The Delay-Friendliness of TCP for Real-Time Traffic

Eli Brosh, Salman Abdul Baset, *Graduate Student Member, IEEE*, Vishal Misra, *Member, IEEE*, Dan Rubenstein, *Member, IEEE*, and Henning Schulzrinne, *Fellow, IEEE*

***Abstract*—TCP has traditionally been considered inappropriate for real-time applications. Nonetheless, popular applications such as Skype use TCP since UDP packets cannot pass through restrictive network address translators (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.**

***Index Terms*—Live video streaming, measurement, performance modeling, TCP congestion control, VoIP.**

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 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 [19], [22], [32] 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 commercial streaming traffic is carried over TCP [18]. Popular media applications such as Skype [8] and Windows Media Services [18] use TCP to pass through restrictive network address translators (NATs) and firewalls that block UDP traffic. Furthermore, TCP is by definition TCP-friendly [19] 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 on top of TCP? 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 a broad set of aspects of the performance of real-time applications, we conduct an extensive performance study using both 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, it is geared toward the performance of file transfers [12], [30], [31] and video streaming [21], [35] from the standpoint of throughput rather than that of delay.

We use both test-bed and Internet experiments to validate the model over a wide range of network environments. 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 [20] and limited transmit [5]. The results obtained yield guidelines for delay-friendly TCP settings and may further be used to compare the performance of TCP with alternative protocols [19], [22] and experimental real-time enhancements for TCP [17], [24], [27]. We analyze two application-level schemes—namely, packet splitting and parallel connections—that we find significantly reduce the delay of live video streaming flows by as much as 30% and 90%, respectively.

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 also reduces the resulting TCP delay.

The main contributions of this paper are the following.

- To the best of our knowledge, we are the first to present a discrete-time Markov model of the delay distribution of a real-time TCP flow (Section IV).

Manuscript received February 27, 2009; revised October 31, 2009; accepted February 02, 2010; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor C. Dovrolis. Date of publication June 28, 2010; date of current version October 15, 2010. This work was supported in part by the National Science Foundation under grants 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 authors are with the Department of Computer Science, Columbia University, New York, NY 10027 USA (e-mail: elibrosh@cs.columbia.edu; salman@cs.columbia.edu; misra@cs.columbia.edu; danr@cs.columbia.edu; hgs@cs.columbia.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2010.2050780

- We derive the working region for VoIP and live streaming flows based on our model and experiments (Section VI-A). We find that under the same network conditions, VoIP flows suffer from lower TCP delays than live video streaming flows. 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 mechanisms in TCP on the TCP delay (Sections VI-C–VI-E). We then provide TCP-level guidelines (Section VI-G) and simple application-level heuristics (Section VII) for improving the performance of real-time applications. We find that using parallel connections with shortest-queue-first policy achieves up to 90% delay reduction.

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 [36]. Although our analysis is general, we focus on CBR sources corresponding to VoIP and live video streaming, as detailed in Section V. Furthermore, we also discuss the applicability of our analysis to variable bit-rate (VBR) flows in Section VI-B. Unlike greedy flows, such as FTP, where the source rate is limited by the network, the sending rate of VoIP flows is a function of media encoding and, thus, may or may not be network-limited. We refer to the periods where the source rate is not limited by the network as application-limited periods. Specifically, in an application-limited period, the TCP throughput satisfies the source's rate requirement.

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 [33]. 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 on the order of seconds [18]. For VoIP, low delay of no more than 400 ms is required in order to maintain acceptable interactivity [17]. 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 [14], [34]. 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 time, 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.

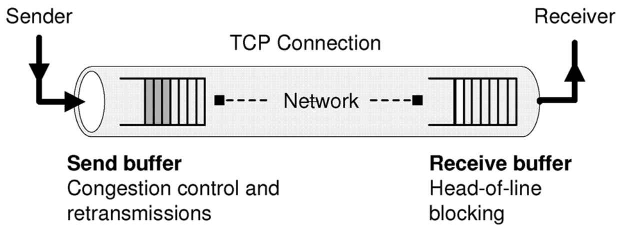

Fig. 1. Transport-layer queueing delays.

III. 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, as depicted in Fig. 1: 1) network delay, which is the time it takes a segment to get across the network; 2) 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 TCP's mechanisms can be found in [33].

TCP uses two buffers to provide congestion-controlled reliable data delivery; a send buffer and a receive buffer. The send buffer serves two functions [17]. 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 for possible retransmission. 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. 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.

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 [33]). Applications usually minimize this backlogging by setting a large receive buffer and operating with nonblocking sockets. Packet backlogging can also occur due to Nagle's algorithm [29] 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

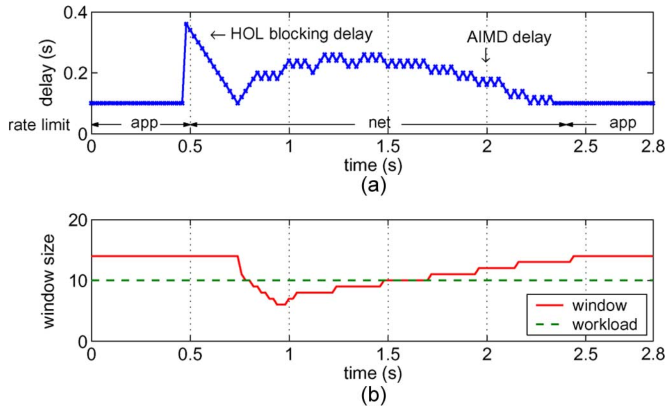

Fig. 2. Evolution of the TCP delay and congestion window size for a video-like CBR source. Time 0 refers to a point when the congestion window has already stabilized. (a) TCP delay. (b) Congestion window size.

practice, many delay-sensitive applications disable this algorithm to reduce transmission delays [39]. We follow this practice in our work.

Fig. 2(a) illustrates the delay experienced by 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 to 0.5 s and from 2.4 to 2.8 s. In this period, the TCP delay is determined by the network delay. A network-limited period is seen from 0.5 to 2.4 s. During this time, 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 to 0.76 s, and packet backlogging, seen from 0.76 to 2.4 s. TCP uses retransmission to recover the lost packet, which in turn causes HOL 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 Fig. 2(b). The mismatch between the input and output rates at the TCP sender results in the quadratic-like delay curve seen in Fig. 2(a). Later, at time 2.4 s, TCP is again application-limited when the rates are matched.

