Receiver Based Management of Low Bandwidth Access Links

Neil T. Spring, Maureen Chesire, Mark Berryman, Vivek Sahasranaman,
Thomas Anderson and Brian Bershad

Abstract—**In this paper, we describe a receiver based congestion control policy that leverages TCP flow control mechanisms to prioritize mixed traffic loads across access links. We manage queueing at the access link to: (1) improve the response time of interactive network applications; (2) reduce congestion-related packet losses; while (3) maintaining high throughput for bulk-transfer applications. Our policy controls queue length by manipulating receive socket buffer sizes. We have implemented this solution in a dynamically loadable Linux kernel module, and tested it over low bandwidth links. Our approach yields a 7-fold improvement in packet latency over an unmodified system while maintaining 94% link utilization. In the common case, congestion-related packet losses at the access link can be eliminated. Finally, by prioritizing short flows, we show that our system reduces the time to download a complex web page during a large background transfer by a factor of two.**

I. Introduction

To handle increasing Internet traffic, network backbones have been equipped with high speed links and fast routers. Overprovisioning backbones alleviates congestion within the network, but also moves it to the edges: the access links of senders and receivers. Studies have shown that Internet traffic is asymmetric: most traffic is sent *from* the server *to* the client [1], [2], [3]. Client access links (e.g. modem or DSL connections) are often the network bottleneck, because they have relatively low bandwidth in comparison with the backbone. Managing contention between incoming traffic flows at the receiver's access link is the focus of this paper.

Although access links are typically used to perform one operation at a time today, this will be less true in the future. Users naturally want to continue to work during long latency operations. For example, a user might browse web pages while listening to real-time streaming audio, download a software package while participating in a chat session with a friend, or download attachments from a mailbox while checking stock quotes. In each of these scenarios, performance suffers because of contention. The aggressive download behavior of web browsers often degrades the quality of the streaming audio by overwhelming the link. A long running download introduces queueing delay that may make the chat session less responsive. Finally, a new web connection may not be able to get a fair share of bandwidth quickly if a long running transfer has filled the access link's queue. We present one aspect of this contention in Figure 1: the response time of a telnet session becomes sluggish as a background transfer adds queueing delay.

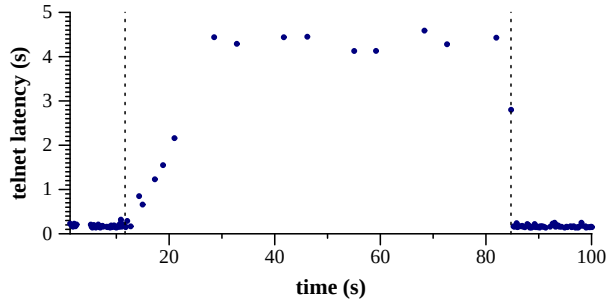


Fig. 1. Effects of queueing delay on the latency of telnet packets during an ftp transfer over a 28.8 modem. Dotted lines at left and right represent the start and end of the ftp transfer. The 4 second delay represents signifi cant buffering at the access link. Round trip time without queueing delay was 0.16 seconds.

It is in the client's interest to reduce queueing at the access link to solve the problems presented by the scenarios described above. Furthermore, the client has all the information necessary to determine the rate at which packets should be sent by the server. The bandwidth of the link is known, because the user typically pays for it. The receiver also knows the number of connections that are active as well as the relative importance of different streams. When there are several concurrent connections using the receiver's access link, it is natural for the receiver to manage the resulting congestion, by sharing state between connections as described in [4] and [5]. We believe that a cooperative congestion control strategy, where the receiver limits congestion at the access link and the server limits congestion in the rest of the network, is the most effective approach. Our receiver based policy manages conges-

email: {nspring, maureen, markb, vivek, tom, bershad} @cs.washington.edu. Neil Spring was partially supported by W. Hunter Simpson of the ARCS Foundation. Mark Berryman was supported by a National Defense Science and Engineering Graduate (NDSEG) Fellowship.

tion by controlling the size of each connection's advertised window.

While TCP is the default protocol for implementing network services, it is not well suited for managing contention at a user's access link. TCP congestion control schemes rely entirely on parameters maintained by the sender (generally the server), which is compelled to infer the network characteristics (bandwidth, sharing, queue capacity) of the receiver's access link. The only signal a TCP sender uses to infer these characteristics is packet loss. Although TCP will adjust to the access link after loss detection, it will overestimate link capacity, filling router buffers (potentially causing loss on unrelated links sharing the router), and will not prioritize among flows.

We have implemented our receiver based policy in the context of a standard TCP/IP protocol. Most importantly, our policy has been implemented entirely within the receiver's stack, and requires no protocol changes.

The rest of the paper is organized as follows. In Section II, we present an overview of our approach. Section III presents our model of network performance, which is used in Section IV to define a receiver based congestion control policy. In Section V, we present a summary of observed performance improvements. In Section VI, we describe existing solutions that share our goals. Finally, we conclude in Section VII.

II. Overview

The primary goal of our work is to reduce the response time of interactive applications contending with bulk-transfer flows. Response time represents user perceptible performance, and contention from background transfers is common. We hope that by reducing response time, the overall utility of these access links is improved.

