The Delay-Friendliness of TCP

Eli Brosh[1], Salman Abdul Baset[1], Vishal Misra[1], Dan Rubenstein[2], Henning Schulzrinne[1]
[1] Department of Computer Science
[2] Department of Electrical Engineering
Columbia University
{elibrosh,salman,misra,danr,hgs}@cs.columbia.edu

ABSTRACT

TCP has traditionally been considered unfriendly for real-time applications. Nonetheless, popular applications such as Skype use TCP since UDP packets cannot pass through many NATs and firewalls. Motivated by this observation, we study the delay performance of TCP for real-time media flows. We develop an analytical performance model for the delay of TCP. We use extensive experiments to validate the model and to evaluate the impact of various TCP mechanisms on its delay performance. Based on our results, we derive the working region for VoIP and live video streaming applications and provide guidelines for delay-friendly TCP settings. Our research indicates that simple application-level schemes, such as packet splitting and parallel connections, can reduce the delay of real-time TCP flows by as much as 30% and 90%, respectively.

General Terms

Performance, Measurement

Keywords

Performance modeling, VoIP, live video streaming, TCP congestion control

1. INTRODUCTION

The popularity of real-time applications, such as VoIP and video streaming, has grown rapidly in recent years. The conventional wisdom is that TCP is inappropriate for such applications because its congestion controlled reliable delivery may lead to excessive end-to-end delays that violate the real-time requirements of these applications. This has led to the design of alternative unreliable transport protocols [17,20,31] that favor timely data delivery over reliability while still providing mechanisms for congestion control.

Despite the perceived shortcomings of TCP, it has been reported that more than 50% of the commercial streaming traffic is carried over TCP [16]. Popular media applications such as Skype [7] and Windows Media Services [16] use TCP due to the wide deployment of NATs and firewalls that often block UDP traffic. Further, TCP is by definition TCP-friendly [17] and is a mature and widely-tested protocol whose performance can be fine tuned.

The gap between the perceived shortcomings of TCP and its wide adoption in real-world implementations motivated us to investigate the delay performance of TCP. Our study seeks to address the following questions: (1) Under what conditions can TCP satisfy the delay requirements of real-time applications? (2) Can the performance of these applications be enhanced using simple application-layer techniques? We address these questions in the context of two real-time media applications that are characterized by timely and continuous data delivery: VoIP and live video streaming.

To understand all aspects of the performance of real-time applications, we conduct an extensive performance study using an analytical model and real-world experiments. The analytical model allows us to systematically explore the delay performance over a wide range of parameter settings, a challenging process when relying on experimentation alone. While there exists an extensive literature on TCP modeling and analysis, it is geared towards file transfers [9,28,29] and video streaming [19,34] rather than delay.

We use both test-bed and Internet experiments to validate the model. We analyze how the delay depends on the congestion control and reliable delivery mechanisms of TCP. We further study the impact of recent extensions such as window validation [18] and limited transmit [4]. The results obtained yield guidelines for delay-friendly TCP settings and may further be used to compare the performance of TCP with alternative protocols [17,20] and experimental real-time enhancements for TCP [15,22,25]. We analyze two application-level schemes, namely, packet splitting and parallel connections which we find to significantly reduce the delay of live video streaming flows.

Our research reveals that real-time application performance over TCP may not be as delay-unfriendly as is commonly believed. One reason is that the congestion control mechanism used by TCP regulates rate as a function of the number of packets sent by the application. Such a packet-based congestion control mechanism results in a significant performance bias *in favor* of flows with small packet sizes, such as VoIP. Second, due to implementation artifacts, the average congestion window size can overestimate the actual load of a rate-limited flow. This overestimation reduces the likelihood of timeouts and consequently the resulting TCP delay.

This material was supported in part by the National Science Foundation under Grants No. CNS-0626795, CCR-0615126 and CNS-0202063, and by FirstHand Technologies. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or FirstHand Technologies.

The main contributions of this paper are:

- We are the first to present a discrete-time Markov model for the delay distribution of a real-time TCP flow (Section 4).
- We find that under the same network conditions, VoIP flows suffer from lower TCP delays than live video streaming flows. We derive the working region for VoIP and live streaming flows based on our model and experiments (Section 6.1). VoIP operates well when the network loss rate is at most 2% and RTT is at most 100 ms. Live video streaming operates well when the network loss rate is at most 3% and RTT is 100 ms.
- We study the impact of various TCP mechanisms on the TCP delay (Sections 6.2–6.6). We then provide simple application-level heuristics (Section 7) and delay-friendly guidelines (Section 8) for improving the performance of real-time applications. The most promising heuristic uses parallel connections with shortest-queue first policy and achieves up to 90% delay reduction. For a single connection, a packet splitting heuristic reduces delay by 30% on average.

2. APPLICATION SETTING

We study a general real-time media application, with a Constant Bit Rate (CBR) source, that sends data across the network using TCP. CBR is the most basic and dominant encoding for media flows in the Internet [35]. Although our analysis is general, we focus on CBR sources corresponding to VoIP and live video streaming, as detailed in Section 5. The VoIP and live video streaming flows are application-limited, i.e., their sending rate is a function of media encoding and not the underlying network. This is in contrast to greedy flows, such as FTP, which are network-limited.

Throughout the paper, we refer to the transmission unit of TCP as a segment and to the TCP payload (i.e., the application-layer data unit) as a packet. The maximum segment size, MSS, is determined by the maximum transmission unit of the network path [32]. A common characteristic of real-time applications is their sensitivity to end-to-end delay which may vary from application to application. For live video streaming, there is usually minimal interactivity involved, so the application can afford a startup delay in the order of seconds [16]. For VoIP, low delay of up to 400 ms is required in order to maintain acceptable interactivity [15]. To reduce end-to-end delays, VoIP often uses small payloads (e.g., 160-byte packets) that correspond to 20 ms or 30 ms of audio. Thus, in the context of this paper, the difference between VoIP and live video streaming flows is their packet sizes and their tolerance of delay.

We define TCP delay as the time it takes the application to get a packet from source to destination through a TCP connection. Packet delay, loss rate, and jitter are key parameters that determine the user-perceived media quality [11, 33]. We therefore use the TCP delay distribution to evaluate the performance of real-time applications. From the delay distribution we derive the portion of packets that arrive beyond their scheduled playout times, i.e., the packet loss rate at the application level. The loss rate metric is determined by the α-percentile delay bound, defined as follows. A delay value d of α-percentile corresponds to $1-\alpha$ portion of packets that are delayed more than d time units.

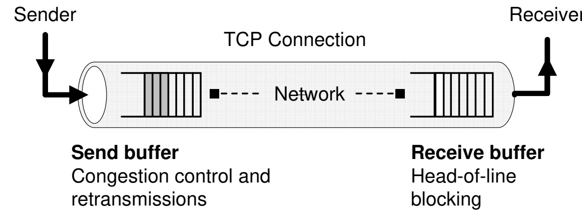

Figure 1: Transport-layer queueing delays

3. TCP DELAY COMPONENTS

Here we examine the various ways in which delay is introduced in a TCP connection with a CBR source. The delay in a TCP connection consists of two main components: (a) network delay, which is the time it takes a segment to get across the network; (b) TCP-level delay, which is an artifact of how TCP reacts to variations in the effective throughput. While throughput variations can occur due to application-level flow control, they are primarily the result of network congestion. To understand TCP-level delays, we briefly describe the transmission behavior of TCP. TCP is a window-based protocol that uses two main mechanisms to regulate its sending rate: Additive-increase-multiplicative-decrease (AIMD) and timeout. These mechanisms may delay data delivery because they require TCP to reduce its sending rate in response to network congestion. In addition, TCP uses packet retransmissions to provide lossless data delivery. This mechanism introduces additional delay for data delivery. A detailed discussion of the mechanisms in TCP can be found in [32].

TCP uses two buffers to provide congestion-controlled reliable data delivery; a send buffer and a receive buffer. The send buffer serves two functions [15]. It absorbs rate mismatches between the application sending rate and the transmission rate of TCP. It also stores a copy of the packets in transit in the network should they be retransmitted. Although these packets are buffered, they do not introduce additional queuing delay for unsent packets. Only the unsent packets held in the send buffer, hereafter referred to as the *backlogged* packets, contribute to the delay of newly admitted packets to the send buffer. The purpose of the receive buffer is to hold out-of-order packets while a loss is being recovered. This buffering results in head-of-line (HOL) blocking delay.

In this paper, we only consider packet backlogging due to network congestion and ignore packet backlogging due to other causes, such as application-level flow-control (e.g., a receiving application that slows down an aggressive sender [32]). Applications usually minimize this backlogging by setting a large receive buffer and operating with non-blocking sockets. Packet backlogging can also occur due to Nagle's algorithm [27] that was added to TCP to limit the transmission of small segments. This algorithm ensures that TCP sends data only when there are at least MSS bytes of available data, and hence improves throughput at the expense of increased transmission delay. In practice, many delay-sensitive applications disable this algorithm to reduce transmission delays [39]. We follow this practice in our work. Figure 1 illustrates the TCP-level delay components. The sender-side delay is caused by the congestion control and reliable delivery mechanisms in TCP, whereas the receiver-side delay is caused by the in-order delivery guarantee of TCP.

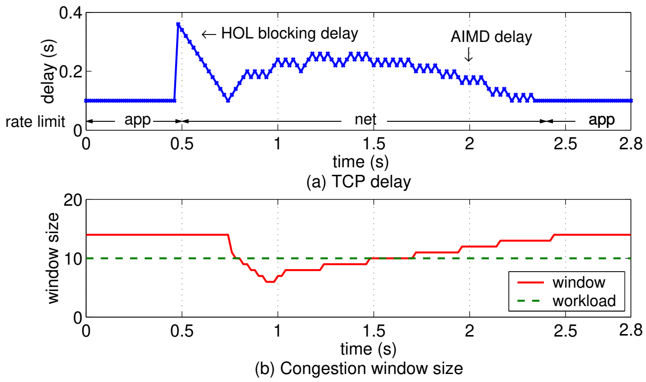

Figure 2: The evolution of the TCP delay and congestion window size for a video-like CBR source.

Figure 2(a) illustrates the delay behavior of a TCP flow driven by a CBR source. The CBR source sends 50 MSS-sized packets per second over a symmetric network with a 200 ms round-trip time (RTT). An application-limited period is seen from 0 s to 0.5 s and from 2.4 s to 2.8 s. We define an application-limited period as a period where the TCP throughput satisfies the source's rate requirement. In this period, the TCP delay is determined by the network delay. A network-limited period is seen from 0.5 s to 2.4 s. In this period, the TCP throughput no longer satisfies the source's rate requirement, resulting in TCP-level delays. TCP moves to a network-limited period when a packet loss occurs. Within the network-limited period there are two subregions: loss recovery, seen from 0.5 s to 0.76 s, and packet backlogging, seen from 0.76 s to 2.4 s. TCP uses retransmission to recover the lost packet, which in turn causes head-of-line blocking delay at the receiver. The receipt of a packet loss indication at time 0.76 s triggers TCP to reduce its congestion window size, resulting in packet backlogging.

Unlike application-limited periods, in network-limited periods TCP probes for additional bandwidth to satisfy the source's rate requirement. In our example, the transmission rate of TCP is governed by the AIMD mechanism and hence is linearly increasing, as seen in Figure 2(b). The mismatch between the input and output rates at the TCP sender results in the quadratic-like delay curve seen in Figure 2(a). TCP moves back to an application-limited period when the rates are matched.

3.1 TCP Interaction with VoIP-like Flows

The performance of real-time applications that use small packets (e.g., VoIP) is directly affected by whether the congestion control mechanism in TCP is byte or packet-based. According to [6], there are two different issues at work. First, a TCP sender can track the congestion control state in terms of outstanding bytes or outstanding packets. Second, a TCP sender can update the congestion control state based on how many bytes are acknowledged, a mechanism known as byte-counting, or by some constant for each ACK arrival, a mechanism known as ACK-counting. A TCP implementation that tracks its congestion control state in terms of bytes, but updates its state in terms of MSS-bytes (e.g., a BSD system) would qualify as an ACK-counting implementation because it does not take into the account the size of the data acknowledged. We compare the performance of ACK and byte-counting mechanisms (Section 6.3) and focus on the latter due to its wide-deployment [23], as also verified by our measurements.

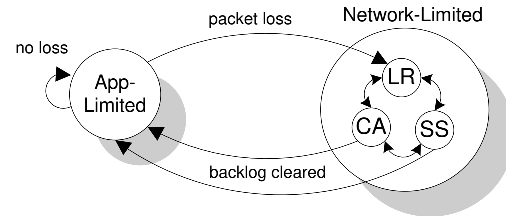

Figure 3: A high-level view of a model for a TCP connection with a CBR source.

