The Delay-Friendliness of TCP

Eli Brosh*, Salman A. Baset*, Vishal Misra*, Dan Rubenstein† and Henning Schulzrinne*
*Department of Computer Science, Columbia University, NY, USA
Emails: {salman,elibrosh,misra,hgs}@cs.columbia.edu
†Department of Electrical Engineering, Columbia University, NY, USA
Email: danr@ee.columbia.edu

***Abstract*—Traditionally, TCP has been considered unfriendly for real-time applications. Nonetheless, popular applications such as Skype use TCP due to the deployment of NATs and firewalls that prevent UDP traffic. This observation motivated us to study the delay performance of TCP for real-time media flows using an analytical model and experiments. The results obtained yield the working region for VoIP and live video streaming applications and provide guidelines for delay-friendly TCP settings. Further, our research indicates that simple application-level schemes, such as packet splitting and parallel connections, can significantly improve the delay performance of real-time TCP flows.**

I. 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 may be 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 UDP-based transport protocols [1]–[3] 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 [4]. Popular media applications such as Skype [5] and Windows Media Services [4] use TCP due to the wide-deployment of NATs and firewalls that often prevent UDP traffic. Furthermore, TCP is by definition TCP-friendly [2], 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 explore 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 these applications, we conduct an extensive performance study using both analytical models and real-world experiments. The analytical models allow us to systematically explore the delay performance over a wide range of parameter settings, a challenging process when relying on experimentation alone. The extensive literature on TCP modeling and analysis is geared towards file transfers assuming either persistent [6], [7] or short-lived flows [8]; and more recently towards video streaming [9], [10]. To the best of our knowledge, we are the first delay-based analytical study of TCP for real-time flows.

We use both test-bed and Internet experiments to validate the models and to measure the TCP delay over a wide range of paths in the Internet. 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 [11] and limited transmit [12]. The results obtained yield guidelines for delay-friendly TCP settings and may further be used to compare the performance of TCP with alternative protocols [2], [3] and experimental real-time enhancements for TCP [13]–[15]. We analyze two application-level schemes, namely, packet-splitting and parallel connections that can significantly improve the performance of a real-time media application.

Our research reveals that real-time applications 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 results in reduced sensitivity to timeouts and an improvement in the delay performance.

The main contributions of this paper are:

- We present a Markov model for the delay distribution of a real-time TCP flow (Section III-B).
- We predict the working region for VoIP and live streaming flows based on our model. VoIP operates well when the packet drop rate is below 2% and the RTT is lower than 200 ms. Live streaming operates well when the drop rate is under 10% (Section V).
- We show that the delay added by TCP is on the order of the network round-trip time when the flow's data rate is roughly one third of the fair-TCP rate (Section V-A).
- We develop guidelines and simple application-level heuristics for improving the performance of TCP-based real-time applications. The most promising heuristic uses parallel-connections with shortest-queue first policy and achieves up to 90% delay improvement. For a single connection, a packet-splitting heuristic improves the delay performance by 30% on average (Section VI, VII).

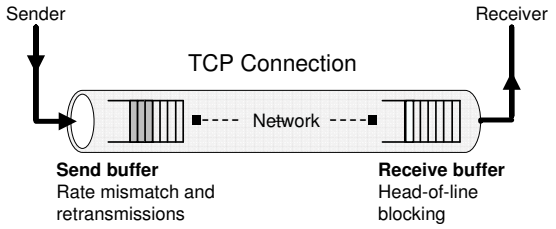

Fig. 1. Transport-layer queueing delays

II. 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 [16]. We assume that Nagle's algorithm is disabled since real-time applications often disable this mechanism to improve sending delays [17]. Hence, each packet can be immediately transmitted after it is written by the sender application.

A common characteristic of real-time applications is their low tolerance for end-to-end delay, the level of which depends on the application. While live video streaming can tolerate delays of a few seconds [4], VoIP typically requires less than 200 ms of end-to-end delay [18]. The low VoIP delays are often obtained by using small packets corresponding to 20 ms or 30 ms of audio. Real-time TCP flows differ from greedy TCP flows in that they are inherently rate-limited and have variable packet sizes. We discuss the implications of these characteristics, on the delay performance of TCP in Section V-B.

We use the packet delay distribution to evaluate the performance of real-time TCP flows. From the delay distribution, we derive the delay jitter which is the standard deviation of the packet delay, and the application-level packet loss rate which is the portion of packets that arrive beyond their playback time. These metrics are closely correlated with user-perceived video and audio quality [18], [19], and hence are used as an approximate performance measure. The packet loss 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.

A. TCP Delay Components

Real-time applications that use TCP run the risk of receiving low perceived quality. The additive-increase-multiplicative-decrease (AIMD) rate regulation scheme of TCP may delay data delivery by introducing throughput fluctuations when congestion occurs. Packet retransmissions, the mechanism used by TCP to provide lossless data delivery, can further introduce undesirable transport-layer delays.

TCP uses two buffers to provide congestion-controlled reliable data delivery: the send buffer and the receive buffer. The purpose of the send buffer is twofold [13]. First, it absorbs rate mismatches between the application sending rate and the transmission rate of TCP. Second, it stores a copy of the packets outstanding in the network should they be retransmitted. Hence, packets are held in the TCP send buffer due to rate mismatch and retransmissions. Note that only the unsent packets held in the send buffer, hereafter referred to as the *backlogged* packets, can contribute to the delay of newly admitted packets to the send buffer. The purpose of the receiver buffer is to hold out-of-order packets while a loss is being recovered. This buffering results in head-of-line (HOL) blocking delay. Figure 1 illustrates the three delay components of a TCP connection: sender-side delay, network delay, and receiver-side delay. 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. f

III. Modeling TCP Delay

Our model builds upon the detailed TCP model in [20] 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 incorporate the limited transmit [12] and window appropriateness [11] mechanisms to accurately capture the loss recovery latency of TCP. Third, we model the effect of window inflation [21] to improve the accuracy of the model for small congestion windows.

We assume that the sender is using a NewReno TCP [22] implementation. Later on in Section III-C we extend the TCP model to support other commonly found TCP variants such as SACK and Reno. Before describing the delay model, we briefly describe the mechanisms provided by TCP. For a detailed description of TCP's behavior, we refer the interested reader to [17], [22].

A. TCP Background

For completeness, we now describe the basic mechanisms used by the TCP protocol. TCP is window-based protocol that uses a feedback-based rate regulation scheme. The idea is to use a congestion window to regulate the amount of data that can be outstanding in the network at any time. TCP relies on losses as congestion feedback. It uses two mechanisms to detect packet losses: fast retransmit and timeout. If the sender receives three duplicate acknowledgements, it assumes that the data indicated by the acknowledgements is lost and immediately retransmits the missing data. This mechanism is called fast retransmit. After sending the missing data, TCP uses the fast recovery algorithm to govern the transmission of new data until a non-duplicate ACK arrives. Due to the round-trip time (RTT) needed for the receipt of the loss indication, fast recovery typically takes on the order of the path's round-trip-time [8].

The other loss recovery mechanism provided by TCP is the timeout mechanism. The TCP sender sets a timer for each transmitted packet. In case it receives less than three duplicate ACKs and the timer expires, the packet is retransmitted and the window size is reduced to one. If the first packet after a timeout is lost, TCP will double the length of the next timeout period. This doubling, usually called exponential backoff, continues

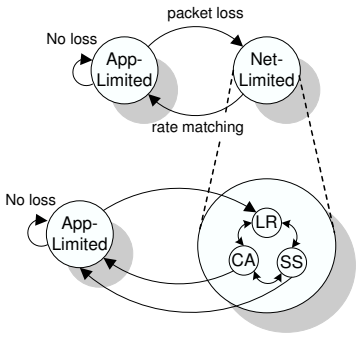

Fig. 2. A high-level view of a model for a TCP connection with a CBR source. The sender alternates between application-limited and network-limited periods. While network limited, it moves between several congestion control algorithms: slow-start (SS), congestion avoidance (CA), and loss recovery (LR).

until the retransmitted packet is successfully delivered. Since the initial timeout duration is typically on the order of several round-trip times (e.g, $4RTT$ according to [2]), timeout-based loss recovery is usually longer than fast recovery.

To regulate its sending rate, TCP uses an AIMD scheme: it increases the congestion window linearly fast as long as there is no loss, and reduces the window size by half upon the receipt of three duplicate ACKs. If a timeout event occurs, it reduces the window size to one.

B. Model for TCP Delay

We consider a data 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 data generation rate. However, transient congestion episodes can lead to TCP throughput fluctuations and to variable delays. As long as no packet is dropped, TCP will not add any delay to the data delivery beyond that of the network. After a packet drop, TCP reduces its throughput, and introduces additional delay for as long as the throughput it provides does not satisfy the sending rate of the flow. Thus, transient congestion episodes cause the flow to oscillate between an application-limited period and a network-limited period. A network-limited period is defined as any period in which the TCP sender is limited by the congestion window. Otherwise, the sender is application-limited. The behavior of a TCP connection with a CBR source is depicted in Figure 2. While the TCP sender is network-limited, it moves between several congestion control algorithms: slow start, congestion avoidance, and loss recovery (e.g., fast recovery and retransmission timeout). Our model captures the subtleties of the slow start algorithm [21] to improve the delay prediction accuracy.