A. 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 [7], there are two permitted approaches. 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 acknowledgment (ACK) arrival, a mechanism known as ACK counting. We compare the performance of ACK- and byte-counting mechanisms (Section VI-D)

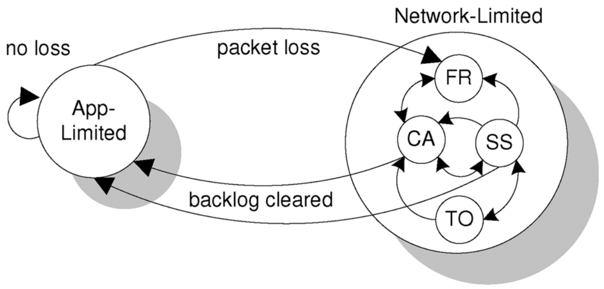

Fig. 3. High-level view of a model for a TCP connection with a CBR source.

and focus on the former due to its wide deployment [25], as also verified by our measurements.

IV. 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 [20] to accurately capture the retransmission timeout probability. Third, we capture the effect of window inflation [7] and the limited transmit mechanism [5] 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 [25], and refer the interested reader to [15] and [33] for a detailed description of TCP NewReno's mechanisms.

A. 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 III.

Mimicking the behavior of a real-world TCP flow, our model consists of two main states: *application-limited* and *network-limited*. The system 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 system 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 system 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 Fig. 3.

We make several simplifying assumptions in our model, as follows. First, we assume that TCP increases the congestion window by one *packet* per RTT, an assumption motivated by the wide deployment of ACK-counting TCP implementations [25]. Second, we assume that the TCP implementation does not

TABLE I
SUMMARY OF MODEL NOTATIONS

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)

increase the congestion window when the TCP sender is application-limited, which is the behavior observed for Linux and Windows XP systems (see Section VI-E). Third, we assume that the slow start threshold is statically set to half of the source's sending rate in packets per RTT. We expect it to stabilize at this value when most losses are not timeouts. From our experience, using a static SS 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 ACKs. Nonetheless, our model can be easily extended to support delayed ACKs using a similar approach to what is done in [31].

Our model characterizes the CBR source 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 RTT. For convenience, we summarize the notations used in this paper in Table I. 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 permits 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. Note that successful delivery happens when the entire window of data is delivered rather than a single packet. Each transition represents a certain number of packet transmissions, and each packet in this transmission experiences 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 data needs to be recovered $(l > 0)$ or not $(l = 0)$. The backlog size value is used to indicate whether the sender is application-limited $(r \leq w, b = 0)$ or network-limited ($0 < w, b \neq 0$ or $w < r, b = 0$). 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 backoff stage. Note that although TCP never uses a window size of zero, it is convenient to use this value to represent timeout states and exponential backoff states since no packets are sent during these periods. Table II lists the rules for classifying an arbitrary state $s = (w,b,l)$ according to the congestion control phases of TCP. We use the notation $AL = \{(w,b,l) : r \leq w, b = 0, l = 0\}$ to denote the set of states for which the application-limited condition holds. The notations CA, SS, FR, and TO are defined in a similar way, as shown in Table II.

TABLE II
STATE CLASSIFICATION

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

B. Delay Performance Model

In this section, we model the three ways in which TCP introduces delays: congestion control, retransmissions, and HOL blocking, as detailed in Section III. We model the time packets buffered at the sender before being transmitted (i.e., the congestion control delay) by scaling the backlog size by the sources's rate. This delay model follows from the observation that the unsent packets left behind after a packet transmission must have arrived to the send buffer while the transmitted packet was being 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 modeling approach introduces an error on the order of several packetization intervals because it captures the backlog size evolution in network-limited states at RTT granularity. The error can be reduced by keeping track of the intersending packet times. However, this will make the state space of the model prohibitively large and hence will limit its usefulness.

We determine the HOL 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 the 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 [12].

Using the above observations, we express the TCP delay of the ith packet sent in a transition from state s to state s' as

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

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 second 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, packets are not sent in transitions to timeout states, and consequently there is no associated delay.

The number of CBR packets sent in a transition from s to s' $n_{s;s'}$ is given by

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

Since our model evolves at packet-level granularity while in an application-limited state (see Fig. 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 [7].

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.

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

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 $\arg\max_x P(D \le x) \le \alpha$, along with other useful statistics such as the mean delay $E[D]$. Note that the input parameters to the model, RTT and network loss rate (p), factor into the numerical computation through the probability transmission matrix $q_{s;s'}$ that captures the probability of packet loss as well as through (1) that represents the delay of a transmitted packet.

We remark that 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 backoff timeout states and w_m nontimeout states with unique window sizes, the size of the state space is $6b_m + 2b_m w_m$. For the numerical computation, we use $w_m = 8r$ and $b_m = 6T_0 f = 24r$. This state space enables us to efficiently evaluate the TCP delay for the range of network environments considered in Section V.

C. Backlog and Congestion Window 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 RTT in the congestion avoidance phase and doubles the window size every RTT in the slow-start phase. However, it is pretty typical for TCP implementations to not increase the congestion window when the TCP sender is application-limited (see Section VI-E). 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 [11].

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

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

where $t_{s;s'}$ is the time taken for the transition from s to s', which can be found in [11]. 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).

V. Model Validation

We evaluate 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 "short-lived" for a TCP connection with short-lived bursty traffic.

We wrote a tool that can send and receive bidirectional CBR over TCP flows with different packet sizes and different packetization (intersending time) intervals. To validate our model, we use CBR sources with packet sizes of 174, 724, and 1448 bytes, and packetization intervals of 20 and 30 ms, as these choices approximately reflect typical one-way voice [32], low-bit-rate interactive video [17] and live video streaming [18]. The size of the packet includes a 12-byte RTP header [32] and 2 bytes for framing RTP packets over TCP [23]. Hence, excluding the header size, the bit rate of the voice flows is 64 and 42 kb/s,

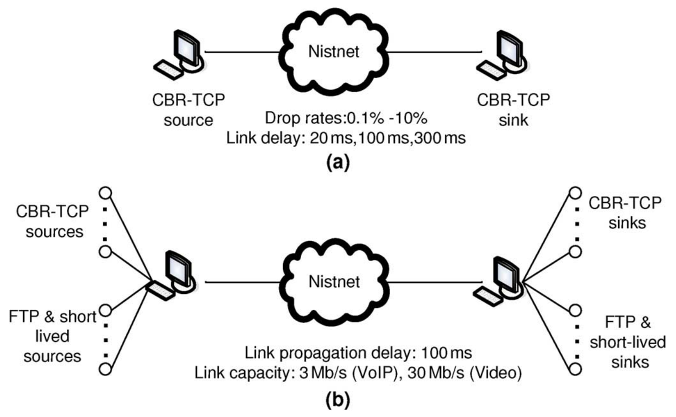
Nistnet
CBR-TCP source
CBR-TCP sink
Drop rates:0.1% -10%
Link delay: 20 ms,100 ms,300 ms
(a)
CBR-TCP sources
FTP & short lived sources
CBR-TCP sinks
FTP & short-lived sinks
Link propagation delay: 100 ms
Link capacity: 3 Mb/s (VoIP), 30 Mb/s (Video)
(b)

Fig. 4. Experiment setup for model verification in a controlled environment. (a) Configured drop rates. (b) Drop-tail queue.