4. MODELING TCP DELAY

Our model builds upon the detailed TCP model in [37] that predicts the performance of TCP from the viewpoint of throughput. We extend this model in three ways. First, we include the TCP buffer dynamics in order to predict the delay performance of TCP. Second, we model the window behavior during application-limited periods [18] to accurately capture the loss recovery latency of TCP. Third, we capture the effect of window inflation [6] and the limited transmit mechanism [4] to improve the accuracy of the model for small congestion windows. We assume that the sender is using a NewReno TCP implementation, the predominant TCP variant in the Internet [23], and refer the interested reader to [12, 32] for a detailed description of TCP NewReno's mechanisms.

4.1 A TCP Model

We consider a CBR source that sends fixed-size packets at regular intervals across the network using TCP. Throughout the paper, we assume that the average throughput provided by TCP satisfies the rate requirement of the CBR source. However, transient congestion episodes in the network can still lead to TCP throughput fluctuations and hence to TCP-level delays. These episodes cause the TCP connection to alternate between application-limited and network-limited periods, as described in Section 3.

Mimicking the behavior of a real-world TCP flow, our model consists of two main states: application-limited and network-limited. It transitions from an application-limited state to a network-limited state when a loss occurs. TCP-level delays are introduced only during network-limited states. The model transitions back to an application-limited state when the TCP sender matches its input and output rates (e.g., when packet backlog is cleared). While in a network-limited state, the model moves among four states corresponding to TCP's congestion control phases: slow-start (SS), congestion avoidance (CA), fast recovery (FR), and retransmission timeouts (TO). A high-level view of a model for a TCP connection with a CBR source is shown in Figure 3; for ease of presentation, we merged the timeout and fast recovery states into a single loss recovery (LR) state.

We make several simplifying assumptions in our model, as follows. First, we assume that TCP increases the congestion window by one *packet* per round-trip time, an assumption motivated by the wide-deployment of ACK-counting TCP implementations [23]. Second, we assume that the TCP im-

Notation	Definition
f	load in packets per second
r	load in packets per round-trip time
a	packet size (bytes)
w	congestion window size (in segments)
b	backlog size (bytes)
l	indicates whether loss recovery is required
p	segment loss probability
L	one-way network delay (seconds)
MSS	maximum segment size (bytes)

Table 1: Summary of model notations

plementation does not increase the congestion window when the TCP sender is application-limited, which is the behavior observed for Linux and Windows XP systems (see Section 6.4). Third, we assume that the slow start threshold is statically set to half of the source's sending rate in packets per round-trip time. From our experience, using a static slow-start threshold rather than a dynamic one has a marginal impact on the model's prediction accuracy. Last, we do not model the effect of delayed acknowledgements (ACKs). Nonetheless, our model can be easily extended to support delayed ACKs using a similar approach as in [29].

The CBR source is characterized by two parameters, the data generation rate in packets per second f and the size of a generated packet a. We let r denote the data generation rate in packets per round-trip time. For convenience, we summarized the notations used in this paper in Table 1. We model the behavior of a TCP source by a discrete-time Markov chain with a finite state space $S = \{(w, b, l)\}$ and a probability transition matrix $Q = [q_{s;s'}]$, $s, s' \in S$. Each state is associated with at most three outgoing transitions representing the following events: the receipt of a fast retransmit loss indication, the receipt of a timeout loss indication, and successful delivery of window data. Each transition is associated with a certain number of packet transmissions, and each packet in this transmission is associated with a delay.

In our model, each state is represented by an ordered triple (w, b, l), where w is the current congestion window size in segments, b is the current backlog size in bytes, and l indicates whether a loss has been detected and needs to be recovered from ($l > 0$) or not ($l = 0$). The backlog size value is used to indicate whether the sender is application-limited ($r \le w, b = 0$) or network-limited. The window size value is used to distinguish between the two loss recovery strategies employed by TCP: fast recovery ($w > 0, l = 1$) and retransmission timeout ($w = 0, l \ge 1$), where l indicates the current exponential back-off stage. Table 2 lists the rules for classifying an arbitrary state $s = (w, b, l)$ according to the congestion control phases of TCP. We use the notation $AL = \{(w, b, l) : r \le w, b = 0, l = 0\}$ to denote the set of states for which the application-limited condition holds. The notations CA, SS, FR, TO are defined in a similar way, as shown in Table 2.

4.2 A Delay Performance Model

In this section, we model the three ways in which TCP introduces delays: congestion control, retransmissions, and head-of-line blocking, as detailed in Section 3. We model the time packets are buffered at the sender (i.e., the congestion control delay) by scaling the backlog size by the sources's rate. This modeling approach can be explained by observing that the (unsent) buffered packets left behind after a packet is transmitted must have been admitted to the send buffer while the transmitted packet was buffered. Since we consider a data source with a constant rate, a transmitted packet that leaves behind a backlog of b bytes must have been buffered for at least $b/(fa)$, the backlog size divided by the source's rate in bytes per second. This approach introduces an error in the order of several packetization intervals. The error arises because the Markov chain captures the backlog size evolution in network-limited states at round-trip time granularity. This error can be reduced by keeping track of the inter-sending packet times. However, this will make the state space of the model prohibitively large and hence will limit its usefulness.

Classification	Condition
AL (Application-limited)	$r \le w, b = 0, l = 0$
NL (Network-limited)	$0 < w, b \neq 0$ or $w < r, b = 0$
CA (Congestion avoidance)	$r/2 < w, l = 0$
SS (Slow start)	$w \le r/2, l = 0$
FR (Fast recovery)	$0 < w, l = 1$
TO (Timeout)	$w = 0, l = 1, \ldots, 6$

Table 2: State classification

We determine the head-of-line and the retransmission delay by the loss recovery latency (i.e., the time it takes TCP to detect and recover a lost packet). TCP interprets receipt of three duplicate ACKs as an indication of a packet loss. It immediately retransmits the lost packet upon the receipt of the third duplicate ACK. Hence, we take the time required to receive a fast retransmit loss indication to be $RTT + 3/f$; the RTT term is the time needed for the first duplicate ACK feedback and the $3/f$ term is the maximum time to generate three duplicate ACKs, which is attained when the loss occurs in an application-limited state. For sake of simplicity, we assume that fast recovery always takes a single RTT regardless of the number of packets lost in a transmission window, as suggested by [9].

Using the above observations, we express the TCP delay of the i^{th} packet sent in a transition from state s to state s' as:

$$d^{(i)}_{s;s'} = L + \begin{cases} b/(fa) + RTT + (3+i)/f & \text{if } s' \in FR \\ 0 & \text{if } s' \in TO \\ b/(fa) & \text{otherwise} \end{cases} \tag{1}$$

where L is the one-way sender to receiver network delay. For loss-free transitions, the delay added by TCP is determined by the backlogged packets, and hence is modeled as $b/(fa)$, as shown by the third case of (1). For transitions to fast recovery states, an additional delay of $RTT + (3 + i)/f$ is introduced by the in-order delivery guarantee of TCP, as shown by the first case of (1). Since the TCP sender is likely to be idle during timeouts, we assume no packets are sent in transitions to timeout states.

The number of CBR packets sent in a transition from s

to s' $n_{s;s'}$ is given by

$$n_{s;s'} = \begin{cases} 1 & \text{if } s \in AL, s' \in AL \\ \lfloor \min(b, wMSS)/a \rfloor & \text{if } s' \in \{CA|SS\} \\ \lfloor \min(b, (w+3)MSS)/a \rfloor & \text{if } s' \in FR \\ 0 & \text{if } s' \in TO \end{cases} \quad (2)$$

Since our model evolves at packet-level granularity while in an application-limited state (see Figure 3), a single packet is sent in a loss-free transition from an application-limited state, as captured by the first case of (2). The second case models the number of packets sent in a loss-free transition from a network-limited state, which is determined by the number of backlogged packets that fit into the congestion window. The third case accounts for the extra transmissions due to the receipt of the duplicate ACKs needed to trigger a fast recovery, a mechanism known as window inflation [6].

We obtain the stationary distribution of the Markov chain for the TCP source, π_s, using standard steady-state discrete-time Markov analysis; see for example [38]. Let N_t be the number of packets successfully sent in some time interval $[0, t]$ and let $N_t(d)$ be the number of packets out of N_t that experience delay d. Then, the portion of packets sent that experience delay d is given by $N_t(d)/N_t$. Let D be the steady state delay distribution of a TCP connection with a CBR source. Assume D is defined over some finite interval A. Using renewal theory [38], we can now compute the steady-state delay distribution.

$$\begin{aligned} D = d \quad & \text{w.p.} \lim_{t\to\infty} \frac{N_t(d)}{N_t} & \forall d \in A \\ = d \quad & \text{w.p.} \frac{\sum_{s\in S} \pi_s \sum_{s'\in S} q_{s;s'} \sum_{i=1}^{n_{s;s'}} I_{d_{s;s'}=d}}{\sum_{s\in S} \pi_s \sum_{s'\in S} q_{s;s'}\, n_{s;s'}} & \forall d \in A \end{aligned} \quad (3)$$

where I is the indicator function, π_s is the steady-state distribution of the chain, and $d_{s;s'}$ and $n_{s;s'}$ are given in (1) and (2), respectively. The numerator and denominator correspond, respectively, to the number of packets sent that experience delay d in steady-state and the number of packets sent in steady-state. Equation (3) can be solved numerically to yield the performance statistics of TCP: the delay jitter σ_D and the α-delay percentile $\arg\max_x P\{D \leq x\} \leq \alpha$, along with other useful statistics such as the mean delay $E[D]$.

4.3 Backlog and Window Size Evolution

In network-limited periods, TCP probes for additional bandwidth to satisfy the rate requirement of the source. Specifically, it increases the window size by one every round-trip time in congestion avoidance phase, and doubles the window size every round-trip time in slow-start phase. However, it is pretty typical for TCP implementations not to increase the congestion window when the TCP sender is application-limited (see Section 6.4). Hence, in the absence of packet loss, the TCP model transitions from state $s = (w, b, l)$ to state $s' = (w + 1, b', l')$ if $s \in CA$, to state $s' = (2w, b', l')$ if $s \in SS$, and to state $s' = (w, b', l')$ if $s \in AL$. A detailed description of the Markov chain is given in Appendix A.

Since TCP is a byte stream protocol, it can assemble a number of small application packets into one TCP segment. An application that uses small packets (e.g., VoIP) yields a TCP flow that dynamically varies its segment size, and hence the packet size on the wire, depending on the congestion in the network. During network-limited periods, the data backlog often enables the TCP sender to use the maximum segment size. In application-limited periods, however, there is no backlog at the sender, and TCP matches the segment size to the application payload size. Let M_s be the size of a segment transmitted in a transition from state s. Hence, $M_s = a$ if $s \in AL$ and $M_s = MSS$ otherwise.

The *backlog evolution* (i.e., the TCP send buffer occupancy evolution) for two successive states, $s = (w, b, l)$ and $s' = (w', b', l')$, is modeled by

$$b' = \begin{cases} \max(0, b + aft_{s;s'} - M_s) & \text{if } s \in AL, s' \in AL \\ \max(0, b + aft_{s;s'} - wM_s) & \text{if } s' \in \{CA|SS\} \\ \max(0, b + aft_{s;s'} - (w+3)M_s) & \text{if } s' \in FR \\ \max(0, b + aft_{s;s'}) & \text{if } s' \in TO \end{cases} \quad (4)$$

where $t_{s;s'}$ is the time taken for the transition from s to s', which can be found below. The first term in (4) $b + aft_{s;s'}$ models the increase in backlog size due to newly admitted packets to the send buffer. The second term models the decrease in backlog size due to the transmission of segments, which is obtained by applying similar reasoning to that used to derive (2).

The time taken for a transition from state s to state s', denoted by $t_{s;s'}$, is modeled by

$$t_{s;s'} = \begin{cases} 1/f & \text{if } s \in AL, s' \in AL \\ RTT + 3/f & \text{if } s \in AL, s' \in FR \\ 2^{\min\{l-1,6\}} T_0 & \text{if } s' \in TO \\ RTT & \text{otherwise} \end{cases} \quad (5)$$

Since our model evolves at packet-level granularity while in an application-limited state (see Figure 3), the time spent in a loss-free transition from an application-limited state is the packetization interval $1/f$, as captured by the first case of (5). The time to receive a fast retransmit loss indication in an application-limited state is $RTT + 3/f$, as discussed in Section 4.2. In the absence of timeouts, the window size is updated every round-trip time [29], as shown by the last case of (5). The time spent in a sequence of timeout states increases exponentially [29], but does not exceed 64 times the base timeout value, denoted by T_0. This time duration is captured by the thrid case of (5).