We make several simplifying assumptions as follows. First, the TCP sender uses a packet-based congestion control mechanism, an assumption motivated by the wide-deployment of packet counting (ACK counting) TCP implementations [23]. Second, we assume that the slow start threshold is equal to half of the load in packets per round-trip time (*ppr*). This assumption is justified when a significant portion of the losses occurs during application-limited periods[1], as is the common case observed in our traces. Third, we do not model the effect of delayed acknowledgements (ACKs) since real-time applications will disable this mechanism to improve the delay performance. For simplicity of notation, we assume that the flow's load in *ppr* is an integer. For convenience, we summarized the notations used in this paper in Table II.

Abbreviation	Definition
CBR	Constant bit-rate
AL	Application limited
BC	Byte-counting
MSS	Maximum segment size
PI	Packetization interval
CVW	Congestion window validation
Bps	Bytes per second
pps	Packets per second
ppr	Packets per round-trip time

TABLE I
SUMMARY OF PAPER ABBREVIATIONS

Notation	Definition
f	load in packets per second
r	load in packets per round-trip time (*ppr*)
a	packet size
w	congestion window size
b	backlog size
l	indicates whether loss recovery is required
p	segment loss probability
T_0	initial timeout duration
L	forward network delay

TABLE II
SUMMARY OF MODEL NOTATIONS

We model the behavior of a TCP source by a 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 represented by an ordered triple (w, b, l), where w is the current congestion window size, b is the current backlog size, 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 ($r \leq w, b = 0$, where r is the load in *ppr*), or network-limited. Due to flows with small packets the window limitation can be either in bytes per RTT ($b > 0$), or in packets per RTT ($w < r, b = 0$), as described in Section V-B. 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 ($w = 0, l \geq 1$, where l indicates the current exponential back-off stage). Table III lists the rules for classifying an arbitrary state $s = (w, b, l)$ according to the congestion control phases of TCP.

The TCP sender changes the congestion window size and the congestion control algorithm based on the packet loss feedback. For example, in the absence of packet loss, the TCP source moves from state $s = (w, b, l)$ to state $s' = (w+1, b', l)$ if it is in congestion avoidance. More specifically, each state

[1] [21] recommends that the slow start threshold should be set to half of the packets in flight when fast recovery is invoked.

Classification	Condition
Application-limited	$r \leq w, b = 0, l = 0$
Network-Limited	$0 < w, b \neq 0$ or $w < r, b = 0$
Congestion avoidance	$r/2 < w, l = 0$
Slow start	$w \leq r/2, l = 0$
Fast recovery	$0 < w, l = 1$
Timeout	$w = 0, l = 1, \ldots, 6$

TABLE III
STATE CLASSIFICATION

has at most three outgoing transitions that correspond to the following events: the receipt of a fast retransmit loss indication, the receipt of a timeout loss indication, and successful data delivery. Detailed description of the Markov chain and the transition probabilities can be found in Appendix A.

Since TCP is a byte stream protocol, it can assemble a number of small packets into one data transmission unit, known as *segment*. Packet assembly can occur when small packets are sent and the sender is backlogged and is avoided when large packets are sent. Due to packet assembly a small-packet TCP flow reacts to congestion *by adapting both the segment size and the congestion window*, as demonstrated below.

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

$$b' = \begin{cases} \max\{0, aft_{s;s'} - wM\} & \text{if } w' > 0, l' = 0 \\ \max\{0, aft_{s;s'} - (w+3)M\} & \text{if } w' > 0, l' = 1 \\ \max\{0, aft_{s;s'}\} & \text{if } w' = 0, l' \geq 1 \end{cases} \tag{1}$$

$$M = \begin{cases} a & \text{if } w = r,\ b = 0,\ l = 0 \\ MSS & \text{otherwise} \end{cases}$$

where f is the load in packets per second, a is the packet size of the flow, MSS is the maximum segment size (typically 536 or 1460 bytes [27]), M is the size of the sent segment which depends on whether the sender is network-limited ($M = MSS$) or not ($M = a$), and $t_{s;s'}$ is the time taken for the transition from s to s', which we will soon define. The first term in (1) $aft_{s;s'}$ models the increase in backlog size due to newly admitted packets to the send buffer. The second term (e.g., wM) models the decrease in backlog size due to the transmission of segments. The number of segments sent is w for a loss-free transition and $w + 3$ for a transition to fast recovery state. The latter case accounts for segment transmissions caused by the receipt of duplicate ACKs, a mechanism known as window inflation [21]. We assume that no segments are sent during a sequence of timeout states.

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

$$t_{s;s'} = \begin{cases} 1/f & \text{if } AL, w' > 0, l' = 0 \\ RTT + 3/f & \text{if } AL, w' > 0, l' = 1 \\ 2^{\min\{l-1,6\}} T_0 & \text{if } \overline{AL}, w' = 0, l' \geq 1 \\ RTT & \text{otherwise} \end{cases} \tag{2}$$

where for brevity we use $AL \triangleq r \leq w, b = 0$ to denote the condition for an application-limited state. In the absence of packet loss, the time spent in an application-limited state is the inter-sending time of the flow, denoted by $1/f$. Due to the round trip-time needed to receive the first duplicate ACK, the time spent in an application-limited state that receives a fast retransmit loss indication is RTT + $3/f$. In the absence of timeout loss indications, the time spent in a network-limited state is taken to be RTT, a common modeling assumption [6], [8]. The time spent in a sequence of timeout states is taken to be $T_0, 2T_0, \ldots, 64T_0$, where T_o is the duration of the initial timeout. The initial timeout duration is approximated by $4RTT$, as suggested in [2].

The delay at the sender is approximated by the backlog size, as follows. Observe that the packets left behind in the send buffer after a successful transition must have arrived while the transmitted packet was backlogged. Hence, a transmitted packet that leaves behind a backlog of size b has been buffered for at least $b/(af)$. The delay of a packet sent in a loss-free state is thus modeled by

$$L + b/(fa) \tag{3}$$

where L is the one-way network delay, which is assumed to be constant.

The head-of-line blocking delay at the receiver is determined by the loss recovery latency. This latency consists of the time it takes TCP to detect a loss and the time it takes the retransmitted packets to arrive to the receiver. Due to the round-trip time needed for the first duplicate ACK feedback, we assume that the time required to receive a fast retransmit loss indication is at most $RTT + 3/f$. Further, we assume that fast recovery always takes a single RTT regardless of the number of packets lost in a transmission window, as suggested by [8]. Since TCP is an in-order protocol, the receiver does not deliver out-of-order packets to the application till the missing packets have arrived. Hence, the delay of the i-th packet sent in a state that receives a fast retransmit loss indication, is modeled by

$$L + RTT + b/(fa) + (3+i)/f, \tag{4}$$

where $i = 0$ represents the delay of the lost packet. Using equations (3) and (4), we can express the TCP delay of the i-th packet sent in a state s that transitions to a state s' as:

$$d^{(i)}_{s;s'} = L + \begin{cases} b/(fa) & \text{if } w' > 0, l' = 0 \\ b/(fa) + RTT + (3+i)/f & \text{if } w' > 0, l' = 1 \\ 0 & \text{if } w' = 0, l' \geq 1 \end{cases} \tag{5}$$

The number of packets sent during this transition $n_{s;s'}$ is given by

$$n_{s;s'} = \begin{cases} \lfloor ft_{s;s'} \rfloor & \text{if } AL, w' > 0, l' = 0 \\ \lfloor \min\{b, wMSS\}/a \rfloor & \text{if } \overline{AL}, w' > 0, l' = 0 \\ \lfloor \min\{b, (w+3)MSS\}/a & \text{if } w' > 0, l = 1 \\ 0 & \text{if } w' = 0, l \geq 1 \end{cases} \tag{6}$$

The first case in (6) corresponds to an application-limited state where a single packet is sent for every loss-free transition. The

second case corresponds to a network-limited state in which the number of sent packets is determined by the number of backlogged packets that fit into the congestion window. The third case accounts for window inflation, and the forth one accounts for idle timeout states.

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 [24]. 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 of 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 defined over some finite interval A. Using renewal theory [24], we can now compute the steady state delay distribution.

$$\begin{aligned} D = d \quad & \text{w.p.} \lim_{t\to\infty} \frac{N_t(d)}{N_t} && \forall d \in A \\ = d \quad & \text{w.p.} \frac{\sum_{s\in S} \pi_s \sum_{s'\in S} d_{s;s'} \sum_{i=1}^{n_{s;s'}} I_{d_{s;s'}=d}}{\sum_{s\in S} \pi_s \sum_{s'\in S} d_{s;s'}\, n_{s;s'}} && \forall d \in A \end{aligned} \tag{7}$$

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 (5) and (6), respectively. The above equation can be solved numerically to yield the application's performance statistics, namely, the TCP delay jitter σ_D and the α-delay percentile $\arg\max_x P\{D < x\} < \alpha$, as well as other useful statistics such as the average delay $E(D)$ and the portion of time that the sender is backlogged $\frac{\sum_{s\in S: s=(w,0,l), r\le w} \pi_s \sum_{s'\in S} q_{s;s'}\, t_{s;s'}}{\sum_{s\in S} \pi_s \sum_{s'\in S} q_{s;s'}\, t_{s;s'}}$.

