TCP-RC: A Receiver-Centered TCP Protocol for Delay-Sensitive Applications

Doug McCreary, Kang Li, Scott A. Watterson, David K. Lowenthal
Dept. of Computer Science, The University of Georgia

ABSTRACT

TCP is the de-facto standard transport-layer protocol in the Internet. However, TCP is generally considered to be inappropriate for delay-sensitive applications such as multimedia. This paper proposes a novel receiver-centered TCP (TCP-RC), which is a TCP modification at the receiver that is intended for delay-sensitive applications. The basic principle behind TCP-RC is that it achieves low latency at the expense of reliability. In particular, TCP-RC forges lost packets, passing them on to an enabled application. This allows low-latency transmission for a class of applications that do not demand full reliability. Results obtained from emulated experiments show that over a range of loss rates and round-trip times, TCP-RC has a significantly smaller average- and worst-case per-packet delay than regular TCP.

1. INTRODUCTION

TCP is the de-facto standard transport-layer protocol in the Internet. It is designed for reliable bulk transfer and so bundles together congestion control, flow control and reliability control. However, TCP is generally considered to be inappropriate for delay-sensitive applications such as multimedia. This is because such applications require low latency, which is difficult to achieve with TCP due to retransmissions forced by its reliability mechanism.

Because many delay-sensitive applications can tolerate loss, they typically use one of several alternatives to TCP. The most frequent alternative is UDP, which, because it does not force lost data to be retransmitted, allows low-latency transmission. However, the disadvantage is that it is a nontrivial task to build congestion control that is friendly to existing TCP flows[1] on top of the UDP layer.

A second alternative is to build a new protocol that is specifically designed for delay-sensitive applications.[2–10] The advantage of these techniques is that it is easier to build such applications. Unfortunately, none of these protocols have been widely deployed. This is because in general, it is difficult to achieve wide deployment of new network protocols that are in competition with TCP—because adoption at both server and client is required.

Accordingly, this paper proposes a novel receiver-centered TCP modification, called TCP-RC, for delay-sensitive applications that achieves low latency by sacrificing full reliability. In particular, TCP-RC forges lost packets, passing them on to a TCP-RC-enabled application—which avoids significant delay in the stream. The receiver then acknowledges the forged packet to the sender. Because our TCP modification is purely on the receiver, it does not require any change to any other machine on the Internet—only the delay-sensitive application need change. This makes deployment a relatively straightforward task, as clients can choose (but are never required) to run TCP-RC.

While TCP-RC avoids delay by forging lost packets, to be a reasonable alternative to a UDP-based protocol, it should also provide TCP-friendly congestion control. We handle this by sending enough acknowledgements for the forged packets such that the sender in turn provides TCP-style congestion control. In particular, we will leverage TCP's fast retransmission mechanism to force the sender to cut its window size. In this way we also avoid TCP timeouts.

Results obtained from emulated experiments show that over a range of bandwidths, loss rates, and round-trip times, that TCP-RC has a median per-packet delay of approximately the one-way trip time (OTT), and the maximum packet delay is slightly more than one round-trip time (RTT); an average delay is only slightly more than the OTT. On the other hand, regular TCP has a maximum packet delay of over seven times the RTT, and the average packet delay is as much as 50% larger than the OTT due to timeouts, retransmissions, and sequential packet delivery. In terms of TCP friendliness, TCP-RC has an identical rate as TCP at low loss rates.

Figure 1. Operation of TCP and TCP-RC, under packet loss.

Problems with Regular TCP for Delay-Sensitive Applications

In TCP, a packet sent from sender to receiver can fail due to overloading of a network router or the receiver. The solution TCP takes in these situations is that the receiver will acknowledge a packet P only if all packets up to and including P are received. Otherwise, either no acknowledgement is made, or a packet previous to P is acknowledged. If not acknowledged, the sender re-sends packet P either due to a fast retransmission or a timeout.