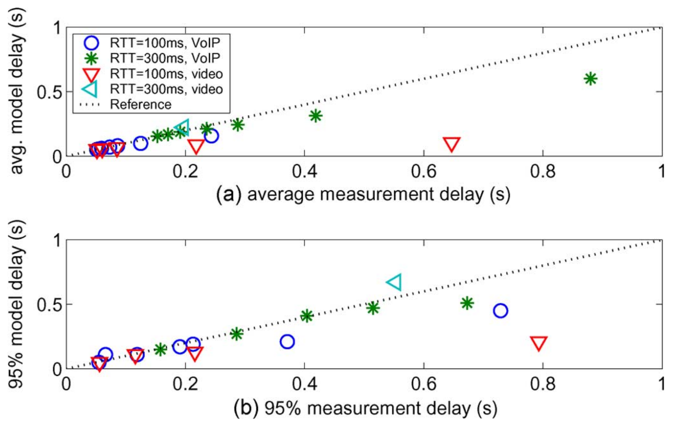

Fig. 5. Predicted versus measured (a) mean delay and (b) 95th percentile TCP delay (in seconds) for VoIP and video flows for various loss rates and RTT of 100 and 300 ms.

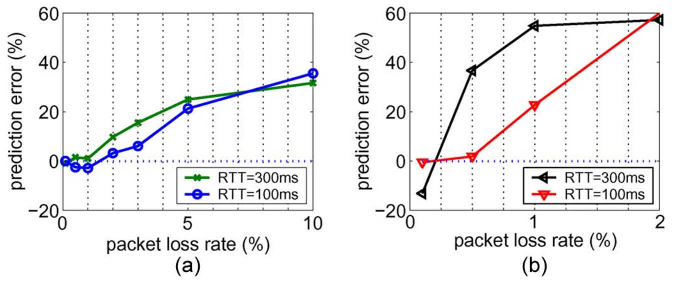

Fig. 6. Relative modeling error for (a) VoIP and (b) video flows as a function of loss rate.

that of interactive video is 284 and 187 kb/s, and that of live video streaming is 573 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, and due to lack of space, only present the results using these parameters. Excluded results are available in [11]. 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 conducted using Linux machines. The system and session-level TCP settings were determined according to the configuration described in Section VI-G.

A. Model 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 Fig. 4(a). We consider a single CBR-TCP flow going through a router running NIST Net [2], 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 irrespective 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, 100, and 300 ms. These delay settings roughly reflect the delay of sites on the same coast in the U.S., U.S. coast-to-coast delays, and transcontinental delays [17]. Note that the network environment does not include background traffic, so there are no queuing delays on the round-trip path. We do not consider loss rates greater than 10% because the average TCP throughput (i.e., the available network bandwidth), as estimated by Padhye’s equation [31], does not satisfy the rate requirement of the CBR-TCP flow for the considered RTTs. For each set of parameters, we ran the experiment for 5 min and repeated each experiment 10 times. We present the average results of these experiments and compare them to the ones obtained using our model. The model assumes random packet losses (see [11]), similar to the considered environment.

Fig. 5(a) and (b) present the predicted versus measured mean and 95th percentile TCP delay, respectively, for VoIP and video flows for various network packet loss rates and RTTs of 100 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. 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 Fig. 6. 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 on the order of 20% for loss rates up to 1% and 0.1%, and RTT of 100 and 300 ms, respectively. The increase in relative error 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.

There are several explanations for the modeling mismatches. First, there are the simplifying assumptions made by the model, such as the recovery of multiple losses in a single transmission window within one RTT, which introduce error. Second, although our model accurately captures the backlog size at RTT granularity, it ignores backlog evolution at smaller time scales. A more detailed discussion of these issues can be found in Sections IV-A and IV-B.

Observe from Fig. 6 that the prediction error increases with the network loss rate. This is because the size of the model’s

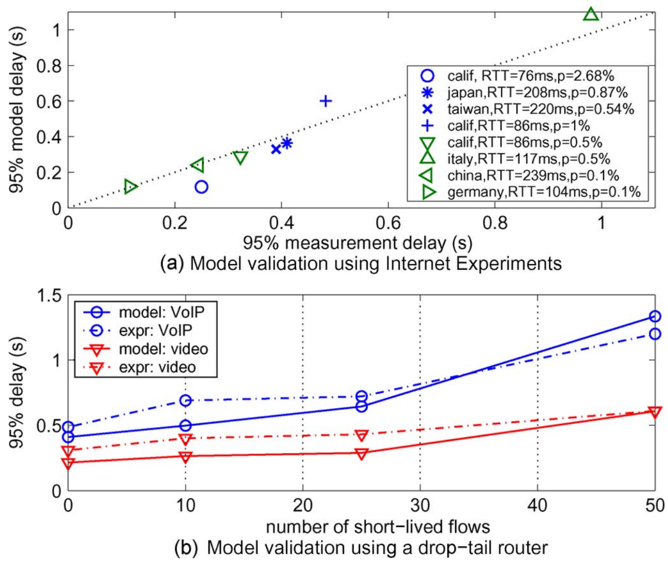

Fig. 7. Model validation using (a) PlanetLab experiments and (b) drop-tail router experiments for VoIP and video flows.

state space is truncated to reduce computational complexity, as explained in Section IV-B. Furthermore, for video flows, the jump in the prediction error at loss rate of 0.5% and 1% for RTT of 100 and 300 ms, respectively, occurs because the achievable TCP throughput is close to or below the bit rate of the video flow. When the CBR rate is close to the TCP throughput, the TCP connection increasingly exhibits sawtooth-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 is 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.

B. Model 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 U.S. (California, New York, Texas), Europe (Germany, Italy, U.K.), and Asia (China, India, Japan, Taiwan). For each sender and receiver pair, we ran our tool to generate VoIP and video flows for 30 min. The DSL experiments were conducted from hosts in the U.S., 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-TCP flows, thereby increasing the congestion on the link and causing higher loss rates for CBR-TCP flows. Fig. 7(a) plots the predicted versus measured 95th percentile delay for VoIP and video flows for a range of sites around the world. All the sites shown had FTP flows running in tandem to increase the link congestion. The top four entries 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 seen by the flows are also indicated. The figure shows a good match between the model and the measured delay.