C. Transition Probabilities of the Markov Chain

We now explain how to obtain the transition probabilities of the Markov chain. 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. In our derivation of the transition probabilities, we consider both correlated and random packet losses, typical for FIFO and RED routers, respectively.

1) Random Packet Losses: The random packet loss model is the most basic loss model. It assumes that a packet is dropped independently of others with some fixed probability, denoted by p. This loss behavior is likely to arise when the bottleneck router implements a RED queueing scheme [25]. We use the random packet loss assumption to validate the model in scenarios with configured packet drop rates (see Section IV-A).

We build on the modeling results of [26] to obtain the transition probabilities of the Markov chain. According to [26], a NewReno sender transitions to fast recovery if three duplicate ACKs are received and none of the retransmitted segments are lost. Thus, given a window size of w and a backlog size of b, the fast retransmit and timeout probabilities $p_f(w,b)$ and $p_t(w,b)$, respectively, can be expressed as

$$p_f(w,b) = \begin{cases} 0 & \text{if } w < 4 \\ \sum_{i=1}^{w-3} B_p(w,i)(1-p)^i & \text{if } 4 \le w, \overline{AL} \\ p\sum_{i=1}^{w-3} B_p(w-1,i-1)(1-p)^i & \text{otherwise} \end{cases} \tag{8}$$

$$p_t(w,b) = \begin{cases} 1 - p_f(w,b) - B_p(w,0) & \text{if } \overline{AL} \\ p - p_f(w,b) & \text{if } AL \end{cases} \tag{9}$$

where AL denotes the condition for an application-limited state (see Section III-B) and $B_p(w,i)$ is the probability to have i losses out of a window of w segments and is given by

$$B_p(w,i) = \binom{w}{i} p^i (1-p)^{w-i} \tag{10}$$

The fast retransmit probability given in (8) accounts for the Markov chain structure. The second case represents the transition probability from a network-limited state where w segments are sent, and the third case represents the transition probability from an application-limited state where a single segment is sent per loss-free transition. Complete description of the transition probabilities can be found in Appendix A. Markov chains for TCP SACK and TCP Reno can be obtained by modifying (8) according to the SACK and Reno fast retransmit probabilities given by equations (4) and (2), respectively, in [26]. The corresponding timeout probability is determined by the expression in (9).

2) Correlated Packet Losses: Under this model, when a packet is lost, all the following packets sent within the same window are also lost. This loss process is likely to arise when the bottleneck router uses drop-tail queuing. Therefore, we apply this loss assumption to validate the model in drop-tail router environments (see Section IV-B). Padhye *et al.* [6] derived the loss recovery probability under the correlated loss assumption[2], which we use to obtain the transition probabilities of the chain. According to [6], given a window size w, the probability that a loss indication is a timeout $\hat{Q}(w)$ is:

$$\hat{Q}(w) = \max\left\{1, \frac{(1-(1-p)^3)(1+(1-p)^3(1-(1-p)^{w-3}))}{1-(1-p)^w}\right\} \tag{11}$$

Taking into account the Markov chain structure as in the random packet loss case, the fast retransmit transition probability can be modeled by:

$$p_f(w,b) = \begin{cases} \hat{Q}(w)(1-B_p(w,0)) & \text{if } \overline{AL} \\ \hat{Q}(w-1)p & \text{if } AL \end{cases} \tag{12}$$

The corresponding timeout probability formula has the same form as (9).

D. Modeling Byte-Based Network Limitations

So far we have assumed that the network-limitation is in packets per second i.e., the bottleneck in the network is a drop-tail queue in units of packets. However, if the bottleneck

[2]The timeout probability derivation in [6] assumes a TCP Reno implementation.

is in bytes per second i.e., the bottleneck in the network is a drop-tail queue in units of bytes, a small packet is less likely to be dropped than a large packet flow [27]. Consequently, small-packet flows may experience lower delays than large-packet flows. Since the performance may differ significantly it is necessary to design both packet-based and byte-based versions of the model.

When the queue size is maintained in bytes, small packets have a higher probability to be admitted to a nearly full queue. We assume that a small packet of size a is A/a times less likely to be dropped than a large packet of size A, as suggested by [28]. Recall from Section III-B, that a TCP flow reacts to congestion by adapting the segment size and that the segment size depends on whether the sender is network-limited (MSS) or not (a). Hence, the packet drop probability used to derive the transition probabilities from application-limited states should be scaled so that it is proportional to the packet size. More specifically, the fast retransmit and timeout probabilities $p_f^b(w,b)$ and $p_t^b(w,b)$, respectively, for a byte-based bottleneck are computed by:

$$p_f^b(w,b) = \begin{cases} p_f(w,b) & \text{if } \overline{AL} \\ \tilde{p}\sum_{i=1}^{w-3} B_{\tilde{p}}(w-1,i-1)(1-\tilde{p})^i & \text{if } AL \end{cases} \tag{13}$$

$$P_t^b(w,b) = \begin{cases} p_t(w,b) & \text{if } \overline{AL} \\ \tilde{p} - p_f^b(w,b) & \text{if } AL \end{cases} \tag{14}$$

where p is the probability that a fully-sized segment is lost, $\tilde{p} = p\frac{a}{MSS}$ is the probability that an a-byte segment is lost, and $p_f(w,b)$ and $p_t(w,b)$ are given by (8) and (9), respectively.

E. Modeling Congestion Window Appropriateness

The congestion window behavior during application-limited periods is implementation dependent. One option is that the congestion window size retains memory of an inflated congestion window used to clear the recent data backlog. This behavior causes the congestion window size to overestimate the actual amount of data sent by the TCP sender. It is, in fact, the common case observed in our test-bed and Internet experiments. Another option is that the congestion window size reflects the actual load of the application. This is likely, if TCP sender applies the congestion window validation (CVW) extension [11].

Since the performance of a rate-limited flow may differ significantly depending on this implementation artifact, our model accounts for both variants, as follows. The congestion window is either decayed to the application's load in *ppr* (CVW) or left intact (non-CVW) upon a transition from a network-limited state to an application-limited state. Complete description of the congestion window evolution can be found Appendix A. The scenario where the window grows arbitrarily large during application-limited periods, pointed out by [11], was not observed in our traces, and hence is not modeled.

F. Modeling the Limited Transmit Mechanism

Here we model the effect of the limited transmit mechanism [12]. This mechanism improves the loss recovery efficiency of TCP when a congestion window is small or when a large number of segments are lost in a single transmission window. It allows the sender to send a new data segment upon the receipt of each of the first two duplicate ACKs, thereby increasing the chance to receive the three duplicate ACKs needed to trigger the fast retransmit algorithm.

Limited transmit can lead to successful loss recovery using fast retransmit and fast recovery if the following conditions hold [26]: (a) three duplicate ACKs are received. That is, one or two duplicate ACKs are received for a single transmission window and at least two or one, respectively, of the corresponding transmitted segments are not dropped in the network (b) none of the retransmitted segments are lost. Let $p_l(w,b)$ be the probability to transition to fast recovery given a congestion window w, a backlog sizes b, and assuming that limited transmit is used. Under the assumption of random packet losses (see Section III-C1), $p_l(w,b)$ can be computed as

$$\begin{aligned} p_l(w,b) = {} & \tilde{B}(w,w-1)(1-p)^{w-1}(1-p)^2 I_{w>1} \\ & + \tilde{B}(w,w-2)(1-p)^{w-2}(1-p^2) I_{w>2} + p_f(w,b) I_{w>3} \end{aligned} \tag{15}$$

$$\tilde{B}(w,b) = \begin{cases} B(w,i) & \text{if } \overline{AL} \\ pB(w-1,i-1) & \text{if } AL \end{cases} \tag{16}$$

where $p_f(w,b)$ is the fast retransmit transition probability assuming limited transmit is not used (e.g., (8)), $B(w,i) = \binom{w}{i} p^i (1-p)^{w-i}$ is the probability to have i losses out of a window of w segments, and I is the indicator function. The corresponding timeout probability formula has the same form as (9). Note that (16) accounts for the fact that the number of segments sent in a state depends on whether the sender is application-limited (where a single segment is sent per loss-free transition) or not (where w segments are sent per transition). As in Section III-D, we account for byte-based bottlenecks by replacing p with a scaled probability $\tilde{p} = p\frac{a}{MSS}$ when computing the transition probabilities from an application-limited state. Similar reasoning can be applied to derive the transition probabilities of the Markov chain when correlated packet losses are assumed.