Clearly, this kind of reliability provided by TCP is not appropriate for many delay-sensitive applications. To see why, consider the left-hand side of Figure 1. The important point is that when a packet P is lost, all packets arrive that after P (in sequence) cannot actually be received by the application. Instead, they must be stored by TCP (in the "out-of-order queue"). This means that the end-host application cannot process these packets until packet P has been received, which causes significant delay in the stream. Furthermore, this delay can be anywhere from one RTT to the timeout period, which can be as much as seconds.

Assumptions

TCP-RC relies on the following assumptions. First, we assume that the delay-sensitive application is aware of TCP-RC. This is necessary because when loss occurs, TCP-RC forges packets. The application must take understand that such packets are meaningless and take appropriate action. Second, we assume packets are periodically generated at the sender and are immediately sent via TCP. This means that in most cases packet loss can be inferred by the reception of several subsequent packets. Third, we also assume that once a packet is delivered to TCP-RC in order, the receiver-side application reads it immediately. This allows us to measure packet delay using the time a packet is received by TCP-RC.

2. IMPLEMENTATION

Our TCP-RC implementation exploits TCP to support delay-sensitive applications. The basic idea is that the receiver takes control of reliability—which is the job of the sender in regular TCP—and forges packets when they are lost to avoid high-latency stalls in the packet stream. As this can cause a flow to be unfriendly to other flows. TCP-RC must closely mimic regular TCP congestion control behavior.

2.1. Receiver-Based Reliability Control

To allow for better performance for delay-sensitive applications, we have designed receiver reliability control. The basic idea (see the right-hand side of Figure 1) is that the receiver will "forge" packet P when the receiver believes it is lost. This way, the client TCP will deliver packet P as well as future packets (assuming they arrive in sequence) to the application.

The key issue is how to determine when a packet is lost. TCP-RC does this by monitoring the packet stream and looking for an out-of-order packet sequence. However, assuming that any out-of-order sequence implies a lost packet is

overly pessimistic. This is because the packet may simply have been reordered and not lost. As there is a cost to mistakenly inferring a packet is lost (see Section 2.2), this approach suffers if there is frequent packet reordering.

Hence, TCP-RC uses a timer-based approach once it detects an out-of-order packet sequence. When TCP-RC detects such a sequence, it sets a timer. Then, if the missing packet does not arrive before the timer fires, the packet is declared missing. The timer value is determined by using the average packet interarrival time and assuming the missing packet will arrive within 3 additional packets. Section 2.2 discusses how TCP congestion control is maintained under this scheme.

2.2. Maintaining TCP Congestion Control

While forging packets can reduce latency on the receiver, simply adding this feature to TCP-RC will most likely cause the flow to be TCP-unfriendly. This is because from the point of view of the sender, packet P arrived at the receiver. So, the sender will continue as if no loss occurred, instead of taking the usual congestion control action on a lost packet. This means in particular that the window size on the sender will never be cut.

Our goal is to signal the sender that loss occurred, so that the sender will apply the usual TCP congestion control mechanism. Recall that in TCP, there are two possible actions that occur on the sender when a packet is determined lost; both involve the retransmission of packet P to the receiver. One is that fast retransmission mode is entered. According to the TCP specification, this occurs when four acknowledgements for the same packet are received by the sender. The other is that a timeout occurs. In most TCP implementations, the sender times out if an acknowledgement for a sent packet does not arrive in a set amount of time.

We want TCP-RC to closely mimic TCP congestion control while avoiding a retransmission, which would cause excessive delay. The basic idea is as follows (also shown in Figure 1). When packet P is declared missing, we send n acknowledgements for P to the sender, where n is determined so that the sender will receive a total of 4 acknowledgements for P. Then, we forge the packet as described in Section 2.1. (The acknowledgement is for packet $P-1$, indicating packet P is next to be received.) This will cause the sender to enter fast retransmission mode, re-send P, and cut its window size in half. In this way TCP-RC is mimicing TCP congestion control behavior. Upon receiving packet P again, TCP-RC will discard P, as it believes it to be a duplicate.

3. PERFORMANCE

This section describes the performance of TCP-RC on a delay-sensitive flow in terms of end-to-end latency and TCP friendliness. The end-to-end latency is the major obstacle to adopting TCP for delay-sensitive applications.

