. For both policing and shaping we evaluate the tradeoffs among the three different approaches to sharing the aggregate token bucket among individual connections.

As a reference case, we first present the throughput of individual TCP connections when no rate control is performed. The throughput is presented as a ratio of the total number of bytes sent so far since the start of transmission. While this does not capture instantaneous fluctuations in the observed throughput, it is useful when comparing the average throughput of long-running connections. Figure 3(a,b) shows the observed throughput for each connection for equal and varying link bandwidths. As discussed in Section 3, TCP is inherently unfair to connections with large `rtt` values. With traffic conditioning rules, we evaluate in Section 5, if and when modifications to TCP's congestion window are required to achieve fairness.

4.2.1 Partitioned Token Bucket

In this scheme the aggregate token bucket is partitioned among the n individual TCP connections. Since no sharing is possible, each connection's token filling rate r_i is an equal share of the aggregate rate r_m, that is $r_i = r_m/n$. We first consider an equal division of the aggregate token bucket depth among each connection, i.e., $b_i = B/n$. The partitioning is equivalent to independent rate control of each TCP connection. The throughput with rate shaping of individual connections, each with a different `rtt` value, for the topology with equal bandwidth links is shown in Figure 4(a). For this topology, the figure shows that each connection is shaped to the assigned rate, the fairness index in steady state being 0.99 and the achieved aggregate throughput being 99.5% of the aggregate rate. Note that TCP's rate of congestion window increase is clocked by the `rtt` duration. With shaping, the congestion window size is limited to the size of the rate window $r_i * rtt_i$. For connections to not lose tokens while waiting for acknowledgments, the token bucket depth should be at least equal to the rate window, i.e., $b_i \geq rtt_i * r_i$. It follows that the aggregate token bucket depth $B \geq \Sigma rtt_i * r_i$, or $B \geq rtt_{av} * r_m$, where rtt_{av} is the average roundtrip time for connections within the aggregate. To avoid losing tokens, we configure each connection with a bucket depth proportionate to it's `rtt` value; the fairness mechanisms work for any configured bucket depth. Figure 4(b) shows the throughput for individual rate-controlled connections with proportionate share of the token bucket; the fairness index in this case is 0.996.

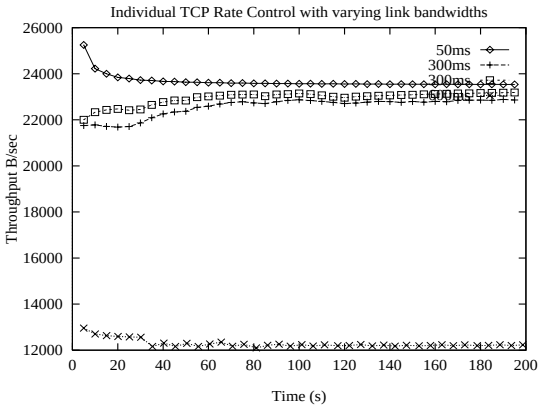
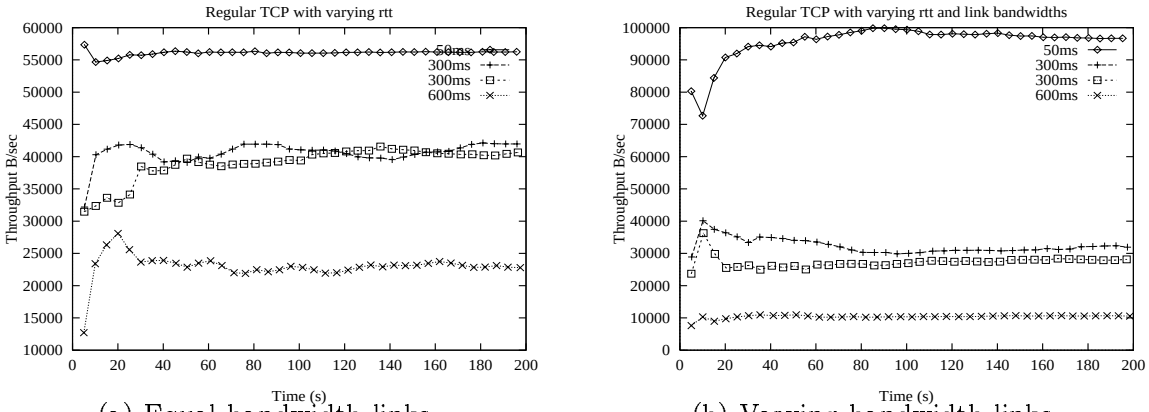


Figure 5: Individual rate control with shaping: Same as Figure 4 except with unequal link bandwidths.