To realize this goal, we leverage TCP's flow control mechanism to limit the size of the sender's sliding window. This window is an abstraction for the maximum number of bytes a sender is allowed to transmit before getting an acknowledgment from the receiver. Three parameters control the size of this window: the congestion window (*cwnd*), the sender's buffer size, and the receiver's advertised window. At any given time, the smallest of these parameters defines the size of the window. The *cwnd* parameter reflects the sender's estimate of the capacity of the network. That is, $\frac{cwnd}{RTT}$ is the rate at which the sender believes the network can absorb traffic. The receiver's advertised window is included in each acknowledgment returned to the sender, and corresponds to the amount of buffer space available to receive additional data. For all but very high bandwidth connections, the size of the sliding window is usually bound by the *cwnd* parameter. However, a receiver can limit the size of the window by allocating a small buffer.

The fundamental idea behind our solution is to control the receiver's advertised window of each open socket by manipulating its receive buffer size. At a high level, we shrink the receive buffers of long lived transfers to reduce the queueing delay experienced by interactive applications and increase the throughput seen by short transfers. Each flow receives a different allocation based on its round trip time and relative priority. Our approach is simple and has several beneficial properties:

Manages queueing delay experienced by incoming traffic: By reducing the size of buffers allocated to connections, we have the ability to limit the number of packets queued at the bottleneck link, control the composition of the queue, and bound queueing delay. This is useful for the following reasons. First, it allows us to reduce the response time observed by users of interactive network applications. Second, controlling the queue enables us to preempt the packet loss that occurs when TCP's adaptive congestion control algorithm overestimates the capacity of the network. Third, a short queue delay allows new connections to progress through connection setup and slow start quickly and achieve a large share of the link's bandwidth.

Preserves link utilization: By allocating bulk-transfer connections buffer sizes equivalent to the bandwidth-delay product, we have the ability to reduce queueing without adversely impacting throughput performance.

Adapts to changing workloads: Since we can dynamically adjust buffer sizes and change the size of the advertised window in each acknowledgment, we can quickly respond to changes in workloads. In later sections, we describe the algorithms used to guide buffer allocation decisions.

Deploys easily: We believe our solution has fewer barriers for acceptance because it is terminal, requires no modifications to the network, server or application software and requires no support from a service provider.

III. Modeling Latency and Throughput of Low Bandwidth Links

To control queueing by adjusting receive buffer sizes effectively, we need to relate a connection's window size to the length of the queue at the bottleneck link. This relationship depends on two factors: the connection's round trip time, and its share of throughput. In this section, we describe a simple, steady-state model that illustrates how these factors influence buffer size selection. The model assumes connections are receiver-window limited and that significant queueing occurs only at the access link. The effects of loss and variance in network delay are ignored.

While this model is highly simplified, it is sufficient for our purposes since we focus on the performance of low-bandwidth links. More general models of TCP performance are presented in [6] and [7].

In the absence of queueing and loss, the window size (i.e., number of bytes sent without waiting for acknowledgments) necessary to keep a network link busy is equal to the product of the connection's bandwidth ($Xput_i$) and its round trip propagation delay ($BaseRTT_i$). If more bytes are sent, those additional packets are queued in the network, typically at the bottleneck link. Restated algebraically, the number of packets in the network for each connection i in steady state is equal to the number of packets waiting in the queue, plus the bandwidth delay product.

$$window\ size_i = Q\ packets_i + Xput_i \times BaseRTT_i$$

The packets making up the first term increase queueing delay, while the packets making up the second preserve throughput.

The queue and link are both shared between all N connections transferring data:

$$Q\ packets_{total} = \sum_{1 \le i \le N} Q\ packets_i$$

$$Xput_{link} \ge Xput_{actual} = \sum_{1 \le i \le N} Xput_i$$

$Xput_{link}$ is the maximum bandwidth of the link and $Xput_{actual}$ is its delivered throughput. The challenge is to globally choose all $window\ size_i$ so that $Xput_{actual} \approx Xput_{link}$ and $Q\ packets_{total} \approx 0$.

A. Reducing Latency

For interactive applications, user response time depends primarily on the latency experienced by that application's packets. This latency is the sum of propagation delay and queueing delay:

$$RTT_i = BaseRTT_i + Q\ delay$$

The delay associated with queued packets is equal to the size of those packets divided by the link's throughput:

$$Q\ delay = \frac{Q\ packets_{total} \times Packet\ size}{Xput_{link}}$$

The way to decrease $Q\ delay$ is by reducing $Q\ packets_{total}$, which is in turn decreased by reducing $window\ size$s.

B. Preserving Throughput

The actual throughput delivered by the link is limited by two factors: the speed of the link, $Xput_{link}$, and the ratio of the number of packets in flight to the round trip time of a connection. The second factor reflects that at most one window size worth of data can be transferred per round trip, and that each round trip, when the link is underutilized, experiences no queueing delay.

$$Xput_{actual} = \min(Xput_{link}, \sum_{1 \le i \le N} \frac{window\ size_i}{BaseRTT_i})$$

To make certain that all available bandwidth is consumed, that $Xput_{actual} = Xput_{link}$, it is desirable to queue a small number of extra packets. Queueing also ensures that the link remains busy while the sender operates with a reduced window size during loss detection and recovery.

IV. Receiver Window Control Strategy

The models described above guide us in improving response time on a low-bandwidth link by manipulating an individual receiver's advertised window. TCP determines the advertised window based on the space available in a socket's receive buffer, which is allocated by the operating system. In this section, we describe a policy for setting the receive buffer sizes of all open connections that prioritizes short, interactive flows to reduce response time.