C. Model Validation Using Drop-Tail Routers

We consider a scenario where multiple CBR-TCP flows compete with FTP and short-lived bursty flows for a bottleneck router with a drop-tail queueing scheme, as shown in Fig. 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.

We used the test-bed from Section V-A and modified NIST Net to incorporate a drop-tail queue. We devised a multiflow setting in which five VoIP CBR-TCP flows compete with five long-lived FTP flows and a varying number of short-lived flows. We repeated the experiment for video flows. We used the SRI and ISI traffic generator [3] to generate exponentially distributed short-lived flows with a mean duration of 50 ms and a constant packet size of 512 bytes. The choice of the number of FTP and short-lived flows and packet size for short-lived flows was inspired by the configuration used to evaluate the performance of TFRC small-packets [16]. The round-trip propagation delay was set to 100 ms for all experiments. The link capacity was set to 3 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 1:10.

The queue at the bottleneck router may be maintained in packets or bytes [22]. If the router maintains its queue in bytes, then small packets are less likely to be dropped than large packets. The preferential drop may cause TCP flows with small packets to experience lower delay than those with large packets. Due to lack of space, we only present the delay results for the drop-tail queue maintained in packets and refer the reader to [11] when the drop-tail queue is maintained in bytes. We configured the packet-based drop-tail queue to hold 100 packets. Although, this choice introduces different queueing delays for link capacities of 3 and 30 Mb/s, our focus is on comparing the results predicted by the model to the experimental results and not a delay comparison between VoIP and video flows. Fig. 8 provides an effective comparison of delay results for VoIP and video flows across similar loss rates and network delays. For each configuration, we ran the experiment for 5 min and repeated it five times, and we present the average of the results.

For each experimental data point shown in Fig. 7(b), we obtained the input parameters for the model, i.e., network loss rate and RTT, which are averaged over five runs. The network RTT consists of round-trip propagation delay of 100 ms and the queuing delay at the router. For the VoIP and video experiments, the network delay of VoIP and video flows was more than 300 and 100 ms, respectively, across all data points. The delays are different because the link bandwidth is different in the two settings. Some of the prediction inaccuracies in the drop-tail

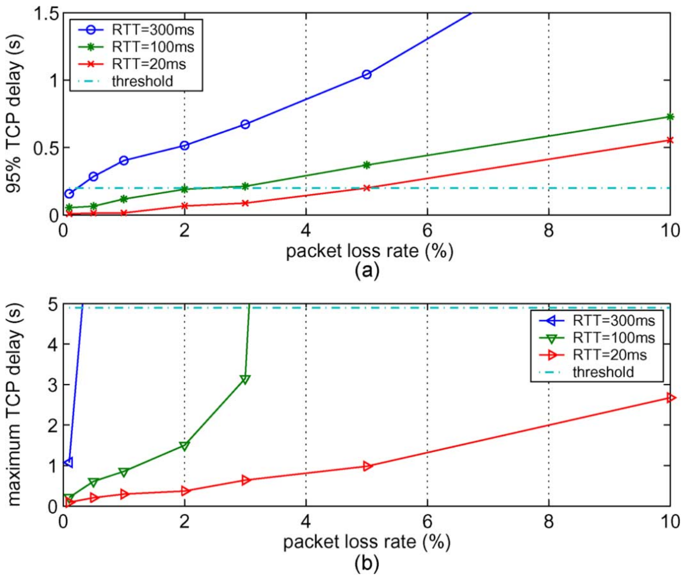

Fig. 8. Working region for (a) VoIP and (b) video streaming as a function of RTT and packet loss rate.

router experiments are caused by inaccurate characterization of the loss process. In this environment, as in our PlanetLab experiments, the model assumes correlated packet losses (see [11]). However, the actual burstiness of losses experienced by the CBR-TCP flows varies depending on the level of statistical multiplexing at the router.

VI. Discussion

In this section, we explore the 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 and 573 kb/s, respectively. We use the model to demonstrate that the working region for other bit rates has a similar characterization. Then, we study the impact of various mechanisms in TCP on its delay performance and use the model to gain an insight into the reason why VoIP flows typically perform better than video flows under the same network conditions. Finally, we use the insights gained to provide guidelines for configuring TCP for real-time applications.

A. 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 [28]. The choice of the delay limit for live video streaming 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 [18]. While VoIP can tolerate up to 5% of packets that miss their playout deadline without a significant effect on intelligibility [28], video viewing quality drops rapidly at 0.1% packet loss [35]. 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 VI-F.

Fig. 8(a) plots the 95th percentile delay for various loss rates from 0.1% to 10% and RTTs of 20, 100, 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 V-A. 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. Fig. 8(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 an 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 V-A.

The bit rates of 64 and 573 kb/s are the highest among the bit rates considered in Section V for VoIP and video 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, the results follow similar pattern as in Fig. 8. Furthermore, the working region can be significantly constrained if the application does not use delay-friendly TCP settings, as discussed in Section VI-G.

B. Performance of VBR Flows

So far, the discussion has focused on modeling and analyzing TCP delay for CBR VoIP and live video streaming flows. However, such flows may not necessarily be CBR. In this section, we extend our discussion and analysis to variable rate flows.

The key to our analysis is characterizing the sources of variability in VoIP and live video streaming flows. For VBR VoIP flows, the bit rate can be broadly characterized by two types of variabilities: 1) switching between different encoding rates of a codec due to network conditions [1]; 2) idle periods due to silence suppression. In the former case, the analytical model can be applied separately for each encoding rate to compute the TCP delay and working region. The working region for multiple bit rates can then be determined by the most constrained region among those computed. In the latter case, since VoIP flows may have idle periods, TCP may reduce its window size, requiring it to ramp up when sending the subsequent talkspurt, which in turn may introduce additional delay. Although our model and delay analysis do not capture these idle periods, it is not strictly necessary, as it is in the applications' interest to transmit packets during silence periods to maintain TCP congestion window. For example, this can be accomplished by sending three packets per RTT during idle periods, as suggested by [26].

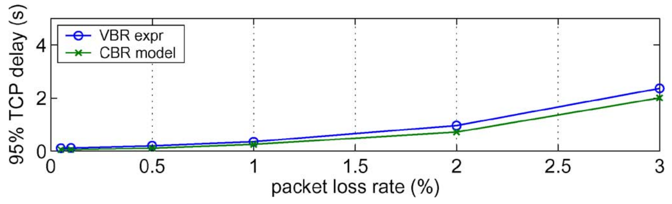

Fig. 9. Comparison of the TCP delay for a measured VBR flow and the delay predicted by our model.