G. Computation Complexity

The complexity of the TCP delay computation directly depends on the size of the state space of the Markov chain. We use trail and error to select a state space size that is small enough and allows us to efficiently evaluate the performance over the set of network environments considered in Section IV. To solve the Markov chain, we select the maximum window size w_{max} to be $w_{max} = 8r$ and the maximum backlog size, measured in units of packets, b_{max} to be $b_{max} = 4T_0 f = 16r$. Since the state space includes the loss recovery variable l which captures the exponential

backoff stage in timeout states, the number of states in the Markov chain is $6b_{max} + 2b_{max}w_{max}$.

IV. Model Validation

We evaluated the model using a controlled network environment and Internet experiments. 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. For the controlled network environment, we consider a single CBR-TCP flow with configured packet drops, and multiple CBR-TCP flows competing with FTP and web flows in a topology consisting of a router with a drop-tail queuing scheme. For the Internet experiments, we consider CBR-TCP flows from broadband residential clients with DSL access and CBR-TCP flows from institutional clients with high-bandwidth access (i.e., PlanetLab clients). Table IV summarizes the configurations for the controlled environment and Internet experiments.

Experiment configurations		
Controlled	Single flow Multi-flow	Linux 2.6, Windows XP Linux 2.6
Internet	Planet Lab DSL	Linux 2.6 Linux 2.6, Windows XP

TABLE IV
Experiment configurations for model validation and TCP delay analysis.

We wrote a tool that can send and receive bidirectional CBR over TCP flows with different packet sizes at constant intervals. The bit-rate of a CBR-TCP flow is determined by the packet size and packetization interval (PI), i.e., the inter-packet generation time. We use packet sizes of 174, 724, and 1448 bytes and packetization intervals of 20 ms and 30 ms as these choices approximately reflect one way voice (64 kb/s) [1] and video flows (300 kb/s and 573 kb/s) [4], [13]. For ease of presentation, we call 174-byte, 724-byte, and 1448-byte packet flows small (VoIP), medium, and large (video) flows, respectively.

We conducted the experiments 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. The flexibility of Linux in configuring system and session-level TCP-related options allows us to assess their impact in isolation. We configured the following system-level options for the experiments. On the sender machine, we set the congestion control algorithm to New Reno [22] and disabled *ssthresh* use from last TCP connection. We enabled the SACK option on both ends to improve the loss recovery latency. We disabled the byte-counting (BC) option because it causes the congestion window to increase as a function of the number of bytes sent and hence results in increased delays. Detailed results on the impact of BC can be found in Section V-D. Note that disabling byte-counting can make the system susceptible to ACK-division attacks [29].

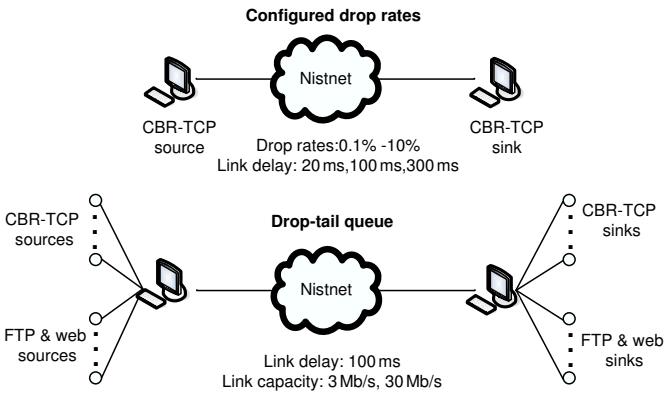

Fig. 3. Experiment setup for model verification in a controlled environment.

Starting with Linux kernel 2.6.18, congestion window validation can be enabled or disabled. As described in Section V-C, the use of congestion window validation can increase sensitivity to timeouts for CBR-TCP applications. We therefore suggest that this option be disabled.

We also configured the following session-level options. We set the TCP buffer size of the receiver application to the maximum value allowed by the system to ensure that a TCP sender is never limited by the receiver window. We disabled Nagle's algorithm and delayed ACKs on both ends. The above system and session-level settings represent a delay-optimized and aggressive configuration for TCP. For model verification and TCP delay analysis, we present the delay distribution, 95th percentile delay, and mean delay.

A. Model Validation Using Configured Drop Rates

We performed model validation experiments for different packet drop rates on a test-bed that emulates a wide range of network environments, as shown in Figure 3.

The router ran NIST Net [30], a network emulation program which can introduce constant delay, and can drop packets with configured loss rates *regardless* of their size. 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 approximately reflects the local, US coast-to-coast, and trans-continental delays. In the experiments, we do not consider loss rates greater than 10% because for such high loss rates the average TCP throughput does not usually satisfy the bit-rate requirement of the CBR-TCP flow. For each set of parameters, we ran the experiment for five minutes and repeated each experiment ten times.

The results for model validation are presented in Figures 4 and 5. Figure 4a and Figure 4b present the predicted vs. measured 95th percentile and mean TCP delay, respectively, for a range of network parameters. Figure 5a shows the predicted vs. measured cumulative TCP delay distribution. Figure 5b shows the relative prediction error with respect to the actual measurement for a loss rate of one percent. The prediction accuracy of the model stems from capturing the send buffer dynamics using the backlog size, as described

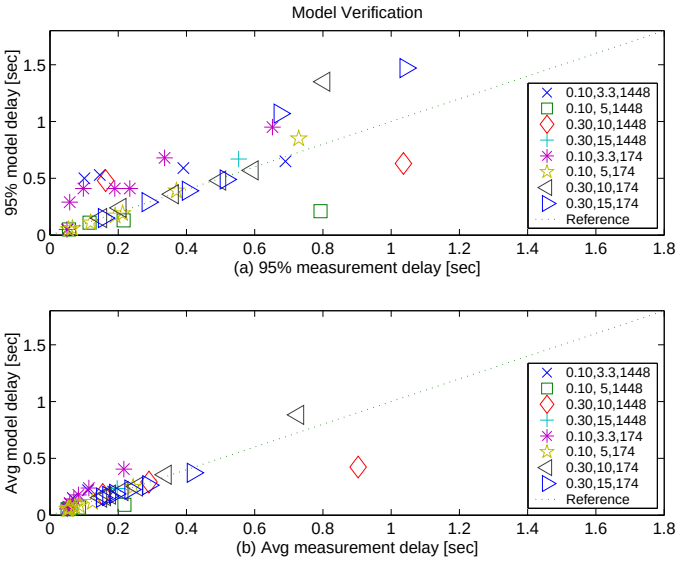

Fig. 4. (a) predicted vs. measured 95th percentile TCP delay for a range of RTTs (leftmost value in the legend label), packets per round-trip time (middle value in the legend label), and packet sizes (rightmost value in the legend label). (b) predicted vs. measured mean TCP delay for the same set of parameters as in (a).

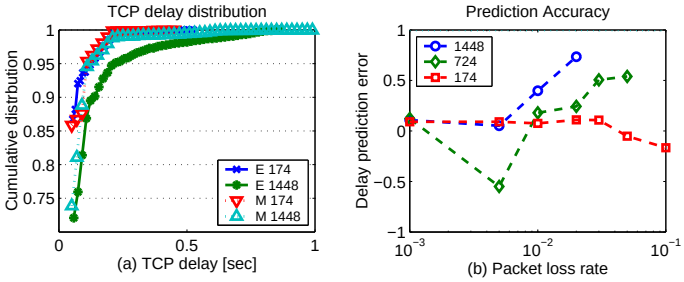

Fig. 5. (a) predicted (M) vs. measured (E) cumulative TCP delay distribution for small (174-byte) and large (1448-byte) packet sizes. (b) relative error of the model for small, medium (724-byte) and large packets.

in Section III-B. In general, the modeling error increases as the packet loss rate increases. This is intuitive because an increase in the packet loss rate results in more frequent throughput fluctuations and hence in larger variations in the backlog size. We elaborate on the model prediction accuracy in the next section.

B. Model Validation Using Routers with Drop-Tail Queues

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 3. The drop-tail queue can be maintained in units of packets and bytes. When the queue size is maintained in bytes, small-packets are less likely to be dropped than large-packets, and hence small-packet flows perform better than large-packet flows under the same network conditions.

We used the Linux test-bed from Section IV-A and modified NIST Net to incorporate a drop-tail queue. We devised a multi-flow setting in which five small and large CBR-TCP flows compete with five long-lived FTP and varying number of web flows. This choice was inspired by the configuration used to evaluate the performance of TFRC-small packets [27]. We used SRI and ISI traffic generator [31] to generate exponentially distributed web traffic with a mean duration of 50 ms and a constant packet size of 512 bytes. The round-trip propagation delay was set to 100 ms for all experiments, and the link bandwidth was set to 3 Mb/s and 30 Mb/s for small and large-packet flows. For each configuration, we ran the experiment for five minutes and repeated it five times.