Implementing an adaptive buffer allocation policy presents several challenges. First, we must define how flows are classified. We classify flows to enable prioritization of interactive and short-lived flows over long-lived bulk-transfer flows. Second, we must decide when the policy makes buffer allocation decisions. Finally, we must determine the amount of buffer space that should be allocated to flows of each class to reduce response time. We address each of these issues in the following sections.

We valued application-transparency in our design, and avoided introducing additional programming interfaces to support application-specific functionality. Specifically, we do not consider real-time traffic, which would likely require an interface for specifying real-time requirements. In addition, we do not support application dictated priorities, like those supported by WebTP [8]. Extensions to support these features are reasonably straightforward.

A. Classifying Flows

Classification allows us to express preferences that influence how limited link resources are partitioned between

competing flows. Since the degree of contention for the link changes as connections are created or destroyed, and since the characteristics of a flow may vary over time, our classification scheme is dynamic.

We categorize flows into four priority classes:

i) **Interactive** flows are sensitive to latency performance, but are insensitive to throughput (e.g. a chat session).
ii) **Short-lived bulk-transfer** flows are sensitive to both latency and throughput performance (e.g., a HTTP connection downloading a small web page.)
iii) **Long-lived bulk-transfer** flows are throughput-intensive (e.g., an ftp download of a large image file)
iv) **Idle** connections neither send nor receive data.

Connections are initially classified by port number and then by their observed behavior. This quickly separates well known applications, like ftp and telnet, into long-lived bulk-transfer and interactive classes, respectively.

The policy classifies flows dynamically by maintaining modest additional connection state. When an unknown connection is opened, it is considered interactive. By keeping track of the number of bytes received by a connection since its last sent packet, we identify bulk-transfer flows. Once a connection receives at least Rcv_{short}, currently 2KB, it is classified as a short-lived bulk-transfer flow. When the amount of data received exceeds Rcv_{long}, currently 8KB, the connection becomes a long-lived bulk-transfer flow. Idle connections are discovered by monitoring the time elapsed since a packet was last received. This threshold is currently 30 seconds.

B. Scheduling Buffer Allocation Decisions

When a connection is established, destroyed or reclassified, receive buffer sizes for all active sockets are recalculated.

Buffer allocation changes are applied conservatively to promote stability. Shrinking the buffer on an unsuspecting connection might confuse the sender by reducing the window size. The Linux implementation of TCP correctly closes the window slowly as new packets fill it. That is, it does not move the right edge of the sliding window to the left, as specified in [9].

Increasing the buffer size quickly could enable a sender to inject several packets simultaneously, compromising the stability of ack pacing, and reducing the ability of the policy to reduce the buffer size again when conditions change. Our policy increases the buffer size by one packet on receipt of each packet until the target size is reached, similar to slow start.

C. Dynamic Buffer Allocation

The size of the receive buffer allocated to a connection depends on the flow's priority and the degree of contention between existing flows. Interactive flows have the highest priority. Short-lived bulk-transfer flows are next, followed by long-lived bulk-transfer flows. We give idle flows the least priority. This prioritization scheme is influenced by process scheduling algorithms in operating systems that improve response time by favoring short jobs over long running processes. To avoid starvation, each connection is guaranteed a minimum buffer size equivalent to one MTU (i.e., one packet). In this section, we describe how the policy determines buffer size allocations for flows in each class, in order of increasing priority.

There are two goals that guide decisions of buffer allocations. First, we control the length of the queue at the upstream router. Second, we allocate bandwidth by giving larger buffers to short lived flows.

There are two target queue lengths our policy attempts to maintain at the access link. First is $Q\ Length_{Delay}$, which represents the queueing delay the user is willing to tolerate to preserve throughput. When there are interactive flows, this value limits the queue length. An empty queue makes it harder for the long lived flow to achieve the full throughput of the link, and this target value allows the user to control the balance between latency and throughput prioritization.

If there are no interactive flows, or if $Q\ Length_{Delay}$ is larger, the queue length is limited by the second target queue length, $Q\ Length_{Loss}$. This is the target length to avoid queue overflow and loss at the access link. In the formulas presented in this section, the applicable target queue length is presented as $Q\ Length_{Target}$.

C.1 Idle Connections

Idle connections receive one-packet buffers. This small buffer size is allocated to avoid unpredictable behavior in case the connection's classification changes. If the connection becomes active, the policy can increase the advertised window to the appropriate size within one round trip time, while the connection is still in slow-start. By detecting connections that become idle, we can redistribute link resources to other flows and improve their performance.

C.2 Long-Lived Bulk-Transfer Flows

There are two cases to consider when selecting buffer sizes for long lived flows. When there are no higher priority flows, link resources are divided equally. When higher priority flows exist, buffer allocation is limited to improve the response time of the higher priority flows. These cases

are presented in more detail below.

Case 1: No Short-lived Connections. In the absence of short-lived flows, the policy's goal is to ensure that long-lived flows use the full bandwidth of the link, while bounding the queue length to $Q\ Length_{Target}$. The buffer size allocated to each long-lived connection i is then defined as:

$$buffer\ size_i = \frac{Xput_{link} \times BaseRTT_i + QLength_{Target}}{N_{bulk}} \quad (1)$$

where N_{bulk} corresponds to the number of bulk-transfer flows in the system. This allocation approximates an equal share of the throughput and buffer space of the link, for connections of varying $BaseRTT$. In Section IV-D we describe how parameter values in this equation are determined.

