Understanding the Behavior of TCP for Real-time CBR Workloads

Salman A. Baset, Eli Brosh, Vishal Misra, Dan Rubenstein, and Henning Schulzrinne
Columbia University

ABSTRACT

In this paper, we examine the feasibility of sending real-time CBR workloads over TCP. This is motivated by the friendliness of NATs and firewalls towards TCP as opposed to UDP as well as by recent improvements in Internet's bandwidth and loss rates. Traditionally, TCP has been considered undesirable for real-time CBR workloads. We evaluate this assertion by developing a novel analytical tool that yields TCP's sender-to-receiver socket delay distribution for CBR workloads. A key insight gained is that the use of smaller than MSS-sized packets in CBR workloads can exploit the TCP's ACK counting mechanism thereby limiting the delay impact of congestion window variations. We leverage this insight to provide a heuristic and system-level guidelines for reducing TCP transport delays.

1. INTRODUCTION

It is widely accepted that UDP is the preferred transport layer for delivery of real-time CBR workloads, such as voice or video. In practice, the presence of NATs and firewalls may limit UDP connectivity. Skype, a popular VoIP application bypasses this problem by using TCP transport whenever UDP connectively is restricted. Furthermore, the recent advances in Internet bandwidth and loss rates, and recent TCP loss recovery enhancements, such as SACK [4], may suggest that TCP based transport of real-time CBR workloads is not as impractical as is traditionally believed [3]. This motivated us to evaluate the feasibility of CBR over TCP.

To this end, we develop an analytical tool that yields the TCP delay distribution, that is, the sender-to-receiver socket delay distribution for a single CBR workload as a function of the network loss rate, round-trip time, and workload rate. Our tool differs from existing analytical models [2, 5], in that it assumes a rate-limited TCP sender that may send smaller than MSS-sized packets rather then a saturated TCP sender that sends full MSS-sized segments. Furthermore, while previous work has been concerned with characterizing TCP's throughput, ours is concerned with characterizing the end-to-end TCP delay as it determines whether a packet can satisfy the play out constraint at the receiver.

Although, it is possible to study TCP's behavior empirically, an analytical delay prediction tool offers several advantages over such an approach. First, it provides an effective way to derive delay bounds over a wide range of network and workload parameters which can be used to compare the protocol delay of TCP to that of UDP. Second, it provides an application with a method to quickly evaluate the quality of multiple candidate routes. Finally, the tool can be used to avoid over provisioning the receiver buffer size for CBR workloads, such as video.

We use our tool to identify the feasible operational region for CBR over TCP and demonstrate that in some cases, such as voice workloads, TCP delivery can satisfy the delay constraints of a real-time application over a wide-range of network settings, e.g., the delay of a CBR flow with 'small' packet sizes (MSS/10) does not exceed a 200 ms delay with probability 0.97 for RTTs below 100 ms and packet loss rates of at most 1%. The low delay results can be explained by observing TCP's congestion control behavior in a delay optimized setting, i.e., when no delay socket option and ACK counting [1] are enabled[1]. In this setting, TCP adjusts its congestion window according to the workload's *packet rate* rather than the total *byte rate*. The use of smaller than MSS-sized packets for CBR workloads such as voice implicitly exploits this behavior thereby limiting the delay impact of congestion window variations. We use our tool to characterize the tradeoff between packet size and packet rate in a quantative manner.

2. DELAY PREDICTION TOOL

Our tool captures TCP delay costs due to packet retransmission and congestion control. For a single or multiple losses within a window, the expected cost of a retransmission is assumed to be RTT[2]. TCP uses a congestion window (CW) to control the allowed transmission rate and thus may induce sender side delays when the offered load is limited by the CW size. A key assumption we make is that ACK counting is enabled at the sender, i.e., TCP increments the CW by a constant amount (MSS) upon the arrival of each ACK.

We capture the TCP delay by keeping track of the window size and the backlog size (the amount of unsent data in TCP's send buffer) evolutions across rounds. A round corresponds to the period between CW updates. We assume that a round duration is RTT during window limited periods and is the interval between losses during workload limited periods. Our tool only captures the congestion-avoidance behavior of TCP and we are currently working towards incorporating slow-start and timeout effects. Let w_i and b_i be the window size and the backlog size at the beginning of the ith round. In the absence of a loss indication, the system

[1]ACK counting is enabled by default in Windows XP and most Linux distributions.

[2]This is motivated by the wide-deployment of new TCP recovery mechanisms, such as SACK and NewReno.