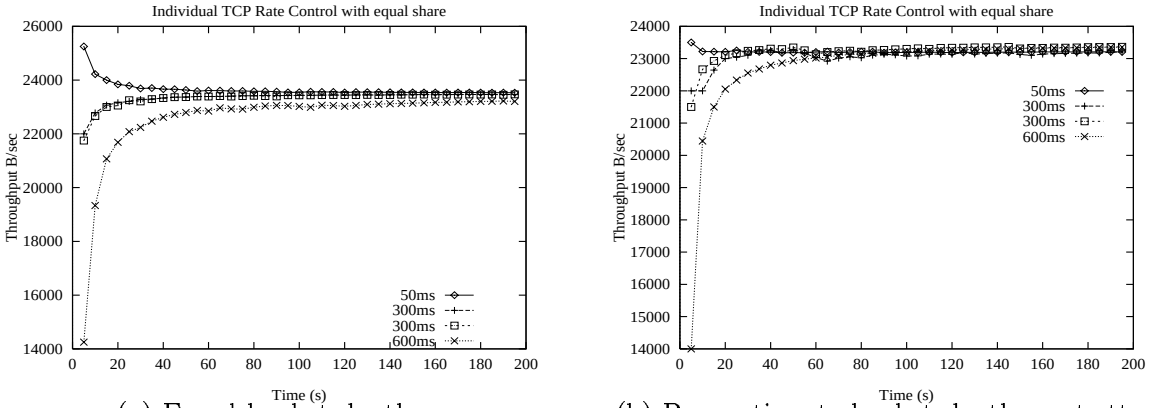
For the topology with equal bandwidths and proportionate share, individual rate shaping with a partitioned token bucket achieves a fair share among individual connections. However, there are two drawbacks with individual rate shaping. First, fine-grain per-connection shaping timers result in high system overheads; a coarse-grained timer per aggregate or per server will lower overheads significantly. As discussed later, coarse-grained timers are feasible only if additional triggers are used. Such triggers are available only when multiple connections are shaped as an aggregate. Second, the aggregate throughput is low when different connections span different link bandwidths. When a connection cannot utilize it's assigned equal share, the extra rate is wasted, decreasing the aggregate throughput. Figure 5 shows that with different bandwidth links the achieved aggregate throughput is reduced to 87% of the aggregate rate. With aggregate token sharing, this unused throughput can be dynamically shared across the remaining connections. Note that with independent rate control it is not possible to determine *a priori* the best rate to assign to a connection.

Figures 6(a,b) compares the compliant and non-compliant throughput with policing and marking of individual connections. The total throughput (compliant and non-compliant) is much higher than the aggregate rate, but the aggregate compliant throughput is lower since the loss of unmarked packets reduces the congestion window below the rate window.



(a) Equal bandwidth links (b) Varying bandwidth links

Figure 3: Regular TCP without rate control: Throughput of TCP connections with different RTTs sharing the same bottleneck link; link bandwidths based on topology shown in Figure 1(b).



(a) Equal bucket depth (b) Proportionate bucket depth w.r.t rtt

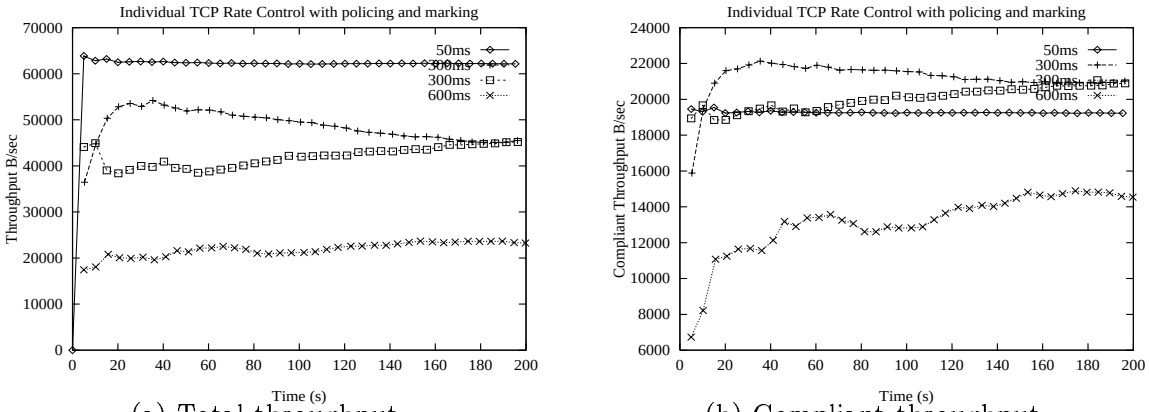
Figure 4: Individual rate control with shaping: Throughput of TCP connections with different RTTs sharing the same bottleneck link with each connection shaped separately; the link bandwidths are equal.

4.2.2 Common Token Bucket

Another approach for aggregated rate control is to use a common token bucket. By sharing the token bucket among connections and using a single timer for shaping, we can alleviate the drawbacks of individual rate control. We first consider a naive approach to rate shaping with a common bucket. In this case, a list of connections is maintained per aggregate; when the shaping timer expires, a sequential scan over the list is done for initiating a data transfer. For TCP connections with different `rtt` values, an aggressive connection, which has a small `rtt` and always has data to send, will consume most of the tokens and starve the less aggressive connections. The fairness index for such a sequential ordering is 0.43 for the throughput values shown in Figure 7(a). A fairer approach is to use a first-come-first-served (FCFS) scheme for assigning tokens. In FCFS, the connections are ordered based on the time when they first became non-compliant. When the shaping timer expires, the connection that became non-compliant the earliest is allowed to transmit first. The fairness index with the FCFS scheme is much higher (0.95), for the throughput values shown in Figure 7(b). Instead of FCFS, another approach is to use a least-recently-used LRU ordering for assigning tokens. On a shaping timer trigger, the connection that was least recently assigned a token goes first. The fairness index with the LRU scheme scheme is similar to that with FCFS (0.95), for the throughput values in Figure 7(c). However, the LRU scheme penalizes short aggressive connections more than FCFS.

Comparing with rate control of individual connections, aggregate shaping with a common timer seems to have poorer fairness. However, when link bandwidths are non-uniform the aggregate throughput is higher than that of the partitioned schemes. This is because sharing the token bucket inherently leads to sharing unused capacity as shown in Figure 7(d). The throughput achieved is 99.5% of the aggregate rate.