Case 2: Contention with Short-lived Flows. To increase the bandwidth available to short-lived flows, we choose to sacrifice the throughput of long-lived flows. Each long-lived flow gets the minimum buffer allocation of one packet. Although this may severely impact the throughput of the long-lived connection, it is not throttled for long: the short-lived connection will either terminate or be quickly demoted as it receives additional data.

C.3 Short-lived bulk-transfer flows

When there are no interactive flows contending with short-lived flows, each connection's buffer size is determined using equation 1 with $Q\ Length_{Target} = Q\ Length_{Loss}$. When interactive flows are introduced, the policy reduces buffer sizes further, using $Q\ Length_{Target} = Q\ Length_{Delay}$.

C.4 Interactive flows

Interactive flows typically receive a few small packets at a time and therefore do not consume much bandwidth. For this reason, the size of the buffer allocated to this type of flow has less importance. To guard against the case where an interactive flow becomes a bulk-transfer flow, we allocate the same buffer size as other bulk-transfer flows, using equation 1 above.

D. Determining Parameters

Values for parameters used by the policy are generally supplied by the user, but could easily be determined dynamically. In this subsection, we describe how we currently set these parameters and strategies that could be used to estimate them dynamically.

$Xput_{link}$, the bandwidth of the access link, is currently a user-specified parameter. Several tools are available for dynamically measuring link throughput ([10], [11]), or the receiver could simply observe the rate at which packets are received.

$Q\ Length_{Loss}$ is set to one half the size of the queue of buffers available at the access link. We estimate the available queue length by noting how many consecutively sent UDP packets are received from a host on the Internet close to the access link. A useful estimate could be derived passively from the pattern of loss during TCP slow start. Tools like those described in [12], [13] could also be used.

$Q\ Length_{Delay}$ is defined by a user-supplied value. This value expresses the maximum increase in latency the user is willing to tolerate due to packets queued at the access link. We have configured this length to correspond to a 0.4 second delay. A reasonable value for this parameter may decrease with increased link speed.

N_{bulk}, the total number of bulk-transfer flows, is a counter maintained by the policy.

Rcv_{short} and Rcv_{long} are set to 2KB and 8 KB, respectively, but could increase with $Xput_{link}$.

Getting an accurate value for $BaseRTT_i$ is difficult because round trip time measurements maintained by TCP variables are aggregated into a smoothed round trip time estimate, *srtt*, which includes queueing delay. Our policy estimates propagation delay using the minimum round trip time observed. This is the same approach used in TCP Vegas [14].

Although there are several important parameters used by the policy, they are reasonably easy to derive. Those parameters that reflect link characteristics can be determined dynamically using simple tools. The effectiveness of the system does not seem to be sensitive to the higher-level parameters like $Q\ Length_{Delay}$ and Rcv_{short}. Automatically determining appropriate values for these parameters is the subject of future work.

V. Results

In this section, we evaluate the effectiveness of our approach. Each of these performance measurements demonstrates an aspect of our goal to improve response time while maintaining high throughput. Specifically, we show that:

- the latency of an interactive application (telnet) competing with a background transfer over a modem can be reduced from over 4 seconds to 0.6 seconds, with only a 4% sacrifice in bulk transfer throughput;
- latency can be controlled even when transferring from a distant host over a variable network, which demonstrates our ability to adapt to changing workloads;

- the time to download a web page while running a large background transfer can be reduced by a factor of two;
- in the common case, congestion related packet losses at the access link can be eliminated.

We present detailed performance measurements from individual executions that are typical of the behavior of the system.

A. Experimental Setup

In each experiment, our client machine was a Pentium running a stable version of the Linux operating system (2.2.7). The system was modified to include a kernel module with an implementation of our congestion control policy. For the first three experiments we present, this machine is connected to the Internet via a 28.8 Kbps modem to the University of Washington dial-in modem pool. For the last two, a similar machine is connected through Dummynet [15] to simulate links of varying speed.

In almost all experiments, we execute ftp transfers from the computer science department's anonymous ftp server. For the distant host scenario, we transfer from an ftp server in Australia.

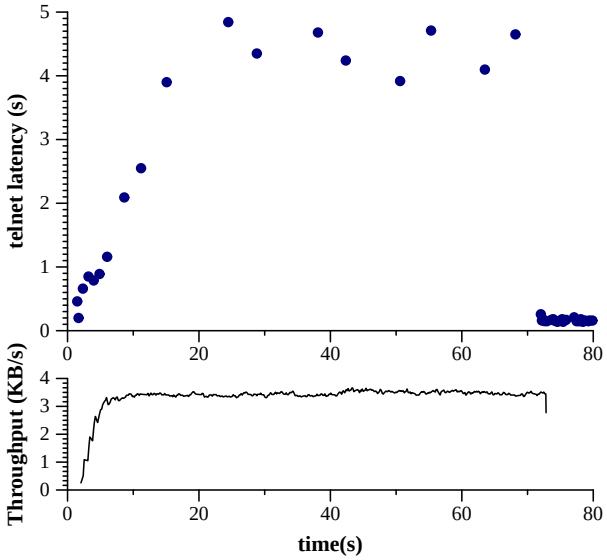
The modem's PPP software was configured to use an MTU of 576 bytes. This is smaller than the default MTU, and was chosen based on initial experiments and a recommendation presented in [16]. We also used this smaller MTU for the otherwise unmodified system.