5. MODEL VALIDATION

We evaluated the model using experiments in a controlled network environment and Internet experiments using PlanetLab and residential machines. We use "CBR-TCP" to denote a TCP connection with a CBR source, "FTP" for a TCP connection with bulk data transfer, and "web" for a TCP connection with HTTP traffic.

We wrote a tool that can send and receive bidirectional CBR over TCP flows with different packet sizes and different packetization (inter-sending time) intervals. To validate our model we use CBR sources with packet sizes of 174, 724 and 1448 bytes, and packetization intervals of 20 ms and 30 ms, as these choices approximately reflect typical one-way voice [31], low bit rate interactive video [15] and live video streaming [16]. The size of the packet includes a 12 byte RTP header [31] and two bytes for framing RTP packets over TCP [21]. Hence, excluding the header size, the bit

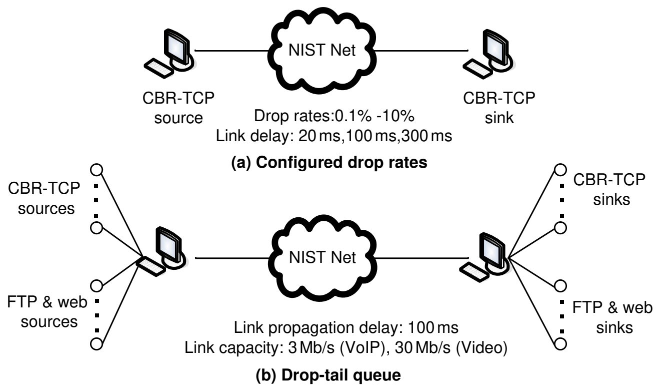

Figure 4: Experiment setup for model verification in a controlled environment.

rate of the voice flows is 64 kb/s and 42 kb/s, that of interactive video is 284 kb/s and 187 kb/s, and that of live video streaming is 573 kb/s and 378 kb/s. Unless stated otherwise, we refer to the voice flow with a bit rate of 64 kb/s as 'VoIP' and the live video streaming flow with a bit rate of 573 kb/s as 'video' flow. We abuse notation and refer to the segment loss rate in the network as the packet loss rate. These rates may be different for VoIP flows because TCP can assemble several small packets into one segment during network-limited periods.

All the experiments, except for those run in the PlanetLab environment, were conducted using Linux (kernel versions 2.6.17.8 and 2.6.9) and Windows XP machines. Both operating systems yielded similar delay performance and hence Windows XP results are not shown. PlanetLab experiments were conduced using Linux machines. The system and session-level TCP settings were determined according to the configuration described in Section 8.1.

5.1 Validation Using Configured Drop Rates

We performed the model validation on a test-bed that emulates a wide range of network settings. The topology of the test-bed is shown in Figure 4(a). We consider a single CBR-TCP flow going through a router running NIST Net [1], a network emulation program which can introduce constant delay and can drop packets according to a configured loss process. We configured NIST Net to drop packets uniformly at random irrespectively of their size.

We varied the network setting, as follows. NIST Net was configured with drop rates of 0.1%, 0.5%, 1%, 2%, 3%, 5% and 10%, and a fixed round-trip propagation delay of 20 ms, 100 ms, and 300 ms. The delay setting choice roughly reflects the delay of sites on the same coast in US, US coast-to-coast delays, and trans-continental delays [15]. Note that in this setting there were no background flows and hence there were no queuing delays on the round-trip. We do not consider loss rates greater than than 10% because the average TCP throughput (i.e., the available network bandwidth) does not satisfy the rate requirement of the CBR-TCP flow for considered RTTs. The average TCP throughput can be computed using Padhye's equation [29]. For each set of parameters, we ran the experiment for five minutes and repeated each experiment ten times. We present the average results of these experiments and compare them to the ones obtained using our model. The model was run with the assumption of random packet losses (see Appendix A.1).

Figure 5(a) and Figure 5(b) present the predicted vs. measured mean and 95th percentile TCP delay, respectively, for VoIP and video flows for various network packet loss rates and RTT of 100 ms and 300 ms. As shown, the model provides satisfactory matching for the majority of cases, specifically, when the measured delay is below 0.6 s. The delay prediction results for the flows with lower bit rates, which are presented in Figure 6 and 7, follow a similar pattern. To better see the modeling accuracy across various loss rates and RTTs, we plot the relative prediction error of the average TCP delay with respect to the actual measurement for VoIP and video flows in Figure 8. Observe that for VoIP flows, the average error is less than 10% for loss rates up to 2%. For video flows, the relative prediction error is in the order of 20% for loss rates up to 1% and 0.1%, and RTT of 100 ms and 300 ms, respectively. The increase in relative error as compared to VoIP flows is due to high variability in sender packet backlog. Video flows have a higher backlog buildup than VoIP flows since they use higher bit rates. Similar results are obtained for flows with lower bit rates, as seen in Figure 9 and 10.

The modeling mismatches are due to several reasons. First, the simplifying assumptions made by the model such as the recovery of multiple losses in a single transmission window within one round-trip time introduce error. Second, although our model accurately captures the backlog size at round-trip time granularity, it ignores backlog evolution at smaller time scales. A more detailed discussion of these issues can be found in Section 4.1 and 4.2.

Observe from Figure 8 that the prediction error increases with the network loss rate. This is because the size of the state-space of the model is truncated to reduce computational complexity, as explained in Appendix A.3. Further, for video flows, the jump in the prediction error at loss rate of 0.5% and 1% for RTT of 100 ms and 300 ms, respectively, occurs because the achievable TCP throughput is close to or below the bit rate of video flow. When the CBR rate is close to the TCP throughput, the TCP connection increasingly suffers from saw-tooth like transmission behavior, resulting in large variability in packet backlog buildup. This variability causes the high modeling error. When the throughput of the TCP connection is below the CBR rate, the CBR-TCP flow can be delayed indefinitely. For TCP throughput of at least twice the bit rate of VoIP and video flows, the modeling error was below 20%.

In general, the modeling error increases as the rate of the CBR source approaches the achievable TCP throughput. The 95th percentile measure pinpoints cases of largest deviation from the measurement, providing a highly conservative measure for the validity of the model; the average measure, demonstrates a better match between the model and the experimental results. Similar results were obtained for the variance and the maximum TCP delay measures, as seen in Figure 11, which illustrates the predicted vs. measured standard deviation of the TCP delay.

5.2 Validation Using Internet Experiments

We performed model validation using the PlanetLab environment and hosts connected to residential DSL and cable modems. We conducted the PlanetLab experiments on machines located in the US (California, New York, Texas), Europe (Germany, Italy, UK), and Asia (China, India, Japan,

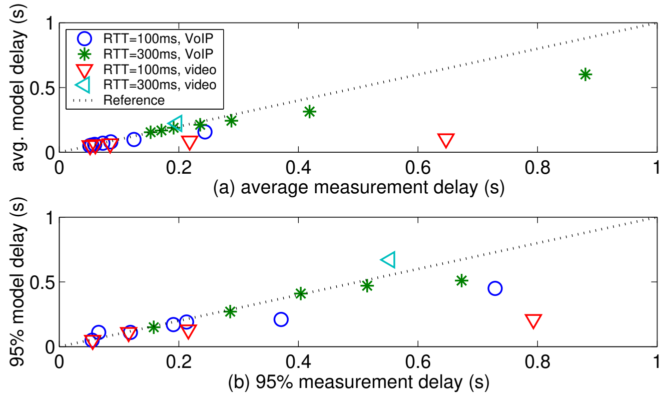

Figure 5: Predicted vs. measured (a) mean (b) 95th percentile TCP delay for VoIP and video flows with bit rates of 64 kb/s and 573 kb/s, respectively, for various loss rates and RTT of 100 ms and 300 ms.

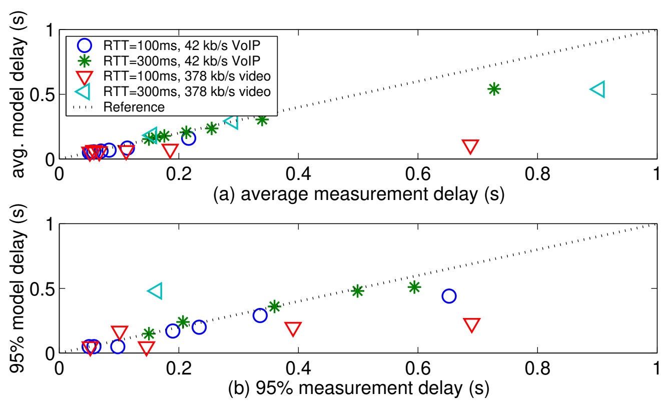

Figure 6: Predicted vs. measured (a) mean (b) 95th percentile TCP delay for VoIP and video flows with bit rates of 42 kb/s and 378 kb/s, respectively, for various loss rates and RTT of 100 ms and 300 ms.

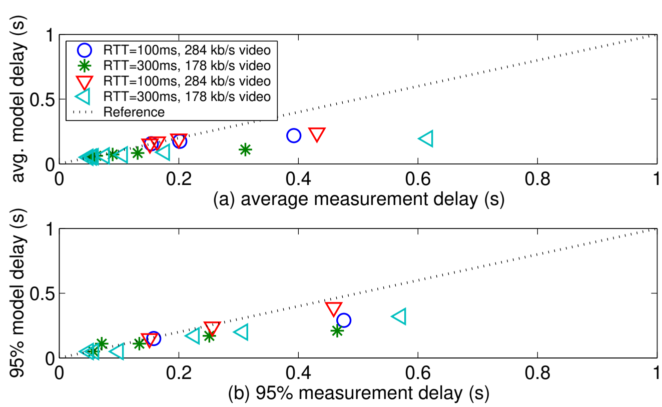

Figure 7: Predicted vs. measured (a) mean (b) 95th percentile TCP delay for interactive video flows with bit rates of 284 kb/s and 178 kb/s for various loss rates and RTT of 100 ms and 300 ms.

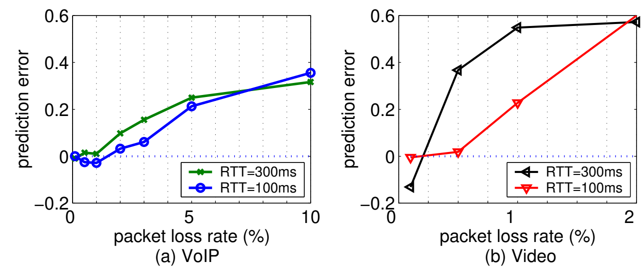

Figure 8: The relative modeling error for VoIP and video flows with bit rates of 64 kb/s and 573 kb/s, respectively, as a function of loss rate.

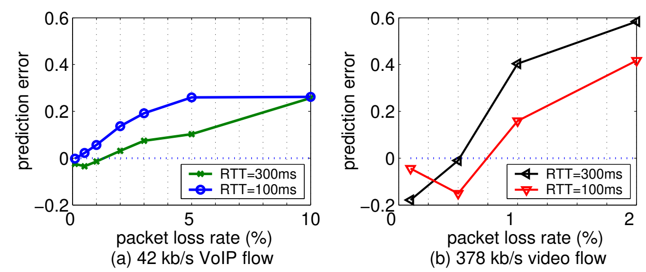

Figure 9: The relative modeling error for VoIP and video flows with bit rates of 42 kb/s and 378 kb/s, respectively, as a function of loss rate.

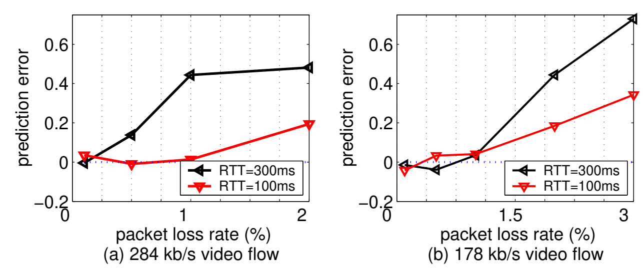

Figure 10: The relative modeling error for interactive video flows with bit rates of 284 kb/s and 178 kb/s as a function of loss rate.