For small CBR-TCP flows, 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 [32]). However, this number overestimates the packet queue equivalent since the packet sizes of CBR-TCP and web flows are less than one MSS. Thus, we set it to 50,000 bytes which approximately reflects the average of CBR-TCP (174 bytes), web (512 bytes), and FTP (1448 bytes) flows, a choice motivated by [27]. Similarly, for large-packet flows, we set the byte queue to 100,000 bytes which approximately reflects the average of CBR over TCP (1448 bytes), web and FTP flows.

Table V and VI present the measured and predicted 95th percentile delay for small and large packet flows, respectively for a packet and byte-based queue. The 95th percentile statistics are abbreviated with 95%. From these tables and Figure 5 we observe that the prediction accuracy of the model decreases as the available network bandwidth seen by the TCP connection decreases. More specifically, the accuracy decreases as the connection utilization, the ratio of the flow's bit-rate to the fair-TCP bit-rate [27], increases. The model prediction accuracy is low when the connection utilization is close to one, because the actual delay grows arbitrary large and hence the size of the state-space of the model should be increased accordingly. However, this is not the case, since we truncate the state-space size to reduce the computational complexity.

Note that the percentile delays for small and large flows are not directly comparable because the link bandwidths for the two settings were different, and hence loss rates and queuing delays are different. Also note that we present results for a packet and byte-based queue for large-packet flow. This is because our experiment setting comprised of competing web flows whose packet size was 512 bytes.

C. Model Validation Using Internet Experiments

We performed model validation using Planet-Lab environment, and machines connected to home DSL. For Planet-Lab experiments, we ran our sender and receiver application on machines located at US (California, New York), Europe (UK, France), Asia (China). For each sender and receiver pair, we ran our tool for small and large-packet flows for thirty minutes and repeated the experiments five times over randomly selected days from February to June, 2007. For DSL experiments, we ran sender on machines located in US, Israel, and Pakistan and receiver application was in New York.

Surprisingly, for the majority of Planet Lab flows we observed a handful of losses ($< 0.5\%$). Hence, the 95th percentile delay was close to the network delay for which

Model validation for small-packet flow (174 bytes)												
Web flows	Packet queue						Byte queue					
	LR (%)	RTT (s)	95% jitter (s)	95% delay (s)	Model (s)	Con. Util.	LR (%)	RTT (s)	95% jitter (s)	95% delay (s)	Model (s)	Con. Util.
0	1.02	0.317	0.136	0.410	0.487	0.272	0.20	0.172	0.069	0.135	0.122	0.065
10	1.60	0.380	0.169	0.498	0.690	0.408	0.55	0.178	0.099	0.183	0.128	0.112
15	2.31	0.371	0.205	0.644	0.721	0.479	0.81	0.180	0.111	0.233	0.130	0.137
20	3.39	0.383	0.263	1.335	1.853	0.601	1.24	0.182	0.140	0.308	0.192	0.172
25	5.61	0.385	0.447	6.173	2.255	0.774	2.41	0.185	0.204	3.057	0.755	0.244

TABLE V
MODEL VALIDATION FOR FIVE SMALL CBR-TCP FLOWS COMPETING WITH FIVE FTP FLOWS AND VARYING NUMBER OF WEB FLOWS.

Model validation for large-packet flow (1448 bytes)												
Web flows	Packet queue						Byte queue					
	LR (%)	RTT (s)	95% jitter (s)	95% delay (s)	Model (s)	Con. Util.	LR (%)	RTT (s)	95% jitter (s)	95% delay (s)	Model (s)	Con. Util.
0	0.75	101	0.101	0.215	0.309	0.618	1.25	101	0.151	0.460	0.57	0.798
10	0.81	101	0.093	0.265	0.401	0.642	1.28	101	0.159	0.594	0.59	0.808
25	0.90	101	0.094	0.289	0.431	0.677	1.34	101	0.174	0.914	0.61	0.826
50	1.20	101	0.110	0.608	0.610	0.782	1.79	101	0.229	1.890	0.67	0.955
100	2.91	101	0.310	4.880	1.670	1.218	3.18	101	0.544	5.978	1.91	1.273

TABLE VI
MODEL VALIDATION FOR FIVE LARGE CBR-TCP FLOWS COMPETING WITH FIVE FTP FLOWS AND VARYING NUMBER OF WEB FLOWS.

the model provided a good match. Similar low loss rates were observed for DSL experiments. The DSL losses were clustered and resulted in consecutive retransmission timeouts. Since these clustered loss events were rare, we did not have enough data to reliably apply the model. The low loss rate observed in the DSL measurements may stem from over-provisioning in the Internet and the low-bandwidth requirement of CBR-TCP flows.

V. DISCUSSION

In this section, we present the working region for VoIP and live video streaming flows carried over TCP and discuss the key factors that influence the performance of CBR-TCP flows, deduced using our model and experiments.

A. Working Region

We now identify the conditions under which TCP can satisfy the delay requirements of VoIP and live media streaming. For VoIP, we assume that the interactive latency limit is 200 ms [18] and for live video streaming we assume it is 5 s, a choice motivated by [4]. The working region for VoIP and live video streaming is defined as the range of loss rates and RTTs where the 95th percentile delay and the maximum delay is below 200 ms and 5 s, respectively.

Figure 6 demonstrates the working region for the two applications, which was obtained using the random-drop experimental setup described in Section IV-A. We observe that the VoIP delay threshold is satisfied when the RTT is less than 200 ms and the packet drop rate is lower than 2%. The streaming threshold is satisfied when the loss rate is below 10%. The RTT has marginal effect on the streaming threshold and hence can be ignored. These regions were obtained under the assumption that the bottleneck limitation is in packets per second. Hence, the working region can be larger when the bottleneck limitation is in bytes per second. Note that a non-delay friendly setting of TCP (see Section IV) can significantly constrain the working region.

The loss rate and the round-trip time jointly determine the fair-TCP rate [27]. The ratio of the CBR rate to the fair-TCP rate, hereafter called the *connection utilization*, represents how much the CBR rate is lower than the throughput provided by the TCP connection. This ratio can be viewed as an approximate measure for the delay performance of TCP, as suggested by [9]. Therefore, we characterize the working region using this measure. Figure 7 shows the delay performance as a function of the connection utilization and the packet size of the flow for a loss rate of one-percent. We observe that the delay curves have similar shapes. At first, the delay increases little or linearly with the connection utilization. However, as network congestion increases, queueing delays at the TCP sender start building up and the TCP delay increases drastically, exhibiting exponential-like growth. The results indicate that, in general, the delay added by TCP is on the order of the path's round trip time when the connection utilization is below one third.

B. The Effect of Packet Size on Performance

Our traces indicate that small-packet flows perform significantly better than large-packet flows under the same network conditions. In this section, we explain the reason for the observed performance bias using TCP traces taken from our test-bed environment.

Figure 8 demonstrates the performance improvement when the flow's packet size is reduced. The figure shows the TCP delay and the corresponding window size vs. time for a large

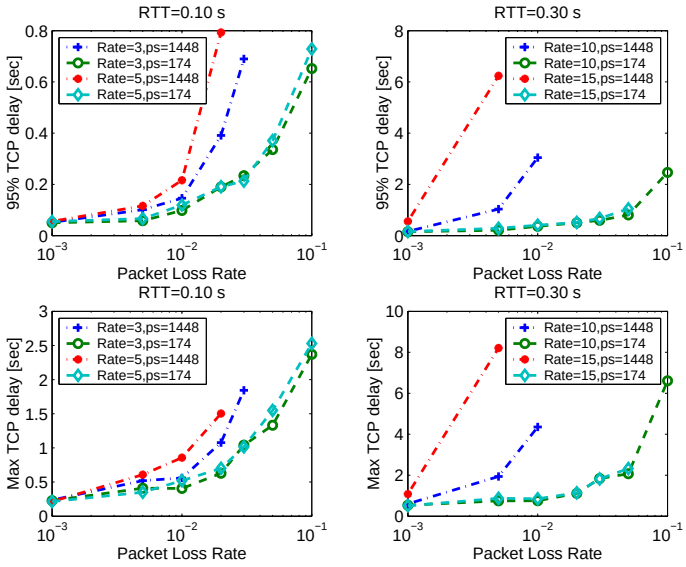

Fig. 6. Feasible region for VoIP (upper plots) and video streaming (lower plots) as a function of sending rate in packets per round-trip time (rate) and packet sizes (ps).

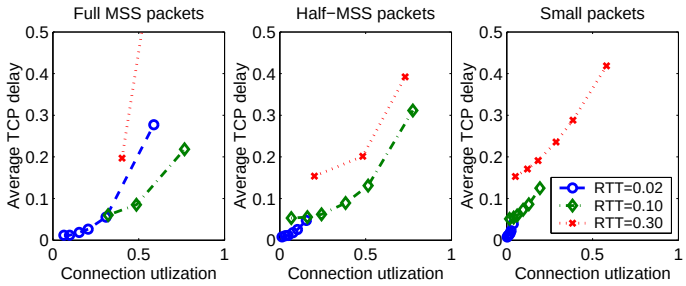

Fig. 7. Average TCP delay as a function of connection util. and packet size.