B. The Classic Scenario

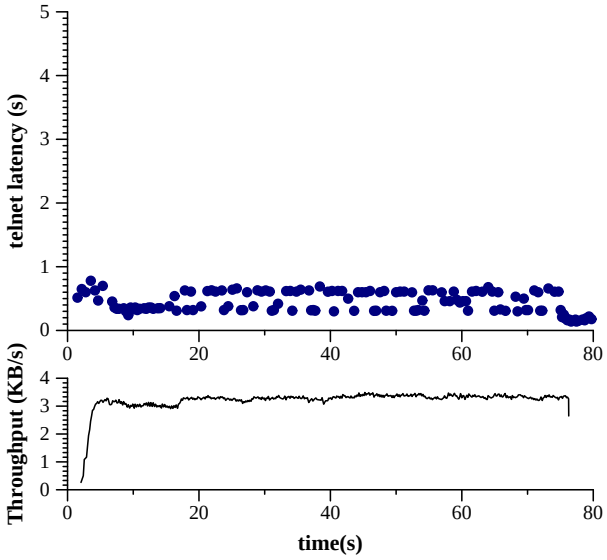
The scenario that served as our early motivation consists of a telnet session running simultaneously with a single ftp download. An Expect script simulates a user typing commands in the telnet session. We use this scenario to show that: (1) the default policy for assigning receive buffer sizes provides poor latency performance to interactive applications; and (2) a smaller window is sufficient to saturate the link, with much less queueing. To demonstrate this, we run the experiment using both the OS default buffer size and our congestion control policy. The telnet latency for each case is graphed in Figure 2.

For this scenario, the Linux default receive buffer (32KB) is far too large, and the connection is actually send buffer limited to 16KB. With a more modest receive buffer of 8KB, the default for Windows 98 and NT [17], the latency in Figure 2(a) could be expected to drop to around 2 seconds from 4.

We notice in Figure 2(b) that telnet latency is reduced to 0.6 seconds ($Q\ Length_{Delay} + BaseRTT$) from four seconds by restricting the size of the buffer allocated to the ftp data connection. Ftp throughput is not affected significantly, confirming that the excess receiver buffer size actually does not improve throughput performance, but only adds queueing delay.



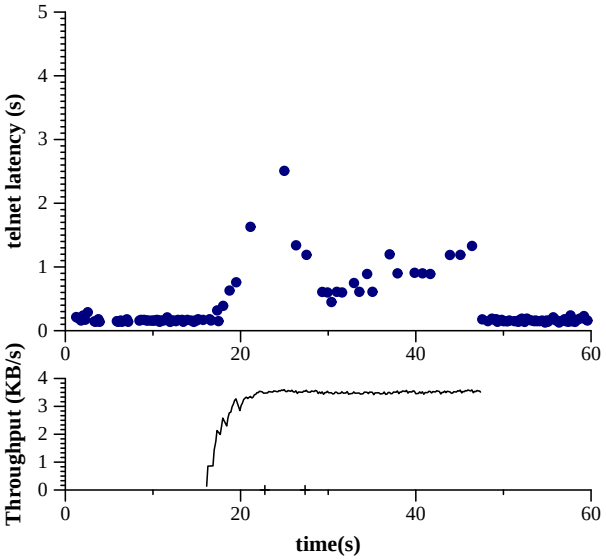
(a) unmodifi ed system



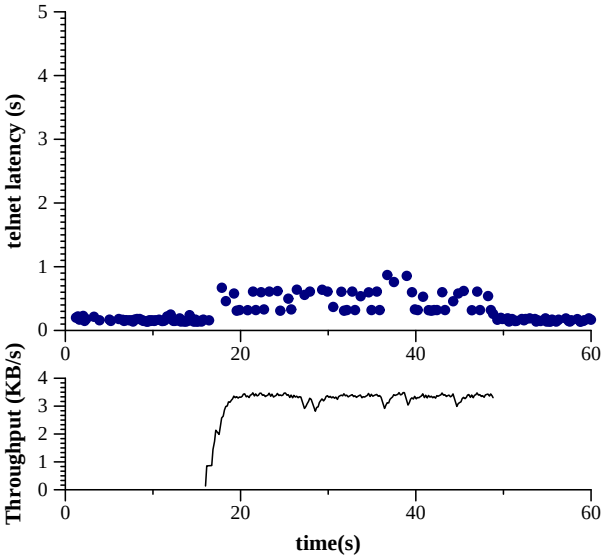
(b) receiver managed

Fig. 2. Latency for telnet packets sent during an FTP download of 240KB download. In (a), the duration of the transfer was 73 seconds, for a throughput of *3.3KB/s*. For (b), 76 seconds, for a throughput of *3.1KB/s*. The larger number of points in the lower graph is due to Nagle's algorithm.

An interesting observation is that Figure 2(a) has fewer data points than Figure 2(b). This is an effect of Nagle's algorithm [18]. Nagle's algorithm restricts the number of small, unacknowledged packets in the network for any connection to one, to prevent connections from sending a large number of very small packets in succession (as would telnet for every keystroke). Because of this restriction, a new telnet packet enters the network only when its predecessor has been acknowledged.