For policing and marking of connection aggregates, we compare the compliant and non-compliant throughput in Fig-



(a) Total throughput (b) Compliant throughput

Figure 6: Individual rate control with policing and marking: Throughput of TCP connections with different RTTs sharing the same bottleneck link, with each TCP connection individually policed and marked at the source. Routers are configured to support ERED; link bandwidths are equal.

ures 8(a,b). With policing, the total (compliant and non-compliant) and compliant throughput is higher than the partitioned scheme with individual rate control.

4.2.3 Common Token Bucket with Logical Partitions

Our goal for aggregate rate control is to achieve the fairness level of a partitioned scheme, while sharing the unused capacity to achieve high aggregate throughput, and using a single common shaping timer to reduce overheads. In order to do this we use a common token bucket but define a logical partition per connection. With this approach each logical token bucket has an equal input rate r_i, where $r_i = r_m/n$ and a weighted share of the aggregate bucket size B, i.e., $b_i = w_i B$. As discussed in Section 4.2.1, the value of b_i should ensure that tokens are not lost due to TCP's ack-based flow control. For a value of $w_i = rtt_i/\Sigma rtt_i$, we can guarantee that bucket depth is at least equal to the rate window, i.e., $b_i \geq rtt_i * r_i$, given that the same requirement holds for the aggregate bucket depth, i.e., $B \geq rtt_{av} * r_m$. Thus each connection has its own logical token bucket given by $< r_i, b_i >$. However, if a connection cannot use its tokens at the rate r_i, due to a limited link capacity on some link in the path, the unused tokens are shared equally among the other active connections.

Consider Figure 9 that illustrates logical token buckets and sharing of unused tokens. When a connection's logical token bucket is full, the extra tokens are shared equally among the remaining non-full token buckets. The unused token sharing scheme satisfies the constraint that at any time, the sum of the tokens available to all connections is not more than the aggregate bucket depth. Given this constraint we can derive the number of unused tokens available for use at a given time

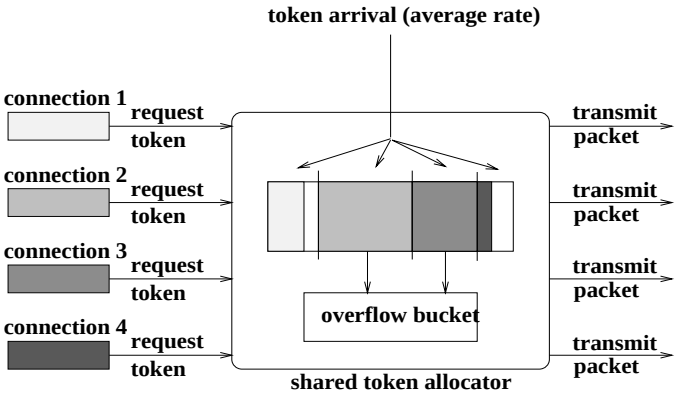
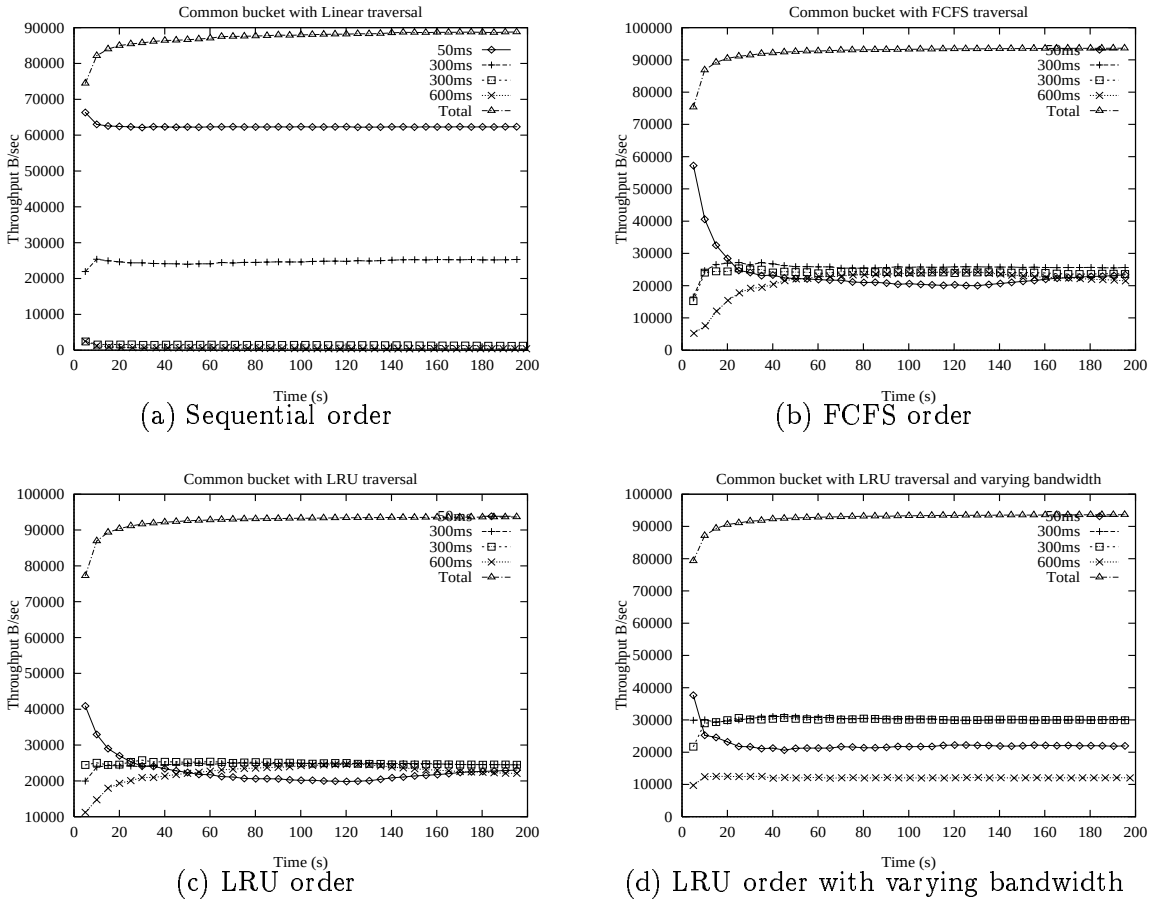