For VBR live video streaming flows, the traffic bit rate is dynamically adjusted to capture the rate of change in the scene. In addition, like VoIP flows, a VBR source may switch between different encoding rates. To accurately capture this behavior, it is necessary to incorporate a multiparameter VBR source model, which is beyond the scope of this paper. Instead, we explore a simpler question: How does the delay of VBR flows that dynamically adjust their rates compare to the delay of CBR flows with an equivalent bit rate?

We study this question using real-world VBR flows. We obtained each VBR flow by establishing a video call between two Skype clients. The caller's camera captured a music video running on another machine and directly transmitted it to the callee Skype client for a duration of 8 min. We examined four VBR Skype flows, two TCP and two UDP with different bit rates. For each flow, we captured the payloads, fed them into our TCP sender, and obtained TCP delays for a range of network loss rates and propagation delays. We found that the results follow a similar trend. We thus show the results for a single VBR flow. The average and standard deviation of bit rate, packet size, and interpacket time of this flow are 605 and 25 kb/s, 1082 and 480 bytes, and 14 and 16 ms, respectively. We ran the model with a CBR flow that has the same bit rate as the VBR flow. We set the model parameters, packet size and interpacket gap, according to the average values of the VBR flow.

In Fig. 9, we show the 95th-percentile delay results for the VBR flow, as well as that predicted by our model, for an RTT of 100 ms and varying loss rates. As expected, the delays increase with loss rates. We observe that the characteristic form of the predicted delay is similar to the experimental one. We repeated the experiment for a large spectrum of network loss rates (0%–3%) and RTTs (20–300 ms) and observe similar behavior. We also observe that the model underestimates the measured delay. The underestimation occurs because the model does not capture the variability in VBR flows. Although not shown, the relative error of the model (in comparison to the measured results) is high for low loss rates and decreases as the loss rates increase. For example, for 0.1% loss rate, the predicted and the measured delay are 50 and 110 ms, respectively, whereas for loss rate of 3%, the predicted and the measured delay are 2.05 and 2.35 s. The relative error at low loss rates is high because the packet backlog at the sender is small and the TCP delay is dominated by the inherent variability of the video source, which is reflected in the packet sizes and the interpacket times. At high loss rates, the model's accuracy improves because the TCP delay becomes more dominated by the packet backlog at the sender. The backlog is similar for the VBR and CBR flows because they have the same average bit rate.

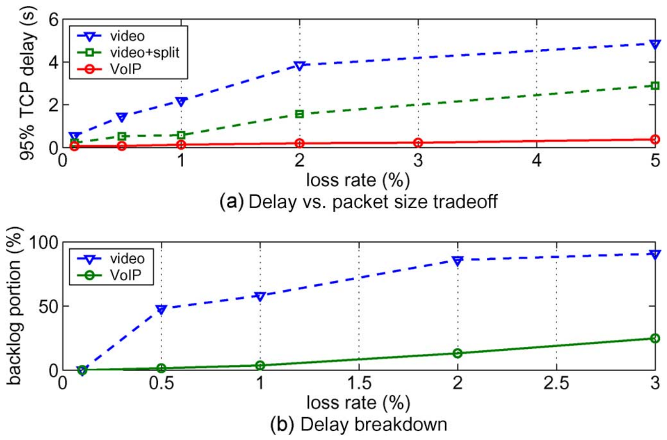

Fig. 10. (a) Delay performance of two TCP flows having the same load in kb/s but different load in pps, and a VoIP flow. (b) Delay breakdown: the portion of TCP-level delays caused by the congestion control mechanism.

In general, although the source variability is not captured by our model, we observe that our model is relevant in understanding the TCP delays of VBR flows. This is because the delays predicted by the model follow the same trend as the measured ones and the relative error is low for loss rates greater than 1%. It can be argued that for video traffic the main concern is the video frame delay (which comprises multiple packets) rather than the packet delay. While this may be true, our analysis is focused on characterizing the TCP delay for packets. The Interframe delay analysis is left for future work.

C. *Effect of Packet Size on Performance*

Our experiments indicate that under the same network conditions, VoIP flows perform significantly better than video flows. Fig. 10(a) plots the 95% delay for VoIP and video flows with the same workload in packets per second (pps), but substantially 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 toward 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 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 is the following: Among two flows having the same workload in kb/s, does TCP have a performance bias toward a flow with a 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 workloads in pps. The results are shown by the curves labeled video and video+split in Fig. 10(a). 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 toward the flow with twice the packet rate of the other flow.

We illustrate the reason for this performance difference in Fig. 11, which plots the TCP delay and congestion window

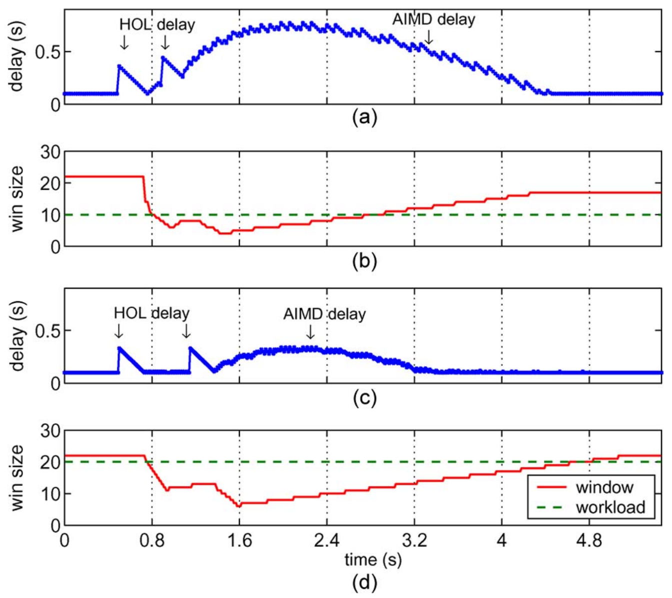

Fig. 11. TCP delay and congestion window evolution for two flows with the same workload in kb/s but different packet sizes, MSS and half-MSS. (a) TCP delay, MSS packets. (b) Congestion window size, MSS packets. (c) TCP delay, MSS/2 packets. (d) Congestion window size, MSS/2 packets.

size for two flows with the same application-level workload in kb/s. The flow in Fig. 11(a) and (b) corresponds to an application that sends 100 MSS-sized packets per second. The flow in Fig. 11(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 a pair of nearby losses. Observe that the flow with half MSS-sized packets experiences lower delay than the other. This happens because the AIMD mechanism updates the congestion control state as a function of the number of *packets* ACKed, rather than as a function of the number of *bytes* ACKed (see Section III-A). Since TCP adapts its congestion control state, and hence its throughput based on the number of packets ACKed, 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 Fig. 11 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 from a larger packet backlog than VoIP flows. However, reducing the packet size has side effects such as increased instances of packet reordering [25]. We analyze the breakdown of TCP-level delays by computing the time packets 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 HOL delay components). Fig. 10(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.