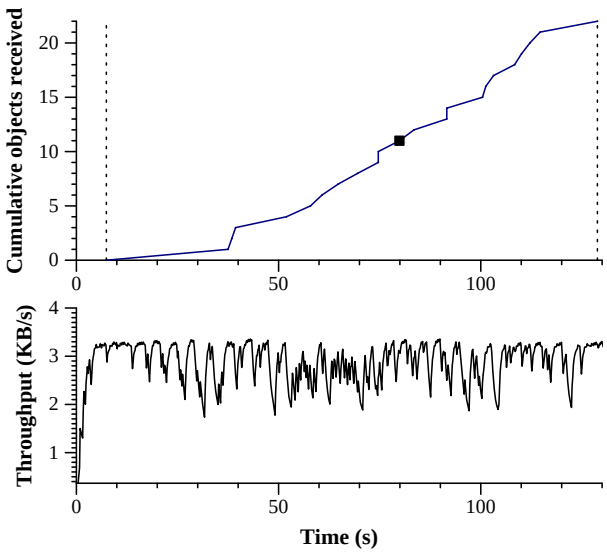


(a) unmodifi ed system

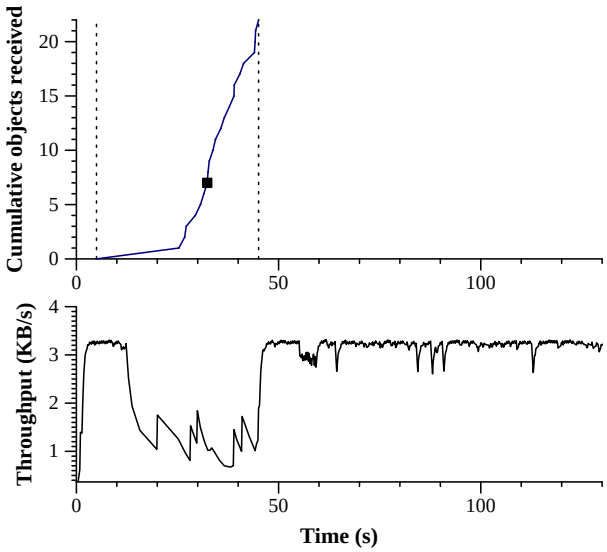


(b) receiver managed

Fig. 3. Latency for telnet packets during one FTP download from Australia. The 106KB download took (a) 38.6 seconds, (b) 38.8 seconds to complete, at an average throughput of (a) 2.7 KB/s (b) 2.7 KB/s.



(a) unmodifi ed system



(b) receiver managed

Fig. 4. Web download in contention with a long running ftp. The dotted lines represent the start and end of the web download, and the square represents the completion of the main page. The web transfer completed in (a) 121 seconds, (b) 40 seconds. The 970KB background transfer, shown in the lower graphs, completed in (a) 5 minutes, 21 seconds, (b) 5 minutes 32 seconds.

C. Large RTT

Connections with a large round trip time are handled gracefully by our system. To demonstrate this, we simulate a telnet session in contention with an ftp download. The background ftp transfer in this case was from ftp.cc.monash.edu.au with a $BaseRTT$ of 560 ms (370ms across the Internet, and around 190 ms from the modem). Figure 3 compares latency and throughput measurements obtained from an unmodified system and a system running the receiver based policy.

D. The Web

Prioritization of short-lived bulk transfer flows enables a reduction in web access response time.

We conducted an experiment to test the improvement in response time downloading complex web pages. While downloading a large (970 KB) file, we loaded a locally mirrored copy of the contents of `http://www.amazon.com/` using Netscape Communicator 4.5. The browser cache was empty, and we do not

consider the time taken for name lookups. The server ran Microsoft IIS on Windows NT 4.0. The throughput observed by the background transfer, along with a cumulative representation of the number of objects from the web page downloaded over time, are shown in Figure 4.

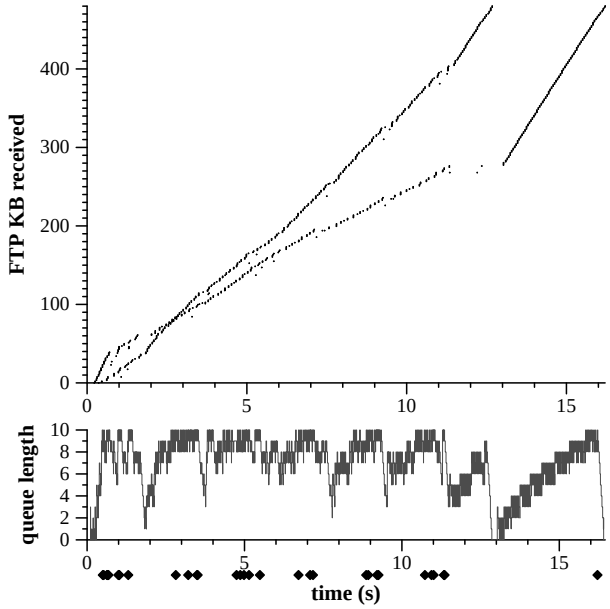
The web request on the system governed by our policy is able to consume more instantaneous bandwidth. In the lower graphs of Figure 4, we show the number of bytes transferred by ftp. The graph only shows the first 130 seconds of the 970KB transfer. Notice that the ftp client under our policy sacrifices more bandwidth while the web transfer completes. This difference in performance is typical when there is limited queue space at the access link.