Figure 9: Token allocation to individual connections using logical buckets within a shared token allocator.

T. If each bucket consists of $\hat{b_i}$ tokens at time T, the maximum number of extra tokens that can be consumed are given by $B - \Sigma\hat{b_i}$. The number of unused token available are $\Sigma\ max(0, r_i * (T - t_i) - b_i)$, where t_i is the last time the logical token bucket i, was empty. Thus the total excess tokens available are

$$min\{B - \Sigma\hat{b_i}, \Sigma max(0, r_i * (T - t_i) - b_i).\}$$

For the throughput values shown in Figure 10(a), the fairness index of 0.996 for a common token bucket and logical partitions equals that of individual connection rate control. Also, with varying link bandwidths the unused capacity is shared among the remaining connections to fully utilize the assigned aggregate rate (as shown in Figure 10(b)). We argue that shaping using a common token bucket with logical partitions is able to balance the tradeoffs of fairness and utilization, assuming that a single shaping timer suffices for a large group of connections. To ensure that a single timer scales to a large number



(a) Sequential order (b) FCFS order

(c) LRU order (d) LRU order with varying bandwidth

Figure 7: Shaping with common bucket: Throughput of TCP connections with different RTTs, with aggregate shaping at the source; link bandwidths are equal in (a,b,c) and different in (d).

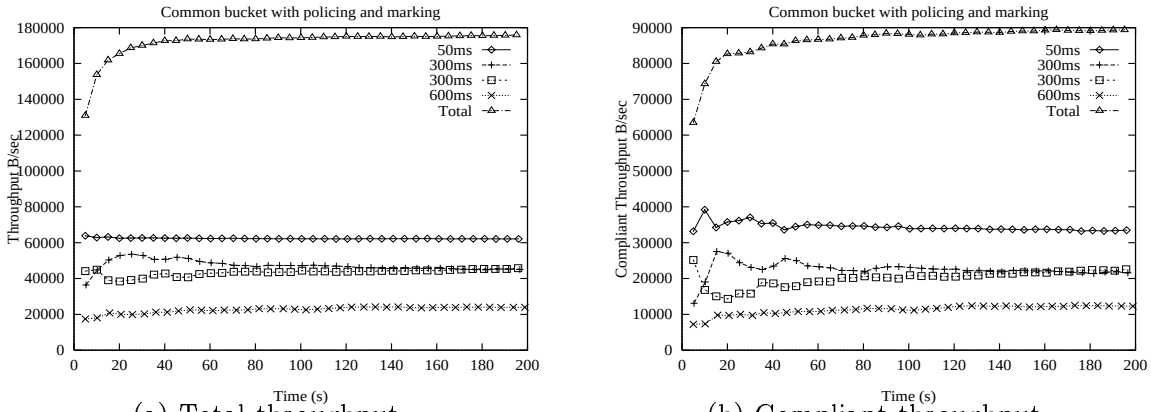
of connections we need additional mechanisms to trigger compliance checks and transmissions. Section 6 discusses triggers for scalability with coarse grained timers.

For policing and marking of connection aggregates, we compare the compliant and non-compliant throughput in Figures 11(a,b). With policing, the total (compliant and non-compliant) and compliant throughput is higher than that with individual rate control. The unused share of a connection is used by other connections to send more marked packets, thereby lowering the loss rate.

5 Adapting TCP Congestion Window

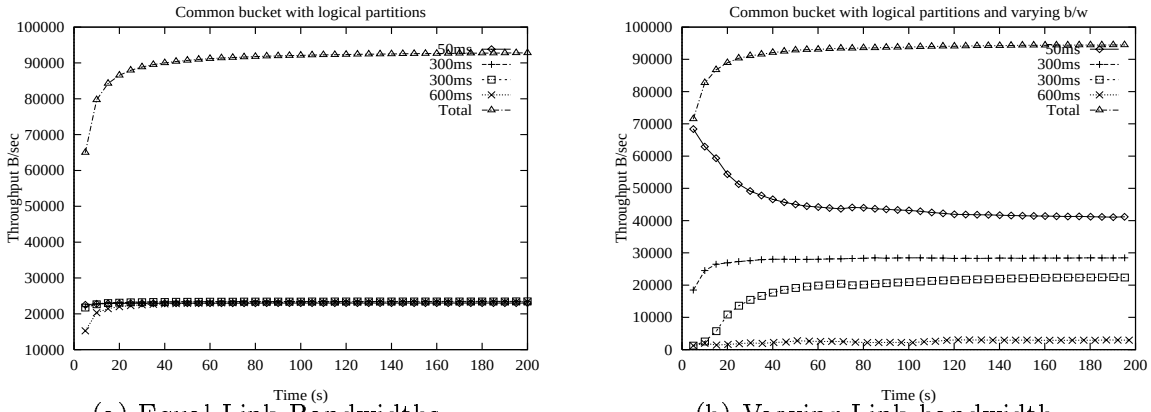
Previously we demonstrated that a common token bucket with logical partitions results in fair bandwidth sharing while offering high bandwidth utilization on the (bottleneck) access link. Our results apply primarily to a mix of long-lived connections that can ramp up to the target rate window. However, short-lived connections may terminate before reaching the target rate window, thereby not benefitting from the fairness policies described earlier. We now consider approaches for fair bandwidth sharing for short-lived connections in the congestion avoidance phase (i.e., ramping up after experiencing packet loss).