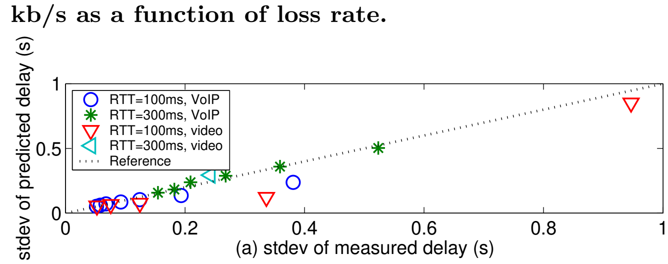

Figure 11: The predicted vs. measured standard deviation of the TCP delay for VoIP and video flows with bit rates of 64 kb/s and 573 kb/s, respectively, for various loss rates and RTT of 100 ms and 300 ms.

Taiwan). For each sender and receiver pair, we ran our tool to generate VoIP and video flows for thirty minutes. The DSL experiments were conducted from hosts in the US, Israel, and Pakistan to hosts in New York and California.

For the majority of the PlanetLab and DSL experiments we observed only a handful of losses (<0.5%), whereas in a few cases, the throughput of the TCP connection did not meet the rate requirement of the CBR-TCP flow. We therefore started multiple FTP flows in tandem with the CBR-

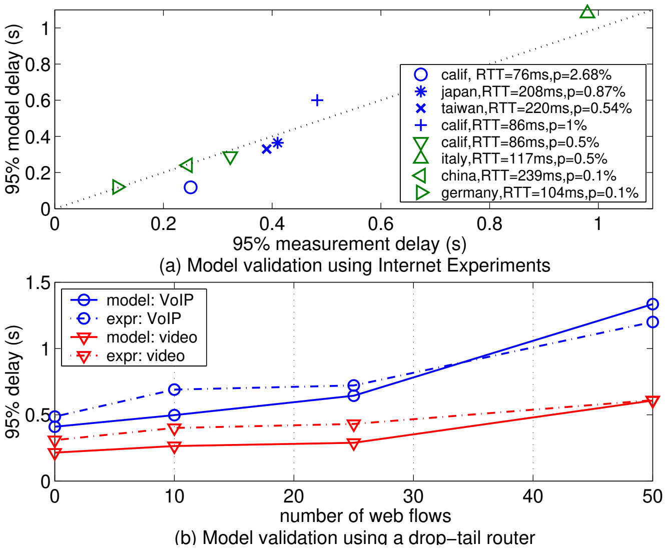

Figure 12: Model validation using (a) PlanetLab experiments (b) drop-tail router experiments for VoIP and video flows. The drop-tail queue is maintained in packets.

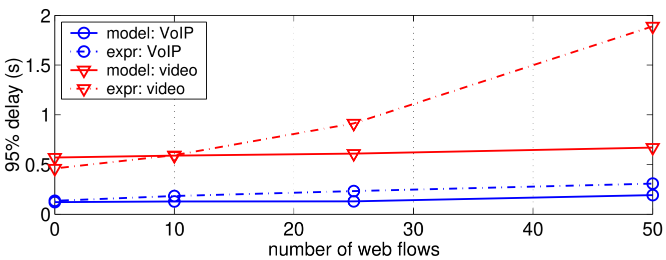

Figure 13: Model validation using drop-tail router experiments for VoIP and video flows. The drop-tail queue is maintained in bytes.

TCP flows, thereby increasing the congestion on the link and causing the CBR-TCP flows to suffer from higher losses. Figure 12(a) plots the predicted vs. measured 95th percentile delay for VoIP and video flows for a range of sites around the world. The top four settings in the legend of the figure refer to VoIP flows, and the bottom four refer to video flows. The network loss rates p and RTTs experienced by the flows are shown as well. As shown, there is a good match between the model and the measured delay.

5.3 Validation Using Drop-Tail Routers

We consider a scenario where multiple CBR-TCP flows compete with FTP and web flows for a bottleneck router with a drop-tail queueing scheme, as shown in Figure 4(b). We note that the lines in the figure connecting the data sources and sinks to the end hosts are for illustration purposes only and do not represent actual links.

The queue at the bottleneck router may be limited in packets or bytes [20]. When the queue size is maintained in bytes, small packets are less likely to be dropped than large packets, and hence TCP flows with small packets may perform better than TCP flows with large packets under the same network conditions. Since Internet router behavior is not well specified, we designed our model to support drop-tail queues in bytes and in packets, as discussed in Appendix A.1.

To validate the model, we used the test-bed from Section 5.1 and modified NIST Net to incorporate a drop-tail queue. We devised a multi-flow setting in which five VoIP CBR-TCP flows compete with five long-lived FTP and varying number of web flows. We repeated the experiment for video flows. We used the SRI and ISI traffic generator [2] to generate exponentially distributed web traffic with a mean duration of 50 ms and a constant packet size of 512 bytes. The choice of the number of FTP and web flows, and packet size for web flows was inspired by the configuration used to evaluate the performance of TFRC-small packets [14]. The round-trip propagation delay was set to 100 ms for all experiments. The link capacity was set to 3 Mb/s and 30 Mb/s for voice and video CBR-TCP flows, respectively, so that the ratio of cumulative bit rate of five CBR-TCP flows to link capacity was one to ten.

For CBR-TCP flows with small packets, the packet and byte queues were configured as 100 packets and 50,000 bytes respectively. Ideally, the byte queue size should be MSS*100 packets or 1,500,000 bytes (MSS is 1500 bytes for Ethernet [30]). However, this number overestimates the packet queue equivalent since the packet sizes of the CBR-TCP and web flows are less than one MSS. Thus, we set it to 50,000 bytes which approximately reflects the average of VoIP CBR-TCP (174-byte packets), web (512-byte packets), and FTP (1448-byte packets) flows, a choice motivated by [14]. Similarly, for TCP flows with large packets, we set the byte queue to 100,000 bytes which approximately reflects the average of video CBR-TCP (1448-byte packets), web and FTP flows. For each configuration, we ran the experiment for five minutes, repeated it five times, and present the average of the results.

We measured the network loss rate and RTT in each of the experiments to obtain the input parameters for the model. In the considered environment, the network round-trip time consists of a round-trip propagation delay of 100 ms and a queuing delay at the router. Table 3 and 4 present the average round trip time and the average network loss rate for VoIP and video flows and a packet and a byte-based queue. It also presents the mean and standard deviation of the 95th percentile TCP delay for these experiments. In the packet-based queue experiments, the measured RTT was roughly 380 ms for VoIP flows and 101 ms for video flows. Hence, the congestion was low for the video experiments. The queuing delays and loss rates are different for these two flows because the link bandwidths in these two settings were different.

Figure 12(b) presents the measured and predicted 95th percentile delay for VoIP and video flows in an environment with a packet-based queue. Some of the prediction inaccuracies in the drop-tail router experiments are caused by inaccurate characterization of the loss process. In this environment, as in the PlanetLab case, we run the model with the assumption of correlated packet losses (see Appendix A.1). However, the actual burstiness of losses experienced by the CBR-TCP flows varies depending on the level of statistical multiplexing at the router.

As seen in Table 3, the use of a byte-based queue yields lower RTTs and network loss rates compared to a packet-based queue. Consequently, TCP flows with a byte-based bottleneck experience lower delays than those with an equivalent packet-based bottleneck. The use of byte-based queues

Network loss rate, RTT, 95th percentile TCP delay and jitter for VoIP flows								
Web flows	Packet queue				Byte queue			
	loss rate (%)	RTT (s)	95% jitter (s)	95% delay (s)	loss rate (%)	RTT (s)	95% jitter (s)	95% delay (s)
0	1.02	0.317	0.027	0.410	0.20	0.172	0.014	0.135
10	1.60	0.380	0.034	0.498	0.55	0.178	0.019	0.183
15	2.31	0.371	0.041	0.644	0.81	0.180	0.022	0.233
20	3.39	0.383	0.052	1.335	1.24	0.182	0.028	0.308
25	5.61	0.385	0.089	6.173	2.41	0.185	0.040	3.057

Table 3: The network loss rate, RTT, 95th TCP percentile delay and jitter for a VoIP flow in an environment with a packet and a byte-based queue.

Network loss rate, RTT, 95th percentile TCP delay and jitter for video flows								
Web flows	Packet queue				Byte queue			
	loss rate (%)	RTT (s)	95% jitter (s)	95% delay (s)	loss rate (%)	RTT (s)	95% jitter (s)	95% delay (s)
0	0.75	0.101	0.020	0.215	1.25	0.101	0.030	0.460
10	0.81	0.101	0.018	0.265	1.28	0.101	0.032	0.594
25	0.90	0.101	0.018	0.289	1.34	0.101	0.035	0.914
50	1.20	0.101	0.022	0.608	1.79	0.101	0.046	1.890
100	2.91	0.101	0.062	4.880	3.18	0.101	0.109	5.978

Table 4: The network loss rate, RTT, 95th percentile TCP delay and jitter for a video flow in an environment with a packet and a byte-based queue.

yield higher delays for video flows than a packet-based queue because video flows, which have a packet size of 1448, compete against web traffic, which has a smaller packet size of 512 bytes. Figure 13 presents the measured and predicted 95th percentile delay for VoIP and video flows in an environment with a byte-based queue. The mismatch between the model and the measurement at 50 web flows occurs because the bit rate of video flow is very close to the maximum achievable TCP throughput, as explained in Section 5.1.

6. DISCUSSION

In this section, we explore the delay performance of real-time delivery over TCP. We experimentally characterize the working region for VoIP and live video streaming applications with bit-rates of 64 kb/s and 573 kb/s, respectively, and use our model to identify the working region for other bit-rates. Then, we study the impact of various mechanisms in TCP on its delay performance.

6.1 Working Region

Here we characterize the working region for VoIP and live video streaming applications, i.e., the conditions under which the performance of these applications is satisfactory. In general, the user perceived media quality is acceptable when the fraction of packets that arrive beyond their playout time is low and the end-to-end delay is low.

For interactive applications, ITU G.114 recommends that the worst-case one-way delay should be 400 ms. Studies show that 200 ms is an acceptable one-way delay limit for VoIP applications [26]. The choice of the delay limit for video is more flexible because people can usually tolerate a few seconds of startup delay. For the analysis we consider a 5 s startup delay, as suggested by [16]. While VoIP can usually tolerate up to 5% of packets that miss their playout deadline without a significant effect on intelligibility [26], video viewing quality drops rapidly at 0.1% [34]. We follow these guidelines and define the working region for VoIP and live video streaming as the range of network loss rates and RTTs where the 95th percentile and maximum TCP delay is at most 200 ms and 5 s, respectively. We explore how the performance varies with the delay limit in Section 6.6.

Figure 14(a) plots the 95th percentile delay for various loss rates from 0.1% to 10% and RTTs of 20 ms, 100 ms, and 300 ms for a VoIP flow with a bit-rate of 64 kb/s. The results shown were obtained empirically using the environment described in Section 5.1. Observe that when the RTT is 100 ms, the delay tolerance for VoIP is satisfied when the network loss rate is at most 2%. However, when the RTT is only 20 ms, the results indicate a tolerance of up to 5%. At the boundary of the working region, the delay added by TCP causes 5% of the packets to miss their playback deadline. Figure 14(b) plots the maximum delay for a live video streaming flow with a bit-rate of 573 kb/s. When the RTT is 100 ms, the streaming threshold is satisfied when the loss rate is at most 3%. For RTT of 300 ms, it is satisfied at a network loss rate of 0.1%. The jump in the maximum delay at a network loss rate of 0.5% and RTT of 300 ms occurs because the 5 s startup delay is no longer sufficient to completely mask TCP delays. This knee of the curve typically occurs when the achievable TCP throughput is close to the bit rate of the video flow, as explained in Section 5.1. The bit rates of 64 kb/s and 573 kb/s are the highest among the bit rates considered in Section 5 for VoIP and video

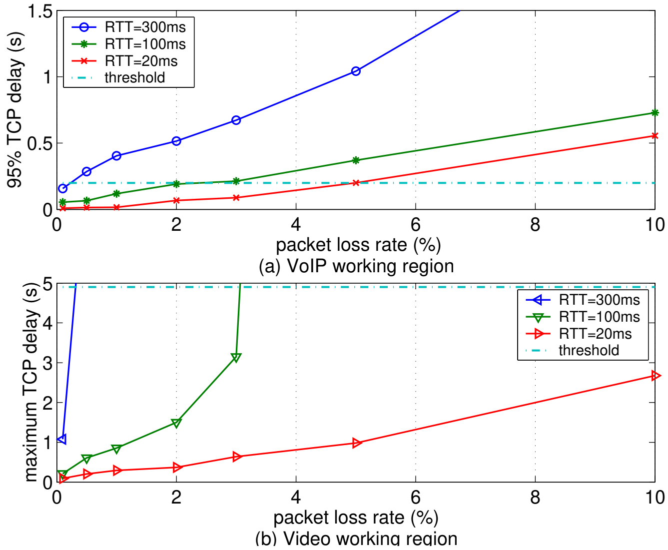

Figure 14: Working region for VoIP and video streaming with bit rates of 64 kb/s and 573 kb/s, respectively, as a function of RTT and packet loss rate.