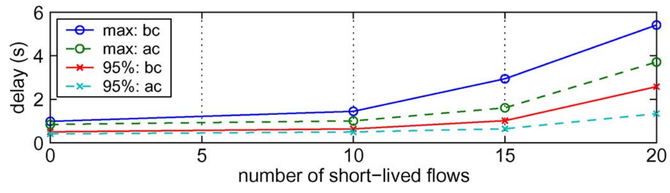

Fig. 12. 95% and maximum delay for a VoIP flow using ACK and byte counting.

D. Sensitivity to Byte Counting

In order to provide a measured response to ACKs that cover only small amounts of data, [4] 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. This begs the question: How does the performance of VoIP flows change when TCP increases its congestion window by the number of bytes sent?

We address this question by measuring the delay of five VoIP flows competing with five long-lived TCP flows and varying number of short-lived bursty flows in a drop-tail queue environment (see Section V-C). Fig. 12 shows the 95th percentile and maximum delay for a VoIP flow with ACK and byte counting. As shown, the use of byte counting degrades the performance. On average, it increases the TCP delay by 10%–20%. The delay increases because TCP with byte counting increases its transmission rate in proportion to the number of bytes sent rather than the number of packets 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.

E. 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 also becomes small. 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 affected by the behavior of TCP during application-limited periods. According to [20], 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 [20]. 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

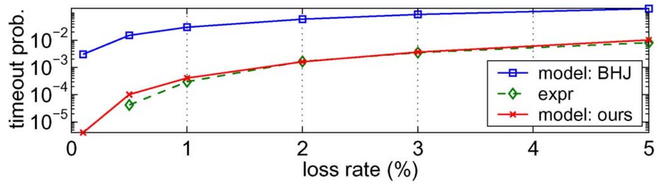

Fig. 13. Timeout probability predicted by our model, Beomjoon *et al.* model (BHJ), and the measured probability for various loss rates.

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 Fig. 11(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 [5], 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.* [9]. The latter model does not consider the impact of the two mechanisms. The derivation of the timeout probability for our model is given in detail in [11]. We show the results in Fig. 13 for an average window size of 3. To validate the results, we also measured the timeout probability of several VoIP flows that send three packets per RTT 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 nonnegligible impact on the timeout probability.

F. Playout Buffer Size Setting

Real-time applications use a playout buffer to compensate for variability in network delay. The receiving application 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 adds transport-layer delay for lost packets. As explained in Section IV-B, 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, often approximated by $4RTT$ [19]. Hence, the minimum playout delay is determined by the minimum of these two thresholds plus the one-way network delay L.

We investigate the sensitively of the application to the playout delay value by measuring the fraction of packets that miss their deadlines, hereafter referred to as *late packets*, as a function of the playout delay. The playout delay remains fixed throughout the lifetime of a flow. Fig. 14 shows the results for VoIP and live video streaming flows obtained in the network environment described in Section V-A. 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. Furthermore, it indicates that VoIP flows require smaller playout delays than video flows for the same fraction of late packets. This happens because video flows suffer from larger packet backlog than VoIP flows (see Section VI-C). In these experiments, TCP can recover from a packet loss using fast retransmit within $1.5RTT + 60$ ms because the packetization interval is 20 ms and the network delay is symmetric. Setting the playout delay for VoIP and video flows to this threshold results in up to 5% of late packets for RTTs of up to 100 ms and network loss rates of up to 2%.

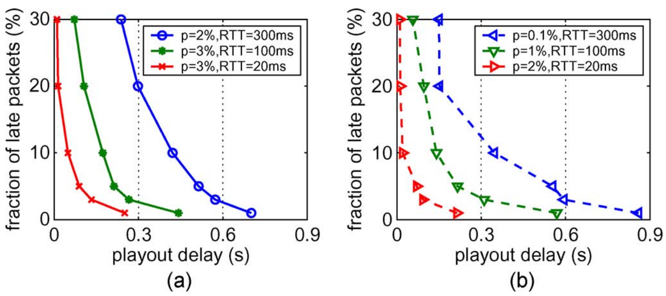

Fig. 14. Required playout delay for (a) VoIP and (b) video flows and various RTTs and loss rates (p).

We use the model to evaluate the effectiveness of this playout setting for the working regions defined in Section VI-A. 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.

G. TCP and OS Settings

We provide a comprehensive set of guidelines for delay-friendly settings of 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.

As discussed in Section III, 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 nonblocking sockets so that the TCP transmission is not limited by the flow control mechanism. To increase the loss efficiency of TCP, SACK should be enabled [13] and limited transmit be used. The latter is helpful for a TCP connection with small windows. Byte counting and congestion window validation during application-limited periods and should be disabled. The initial window size should be set to four segments as it can remove delays of up to three RTTs and a timeout during the initial slow-start period [6].

VII. 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.

A. Packet Splitting

As described in Section VI-C, 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 number of small packets generated. 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. Furthermore, reducing the packet size can increase instances of packet reordering [25].

We analyze 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 V-A. The *split-2* scheme reduced the 95th delay percentile by 60% on average. See [11] for detailed results. This is consistent with the observation made in Section VI-C 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* within 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, as described in Section V-C. As shown in Fig. 15(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 short-lived flows), a TCP sender using a *split-N* scheme is heavily backlogged and is hence unable to improve performance using the *split-N* 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 Fig. 15(b), the *split-N* scheme impacts the throughput of the

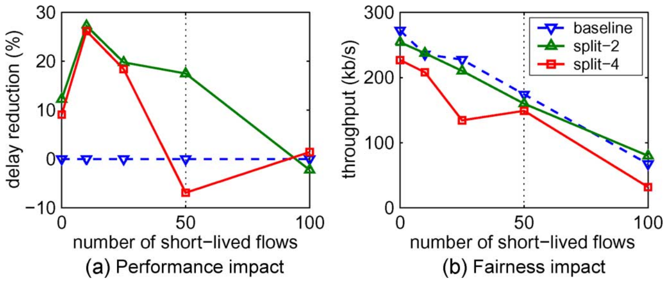

Fig. 15. (a) Performance impact. Reduction in the 95th delay percentile of a video flow using *split-N* in a drop-tail queue environment. (b) Fairness impact. Throughput of background FTP flows in the same environment.