To be fair when ramping up, TCP's congestion window increase must be proportional to the round-trip time of the connection. We now derive an expression for the throughput of connection i, R_i, in terms of its round-trip time, rtt_i; the derivation is similar to the derivation in [6] for Constant Rate window increase. Assume that the available link capacity on the bottleneck link where fair share is desired (for us this is the link between the link and the network provider) is M. On this link assume that the packet loss rate for connection i is p. That is, the average time between packet drops on connection i is $1/p$. With TCP fast recovery, the congestion window halves on a packet loss and linearly increases till the next packet loss. Between successive packet losses the window increases from `cwnd/2` to `cwnd` during a time duration $1/p$. The total bytes sent during this time interval is the sum of the window sizes,



(a) Total throughput (b) Compliant throughput

Figure 8: Common token bucket with policing and marking: Throughput of TCP connections with different RTTs sharing the same bottleneck link, with aggregate policing and marking at the source. Routers configured to support ERED; link bandwidths are equal.



(a) Equal Link Bandwidths (b) Varying Link bandwidth

Figure 10: Shaping with common bucket and logical partitions: Throughput of different TCP connections with different RTT values with aggregate shaping at the source; the bandwidth of individual links is equal.

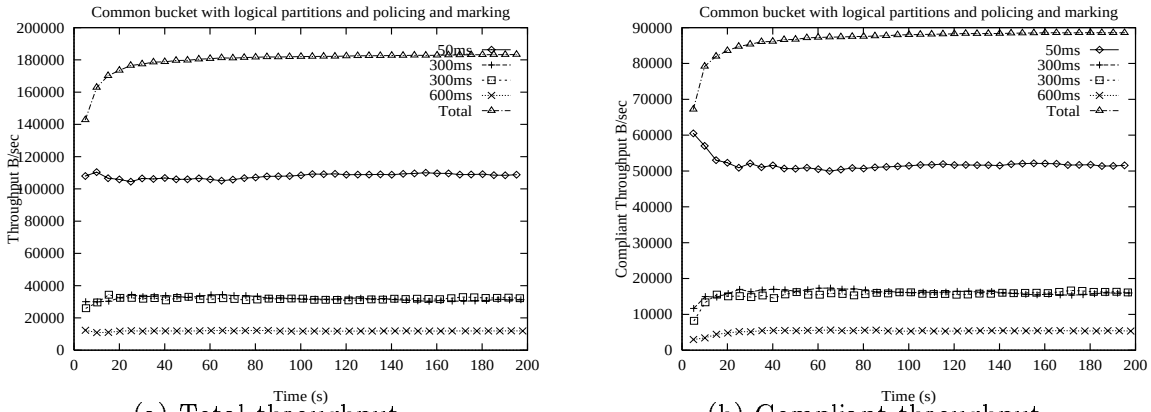
from `cwnd/2` to `cwnd`, increasing at the rate of 1 per rtt_i. The average throughput for the connection is then the ratio of the total bytes sent to the time interval and is given by

$$R_i = \frac{1}{2p * rtt_i}(cwnd + \frac{1}{p * rtt_i}).$$

The ratio of the congestion window `cwnd` that triggers a loss, and the rtt_i is the achievable share of link throughput, $\hat{M}$, where $\hat{M} = \frac{Mp}{\lambda}$, for a packet arrival rate of λ. Thus, $cwnd = \hat{M} * rtt_i$ and $R_i = \frac{Mp}{2\lambda} + \frac{1}{2p * rtt_{av}^2}$. For a generalized window increase algorithm where `cwnd` increases by $g(rtt_i)$ instead of 1 in every rtt_i, the throughput of connection i is

$$R_i = \frac{Mp}{2\lambda} + \frac{g(rtt_i)}{2p * rtt_i^2}.$$

From the above equation, the throughput of a connection is proportional to g_i/rtt_i^2. For fair bandwidth sharing across all connections, g_i should be proportional to rtt_i^2, i.e., $g_i = a * rtt_i^2$ [6], where a is a constant that controls the rate of increase of the congestion window. No criterion for selecting an appropriate value for a is proposed in [6]. One possible interpretation of a is provided in [7] by equating the aggressiveness of a Constant Rate connection with a given value of a to that of a standard TCP connection with a certain RTT. However, proper selection of a depends on the network topology and the number of peer connections [7]; as a result, it is difficult to determine a in a distributed manner and selection of a is typically ad hoc. Another important concern raised by [7] is regarding the increased losses triggered by the Constant Rate connection because the aggressive window increase results in very bursty send patterns. The fix proposed in [7] is to bound



(a) Total throughput (b) Compliant throughput

Figure 11: Policing and marking with common token buckets and logical partitions: Throughput of different TCP connections with different RTT values (compliant packets sent as marked, non-compliant packets sent unmarked). Routers are configured to support ERED; the bandwidth of individual links is different.

the congestion window increase per ACK by 1 segment, so that a Constant Rate TCP connection is never more bursty that a TCP connection in slow start. Alternately, one could smooth the sending of several segments across a longer time period, which is readily achieved via the average and peak rate traffic shaping mechanisms considered previously.