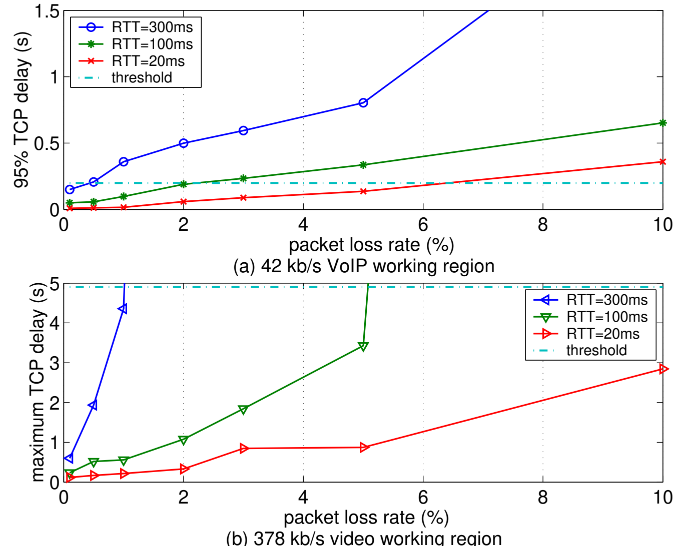

Figure 15: Working region for VoIP and video streaming with bit rates of 42 kb/s and 378 kb/s, respectively, as a function of RTT and packet loss rate.

flows and therefore, they give the most conservative estimate of the working region. We used the model to compute the working region for the other bit rates. While the working region was less constrained due to the lower bit-rates, as seen in Figure 15 and 16, the results follow similar pattern as in Figure 14. Furthermore, the working region can be significantly constrained if the application does not use delay-friendly TCP settings, as discussed in Section 8.1.

6.2 The Effect of Packet Size on Performance

Our experiments indicate that under the same network conditions, VoIP flows perform significantly better than video flows. Figure 17(a) plots the 95% delay for VoIP and video flows with the same workload in packet per second (pps) but quite different workload in terms of bits per second (kb/s), i.e., the VoIP flow has a bit rate of 64 kb/s whereas the video flow has a bit rate of 573 kb/s. The figure clearly shows a performance bias towards the VoIP flow. This happens because a video flow has a higher bit rate than a VoIP flow. Hence, during network-limited periods, a TCP sender transmitting a video flow builds up a larger packet backlog and consequently, it requires more time to drain this backlog. For VoIP flows, the TCP sender groups several queued VoIP packets into one transmission packet as permitted by the MSS. This further increases the queue drain rate, thereby reducing the queuing delay at the TCP sender.

An interesting question to ask is that among two flows having the same workload in kb/s, does TCP have a performance bias towards a flow with larger workload in pps? To address this question, we measured the delay performance of two flows having the same workload in kb/s but different workload in pps. The results are shown by the curves labeled video and video+split in Figure 17(a). Specifically, the packet rate of video+split flow is twice of the video flow but the application-level workload rate in bytes is the same. Surprisingly, there is a performance bias towards the flow with twice the packet rate of the other flow.

To illustrate the reason for this performance difference, we

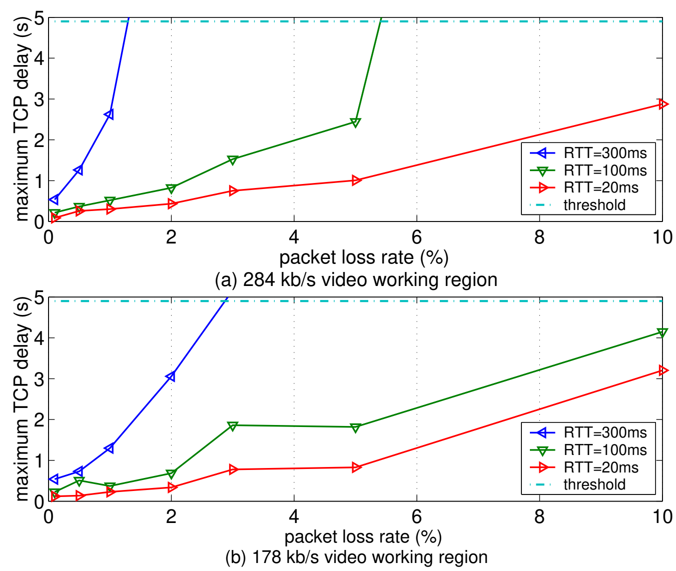

Figure 16: Working region for interactive video streaming with bit rates of 284 kb/s and 178 kb/s as a function of RTT and packet loss rate.

plot the TCP delay and congestion window size for two flows with the same application-level workload in kb/s in Figure 18. The flow in Figures 18(a) and (b) corresponds to an application that sends 100 MSS-sized packets per second. The flow in Figures 18(c) and (d) corresponds to an application that sends 200 half MSS-sized packets per second. Both flows operate over a symmetric network with 200 ms RTT and experience two close-by losses. Observe that the flow with half MSS-sized packets experiences lower delay than the other one. This happens because the AIMD mechanism updates the congestion control state as a function of the number of *packets* sent, rather than as a function of the number of *bytes* sent (see Section 3.1). Since TCP

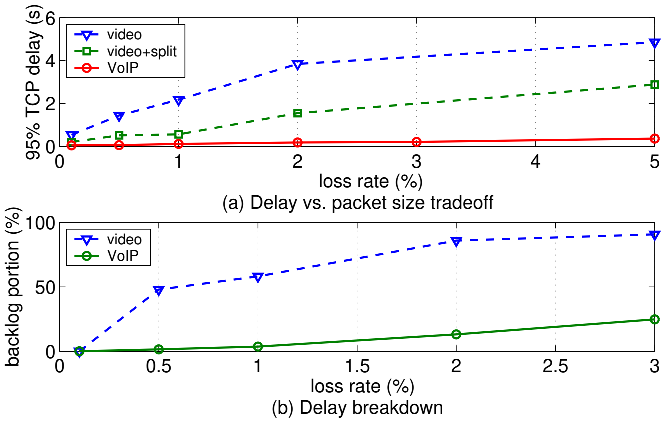

Figure 17: (a) The delay performance of two TCP flows having the same load in kb/s but different load in pps, and a VoIP flow. (b) Delay breakdown: the portion of TCP-level delays caused by the congestion control mechanism.

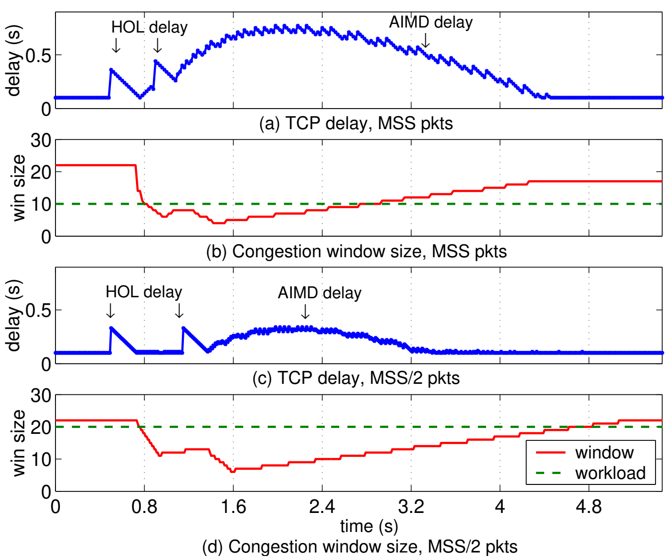

Figure 18: TCP delay and congestion window evolution for two flows with the same workload in kb/s but different packet sizes, MSS and half-MSS.

adapts its congestion control state and hence its throughput based on the number of packets sent, the magnitude of the throughput fluctuations (in bytes) is smaller for the flow with smaller packet size and higher packet rate, resulting in lower delays. For example, the peak delay of the flow with half MSS-sized packets in Figure 18 is 45% lower than that of the one with MSS-sized packets. As shown, the performance gain of a TCP flow with small packets (e.g., VoIP) comes from the reduction in the delays caused by the AIMD mechanism. That is, video flows suffer more from packet backlogging than VoIP flows. However, reducing the packet size has side effects such as increased instances of packet reordering [23].

The performance bias of TCP creates an incentive for real-time applications to improve their performance by manipulating their packet size. We discussed this issue in Section 7.1. We analyze the breakdown of TCP-level delays by computing the time packets are backlogged at the sender (i.e., the congestion control delay component) and the time it takes the TCP sender to get a packet to the receiving application (i.e., the retransmission and head-of-line delay components). Figure 17(b), shows the delay breakdown in terms of these two components for VoIP and video flows. As shown, the delays of a VoIP flow over TCP tend to be dominated by the loss recovery latency, whereas those of a video flow tend to be dominated by the delays caused by the congestion control mechanism. Similar results were obtained for CBR sources with other bit rates.

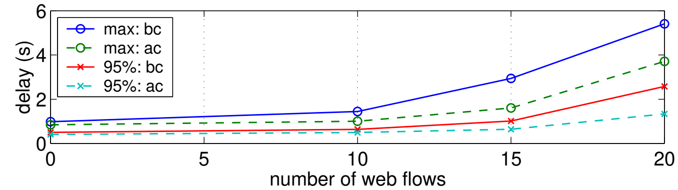

Figure 19: 95% and maximum delay for a VoIP flow using ACK and byte-counting.

6.3 Sensitivity to Byte-counting

In order to provide a measured response to ACKs that cover only small amounts of data, [3] proposes to increase the congestion window based on the number of bytes acknowledged by each incoming ACK rather than on the number of ACKs received. This mechanism is known as byte-counting. Byte-counting is configured on a per-system rather than per-connection basis in Linux, and is disabled by default. It is not implemented in Windows XP. A question arises how does the performance of VoIP flows change when TCP increases its congestion window by the number of bytes sent.

To answer this question, we measured the delay of five VoIP flows competing with five long-lived TCP flows and varying number of web flows in a drop-tail queue environment (see Section 5.3). Figure 19 shows 95th percentile and maximum delay for a VoIP flow using ACK and byte-counting. It highlights the gain of a VoIP flow when byte-counting is not used. On average, the use of byte-counting increases the TCP delay by 10-20%. The delay increases because TCP with byte-counting increases its sending rate in proportion to the number of bytes sent. Hence, a byte-counting TCP can be viewed as more fair than ACK-counting TCP with respect to the congestion control behavior. The support for byte-based congestion control mechanism must come from the underlying operating system. However, since Linux and Windows XP use ACK-counting by default, VoIP flows implicitly benefit from it.

6.4 The Effect of Timeouts on Performance

Since a real-time flow is rate-limited, it has the potential of causing the connection's congestion window to be small. Hence, the chance of sending enough segments for the receiver to generate the three duplicate ACKs becomes small, too. This can harm the delay performance as the sender may need to rely on lengthy retransmission timeouts for loss recovery. Nonetheless, our traces show that the likelihood of timeouts is low.

The likelihood of timeouts is directly effected by the behavior of TCP during application-limited periods. Accord-

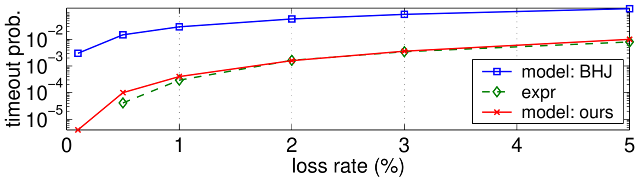

Figure 20: The timeout probability predicted by our model, Beomjoon *et al.* model (BHJ), and the measured probability for various loss rates.

ing to [18] there are three possibilities. A TCP sender can reduce the congestion window so that it would reflect the actual amount of data sent, as suggested by the window validation extension [18]. It may increase the congestion window, resulting in an arbitrarily large window value, or it may maintain the same congestion window, resulting in an invalid window value. We focus on the latter case, as it is the one observed in our measurements for Windows XP and Linux systems. The invalid congestion window overestimates the actual amount of data sent, and hence reduces the likelihood of timeouts. This overestimation happens implicitly for CBR-TCP flows, because during application-limited periods, the TCP sender retains memory of an 'inflated' congestion window used to clear the recent data backlog. The congestion window behavior can be observed in Figure 18(b) and (d). The window value overestimates the actual load by 30% and 40%, respectively, for the flow with MSS-byte and MSS/2-byte packets. The other factor that influences the likelihood of retransmission timeouts is the limited transmit mechanism [4], which is enabled by default in Linux 2.6 and Windows XP. This mechanism allows a TCP sender to send a new data segment upon the receipt of each of the first two duplicate ACKs, thereby increasing the chances of receiving the three duplicate ACKs required to trigger a fast retransmission.