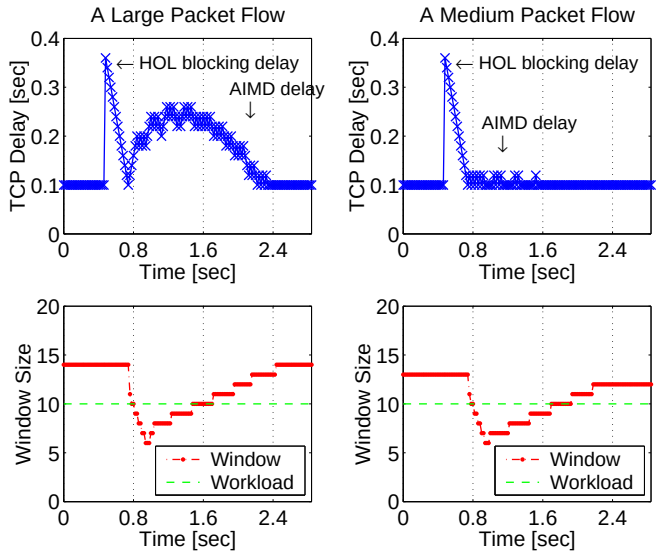

Fig. 8. TCP delay and congestion window size for flows with full and half MSS-sized packets that experiences a single loss.

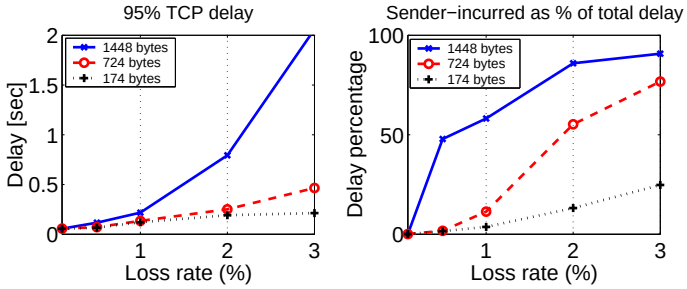

Fig. 9. 95th percentile TCP delay for small, medium and large packets and the sender-incurred delay as a percentage of TCP delay. RTT is 100 ms.

1448-byte (left side plots) and a medium 724-byte packet flow. Both send 100 packets per second over a link with RTT of 200 ms. We used Nist Net to drop a single packet from each flow so that both flows detect a loss at time 0.5 s and become network-limited. Since the loss is recovered using fast retransmit, the HOL blocking delay, given in (5), is at most $1.5 \cdot RTT + 3 \cdot 20\,ms = 360\,ms$.

Although both flows are limited by the same congestion window size during network-limited period, their loads in bytes per second (Bps) differ; the load in Bps of the medium-packet flow is half of the load in Bps of the large-packet flow. For the large-packet flow, the rate mismatch between the the constant load and linearly increasing throughput of TCP will result in data backlog buildup with quadratic-like shape, as shown in time interval [0.76 s, 2.4 s]. For the medium-packet flow, the packet assembly capability of TCP kicks in and reduces backlog buildup. The resulting delays are now on the order of the packet inter-sending time, i.e., 20 ms. We omit similar results for small-packet flows.

The reason for the performance gain observed is that the congestion control mechanism varies the TCP throughput as a function of the number of *packets* in flight, rather than as a function of the number of *bytes* in flight. Although both flows experience the same throughput fluctuations in pps, the throughput fluctuations in Bps can be significantly smaller for the small-packet flow. The smaller the packet size, the smaller the magnitude of the throughput fluctuations, and the lesser the delay. Thus, packet-based congestion control results in a significant performance bias *in favor* of small-packet flows.

In general, the delays of small-packet flows tend to be dominated by loss recovery latency, whereas those of large-packet flows tend to be dominated by the delays induced by congestion control, as demonstrated in Figure 9.

The performance bias described above creates an incentive for real-time applications to improve their performance without network assistance. The reasoning is simple: if a large packet flow experiences worse performance than a small packet flow, then why not masquerade a large packet flow as a small packet flow? The application can simply send few small packets at evenly spaced intervals instead of a large packet, thus improving its own performance while still maintaining the same sending rate in bytes per second.

Figure 10 demonstrates the potential performance improvement obtained via such an approach. The flow in left side plots is the baseline. It sends 100 packets per second over a

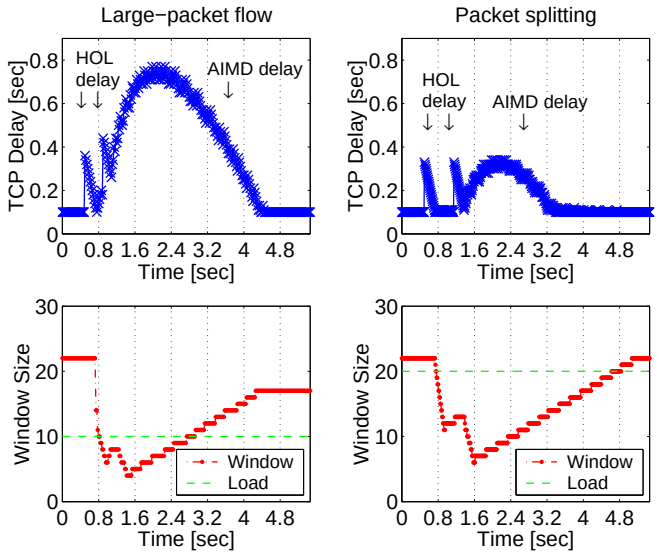

Fig. 10. TCP delay and congestion window size for a large-packet flow (left side plots). The right side plots shows delay and window size for a similar flow where each MSS-size packet is split by half.

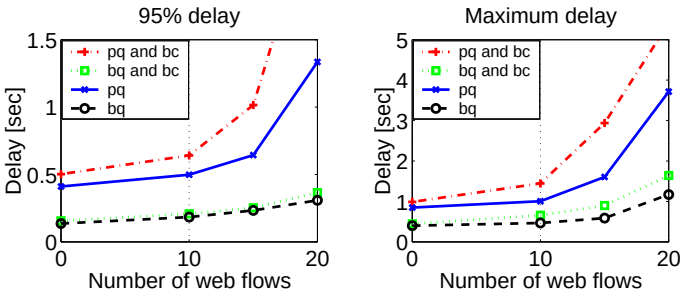

Fig. 11. 95% and maximum end-to-end delay for packet and byte-based queue with and without byte-counting. 'pq', 'bq', 'pc', and 'bc' abbreviate packet-queue, byte-queue, packet-counting, and byte-counting respectively. Five small-packet flows compete FTP flows with varying number of web flows.

300 ms RTT symmetric network and uses MSS-sized packets. The flow in the right side plots sends 200 packets per second and uses a half MSS-sized packets. Both flows have the same bandwidth requirement in bytes per second and both experience two close-by losses.

As expected, both flows exhibit the same performance during application-limited and loss recovery periods. Since TCP adapts its throughput based on the number of packets in flight, the magnitude of the throughput fluctuations is significantly smaller for the medium packet flow, resulting in lower delays (e.g., the peak delay in our example is lower than that of the large packet flow by 45%). From these examples it can be clearly seen that the performance gain of packet splitting comes from the reduction in the AIMD-induced delays.

C. The Effect of Timeout 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.

One reason for the small number of timeouts observed is TCP's use of an invalid congestion window which overestimates the actual amount of data sent. 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. This implementation artifact is in fact the common case[3] observed in our traces. The scenario where the window grows arbitrarily large during application-limited periods, pointed out by [11], was not observed in our traces. For example, the congestion window value in Figure 8 overestimates the actual load of large and medium-packet flow by 30% and 40%, respectively. Alternatively, a TCP sender can use congestion window validation [11] and have the window size reflect the actual amount of data sent. This will result in more timeouts. Retransmission timeouts can also be avoided when the limited transmit mechanism [12] is used, which is enabled by default in Linux 2.6 and Windows XP. This mechanism enables 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 the three duplicate ACKs to arrive. For instance, when the window overestimates the load by 40% and the packet loss rate is 10%, our model yields that, on average, 15% of the timeouts are avoided by window invalidation and 35% are avoided when both mechanisms are in use. The magnitude of savings is significant for lower loss rates and for smaller windows.

D. Sensitivity to Bottleneck Router Limitation and Byte-counting

As Internet router behavior is not well specified, bottlenecks in the network can include limitations in pps, in Bps, and even in both [27]. Different network limitations produce different packet drop rates: large and small packets will experience the same packet drop rates when the network limitation is in pps, whereas, in the other case, small-packet flows can experience lower drop rates than large packet flows. Since TCP's delay performance improves as the packet drop rate reduces, a byte-based bottleneck is biased in favor of small packet flows, as shown in Table VII . A typical TCP sender will most likely operate in the lower two quadrants of the table due to the dominance of packet-based congestion control TCP implementations.

Allman [29] proposed a modification to how TCP increases its congestion window during slow-start and congestion-

[3]Since Windows XP does not allow applications to sample the window state, we used similar techniques to those in [23] to verify this observation.