We argue that for aggregated rate control at a server (or proxy), it is possible to derive a meaningful value for the constant a that not only ensures fairness between connections, but also does not result in overly aggressive behavior relative to the rest of the network. Exploiting the idea of equivalent aggressiveness [7], we introduce the notion of *aggregate fairness*.

Aggregate Fair: The set of connections comprising an aggregate is *aggregate fair*, relative to other TCP connections in the network, if the total (average) throughput of the aggregate can be limited to that of a single TCP connection with RTT roughly equal to the average RTT of the connections in the aggregate. An aggregate fair set of connections is not overly aggressive relative to other TCP connections in the network.

Consider a set of n connections in an aggregate, with a round-trip time rtt_i for connection i, and an average round-trip time $rtt_{av} = \frac{\Sigma R_i}{n}$. Using the expression for throughput derived earlier, the average aggregate throughput is $R_{agg} = \frac{nMp}{2\lambda} + \frac{n}{2p*rtt_{av}^2}$. We use this as the upper bound on the aggregate throughput of a set of connections that is aggregate fair, i.e., $R_{agg} = \Sigma R_i$. Solving for a we get $a = \frac{1}{rtt_{av}^2}$, for a corresponding rate of window increase $g(rtt_i) = \frac{rtt_i^2}{rtt_{av}^2}$. Note that $g(rtt_i)$ is less than 1 for a connection with $rtt_i < rtt_{av}$, greater than 1 for a connection with $rtt_i > rtt_{av}$, and exactly 1 for a connection with $rtt_i = rtt_{av}$. For this reason, and since outgoing traffic is shaped at average and peak rates, the resulting connection window increase is not excessively bursty.

We plan to evaluate the above approach to verify that each connection recovers from loss in an aggressive but fair manner while improving bandwidth utilization. We are applying the above insights to the more conservative Increase-by-K window increase policy [7], in which $g(rtt_i) = K$ for long RTT connections, where K might be constant or a function of RTT. These experimental studies would allow us to validate and/or refine the notion of aggregate fairness.

6 Protocol Stack Extensions

We have developed a number of server extensions to implement fair rate control of aggregated TCP connections in AIX, which has a BSD-style UNIX protocol stack. The key components of these extensions are depicted in Figure 12, and have evolved from an architecture developed earlier by the authors [14].

The policy agent (`pagent`) is responsible for querying a global policy repository to obtain the list of policies applicable to the server, translating global policies to local policies, and interacting with the kernel-resident QoS Manager module to install policies in the protocol stack. The policy agent interacts with the QoS Manager via an enhanced socket interface by sending (receiving) messages to (from) special control sockets [14]. In the current implementation, the policy agent runs as a system daemon in user space, and communicates with the policy database server via the LDAP directory access protocol.

QoS Manager is the key component in our architecture, playing a critical role in the control as well as data planes of the protocol stack. It is entrusted with maintaining kernel state for configured policies, managing network resources such as buffers and link bandwidth, and managing the association be-

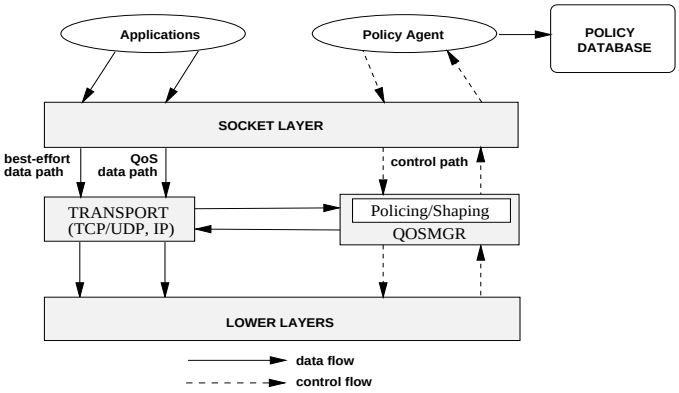


Figure 12: Protocol stack extensions for policy-based service differentiation on Internet servers.

tween individual TCP connections (or UDP sessions) and the appropriate policies. QoS Manager also implements the necessary traffic conditioning functions such as traffic policing, marking, shaping, and buffer allocation based on the desired action on a per-policy basis. A number of minor, but carefully placed, modifications were applied to the socket and transport layers of the protocol stack to allow them to invoke the aforementioned functions. In the current prototype, QoS Manager is implemented as a loadable kernel extension.

6.1 Connection Aggregation and Traffic Conditioning

The QoS Manager module was originally designed for application-initiated signaled QoS using per-flow (connection or session) reservations [14]. As such, the QoS Manager supported a one-to-one association between a connection and a reservation. However, with policy-based service differentiation, a many-to-one association is instead desirable since a policy would frequently control many connections simultaneously. As before, each data socket corresponding to each connection is tagged with a QoS handle that directly identifies the associated policy. The QoS handle reduces the task of packet classification to a single direct lookup, and is used subsequently to correctly handle traffic originating on the data socket. This association is established at connection setup time and removed when the connection terminates. Since a newly installed policy may apply to existing connections, the many-to-one mapping is correctly maintained by searching the entire list of existing data sockets and tagging each socket that matches the specified filter in the new policy being installed.