E. Congestion Related Losses

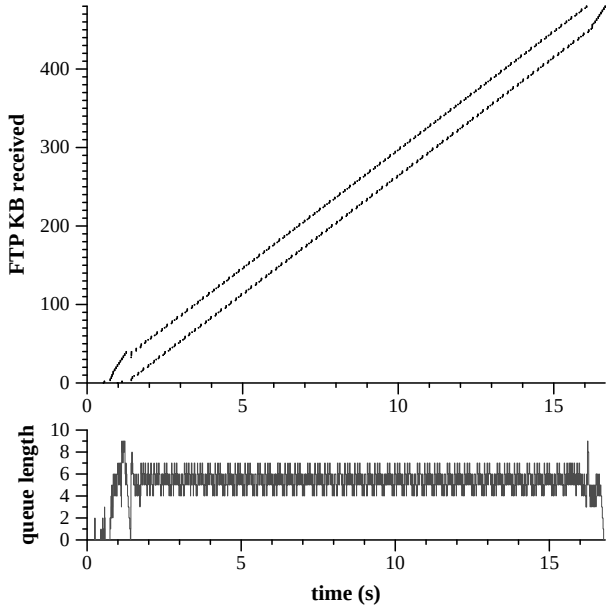
In this section, we demonstrate the performance of the congestion avoidance policy in the absence of interactive traffic. The experimental setup consists of an ADSL link simulated using the Dummynet package for FreeBSD. The downlink bandwidth is 512Kbits/s, and the simulated latency of the link is 22ms. The maximum queue capacity was set to 10, 536 byte packets. We set $QLength_{Loss}$ in the policy to 6 packets. For this experiment, we run two concurrent file transfers, the second of which is started 50ms after the first. This separation gives the first a little time to get started, but not enough time to fill the queue. In the unmodified system, the second transfer would starve if it started when there was a full queue because of synchronization effects [19].

In Figure 5, we show the cumulative number of bytes transferred and the queue length at the simulated ADSL link over time for both the default system and a system managed by our policy, respectively. Beneath the queue length graph, diamond symbols indicate losses due to queue overflow.

Three things are apparent in Figure 5b. First, the number of packets queued remains stable at around 6, demonstrating that the connections are receiver window limited by our policy. The variation is likely the effect of delayed acknowledgments. Second, there are no congestion related losses, even before the connections reach stability. Third, the transfers have reasonably fair throughput. Each of these represents potential performance improvement. Although the time to download both files is similar, server performance is improved because resources are not wasted on the retransmission of lost packets.



(a) unmodifi ed system



(b) receiver managed

Fig. 5. Top: Cumulative bytes transferred by two ftp's under the default (congestion limited) system. Bottom: Queue length at the bottleneck link. Diamonds beneath the graph represent congestion losses. There were no congestion losses for the receiver managed system.

VI. Related Work

There is extensive literature on managing network queueing. We present those with similar goals: to reduce queueing and prioritize flows. We separate these queueing strategies by the goals they address.

A. Queue Reduction

Random Early Detection (RED) gateways reduce queueing by monitoring the average queue length and randomly selecting packets to be dropped before the queue becomes full [19]. Packet losses signal the sender to de-

crease the send rate. This reduces queue occupancy, and thus delay. Although clever, RED is not enabled on many routers at access links.

TCP Vegas senders reduce queueing by monitoring the rate at which packets are accepted by the sender, and use a congestion window sized slightly larger than the bandwidth delay product [14]. Unfortunately, Vegas is a server-side solution, so its advantages can only be realized at the access link if it is deployed on all servers. Our solution is similar in spirit, but receiver based for explicit control over the access link.

B. Prioritized Flows

In Weighted Fair Queueing [20] (WFQ) and related queue management schemes, packets from each flow are queued separately. Each queue is given a weight, corresponding to the share of bandwidth the router will allocate to it. Interactive flows experience less contention. Although WFQ would solve most of the issues raised in this paper, it is not widely deployed because of implementation complexity.

We also share motivation with Packeteer, which seeks to "condition" incoming traffic by delaying acknowledgements returned to servers [21]. Packeteer targets a business environment where several users share a medium bandwidth (T1 or T3) link, and uses specialized hardware.

WebTP is an alternative protocol to TCP specifically designed for web traffic [8]. WebTP shares our goal of supporting the prioritization of incoming traffic. WebTP achieves this by giving the browser application explicit control over which packets to download, and the opportunity to accept out of order data delivery. WebTP is designed specifically for web traffic and would require modification both to clients and servers or widespread deployment of WebTP proxies.

C. Buffer Tuning for Performance

Semke, Mahdavi, and Mathis developed a mechanism that tunes sender buffer sizes to improve throughput on high bandwidth networks [22]. Allocating a send buffer that is too small underutilizes a link, while allocating one too large consumes valuable server memory which can ultimately impact the throughput when several connections are active. Our work is complementary to theirs: while they manage contention between outgoing flows for shared memory buffers, we manage contention between incoming flows for shared queue space at the access link.