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 reduces quickly with the split factor.

B. 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 [36], which can result in lower TCP delays. Previous exploration of parallel TCP connections for streaming and data-intensive applications has focused mainly 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 a 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 show the performance improvement and impact on fairness of this scheme for video flows using the drop-tail queue environment described in Section V-C. Fig. 16(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 HOL blocking per connection, and hence the TCP delay. The performance gain was negligible for VoIP flows. See [11] for detailed results. 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 VI-E). The small congestion windows are due to the low load per connection. We note that the scheme yielded diminishing gains when more than five connections were used.

We propose a delay-aware ("intelligent") scheme that selects a connection for packet transmission that has the smallest TCP send queue and is not in the timeout state, and we show the results in Fig. 16(a). The "intelligent" scheme outperforms the "blind" scheme because it dynamically avoids connections with large queues and in timeout states. Furthermore, 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

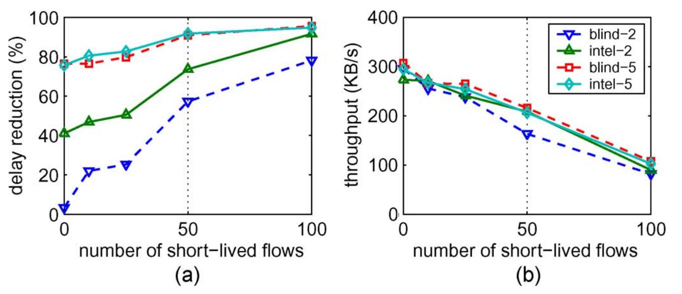

Fig. 16. (a) Performance impact. Reduction in the 95th delay percentile for a "blind" and an "intelligent" scheme with N connections in a drop-tail queue environment. (b) Fairness impact. Throughput of background FTP flows in the same environment.

ranges from a single flow 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 Fig. 16(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.

VIII. Related Work

There is extensive literature on analytical and experimental evaluation of TCP. We present only those studies closely related to ours and refer the reader to [30] for a comprehensive survey of TCP modeling. The majority of TCP modeling studies are geared toward file transfers assuming either persistent [31] or short-lived flows [12]. Our work differs from past work in that we consider nongreedy rate-limited flows with real-time delivery constraints. More recently, the performance of TCP-based video streaming has been analytically analyzed by [35]. The receive buffer size requirement for TCP streaming has been determined in [21]. 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.* [36] have performed a comprehensive analytical study of the performance of multipath video streaming using TCP. Their 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.* [17] 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 [24] and [27]. However, unlike our study, these papers propose a modification to the TCP stack. Application-layer heuristics for improving the loss recovery latency of TCP are suggested [26]. These heuristics are geared toward bursty traffic flows and hence may not be effective for real-time flows. This paper expands on our earlier work [10] by including a discussion on the computational complexity of the proposed model (Section IV-B), analysis of VBR VoIP and live video streaming flows (Section VI-B), and recommendations for setting the playout buffer for real-time applications (Section VI-F).

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 experiments and the model to derive the working region of these flows, and 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.

References

[1] "AMR" [Online]. Available: http://en.wikipedia.org/wiki/Adaptive_multi-rate_compression
[2] "NIST Net" [Online]. Available: http://www-x.antd.nist.gov/nistnet/
[3] "SRI and ISI traffic generator" [Online]. Available: http://www.postel.org/tg/tg.html
[4] M. Allman, "TCP congestion control with appropriate byte counting (ABC)," RFC 3465 (Experimental), Feb. 2003.
[5] M. Allman, H. Balakrishnan, and S. Floyd, "Enhancing TCP's loss recovery using limited transmit," RFC 3042, Jan. 2001.
[6] M. Allman, S. Floyd, and C. Patridge, "Increasing TCP's initial window," RFC 3390, Oct. 2002.
[7] M. Allman, V. Paxson, and W. Stevens, "TCP congestion control," RFC 2581, Apr. 1999.
[8] S. A. Baset and H. Schulzrinne, "An analysis of the Skype peer-to-peer Internet telephony protocol," in *Proc. IEEE INFOCOM*, Barcelona, Spain, Apr. 2006.
[9] K. Beomjoon, C. Yong-Hoon, and L. Jaiyong, "An extended model for TCP loss recovery latency with random packet losses," *IEICE Trans. Commun.*, vol. 89, no. 1, pp. 28–37, Jan. 2006.
[10] E. Brosh, S. A. Baset, V. Misra, D. Rubenstein, and H. Schulzrinne, "The delay-friendliness of TCP," in *Proc. ACM SIGMETRICS*, Jun. 2008, pp. 49–60.
[11] E. Brosh, S. A. Baset, V. Misra, D. Rubenstein, and H. Schulzrinne, "The delay-friendliness of TCP," Department of Computer Science, Columbia University, New York, Tech. Rep. CUCS-014-08, Mar. 2008.
[12] N. Cardwell, S. Savage, and T. Anderson, "Modeling TCP latency," in *Proc. IEEE INFOCOM*, Tel-Aviv, Israel, Mar. 2000, vol. 3, pp. 1742–1751.
[13] K. Chen, C. Huang, P. Huang, and C. Lei, "An empirical evaluation of TCP performance in online games," in *Proc. ACM SIGCHI ACE*, Montréal, QC, Canada, Apr. 2006, Article no. 5.
[14] K. Chen, C. Huang, P. Huang, and C. Lei, "Quantifying Skype user satisfaction," in *Proc. ACM SIGCOMM*, Pisa, Italy, Sep. 2006, pp. 399–410.
[15] S. Floyd, T. Henderson, and A. Gurtov, "The NewReno modification to TCP's fast recovery algorithm," RFC 3782, Apr. 2004.
[16] S. Floyd and E. Kohler, "TCP friendly rate control (TFRC): The small-packet (SP) variant," RFC 4828 (Experimental), Apr. 2007.
[17] A. Goel, C. Krasic, K. Li, and J. Walpole, "Supporting low-latency TCP based media streams," in *Proc. IWQoS*, Miami, FL, May 2002, pp. 193–203.

[18] 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 *Proc. IMC*, Rio de Janeiro, Brazil, Oct. 2006, pp. 217–230.

[19] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "TCP friendly rate control (TFRC): Protocol specification," RFC 3448, Jan. 2003.

[20] M. Handley, J. Padhye, and S. Floyd, "TCP congestion window validation," RFC 2861 (Experimental), Jun. 2000.