The QoS Manager provides efficient traffic conditioning support (marking, policing, and shaping) for each policy installed on the server. Traffic conditioning functions for a given policy are invoked for traffic on any connection controlled by that policy; this ensures proper rate control of the entire aggregate. Traffic marking is implemented efficiently by storing the desired packet marking in the connection's protocol control block at connection setup time. Policing is implemented for the specified average as well as peak rates; different packet markings can be generated on the fly for in-profile and out-of-profile packets. Traffic shaping is implemented using system timers to delay packets (by withholding buffers) till compliance; this can impose significant overheads, as discussed below. As an optimization, the policing and shaping functions are invoked only when TCP's congestion window allows it to transmit data.

CPU Type	133MHz PowerPC	33MHz POWER
Set timer	7.4μs	14.0μs
Handle timer	7.1μs	30.1μs
Cancel timer	6.5μs	9.6μs

Table 2: Overheads of system timer operations.

6.2 Traffic Shaping Overheads

For accurate traffic shaping a packet must only be delayed as long as necessary to meet compliance. Achieving the desired delay via per-flow shaping timers does not scale with the number of flows due to the significant overheads imposed by system timers. In an earlier paper we explored the performance impact of supporting traffic conditioning functions in TCP/IP protocol stacks [15]. Table 2 summarizes our measured overheads of timer operations under AIX 4.2; as can be seen, these overheads constitute a significant performance burden for a well-optimized protocol stack, especially for the dominant class of network traffic (i.e., TCP).

To keep system overhead low, it is desirable to reduce the number of timers active simultaneously and/or avoid using very fine-grain system timers. Our implementation employs a single system shaping timer to shape traffic belonging to all active policies; using per-policy shaping timers is possible but the overhead would still be excessive. While this eliminates the overheads due to per-flow shaping timers, it will often be the case that compliant packets must wait for the shaping timer to invoke transmission. This is because, depending on the policies configured and application behavior, traffic associated with different policies may need to be shaped for a wide range of delays. One alternative is to use a fine-grain shaping timer (e.g., an interval of 1-5 ms), but this may impose excessive overhead since all policies may not require traffic shaping. Our implementation provides a configurable shaping timer that we are currently experimenting with in order to determine a reasonable value for the shaping timer interval.

To facilitate accurate traffic shaping with relatively coarse shaping timers, we exploit incoming ACKs on a given con-

nection as a trigger for transmitting compliant packets from that connection, if the congestion window is open. Thus, compliant packets on a connection do not have to necessarily wait for the shaping timer to expire and initiate transmission. This is true for each TCP connection being shaped by a policy action. The shaping timer and per-connection ACKs are the only viable triggers for transmitting previously non-compliant packets on a single TCP connection. Note that for accurate traffic shaping, the ACKs must arrive regularly spaced at the server. This, however, is often unlikely given the bursty nature of network congestion and the observed phenomenon of ACK compression. For aggregate rate control, however, we exploit a number of additional triggers to realize accurate traffic shaping.

6.3 Other Shaping Triggers

With multiple connections being shaped together, the following triggers are likely to occur with reasonable frequency: (i) receipt of ACK on a connection with no data to send, (ii) receipt of duplicate ACKs on a connection, (iii) application send on a connection whose congestion window is closed, and (iv) new connection requests associated with the same policy rule.

These triggers can be used in addition to the two triggers mentioned earlier. However, the viability and efficacy of these triggers depends on the actual workload and design complexity relative to a common fine-grain shaping timer. Each of the above triggers (including the shaping timer) must perform the following steps very efficiently for them to be a viable alternative: (a) check if a previously non-compliant policy rule is now compliant, and (b) apply the fairness criterion to select the appropriate connection to send a packet.

The details of our prototype implementation (including support for logical token buckets and the TCP window modifications outlined in Sections 4 and 5), performance optimizations, and experimental results are beyond the scope of this paper, and will appear in a forthcoming paper [16].

7 Related Work

In our work we have built upon several key areas of TCP-related research: TCP fairness, TCP rate control, and fair flow queuing. We discuss related work in each of these areas below. A summary of key TCP-related research efforts, although in the context of satellite networks, can be found in [11].

TCP Fairness: Fairness between TCP connections has been the subject of many recent research efforts. Router mechanisms to enhance fairness and end-to-end performance include active queue management schemes such as RED [9] and longest queue drop [17]. The importance of TCP unfairness in the congestion avoidance phase for different round-trip times is explored and addressed via congestion window modifications in [6] and [7]. An approach of using weighted proportional fairness to achieve differentiated services in the Internet is presented and evaluated in [18]. An integrated congestion control and loss recovery scheme to improve the performance of parallel TCP connections from a server to the same client (i.e., a kind of aggregate) is proposed in [19]. While not directly relevant, this work demonstrates the benefits of sharing state across different, but related, connections.

TCP Rate Control: Several proposals provide some form of rate-based pacing to TCP so as to smoothen out its data flow. While rate-based pacing [20] is primarily meant for improved TCP performance, the proposed scheme in [8] focuses on the dynamics and rate control of a TCP connection in the context of Integrated Services, and considers TCP congestion window modifications to provide a TCP connection with the desired policed rate. Our proposed window modifications for fair rate control of aggregated TCP connections are less aggressive than those proposed in [8]. Ack-based pacing [12] by a network device external to the traffic source provides another mechanism for rate control of TCP connections; however, our work focuses on timer-based pacing inside the traffic source. A detailed study of traffic conditioning overheads in the context of per-flow rate control is provided in [15]. Traffic aggregation provides both opportunities to reduce some of these overheads, as well as challenges in ensuring fairness while maintaining scalability.