To quantify the impact of these mechanisms (i.e., limited transmit and invalid congestion window) on TCP's loss recovery efficiency, we compared the timeout probability predicted by of our model to that predicted by a recent detailed loss recovery model proposed by Beomjoon *et al.* [8]. The latter model does not consider the impact of the two mechanisms. We show the results in Figure 20 for a window size of three and an overestimation factor of 30%. To validate the results we measured the timeout probability of several VoIP flows with an average window size of three in an environment with random packet losses. The figure shows that for small windows the absence of limited transmit and invalid congestion window has a non-negligible impact on the timeout probability.

6.5 The Effect of Bottleneck Limitation on Performance

While the queue at a bottleneck router may be limited in packets, in some cases it may be limited in bytes [14]. The former will give both small and large packets the same drop probability, whereas the latter will preferentially drop large packets. As Interment router behavior is not well specified [20], a question of interest is how does the delay of TCP flows affected by the network bottleneck limitation.

To address the question, we measure the 95th delay percentile of a VoIP flow in an environment with a drop-tail

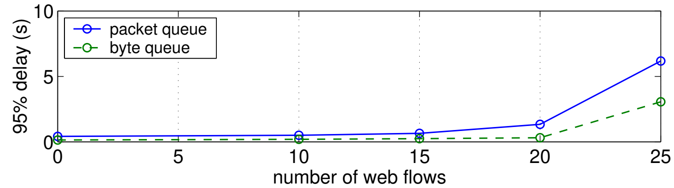

Figure 21: 95th percentile of the TCP delay in an environment with a drop-tail queue maintained in bytes and packets.

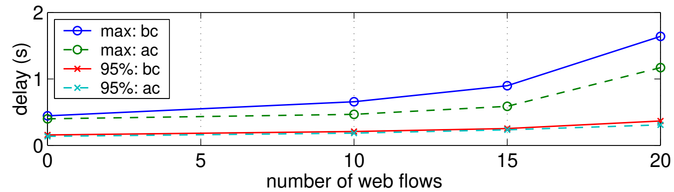

Figure 22: 95% and maximum delay for a VoIP flow using ACK and byte-counting in an environment with a drop-tail queue maintained in bytes.

queue in bytes and packets (see Section 5.3). Both queues have the same size in bytes. As shown in Figure 21, the delays experienced by a CBR-TCP flow with small packets, under a byte-based queue, are lower than those under an equivalent packet-based queue. Recall from Section 4.3 that a TCP flow with small packets dynamically varies its segment size depending on the network congestion. Consequently, it sees lower packet loss rates for a queue in bytes than a queue in packets.

We then study whether the performance bias in favor of flows with small packets is affected by TCP's congestion control implementation. Figure 22 shows the 95th percentile and the maximum delay for a VoIP flow using ACK and byte-counting in an environment with a drop-tail queue in bytes. As shown, a TCP flow with ACK-counting experiences lower delays than those of a TCP flow with byte-counting. Comparing these results to those obtained in an environment with a queue in packets (see Figure 19), we find that a byte-based queue results in lower TCP delays than a packet-based queue for both byte and ACK-counting TCP implementations.

The discussion above is summarized in Table 5. It qualitatively shows the bias in favor of TCP flows with small packets (e.g., VoIP) for different bottleneck queues and different congestion control implementations. In general, a byte-based bottleneck queue exacerbates the bias in favor of TCP flows with small packets.

6.6 Playout Buffer Size Setting

Real-time applications use a playout buffer to compensate for variable network delays. The receiving application usually delays the playout of received media packets for some time so that a large fraction of the packets is received before their scheduled playout times. A question of interest is how should an application factor in the TCP-level delays in computing the appropriate playout buffer size.

TCP is a reliable and in-order delivery protocol that deals

	Byte-based queue	Packet-based queue
Byte-based congestion control	+	None
Packet-based congestion control	+++	++

Table 5: The performance bias in favor of TCP flows with small packets. '+++' represents the largest bias and 'None' represents no bias.

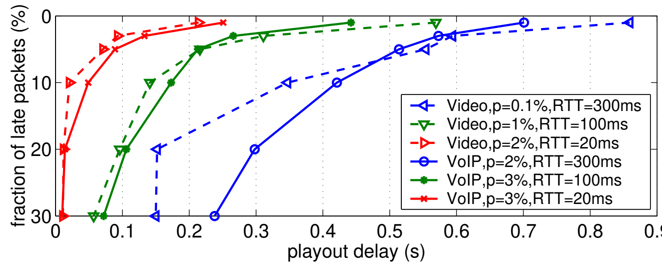

Figure 23: The required playout delay for VoIP and video flows and various RTTs and loss rates (p).

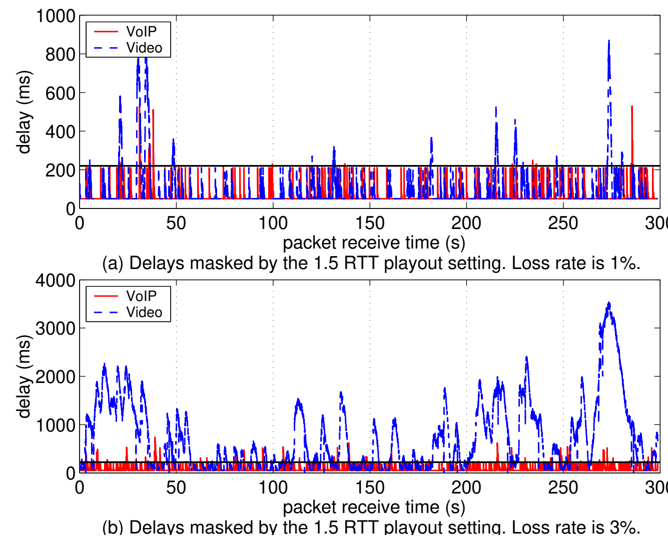

Figure 24: The TCP delays masked out by a $1.5RTT + 3/f$ playout delay for a network loss rate of (a) 1% and (b) 3%.

with packet loss by introducing delay. As explained in Section 4.2, TCP needs at least $RTT + 3/f$ to detect and recover a lost packet using a fast retransmit, where $1/f$ is the packetization interval. The time TCP needs to detect a lost packet using a retransmission timeout is at least the base timeout value. According to [17], this value can be approximated by $4RTT$. Setting the playout delay below the sum of the one-way network delay L and the minimum of these two thresholds will always cause the application to overrun its playout buffer in response to a packet loss.

We investigate the sensitively of a media application to the playout delay value by measuring the fraction of packets that miss their deadlines, which we refer to as *late packets*, as a function of the playout delay. For the analysis we consider a fixed playout delay. Figure 23 shows the results for VoIP and video streaming obtained in the network environment described in Section 5.1. The plot indicates that the application's sensitivity to the playout delay value decreases with RTT. That is, the decrease in the fraction of late packets due to an increase in the playback delay decreases with RTT. It also shows that VoIP flows require smaller playout delays than video flows for the same fraction of late packets. This happens because video flows suffer more from packet backlog than VoIP flows, as explained in Section 6.2.

In the considered environment, TCP can recover from a packet loss using a fast retransmit within $1.5RTT + 60\,ms$, because the packetization interval is 20 ms and the network is symmetric. As seen in the plot, setting the playout delay to the loss recovery latency of TCP yields up to 5% of late packets for RTTs of up to 100 ms and network loss rates of up to 2%. We then use the model to evaluate the effectiveness of this playout setting for the workings regions defined in Section 6.1. We find that a buffer with a playout delay of $L + RTT + 3/f$ can mask out TCP delays for a large portion of the VoIP working region. This setting, however, does not mask out TCP delays for the video working region, because the delay of video flows is dominated by the packet backlog rather than by the loss recovery latency. Figure 24 shows the delays masked out by the proposed playout setting for VoIP and video flows in a network with a 100 ms RTT and packet loss rates of 1% and 3%.

7. DELAY REDUCTION APPROACHES

In this section we discuss application-level heuristics that can improve the performance of real-time media applications without additional help from the network. We analyze whether the delay reduction comes at the expense of other flows, in particular long-lived FTP flows. In the following, we first discuss a packet splitting scheme and then consider the use of parallel connections. We show that both schemes are effective for video flows but have only a marginal impact on VoIP flows.

7.1 Packet Splitting

As described in Section 6.2, the congestion control mechanism of TCP results in a performance bias in favor of flows with small packets. A question of interest is whether the delay performance of real-time applications can be improved by masquerading TCP flows with large packets as flows with small packets. The application can split every large packet into a few smaller ones, while maintaining the same workload in bytes per second. We call this scheme *split-N*, where N is the packet split factor. Packet splitting, however, may also backfire: if all CBR-TCP flows started using packet splitting, the network could quickly become congested due to the TCP header overhead. Hence, a wide-scale adoption of such an approach runs the risk of degrading the performance of all flows. Further, reducing the packet size can increase instances of packet reordering [23].

We analyzed the upper bound on the delay reduction of a *split-N* scheme for both video and VoIP flows by applying our model in the environment with configured packet drops described in Section 5.1. As shown in Figure 25, the *split-2* scheme reduced the 95th delay percentile by 60%

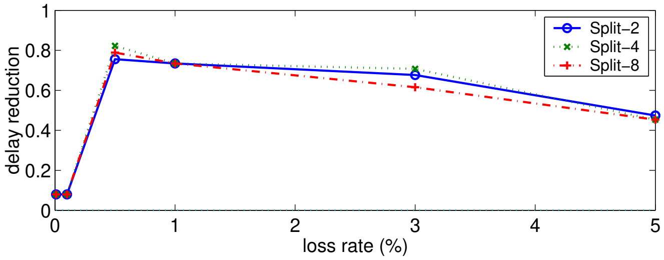

Figure 25: The reduction in the 95th delay percentile of a video flow using *split-N* in an environment with configured packet drops.

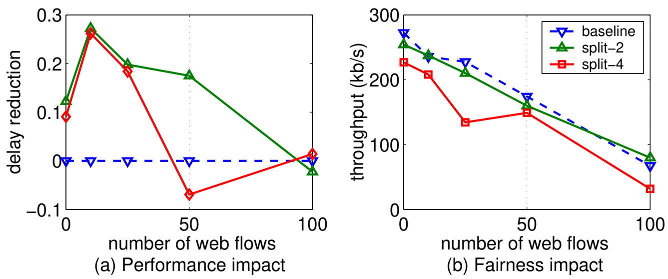

Figure 26: (a) The reduction in the 95th delay percentile of a video flow using *split-N* in a drop-tail queue environment (b) Throughput of background FTP flows in the same environment.

on average. This is consistent with the observation made in Section 6.2 that a TCP flow with small packets experiences smaller packet backlogs, and hence smaller delays, than that of a flow with large packets. For VoIP flows, the scheme yielded diminishing gains due to the low backlog levels experienced by these flows.

To understand the performance of *split-N* in a wide-scale deployment, we measured the delay of a video flow (i.e., a 573 kb/s video source) in an environment with a drop-tail queue described in Section 5.3. As shown in Figure 26(a), the *split-2* scheme reduces TCP delay by up to 30% under low and moderate loss rates, whereas schemes with higher split factors yield diminishing gains or perform even worse than a no-split scheme. The performance degradation is partially due to the increase in the burstiness of the flow with packet splitting. This burstiness can be reduced to some extent by evenly spacing split-packets over the packetization interval. However, perfect pacing may be difficult to achieve at the application layer due to the small packetization intervals (e.g., 20 ms) used in practice.

During periods of high congestion (100 web flows), a TCP sender using a *split-N* scheme is heavily backlogged and hence it is unable to obtain a performance bias by using a *split-N* backlogged scheme. We used the drop-tail queue environment to study the fairness implications of this scheme. In particular, we measured the throughput of long-lived TCP flows that share a congested link with video flows employing packet splitting. As shown in Figure 26(b), the *split-N* scheme impacts the throughput of the long-lived TCP flows. For example, the use of *split-4* reduces the throughput of a background TCP flow by 27% on average. From the plot, we observe that the throughput reduction quickly increases with the split factor.

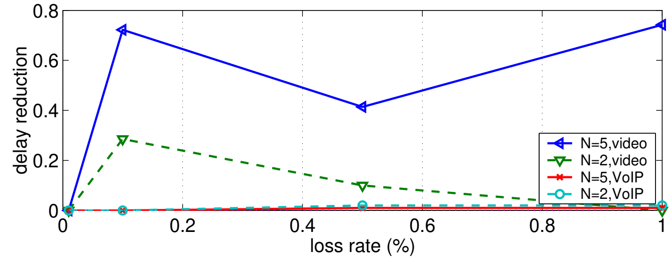

Figure 27: The reduction in the 95th delay percentile for VoIP and video flows using a 'blind' scheme with N connections in an environment with configured packet drops.