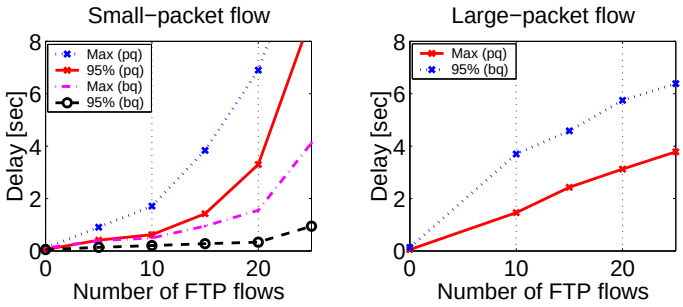

Fig. 12. 95% end-to-end delay for five VoIP and varying number of FTP flows. The 95% statistics are averaged over five VoIP flows.

	Byte-based drops	Packet-based drops
Byte-based cong. control	+	None
Packet-based cong. control	+++	++

TABLE VII
THE PERFORMANCE BIAS IN FAVOR OF SMALL PACKET FLOWS; '+++' REPRESENTS THE LARGEST BIAS AND 'NONE' REPRESENTS NO BIAS.

avoidance. The proposed modification known as byte-counting increases the congestion window based on the number of bytes being acknowledged by each arriving ACK, rather than the number of ACKs. The purpose of this mechanism is to ensure that TCP rate-increase fairly represents the application rate. The mechanism is not enabled by default in Linux and has not been implemented in Windows XP. If enabled, it can reduce the performance gain of real-time small-packet flow such as VoIP, as the congestion window is incremented only after MSS-sized data has been acknowledged.

Figure 11 shows the 95% delay for small-flows competing with web flows in a packet vs. byte queue setting and empirically highlights the gain in favor of small packet flows for byte-based queues. This figure also shows that byte-based queues have a relatively significant impact than disabling byte-counting mechanism on the performance of a small-packet flow.

E. Sustainable FTP flows

A question of interest is how many FTP flows can be sustained with CBR-TCP flows going through a bottleneck router, if the x percentile delay of CBR-TCP flow must be below a certain threshold. Interactive CBR-TCP flows such as VoIP and video chat have low tolerance for delay, and hence a small delay threshold while streaming traffic can tolerate delays of few seconds. Using the bottleneck router configuration from Section IV-B, we run an experiment where five small and large CBR-TCP flows compete with varying number of FTP flows for a drop-tail router. The drop-tail router maintain its queue in units of packets and bytes. Figure 12 shows the 95% and maximum delay for small and large CBR-TCP flows.

Note that the delays of small and large-packet flows are not directly comparable due to the difference in bottleneck bandwidth and drop-tail queue size.

VI. DELAY REDUCTION APPROACHES

We discuss two application-level heuristics that can improve the delay of large-packet flows by reducing the delays induced by the congestion control mechanism.

A. Packet-Splitting

As described in Section V-B, TCP has a bias in favor of small packet flows. The bias for small packet flows comes from the fact that during network-limited periods, TCP can combine several application packets into one transmission segment. A question arises whether a large CBR-TCP flow can improve its delay performance by splitting every MSS-sized packet into smaller packets, while maintaining the same workload byte rate. We call this scheme *splitN*, where N is the packet-split factor.

Using our model and test-bed, we analyzed the performance of this scheme with various split factors for a wide-range of network environments. The results presented in Table VIII are for the bottleneck router considered in Section IV-B that maintains its queue in units of packets and bytes. We find that there is a tradeoff between the delay performance and the split factor. A split2 scheme gives the best performance under a wide-range of network settings, whereas schemes with a higher split factor yielded diminishing gains or performed even worse than a no-split scheme. The performance degradation occurs due to the increase in the burstiness of the flow.

Note that the performance of *splitN* is relatively better for a byte-based bottleneck queue. This stems from the fact that byte-based bottleneck queue are biased in favor of small packet flows, as explained in Section V-D.

There are at least two ways to implement a *splitN* scheme. One way is to split every generated packet into N packets and send them immediately. The other mechanism is to send the split portions of the packet over the packetization interval. In practice, this pacing may be difficult to achieve as packetization intervals are typically 20 ms or 30 ms and a split factor of four will require sending packets at a granularity of 5 ms or 7 ms, a challenging task with current operating-systems. Thus, we use the former approach to implement the *splitN* scheme.

Note that even though the *splitN* scheme does not change the workload byte-rate, the total byte-rate of a flow increases due to the overhead of TCP/IP headers for additional packets. Hence, a wide-scale adoption of such an approach runs the risk of degrading the performance of all flows due to higher network congestion.

B. Parallel-Connections

A straight-forward approach to improve the delay performance of a CBR-TCP flow is to stripe its load across multiple TCP connections. This scheme can be considered a 'hack' as it attempts to lower delays by increasing TCP throughput. A 'blind' parallel-connection scheme will send packets over multiple TCP connections in a round-robin fashion. This will improve performance if loss recovery is not dominated by timeout retransmissions, which is the common case observed in our traces. However, it can potentially degrade performance

	No heuristic		Packet-splitting (pq)				Packet-splitting (bq)				Parallel-conn	
Web flows	baseline		split2		split4		split2		split4		par2	par5
	LR (%)	95% delay	LR (%)	% Improv.	LR (%)	% Improv.	LR (%)	% Improv.	LR (%)	% Improv.	% Improv.	% Improv.
0	0.75	0.215	0.74	12.2	0.81	-15.4	0.73	50.4	0.65	43.3	41.1	75.7
10	0.81	0.265	0.72	27.2	0.91	13.1	0.83	44.5	0.64	51.7	46.8	80.5
25	0.90	0.289	0.77	19.7	1.10	-52.8	0.86	50.2	0.82	47.6	50.4	82.7
50	1.20	0.608	1.10	17.5	1.58	-90.8	1.32	1.12	1.21	21.4	73.6	91.7
100	2.91	4.886	2.88	-2.2	3.18	-13.3	3.27	-1.55	3.25	-3.73	91.6	94.8

TABLE VIII
THE RELATIVE PERFORMANCE OF PACKET-SPLITTING AND PARALLEL-CONNECTION SCHEME. 'PQ' AND 'BQ' ABBREVIATE PACKET-QUEUE AND BYTE-QUEUE RESPECTIVELY.

because load per-connection is reduced resulting in small window sizes (see Section V-C).

To overcome these shortcomings, we devised an 'intelligent' scheme which selects a connection for packet transmission that is not in the timeout state and has the smallest TCP send queue. While both schemes attempt to improve the delay performance by increasing the aggregated TCP throughput, the 'intelligent' scheme outperforms the 'blind' scheme because it dynamically avoids connections with large queues and in timeout states. Further, this strategy will cause the average load on non-timeout connections to be higher than that on timeout connections which in turn improves loss recovery efficiency.

We used the model to evaluate the performance of the 'blind' scheme over the range of network environments considered in Section IV. The blind scheme improved the performance only when the TCP's loss recovery was not dominated by timeouts and yielded diminishing gains when more than four connections were used.

We implemented the 'intelligent' parallel connection scheme in Linux because unlike Windows XP, it allows to sample the TCP state and send queue size. For simplicity we denote this scheme by *parN*, where N is the number of parallel-connections used. We evaluate the performance of *par2* and *par5* schemes for small and large-packet flows using the experimental settings of Section IV-B, and present the results in Table VIII. The RTT was 100 ms and the packetization interval was 20 ms, so the load per connections for par5 scheme is one ppr. Thus, the parallelization spectrum for a CBR-TCP flow ranges from a single flow to having as many flows as the packet rate per RTT.

Parallel-connection scheme outperforms the packet splitting scheme because it increases the aggregated throughput and does not introduce any additional traffic apart from connection setup and tear-down. par5 performs better than par2 because, on average, it has more non-timeout connections to choose from, which increases the overall throughput.

VII. DELAY-FRIENDLY GUIDELINES

We present delay-friendly guidelines for VoIP and live video streaming applications using TCP. We categorize them into TCP-level and application-level guidelines.

A. TCP-level guidelines

- Nagle's algorithm and delayed ACKs should always be disabled.
- To increase the loss efficiency of TCP, we suggest that SACK be enabled, limited-retransmit should be used, and congestion window validation during application-limited periods and byte-counting should be disabled.
- The initial window size should be set to four segments [33].
- Some operating systems (e.g., Linux) use the *ssthresh* value from last connection. We suggest that *ssthresh* inheritance be disabled.
- Application should set the TCP receiver buffer size such that the TCP transmission mechanism is only limited by TCP congestion-control mechanisms.

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.

B. Application-Level guidelines

- **Playout buffer** Applications such as VoIP and interactive-video have tight playout requirements whereas live video streaming may tolerate delays on the order of few seconds. Ideally, an application should dynamically set the playout buffer on the order of the network round-trip time. However, an application using TCP must be aware that TCP is a reliable and in-order delivery protocol and any loss it suffers adds a latency on the order of network round-trip time. For a CBR-TCP application, it at least takes $L + RTT + 3/f$[4] to recover from a single TDACK loss and at least $4 \times RTT$ to recover from a TO loss.