Fair Flow Queuing: In recent years many research efforts have explored link-level fair queuing and scheduling mechanisms and analyzed their bandwidth and delay allocation properties in the context of per-hop and end-to-end quality of service guarantees. Examples include weighted fair queuing (WFQ) and its variants, and class based queuing (CBQ) for hierarchical bandwidth sharing. Our focus is instead on providing fair sharing mechanisms at a traffic source that must aggregate TCP connections for purposes of policy-based rate control. Being at the transport layer, our work is complementary to link-level fair queuing mechanisms. While not directly related, the implications of per-flow queuing on TCP [21] might apply to aggregated TCP connections if, contrary to our transport-layer based support, the aggregation is performed at the link layer.

8 Conclusions and Future Work

In this paper we explored design considerations for policy-based fair rate control of aggregated TCP connections at Internet servers and proxies. We proposed and evaluated different approaches for rate control of aggregated TCP connections via policing with marking or shaping. Specifically, we proposed an approach that employs a common token bucket with logical partitions, and simultaneously achieves high bandwidth utilization and a high degree of fairness. Our proposed modifications to the TCP congestion window increase during congestion avoid-

ance achieves fairness amongst a mixture of connections with long and short lifetimes. We also described the protocol stack extensions in our prototype implementation, and techniques to reduce implementation overheads.

For ongoing and future work we are evaluating our proposed congestion window modifications for a mix of long and short TCP connections. For the prototype implementation, we are also developing aggressive performance optimizations to support traffic aggregation in a scalable fashion. A key challenge is to exploit the available per-connection state for accurate traffic conditioning while minimizing the complexity of maintaining policy-connection associations, implementing logical token buckets, and sharing the unused rate. Each of these aspects are the subject of a forthcoming paper [16].

References

[1] Y. Bernet, J. Binder, S. Blake, M. Carlson, S. Keshav, E. Davies, B. Ohlman, D. Verma, Z. Wang, and W. Weiss, "A Framework for Differentiated Services," October 1998, Work in Progress Internet Draft draft-ietf-diffserv-framework-01.txt.

[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An Architecture for Differentiated Services," *Request for Comments RFC 2475*, December 1998.

[3] The Apache Group, "http://www.apache.org/docs/vhosts/index.html," Apache Virtual Hosts Documentation.

[4] E. Nahum, T. Barzilai, and D. Kandlur, " Performance Issues in WWW Servers," in *Proceedings of ACM SIGMETRICS 99*, May 1999.

[5] T. Lakshman and U. Madhow, "The Performance of TCP/IP for Networks with High Bandwidth-Delay Products and Random Loss," *IEEE/ACM Transactions on Networking*, June 1997.

[6] S. Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks, Part 1: One-way Traffic," *ACM Computer Communications Review*, pp. 30–47, October 1991.

[7] T. R. Henderson, E. Sahouria, S. McCanne, and R. H. Katz, "On Improving the Fairness of TCP Congestion Avoidance," *GLOBECOM*, November 1998.

[8] W. Feng, D. Kandlur, D. Saha, and K. G. Shin, "Understanding TCP Dynamics in an Integrated Services Internet," in *Proceedings of NOSSDAV 97*, May 1997.

[9] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *ACM/IEEE Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

[10] R. Guerin, S. Kamat, V. Peris, and R. Rajan, " Scalable QoS Provision Through Buffer Management," in *Proceedings of SIGCOMM 98*, August 1998, pp. 29–41.

[11] M. Allman, S. Dawkins, et al., "Ongoing TCP Research Related to Satellites," November 1998, Work in Progress Internet Draft draft-ietf-tcpsat-res-issues-05.txt.

[12] R. Satyavolu, K. Duvedi, and S. Kalyanaraman, "Explicit Rate Control of TCP Applications," *ATM-Forum/98-0152R1*, February 1998.

[13] S. McCanne and S. Floyd, "http://www-nrg.ee.lbl.gov/ns/," ns - LBNL Network Simulator, 1996.

[14] T. Barzilai, D. Kandlur, A. Mehra, and D. Saha, "Design and Implementation of an RSVP-based Quality of Service Architecture for Integrated Services Internet," *Journal of Selected Areas of Communications*, April 1998.

[15] R. Engel, D. Kandlur, A. Mehra, and D. Saha, "Exploring the Performance Impact of QoS Support in TCP/IP Protocol Stacks," in *Proceedings of IEEE INFOCOM 98*, March 1998.

[16] A. Mehra, T. Barzilai, Z. Dodson, R. Tewari, and D. Kandlur, "Protocol Stack Extensions for Policy-Based Rate Control," May 1999, In preparation.

[17] B. Suter, T. V. Lakshman, D. Stiliadis, and A. Choudhury, "Efficient Active Queue Management for Internet Routers," *Lucent Tech-Report*, November 1997.

[18] J. Crowcroft and P. Oechslin, "Differentiated End-to-End Internet Services Using a Weighted Proportional Fair Sharing TCP," *Computer Communication Review*, pp. 53–69, July 1998.

[19] S. Seshan, H. Balakrishnan, V. N. Padmanabhan, M. Stemm, and R. Katz, " TCP Behavior of a Busy Internet Server: Analysis and Improvements," in *Proceedings of INFOCOM 98*, March 1998.

[20] V. Visweswaraiah and J. Heidmann, "Rate Based Pacing," *www.isi.edu/lsam/publications/rate_based_pacing*, 1997.

[21] B. Suter, T. Lakshman, D. Stiliadis, and A. Choudhury, " Design Considerations for Supporting TCP with Per-Flow Queueing," in *Proceedings of INFOCOM 98*, March 1998, pp. 299–306.