7.2 Parallel Connections

A straightforward approach to improve the delay performance of a CBR-TCP flow is to stripe its load across parallel TCP connections. The idea is that several TCP streams are more aggressive than one TCP stream with respect to the congestion control behavior [35], which can result in lower TCP delays. The use of parallel TCP connections for streaming and data-intensive applications has mainly focused on enhancing the throughput. However, we focus on reducing the delay. Specifically, we provide insights on the delay performance of parallel connection schemes for real-time applications.

Packet striping can be done in delay-agnostic or delay-aware fashion. The simplest approach is to use a delay-agnostic ('blind') parallel connection scheme that sends packets over parallel TCP connections in a round-robin fashion. We study the performance of the 'blind' scheme for VoIP and video flows by applying the model in the environment with configured packet drops described in Section 5.1. Figure 27 shows the delay reduction of the 'blind' scheme. The plot indicates that five parallel connections reduce the 95th delay percentile by 60% on average. As shown, the performance gain was negligible for VoIP flows. The gain was negligible because the delay reduction is offset by the decrease in TCP's loss recovery efficiency caused by small congestion windows (see Section 6.4). The small congestion windows are due to the low load per connection. We then empirically study the performance improvement of this scheme for video flows in the drop-tail queue environment described in Section 5.3. Figure 28(a) shows that five parallel connections reduce the 95th delay percentile by 90% on average. The delay reduction stems from lowering the load per connection, which in turn reduces sender backlog buildup and receiver head-of-line blocking per connection, and hence the TCP delay. We note that the scheme yielded diminishing gains when more than five connections were used.

We propose a delay-aware ('intelligent') scheme which selects a connection for packet transmission that has the smallest TCP send queue and is not in the timeout state and show the results in Figure 28(a). The 'intelligent' scheme outperforms the 'blind' scheme because it dynamically avoids connections with large queues and in timeout states. Further, due to its dynamic nature, this scheme copes better with connections with small congestion windows. Similar to the 'blind' scheme, we observe that using more than five parallel connections results in diminishing gains. We note that the parallelization spectrum ranges from a single flow

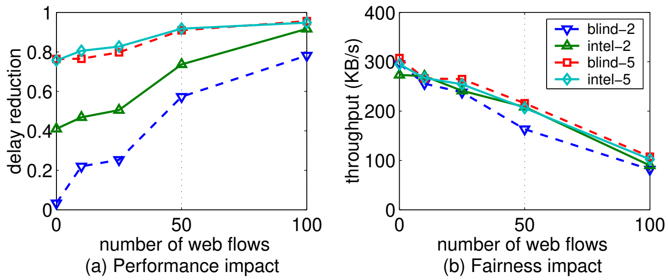

Figure 28: (a) The reduction in the 95th delay percentile for a 'blind' and an 'intelligent' scheme with N connections in a drop-tail queue environment (b) Throughput of background FTP flows in the same environment.

to having as many flows as the packet rate per RTT. Similar to packet splitting, we study the fairness impact of these schemes on the background traffic using a drop-tail queue environment. We present the results in Figure 28(b). As shown, both 'intelligent' and 'blind' schemes have a negligible impact on the throughput of the background long-lived FTP flows. The impact is negligible because these schemes do not introduce additional traffic besides session setup and teardown. Though parallel TCP streams are more aggressive than a single TCP stream, their aggregated throughput is still limited by the rate of the CBR source.

8. DELAY-FRIENDLY GUIDELINES

We present delay-friendly guidelines for TCP-based VoIP and live video streaming applications. We categorize them into TCP and OS-level guidelines and application-level guidelines. In the following, we first provide a comprehensive set of guidelines for setting TCP and OS parameters. While several settings such as disabling Nagle's algorithm and using large receive buffers are common practices in delay-sensitive applications, the impact of others, specifically, window validation, byte counting and limited transmit is less obvious. We then provide guidelines for setting application-level parameters.

8.1 TCP and OS-level guidelines

- As discussed in Section 3, Nagle's algorithm should be disabled as it introduces transmission delays at the TCP sender.

- CBR-TCP applications should set a large receive buffer and operate with non-blocking sockets so that the transmission of TCP is not limited by its flow control mechanism.

- To increase the loss efficiency of TCP, SACK should be enabled [10] and limited transmit be used. The latter is primarily applicable for a TCP connection with small windows.

- Congestion window validation during application-limited periods, and byte-counting should be disabled. These settings are disabled by default on Linux and Windows XP systems.

- The initial window size should be set to four segments as it can remove delays up to three RTTs and a timeout during the initial slow-start period [5]. Some operating systems (e.g., Linux) inherit the *ssthresh* value from last connection for the current connection. This may cause the current connection to enter congestion avoidance earlier even if it does not suffer from any loss, thereby incurring additional start-up delay during initial slow-start. Therefore, we suggest that *ssthresh* inheritance be disabled.

As a general rule, the above TCP-level configurations should be set on a per-connection basis if the underlying operating system supports it. Note that these guidelines do not require any change in the TCP stack.

8.2 Application-Level guidelines

- **Playout buffer setting.** As discussed in Section 6.6, a CBR-TCP application with small packets can statically set the playout delay to the loss recovery latency $L+RTT+3/f$. This setting masks out TCP delay variations for a wide range of environments, in particular, for a large portion of the VoIP working region.

- **Video flows.** The delay performance of video flows can be improved by parallelizing the flow over multiple connections. If supported by the underlying operating system (e.g., Linux), an application should preferentially send packets over the connection that has the smallest send buffer size and is not in a timeout state. Splitting MSS-sized packets by half reduces delay for video flows if a single connection is preferred.

- **Idle periods.** VoIP over TCP may have somewhat different properties than CBR over TCP. In particular, VoIP has idle periods, requiring TCP to ramp back up to the old sending rate after idle periods. We therefore suggest that an application should continue sending minimal traffic during idle periods (e.g., 3 packets per round-trip time, as suggested by [24]) to avoid having very small congestion windows. The size of the packets sent during silence periods can be as low as one-byte if the TCP implementation uses ACK-counting. This is because an ACK-counting TCP increases the congestion window size based on the number of packets sent rather than the number of bytes sent.

- **MSS to packet size ratio.** We recommend that the ratio of MSS to packet size should be an integer. This setting insures that the packet assembly capability of TCP will give the lowest possible sending rate in packets per second when the TCP sender is backlogged. The lower sending rate results in lower delays compared to a flow with an equivalent load in kb/s and a non integer MSS to packet size ratio.

- **Proactive packet drop.** By examining the size of the TCP send buffer, an application can potentially infer the delays a new packet will experience. Thus, it may choose to drop packets at the sender, if the buffer size crosses a certain inferred delay threshold.

- **In-order delivery mechanism.** As described in Section 6.2, the delays of TCP flows with small packets tend to be dominated by the in-order delivery mechanism of TCP. That is, packets are held in the receive buffer while waiting for a lost packet(s) to arrive. A potential modification to the TCP operating system API can allow the application to peek into its receive buffer and extract out-of-order packets. A similar approach has been proposed by [22]. Although this modification requires changing the receive API to receive our-of-order packets, it does not change the network semantics of TCP.

9. RELATED WORK

There is an extensive literature on analytical and experimental evaluation of TCP. We present only those studies closely related to ours and refer the reader to [28] for a comprehensive survey of TCP modeling. The majority of TCP modeling studies are geared towards file transfers assuming either persistent [29] or short-lived flows [9]. Our work differs from past work in that we consider non-greedy rate-limited flows with real-time delivery constraints. More recently, the performance of TCP-based video streaming has been analytically analyzed by [34]. The receive buffer size requirement for TCP streaming has been determined in [19]. These papers combine TCP throughput and application-layer buffering models to compute the portion of late packets, whereas we directly model the transport-layer delay of TCP. Our work further differs from those above in that we consider applications with tight delay constraints such as VoIP. Wang *et al.* [35] have performed a comprehensive analytical study of the performance of multipath video streaming using TCP. This work explains that the performance of TCP streaming increases as the ratio of the aggregated TCP throughput to video encoding rate increases. However, our contribution is the insights on the TCP delay performance.

Goel *et al.* [15] present an empirical study of kernel-level TCP enhancements to reduce the delays induced by congestion-control for streaming flows. The performance of TCP for real-time flows has also been considered by [22, 25]. However, unlike our study, these papers propose a modification to the TCP stack. Application-layer heuristics for improving the loss recovery latency of TCP have been suggested [24]. These heuristics are geared towards bursty traffic flows and hence may not be effective for real-time flows.

10. CONCLUSION AND FUTURE WORK

We have presented a Markov-chain TCP delay model for CBR-TCP flows. The model captures the behavior of VoIP and streaming flows. We used experiments and the model to derive the working region of these flows. We verified the model in a test-bed and in PlanetLab. We explored the impact of TCP mechanisms and presented guidelines for improving the delay friendliness of CBR-TCP applications. The delay performance of a video flow can be improved using packet splitting or parallel connection heuristics.

This study provides insights on the use of TCP for VoIP and live-video streaming applications. A direct comparison of real-time delivery over TCP versus unreliable protocols is left for future work. We have used delay percentiles to evaluate the performance of CBR-TCP flows. However, Mean Opinion Score (MOS) is considered a better metric for evaluating user-perceived performance. This is another potential topic for future work.

11. ACKNOWLEDGEMENTS

We especially thank Sally Floyd for her detailed and valuable feedback on the initial technical report. We would also like to thank the anonymous reviewers, Oren Laadan, and Jitendra Padhye for their valuable feedback on the paper. Finally, we sincerely thank our shepherd Martin Arlitt whose guidance helped us refine the final version.

12. REFERENCES

[1] NIST Net. http://www-x.antd.nist.gov/nistnet/.

[2] SRI and ISI traffic generator. http://www.postel.org/tg/tg.html.

[3] M. Allman. TCP Congestion Control with Appropriate Byte Counting (ABC). RFC 3465 (Experimental), February 2003.

[4] M. Allman, H. Balakrishnan, and S. Floyd. Enhancing TCP's Loss Recovery Using Limited Transmit. RFC 3042, January 2001.

[5] M. Allman, S. Floyd, and C. Patridge. Increasing TCP's Initial Window. RFC 3390, October 2002.

[6] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, April 1999.

[7] S. A. Baset and H. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *IEEE INFOCOM*, Barcelona, Spain, April 2006.

[8] K. Beomjoon, C. Yong-Hoon, and L. Jaiyong. An Extended Model for TCP Loss Recovery Latency with Random Packet Losses. *IEICE Transactions on Communications*, 89(1):28–37, January 2006.

[9] N. Cardwell, S. Savage, and T. Anderson. Modeling TCP Latency. In *IEEE INFOCOM*, Tel-Aviv, Israel, March 2000.

[10] K. Chen, C. Huang, P. Huang, and C. Lei. An Empirical Evaluation of TCP Performance in Online Games. In *ACM SIGCHI*, Montréal, Canada, April 2006.

[11] K. Chen, C. Huang, P. Huang, and C. Lei. Quantifying Skype User Satisfaction. In *SIGCOMM*, Pisa, Italy, September 2006.

[12] S. Floyd, T. Henderson, and A. Gurtov. The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 3782, April 2004.

[13] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, 1993.

[14] S. Floyd and E. Kohler. TCP Friendly Rate Control (TFRC): The Small-Packet (SP) Variant. RFC 4828 (Experimental), April 2007.

[15] A. Goel, C. Krasic, K. Li, and J. Walpole. Supporting Low-Latency TCP Based Media Streams. In *IWQoS*, Miami, Florida, USA, May 2002.

[16] L. Guo, E. Tan, S. Chen, Z. Xiao, O. Spatscheck, and X. Zhang. Delving into Internet Streaming Media Delivery: a Quality and Resource Utilization Perspective. In *IMC*, Rio de Janeiro, Brazil, October 2006.

[17] M. Handley, S. Floyd, J. Padhye, and J. Widmer. TCP Friendly Rate Control (TFRC): Protocol Specification. RFC 3448, January 2003.