CoNEXT '06, Lisbon, Portugal

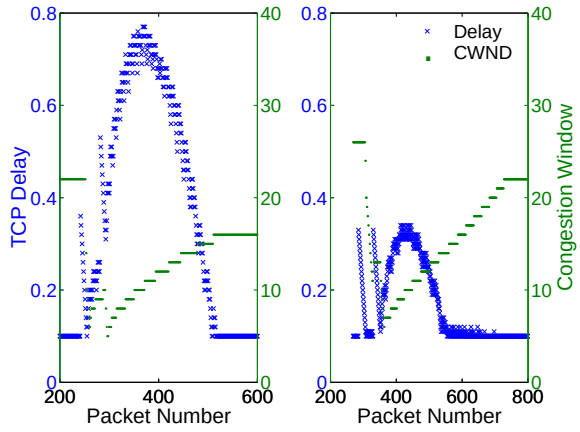

Figure 1: TCP delay for CBR workloads of 10 (left) and 20 (right) packets per RTT.

evolution can be expressed by

$$w_{i+1} = \begin{cases} w_i + 1 & \text{if } b_i > 0 \\ \min(r, w_i + 1) & \text{if } b_i = 0 \end{cases} \quad (1)$$

$$b_{i+1} = \begin{cases} \max\{0, b_i + rs - w_i MSS\} & \text{if } b_i > 0 \\ 0 & \text{if } b_i = 0 \end{cases} \quad (2)$$

where r is the workload packet rate per RTT and s is the workload packet size. The backlog size change is represented by the newly added data rs minus the transmitted data which can be at most $MSSw_i$. If a loss indication is received in the ith round then:

$$w_{i+1} = \begin{cases} \lfloor w_i/2 \rfloor & \text{if } b_i > 0 \\ \lfloor \min(r, w_i)/2 \rfloor & \text{if } b_i = 0 \end{cases} \quad (3)$$

$$b_{i+1} = \begin{cases} \max\{0, b_i + rs - \lfloor w_i/2 \rfloor MSS\} & \text{if } b_i > 0 \\ \max\{0, b_i + rs - \lfloor \min(r, w_i)/2 \rfloor MSS\} & \text{if } b_i = 0 \end{cases} \quad (4)$$

The above equation captures TCP's window adaptations for workload-limited flows so that at the end of loss recovery the CW is decreased to half of the number of packets in flight, regardless of the window size prior to the loss [1].

The sender side delay of a packet can be approximated by the backlog size observed just prior to its transmission, since the packets that a transmitted packet leaves behind must have arrived during its stay in the system. The system evolution equations can be directly translated into a Markov chain which can be solved numerically[3] to yield the delay distribution. We validated the tool by comparing its cumulative delay distribution to those obtained via real measurements. Our test bed consists of sender and receiver Linux machines connected through a router running NISTnet which emulates a network with a constant RTT delay and a packet loss distribution. The obtained results indicate that TCP can satisfy the delay constraints of a real-time application, as described in the previous section.

3. CBR INTERACTIONS WITH TCP

We give an illustrative example to demonstrate the relationship between CBR workload characteristics and TCP delays. Consider Figure 1 which plots the sender-to-receiver TCP delay and window size vs packet number for two CBR over TCP flows of full (right plot) and half-sized (left plot) MSS packets in a symmetric network with 100 ms RTT. The total byte rate is the same for both flows but their packet sending rate is 100 (10) and 200 (20) packets/s (packets/RTT), respectively. Both flows experience two close-by losses. The delay spikes shown are caused by receiver-side buffering due to TCP's in-order delivery. The spike height is 1.5 RTT plus the triple duplicate ACK overhead, e.g., three packetization intervals.

The quadratic delay curve is caused by sender side buffering due to congestion window limitation and is captured by eq. (2). The first flow (left plot) fully utilizes the connection's throughput in steady-state and immediately experiences sender side buffering after the first loss due to CW reduction. The second flow utilizes only half of the available throughput in steady-state since its packet rate is doubled, while the byte rate remains the same. The second flow will experience sender side buffering only after the second loss, i.e., when the CW is reduced by more than two. Specifically, the second flow does better than the first flow because TCP can combine half-MSS sized packets to fully utilize the throughput. The delay reduction comes at the expense of increased TCP header overhead due to increased packet rate. Note that the CW is determined by the packet rate and that the steady-state CW is higher than the workload packet rate due to backlog clearing time.

This explains why CBR workloads such as voice may not be adversely impacted by TCP's window variations. Our tool can be used to systematically evaluate tradeoffs between packet size and rate. Given enough bandwidth, a general recommendation for MSS to packet size ratio would be to keep it at least above two to avoid the build up of sender side backlog.

4. ON GOING WORK

Based on our experiments we recommend TCP parameter settings for optimizing TCP delays. The key points are that Nagle's algorithm and delayed ACKs should be disabled and ACK-counting should be enabled. Detailed results that quantify the impact of these parameters are omitted due to space considerations. Currently, we are investigating the effect of dynamically adjusting packet size on TCP delay and are running planet-lab delay experiments. We intend to use real-voice transmission experiments to quantify voice over TCP delays.

5. REFERENCES

[1] M. Allman *et al.* TCP Congestion Control . RFC 2581, 1999.

[2] E. Altman *et al.* A stochastic model of TCP/IP with stationary random losses. In *SIGCOMM*, Sept. 2000.

[3] A. Goel *et al.* Supporting low-latency tcpbased media streams. In *IWQos*, May 2002.

[4] M. Mathis *et al.* TCP Selective Acknowledgement Options. RFC 2018, Oct. 1996.

[5] J. Padhye *et al.* Modeling TCP throughput: A simple model and its empirical validation. In *SIGCOMM*, Sept. 1998.

[3]This requires delay pattern computations, the details of which have been omitted.