[21] T. Kim and M. H. Ammar, "Receiver buffer requirement for video streaming over TCP," in *Proc. SPIE*, San Jose, CA, Jan. 2006, p. 607718.

[22] E. Kohler, M. Handley, and S. Floyd, "Designing DCCP: Congestion control without reliability," in *Proc. ACM SIGCOMM*, Pisa, Italy, Sep. 2006, pp. 27–38.

[23] J. Lazzaro, "Framing real-time transport protocol (RTP) and RTP control protocol (RTCP) packets over connection-oriented transport," RFC 4571, Jul. 2006.

[24] D. McCreary, K. Li, S. A. Watterson, and D. K. Lowenthal, "TCP-RC: A receiver-centered TCP protocol for delay-sensitive applications," in *Proc. MMCN*, San Jose, CA, Jan. 2005, pp. 126–130.

[25] A. Medina, M. Allman, and S. Floyd, "Measuring the evolution of transport protocols in the Internet," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, pp. 37–52, Apr. 2005.

[26] A. Mondal and A. Kuzmanovic, "When TCP friendliness becomes harmful," in *Proc. IEEE INFOCOM*, Anchorage, AK, May 2007, pp. 152–160.

[27] B. Mukherjee and T. Brecht, "Time-lined TCP for the TCP-friendly delivery of streaming media," in *Proc. ICNP*, Osaka, Japan, Nov. 2000, pp. 165–176.

[28] S. Na and S. Yoo, "Allowable propagation delay for VoIP calls of acceptable quality," in *Proc. AISA*, London, U.K., Aug. 2002, pp. 47–56.

[29] J. Nagle, "Congestion control in IP/TCP Internetworks," RFC 896, Jan. 1984.

[30] J. Olsen, "Stochastic modeling and simulation of the TCP protocol," Ph.D. dissertation, Uppsala University, Uppsala, Sweden, Oct. 2003.

[31] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP throughput: A simple model and its empirical validation," in *Proc. ACM SIGCOMM*, Vancouver, BC, Canada, Sep. 1998, pp. 303–314.

[32] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A transport protocol for real-time applications," RFC 3550, Jul. 2003.

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

[34] S. Tao, J. Apostolopoulos, and R. Gu, "Real-time monitoring of video quality in IP networks," in *Proc. NOSSDAV*, Stevenson, WA, Jun. 2005, pp. 129–134.

[35] B. Wang, J. Kurose, P. Shenoy, and D. Towsley, "Multimedia streaming via TCP: An analytic performance study," in *Proc. ACM Multimedia*, New York, Oct. 2004, pp. 908–915.

[36] B. Wang, W. Wei, and D. Towsley, "Multipath live streaming via TCP: Scheme, performance and benefits," in *Proc. CoNEXT*, New York, Dec. 2007, Article no. 11.

[37] A. Wierman, T. Osogami, and J. Olsen, "A unified framework for modeling TCP-Vegas, TCP-SACK, and TCP-Reno," in *Proc. MASCOTS*, Orlando, FL, Oct. 2003, pp. 269–278.

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

[39] X. Zhang and H. Schulzrinne, "Voice over TCP and UDP" Department of Computer Science, Columbia University, Tech. Rep. CUCS-033-04, Sep. 2004.

Eli Brosh received the B.Sc. degree in computer science, statistics, and operations research and the M.Sc. degree in electrical engineering from Tel-Aviv University, Tel-Aviv, Israel, in 1998 and 2003, respectively. He is currently a Ph.D. candidate with the Department of Computer Science, Columbia University, New York, NY.

He worked in the telecom industry as a Systems Engineer and Architect. His research interests include performance evaluation and mathematical modeling of network systems and design and analysis of P2P networks.

Salman Abdul Baset (S'06) received the B.S. degree in computer systems engineering from GIK Institute, Topi, Pakistan, in 2001, and the M.S. degree in computer science from Columbia University, New York, NY, in 2004, where he is currently a Ph.D. candidate with the Department of Computer Science. His research is focused on the reliability, session quality, protocol design, measurement, and energy efficiency issues in peer-to-peer communication systems.

Vishal Misra (S'98–M'99) received the B.Tech. degree from the Indian Institute of Technology, Bombay, India, in 1992, and the Ph.D. degree from the University of Massachusetts Amherst, Amherst, in 2000, both in electrical engineering.

He is an Associate Professor of computer science with Columbia University, New York, NY. His research emphasis is on mathematical modeling of computer systems, bridging the gap between practice and analysis. His recent work includes the areas of Internet economics, peer-to-peer networks, and efficient scheduling policies.

Dr. Misra has received an NSF CAREER Award, a DoE CAREER Award, and IBM Faculty awards. He has served as the Guest Editor for the *Journal of Performance Evaluation* and as Technical Program Committee Chair and General Chair of the ACM SIGMETRICS conference. He has participated as a member of program committees for conferences such as the IEEE INFOCOMM, ACM SIGMETRICS, ACM SiGCOMM, IFIP Performance, and IEEE ICNP. He serves on the Board of Directors of ACM SIGMETRICS.

Dan Rubenstein (M'00) received the B.S. degree in mathematics from the Massachusetts Institute of Technology, Cambridge, in 1992; the M.A. degree in math from the University of California, Los Angeles, in 1994; and the Ph.D. degree in computer science from the University of Massachusetts Amherst, Amherst, in 2000.

He is an Associate Professor with the Department of Computer Science, Columbia University, New York, NY. His research interests are in network technologies, applications, and performance analysis, with a recent emphasis on resilient, secure, and ultra-low power networking.

Dr. Rubenstein received an NSF CAREER Award, IBM Faculty Award, the Best Student Paper Award from the ACM SIGMETRICS 2000 conference, and Best Paper awards from the IEEE ICNP 2003 Conference and ACM CoNext 2008 Conference. He is an Editor for the IEEE/ACM TRANSACTIONS ON NETWORKING.

Henning Schulzrinne (F'06) received the Ph.D. degree from the University of Massachusetts Amherst, Amherst, in 1992.

He is the Levi Professor of Computer Science at Columbia University, New York, NY. He was a Member of Technical Staff with AT&T Bell Laboratories, Murray Hill, NJ, and an Associate Department Head at GMD-Fokus, Berlin, Germany, before joining the Computer Science and Electrical Engineering departments at Columbia University. He served as Chair of Computer Science at the university from 2004 to 2009. Protocols co-developed by him, such as RTP, RTSP, and SIP, are now Internet standards, used by almost all Internet telephony and multimedia applications. His research interests include Internet multimedia systems, ubiquitous computing, and mobile systems.