Kalampoukas, Varma, and Ramakrishnan simulate limiting the receiver's window to reduce queueing and synchronization effects, and show performance improvements even when the access router uses intelligent discard policies like RED [23]. Although they employ the same mechanism, their goals are fairness and throughput, while our goals include prioritization and reduced delay. We believe our work demonstrates that their congestion avoidance scheme works in practice, and that their simulations demonstrate the robustness of congestion avoidance using the receiver's advertised window.

D. Summary

Although well developed and studied, these strategies are not widely deployed and available for use today. Our receiver based solution can be installed by the client and provides the relevant benefits without relying on ISP's or web servers.

VII. Conclusions and Future Work

In this paper, we have shown that there is potential to reduce interactive delay in the presence of contention on a dedicated, low-bandwidth link. We have developed a mechanism and policy for manipulating receive buffer sizes to improve performance, and shown that it can be applied in a variety of scenarios, including web browsing.

Our work is preliminary, in that it does not address application level priority control or real time constraints. It would be simple to allow an application to dictate the priority class of each of its connections, so that the system could leverage application specific knowledge. We provide a form of real-time latency guarantee, in the form of $Q\ Length_{Delay}$, but our system was not designed to address general real-time issues.

Finally, when the access link is a shared medium, such as a cable modem, contention from traffic received by other users is significant. Since our system adopts some of the strategies of TCP Vegas, it is likely that similar unfairness will result, and aggressive receivers will tend to obtain an unfair share of bandwidth. Isolating receivers, either by queueing each receiver's packet separately or reducing the degree of sharing on a segment, would both help individual receivers in the presence of contention and enable receiver based prioritization.

References

[1] Kevin Thompson, Gregory J. Miller, and Rick Wilder, "Wide-area internet traffic patterns and characteristics," *IEEE Network*, vol. 11, no. 6, pp. 10–23, Nov. 1997.

[2] K. Claffy, Greg Miller, and Kevin Thompson, "The nature of the beast: Recent traffic measurements from an Internet backbone," in *Proceedings of INET '98*, July 1998.

[3] CAIDA, "Traffic workload overview," `http://www.caida.org/Learn/Flow/tcpudp.html`, June 1999.

[4] Joe Touch, "TCP control block interdependence," April 1997, RFC 2140.

[5] Hari Balakrishnan, Hariharan Rahul, and Srinivasan Seshan, "An integrated congestion management architecture for internet hosts," in *Proceedings of ACM SIGCOMM'99*, September 1999.

[6] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *Computer Communication Review*, July 1997.

[7] Jitendra Padhye, Victor Firoiu, Don Towsley, and Jim Kurose, "Modeling TCP throughput: A simple model and its empirical validation," *Computer Communications Review*, vol. 28, no. 4, October 1998, a publication of ACM SIGCOMM.

[8] Rajarshi Gupta, Mike Chen, Steven McCanne, and Jean Walrand, "WebTP: A receiver-driven web transport protocol," in *Proceedings of IEEE INFOCOM'99*, May 1999.

[9] Jon Postel et al., "Transmission control protocol specifi cation," 1981, ARPA Working Group Requests for Comment DDN Network Information Center, SRI International, Menlo Park, CA, RFC-793.

[10] Robert L. Carter and Mark E. Crovella, "Dynamic server selection using bandwidth probing in wide-area networks," in *Proceedings of IEEE INFOCOM'97*, 1997.

[11] Van Jacobson, "Pathchar," `http://www.caida.org/Pathchar/`.

[12] Matthew Mathis, "Windowed ping: an IP layer performance diagnostic," in *Proceedings of of INET'94/JENC5*, June 1994.

[13] Matthew Mathis, "Diagnosing internet congestion with a transport layer performance tool," in *Proceedings of INET'96*, June 1996.

[14] Lawrence Brakmo and Larry Peterson, "TCP Vegas: End to end congestion avoidance on a global internet," *IEEE Journal on Selected Areas in Communication*, vol. 13, no. 8, pp. 1465–1480, October 1995.

[15] Luigi Rizzo, "Dummynet: a simple approach to the evaluation of network protocols," *ACM Computer Communication Review*, vol. 27, no. 1, pp. 31–41, January 1997.

[16] Van Jacobson, "Compressing TCP/IP headers for low-speed serial links," February 1990, RFC 1144.

[17] Jamshid Mahdavi, "Enabling high performance data transfers," `http://www.psc.edu/networking/perf_tune.html`.

[18] John Nagle, "Congestion control in IP/TCP internetworks," January 1984, RFC 896.

[19] Sally Floyd and Van Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

[20] Alan Demers, Srinivasan Keshav, and Scott Shenker, "Analysis and simulation of a fair queuing algorithm," in *Proceedings of ACM SIGCOMM*, 1989, pp. 1–12.

[21] Packeteer, "Technology for intelligent bandwidth measurement," `http://www.packeteer.com/technology/technology.htm`.

[22] Jeffrey Semke, Jamshid Mahdavi, and Matthew Mathis, "Automatic TCP buffer tuning," *Computer Communications Review*, vol. 28, no. 4, October 1998, a publication of ACM SIGCOMM.

[23] Lampros Kalampoukas, Anujan Varma, and K. K. Ramakrishnan, "Explicit window adaptation: A method to enhance TCP performance," in *Proceedings of IEEE INFOCOM'98*, March 1998.