  If timeouts are uncommon and the losses TCP suffers are likely to be TDACKs, a CBR-TCP application can statically set its playout buffer to $L + RTT + 3/f$. Such a setting works rather well for small-packet flows such as VoIP, as during network-limited periods TCP can combine the transmission of several small packets into a single packet. This is a 'sweet spot' which masks TCP

[4] L is the one-way network latency and f is the packet sending rate

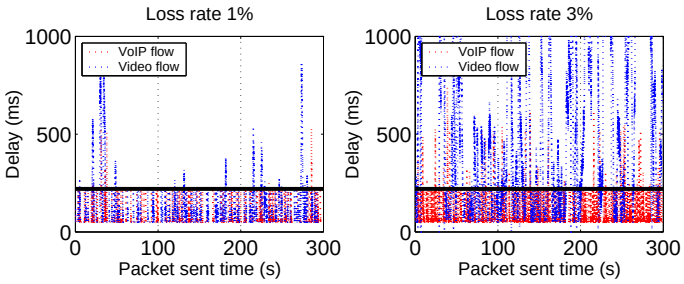

Fig. 13. Masking TCP delay variations for small and large packet flows for one and three percent losses. The round-trip link delay was 100 ms.

delay variations over a wide range of environments, as shown in Figure 13.

- **Large-packet flows** The delay performance of large-packet flows can be improved by parallelizing the flow over multiple connections. If supported by the underlying operating system, an application should take into consideration the TCP buffer size and congestion-avoidance state to decide which connection to use. Splitting MSS-sized packets by half is beneficial for large-packet flows if a single connection is preferred.
- **Silence suppression** can reduce the TCP congestion window during idle periods, and hence reduce the loss recovery efficiency of TCP [34]. We therefore suggest that an application should continue sending minimal traffic during silence periods to maintain its congestion window. Specifically, a CBR-TCP application with a rate of x packets per RTT should send at least x packets during silence periods. The size of the packets sent during silence periods can be one-byte if TCP ACK-counting mechanism is being used, which increases the congestion window based on the number of packets sent rather than bytes.
- **MSS to packet-size ratio** We recommend that the ratio of MSS to packet size be an integer so that the packet assembly capability of TCP will yield the lowest sending rate in pps during congestion.
- **Proactive packet drop** The queue buildup at the TCP sender can give good clues about delays packets will experience. By examining the queue size, an application can potentially infer the delays a new packet will experience. Thus, it may choose to drop packets at the sender, if the queue size crosses a certain inferred delay threshold.
- **In-order delivery mechanism** The delays of small packets tend to be dominated by in-order delivery mechanisms, i.e., during network-limited periods, packets lie in the receiver buffer waiting for 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. Although this modification requires changing the receive API to receive our-of-order packets, it does not change the network semantics of TCP.

VIII. Related Work

There is an extensive literature on analytical and experimental evaluation of TCP. We present only those studies closely related to ours (see [7] for a comprehensive survey). The majority of TCP modeling studies are geared towards file transfers assuming either persistent [6] or short-lived flows [8]. 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 [9]. The receiver buffer size requirement for TCP streaming has been determined in [10]. These papers combine a 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 constraints such as VoIP.

Goel et *al.* [13] 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 [14], [15]. 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 [34]. These heuristics are geared towards bursty traffic flows and hence may not be effective for real-time flows.

IX. 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 the model to predict the working region of these flows and verified it using a real-test bed and in Planet-Lab. We explored the impact of TCP mechanisms and presented system and application-level guidelines for improving the delay friendliness of CBR-TCP applications. The delay performance of a large-packet flow can be improved using packet-splitting or parallel-connection heuristic.

This study is the first-step in understanding the use of TCP for VoIP and live-video streaming applications, and by no means advocates the use of TCP over UDP for these applications. We have used delay and jitter to evaluate the performance of CBR-TCP flows. However, user-perceived performance typically evaluated through Mean Opinion Score (MOS) is a better metric. The user-perceived performance of real-time CBR-TCP flows is the subject of future work.

Appendix A
Model for Real-Time TCP Flows

We model a TCP connection with a CBR source using a finite state 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 $t_{(w,b,l);(w',b',l')}$ be the time taken for this transition. 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 is the ith element of the list and n is the list's length. For brevity, we use the notation $\{x\}^+ \triangleq \max\{0, x\}$. Our model accounts for CVW and non-CVW TCPs by either decaying the congestion window or leaving it intact when entering an application-limited state, as follows. For a CVW TCP, we use the notation

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

whereas for a non-CVW TCP we use

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

The CBR source is characterized as follows. Let f be the load in packets per second, $r = fRTT$ be the load in packets per round-trip time, and a be the packet size. Let L be the forward network latency and T_0 be the duration of the initial timeout. The state transition probabilities, the delays associated with the transitions, and the times taken for the transitions are given in Table IX. The states in this table are grouped into four categories that correspond respectively to the states of a TCP sender (a) application-limited (b) network-limited and loss-free (c) fast recovery (d) and retransmission timeout

REFERENCES

[1] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," RFC 3550, Jul. 2003.

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

[3] E. Kohler, M. Handley, and S. Floyd, "Designing DCCP: Congestion control without reliability," in *SIGCOMM*, Pisa, Italy, Sep. 2006.

[4] 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 Janeriro, Brazil, 2006, pp. 217–230.

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

[6] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: a simple model and its empirical validation," Vancouver, British Columbia, Canada, 1998, pp. 303–314.

[7] J. Olsen, "Stochastic Modeling and Simulation of the TCP Protocol," Ph.D. dissertation, Uppsala Univeristy, Oct. 2003.

[8] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP latency," in *IEEE INFOCOM*, Tel-Aviv, Israel, Mar. 2000.

[9] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia Streaming via TCP: an Aanalytic Performance Study."

[10] T. Kim and M. H. Ammar, "Receiver Buffer Requirement for Video Streaming over TCP," *Proceedings of SPIE*, vol. 6077, 2006.

[11] M. Handley, J. Padhye, and S. Floyd, "TCP Congestion Window Validation," RFC 2861 (Experimental), Jun. 2000.

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

[13] A. Goel, C. Krasic, K. Li, and J. Walpole, "Supporting low-latency TCPbased media streams," in *IWQoS*, May 2002.

[14] B. Mukherjee and T. Brecht, "Time-lined TCP for the TCP-friendly Delivery of Streaming Media," *ICNP*, pp. 165–176, 2000.

[15] D. McCreary, K. Li, S. A. Watterson, and D. K. Lowenthal, "TCP-RC: A Receiver-Centered TCP Protocol for Delay-Sensitive Applications," in *MMCN*, Jan 2005.

[16] M. Li, M. Claypool, R. Kinicki, and J. Nichols, "Characteristics of Streaming Media Stored on the Web," *ACM Trans. Inter. Tech.*, vol. 5, no. 4, pp. 601–626, 2005.

[17] W. R. Stevens, *TCP/IP Illustrated*. MA: Addison-Wesley, Nov. 1994, vol. 1.

[18] K. Chen, C. Huang, P. Huang, and C. Lei, "Quantifying Skype User Satisfaction," in *SIGCOMM'06*, Pisa, Italy, september 2006, pp. 399–410.

[19] S. Tao, J. Apostolopoulos, and R. Gu, "Real-time Monitoring of Video Quality in IP Networks," in *NOSSDAV*, Stevenson, Washington, USA, 2005, pp. 129–134.

[20] A. Wierman, T. Osogami, and J. Olsen, "A Unified Framework for Modeling TCP-Vegas, TCP-SACK, and TCP-Reno," in *MASCOTS'03*, Oct. 2003, pp. 269–278.

[21] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," RFC 2581, Apr. 1999.

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

[23] A. Medina, M. Allman, and S. Floyd, "Measuring the evolution of transport protocols in the internet," *SIGCOMM*, vol. 35, no. 2, pp. 37–52, 2005.

[24] R. W. Wolff, *Stochastic Modeling and the Theory of Queues*. New York: Prentice-Hall, 1989.

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

[26] K. Beomjoon, C. Yong-Hoon, and L. Jaiyong, "An extended model for tcp loss recovery latency with random packet losses(network)," *IEICE transactions on communications*, vol. 89, no. 1, pp. 28–37, 2006.

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

[28] 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.*, vol. 34, no. 2, pp. 137–151, 2004.

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

[30] "Nist Net. http://www-x.antd.nist.gov/nistnet/."

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

[32] J. Postel, "The TCP Maximum Segment Size and Related Topics," RFC 879, Nov. 1983.

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

[34] A. Mondal and A. Kuzmanovic, "When TCP Friendliness Becomes Harmful," in *IEEE INFOCOM*, May 2007.

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

TABLE IX

TCP DELAY MODEL: DEFINITION OF THE STATE TRANSITION PROBABILITIES, TIMES TAKEN FOR THE TRANSITIONS AND THE DELAYS ASSOCIATED WITH THE TRANSITIONS.