[18] M. Handley, J. Padhye, and S. Floyd. TCP Congestion Window Validation. RFC 2861

(Experimental), June 2000.
[19] T. Kim and M. H. Ammar. Receiver Buffer Requirement for Video Streaming over TCP. In *Proceedings of SPIE*, San Jose, CA, USA, January 2006.
[20] E. Kohler, M. Handley, and S. Floyd. Designing DCCP: Congestion Control Without Reliability. In *SIGCOMM*, Pisa, Italy, September 2006.
[21] J. Lazzaro. Framing Real-time Transport Protocol (RTP) and RTP Control Protocol (RTCP) Packets over Connection-Oriented Transport. RFC 4571, July 2006.
[22] D. McCreary, K. Li, S. A. Watterson, and D. K. Lowenthal. TCP-RC: A Receiver-Centered TCP Protocol for Delay-Sensitive Applications. In *MMCN*, San Jose, California, USA, January 2005.
[23] A. Medina, M. Allman, and S. Floyd. Measuring the Evolution of Transport Protocols in the Internet. *SIGCOMM CCR*, 35(2):37–52, April 2005.
[24] A. Mondal and A. Kuzmanovic. When TCP Friendliness Becomes Harmful. In *IEEE INFOCOM*, Anchorage, Alaska, USA, May 2007.
[25] B. Mukherjee and T. Brecht. Time-lined TCP for the TCP-friendly Delivery of Streaming Media. In *ICNP*, Osaka, Japan, November 2000.
[26] S. Na and S. Yoo. Allowable propagation delay for voip calls of acceptable quality. In *AISA*, London, UK, August 2002.
[27] J. Nagle. Congestion Control in IP/TCP Internetworks. RFC 896, January 1984.
[28] J. Olsen. *Stochastic Modeling and Simulation of the TCP Protocol*. PhD thesis, Uppsala University, October 2003.
[29] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *SIGCOMM*, Vancouver, British Columbia, Canada, September 1998.
[30] J. Postel. The TCP Maximum Segment Size and Related Topics. RFC 879, November 1983.
[31] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550, July 2003.
[32] W. R. Stevens. *TCP/IP Illustrated*, volume 1. Addison-Wesley, MA, November 1994.
[33] S. Tao, J. Apostolopoulos, and R. Gu. Real-time Monitoring of Video Quality in IP Networks. In *NOSSDAV*, Stevenson, Washington, USA, June 2005.
[34] B. Wang, J. Kurose, P. Shenoy, and D. Towsley. Multimedia Streaming via TCP: An Analytic Performance Study. In *ACM Multimedia*, New York, NY, USA, October 2004.
[35] B. Wang, W. Wei, and D. Towsley. Multipath Live Streaming via TCP: Scheme, Performance and Benefits. In *CoNEXT*, New York, NY, USA, December 2007.
[36] J. Widmer, C. Boutremans, and J. L. Boudec. End-to-end congestion control for tcp-friendly flows with variable packet size. *SIGCOMM Comput. Commun. Rev.*, 34(2):137–151, 2004.
[37] A. Wierman, T. Osogami, and J. Olsen. A Unified Framework for Modeling TCP-Vegas, TCP-SACK, and TCP-Reno. In *MASCOTS*, Orlando, Florida, USA, October 2003.
[38] R. W. Wolff. *Stochastic Modeling and theory of Queues*. Prentice-Hall, New York, 1989.
[39] X. Zhang and H. Schulzrinne. Voice over TCP and UDP. Technical Report, CUCS-033-04, Department of Computer Science, Columbia University, September 2004.

APPENDIX

A. MODEL FOR A REAL-TIME TCP FLOW

We model a TCP connection with a CBR source using a finite state discrete-time Markov chain. Each state represents a triple, (w, b, l), where w is the current congestion window size, b is the current backlog size, and l is either 1 or 0 depending on whether any loss needs to be recovered in the current state ($l = 1$), or not ($l = 0$). Let $q_{(w,b,l);(w',b',l')}$ be the probability associated with a transition from state $s = (w, b, l)$ to state $s' = (w', b', l')$. Let $d_{(w,b,l);(w',b',l')}$ be the delays associated with packets sent in this transition. Packet delays are represents by an ordered list denoted as $(d_i)_{i=1}^n$, where d_i denotes the ith member of the list and n denotes its length. Let T_0 be the duration of the initial timeout. The state transition probabilities and the delays associated with the transitions are given in Table 6. The states in this table are grouped into four categories that correspond, respectively, to the states of a TCP connection (a) application-limited (b) network-limited and loss-free (c) fast recovery (d) and retransmission timeout. For ease of presentation, we use $(x)^+ \triangleq \max(0, x)$ and

$$\{w+1|2w\} \triangleq \begin{cases} w+1 & \text{if } r/2 < w,\ 0 < b' \\ 2w & \text{if } w \le r/2,\ 0 < b' \end{cases}$$

.

A.1 Transition Probabilities of the Markov Chain

In our model, each state is associated with at most three outgoing transitions representing the following events: the receipt of a fast retransmit (fast recovery) loss indication, the receipt of a timeout loss indication, and successful delivery of window data. We support a wide range of network environments by considering both correlated and random segment loss models, typical for RED [13] and FIFO drop-tail routers, respectively.

In the random segment loss model, each segment is dropped uniformly with a fixed probability p. We determine the transition probabilities for the Markov chain using the timeout and fast retransmit probabilities for TCP-SACK given in [37]. The probabilities in [37], however, cannot be used to determine the transition probabilities from application-limited states, because the model evolves at packet-level granularity rather than at window-level granularity while the sender is application-limited. We derive these transition probabilities by repeating the loss recovery analysis in [37] under the assumption that the first segment in a transmission window is always lost. That is, we replace $P_w(i)$, the probability of i losses out of a window of w segments, in the probability expressions for fast recovery and timeout $P^S_{fr/fr}(w)$ and $P^S_{to}(w)$, respectively, with $pP_{w-1}(i-1)$, the probability of i losses given that the first segment is lost. The resulting transition probabilities are shown in Table 6.

$q_{(w,0,0);(w,0,0)}$	$= 1 - p$	$r \leq w$
$d_{(w,0,0);(w,0,0)}$	$= L$	$r \leq w$
$q_{(w,0,0);(\lfloor (r+3)/2 \rfloor,3,1)}$	$= \sum_{i=1}^{w-3} \binom{w-1}{i-1} p^i (1-p)^w$	$r \leq w$
$d_{(w,0,0);(\lfloor (r+3)/2 \rfloor,3,1)}$	$= (L + RTT + i/f)_{i=1}^{3+w}$	$r \leq w$
$q_{(w,0,0);(0,T_0 fa,1)}$	$= p - \sum_{i=1}^{w-3} \binom{w-1}{i-1} p^i (1-p)^w$	$r \leq w$
$q_{(w,b,0);(\{w+1 \mid 2w\},(b+RTTfa-wMSS)^+,0)}$	$= (1-p)^w$	$0 < w, 0 < b$ or $r \leq w, b = 0$
$d_{(w,b,0);(\{w+1,2w\},(b+RTTfa-wMSS)^+,0)}$	$= (L + b/(fa))_{i=1}^{\lfloor \min\{b,wMSS\}/a \rfloor}$	$0 < w, 0 < b$ or $r \leq w, b = 0$
$q_{(w,b,0);(\lfloor (w+3)/2 \rfloor,(b+RTTfa-(w+3)MSS)^+,1)}$	$= \sum_{i=1}^{w-3} \binom{w}{i} p^i (1-p)^w$	$0 < w, 0 < b$ or $r \leq w, b = 0$
$d_{(w,b,0);(\lfloor (w+3)/2 \rfloor,(b+RTTfa-(w+3)MSS)^+,1)}$	$= (L + RTT + b/(fa) + (3+i)/f)_{i=1}^{\lfloor \min\{b,(w+3)MSS\}/a \rfloor}$	$0 < w, 0 < b$ or $r \leq w, b = 0$
$q_{(w,b,0);(0,b+T_0 fa,1)}$	$= 1 - (1-p)^w - \sum_{i=1}^{w-3} \binom{w}{i} p^i (1-p)^w$	$0 < w, 0 < b$ or $r \leq w, b = 0$
$q_{(w,b,1);(w,(b+RTTfa-wMSS)^+,0)}$	$= 1$	$0 < w$
$d_{(w,b,1);(w,(b+RTTfa-wMSS)^+,0)}$	$= (L + b/(fa))_{i=1}^{\lfloor \min\{b,wMSS\}/a \rfloor}$	$0 < w$
$q_{(0,b,l);(1,(b+RTTfa-MSS)^+,0)}$	$= 1 - p$	$1 \leq l \leq 6$
$d_{(0,b,l);(1,(b+RTTfa-MSS)^+,0)}$	$= (L + b/(fa))_{i=1}^{\lfloor \min\{b,MSS\}/a \rfloor}$	$1 \leq l \leq 6$
$q_{(0,b,l);(1,b+2^l T_0 fa,\min\{l+1,6\})}$	$= p$	$1 \leq l \leq 6$

Table 6: TCP delay model: definition of the state transition probabilities and the delays associated with the transitions.

We use similar approach to derive the transition probabilities under the correlated segment loss model. In this model, if a segment is lost, so are all the following segments sent within the same window. We determine the corresponding transition probabilities using the probability expressions in [29]. As in the random loss case, we derive the transition probabilities from application-limited states by replacing $A(w,k)$, the probability that k segments are successfully transmitted in a round, with $pA(w-1,k-1)$, in the timeout probability expression in equation (22) of [29].

So far we have assumed that the bottleneck router manages its flow in packets so that both small and large packets have the same drop probability. However, If the bottleneck router manages its flow in bytes, then small packets are less likely to be dropped. In this environment (e.g., a drop-tail queue in units of bytes), TCP flows with small packets would experience lower delays than those with large packets. Since Internet router behavior is not well specified (see [20]), we extend our model to support a drop-tail queue in bytes. We assume that a small segment of size a is A/a times less likely to be dropped than a large segment of size A, as suggested by [36]. We derive the transition probabilities of the Markov chain for a byte limited environment by replacing the original segment loss probability p in Table 6 by the scaled loss probability $p\frac{a}{M_s}$, where M_s is the size of a segment transmitted in a transition from state s. As described in Section 4.3, we let $M_s = a$ if $s \in AL$ and $M_s = MSS$ otherwise.

A.2 Modeling Limited Transmit

The limited transmit mechanism [4] allows the TCP sender to transmit a new data segment upon the receipt of each of the first two duplicate ACK, thereby increasing the chance of receiving the three duplicate ACKs needed to trigger a fast retransmission.

We obtain the Markov chain for a TCP source with limited transmit by modifying the transition probabilities given in Table 6. The transition probability from state s to a fast retransmit state $s' \in FR$ is modified to

$$q^{LT}_{s;s'} = P_w(w-1)(1-p)^{w-1}(1-p)^2 I_{w>1} + P_w(w-2)(1-p)^{w-2}(1-p^2) I_{w>2} + q_{s,s'} I_{w>3} \quad (6)$$

where $P_w(i) = \binom{w}{i} p^i (1-p)^{w-i}$ is the probability of i losses out of a window of w segments and I is the indicator function. The first term in (7) corresponds the case where a single duplicate ACK is received for a single transmission window and it leads to two successful segment transmissions. The second term corresponds to the case where two duplicate ACKs are received for a single transmission window and they lead to at leat one successful segment transmission. The probability for a transition from state s to a timeout state $s'' \in TO$ is modified to $p - q^{LT}_{s;s'}$ if $s \in AL$ and to $1 - (1-p)^w - q^{LT}_{s;s'}$ otherwise.

We apply a similar approach to derive the transition probabilities for a TCP source with limited transmit and correlated segment losses. That is, we modify the transition probabilities from state s to a timeout state $s' \in TO$ to

$$q^{LT}_{s;s'} = A(w,1) p I_{w>1} + A(w,2)(p + (1-p)p) I_{w>2} + q_{s,s'} I_{w>3} \quad (7)$$

where $A(w,k)$ is the probability that k segments are successfully transmitted in a round, which is given by equation (22) in [29], and $q_{s;s'}$ stands for the transition probability for a TCP source under the correlated loss model, which is described in Section A.1. The probability for a transition from state s to a fast recovery state $s'' \in FR$ is modified to $p - q^{LT}_{s;s'}$ if $s \in AL$ and to $1 - (1-p)^w - q^{LT}_{s;s'}$ otherwise.

A.3 Computation Complexity

The complexity of solving the Markov chain is directly affected by the size of the state space. Let w_m and b_m be the maximum supported congestion window size and backlog size, respectively. Since we have six back-off timeout states and w_m non-timeout states with unique window sizes, the total number of states is $6b_m + 2b_m w_m$. For the analysis we use $w_m = 8r$ and $b_m = 4T_0 f = 16r$. This state space is small enough to allow us to efficiently evaluate the TCP delay for the range of network environments considered in Section 5.