3.1. Experimental Methodology

Our methodology is as follows. Note that we assume that the receiver is ready to process packets as soon as they are available (see Section 1). In practice, the buffering delay in TCP (at both sender and receiver) has been shown to contribute considerably to overall end-to-end delay, especially at low loss rates. However, recent work on dynamic buffer tuning by Semke[11] and Ashvin et al.[12] allows us to assume that we can reduce these buffering delays to nearly zero.

As a result, the major components left in TCP end-to-end delay are the network delay, out-of-order delay, and the receiver application delay. We focus specifically in this paper on reducing the network and out-of-order delays. The part of network delay that we can affect is that part primarily caused by data retransmissions. It occurs because a packet may need to be sent multiple times, incurring multiple network delays (or some part thereof), along with an additional penalty to detect the loss. In addition, it causes out-of-order delay for subsequent packets because TCP delivers packets in order, and a lost packet temporarily blocks further packets from being delivered to the application. We use Nistnet[13] to regulate the RTT between a client and server, as well as drop packets to create loss.

3.2. End-to-End Latency

For each experiment, a trace is generated at both sender and receiver. We measure the network delay by capturing the departure time from the server and the arrival time at the client for every packet. Then, packet delay is computed as the difference between the sending and the receiving times. Finding the sending time of a packet is straightforward, as we use the timestamp in the entry in the sender side trace. The receiving time is computed as the smaller of two times: (1) the time at which the packet is received by the receiver, and all previous packets have been received, and (2) the time that the packet is forged by TCP-RC, if that occurs. This is because due to loss, the actual arrival time of a packet is not necessarily the

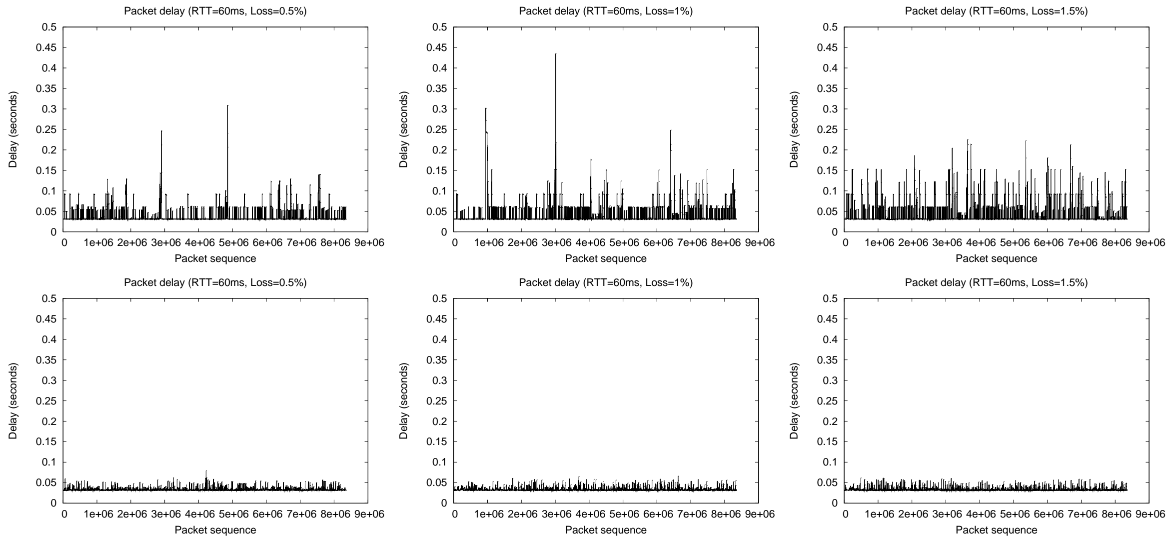

Figure 2. TCP (top) and TCP-RC (bottom) network delay for several different loss rates.

time the packet is delivered to the application. In this short paper, we present only experiments that use a fixed RTT and loss rate.

Figure 2 (top) shows the network delay for a regular TCP flow with a round trip time of 60 *ms* and loss rates that vary from 0.5% to 2%. For most of the packets, the delay is approximately equal to the one way trip time (OTT), 30 *ms*. However, when a packet is lost, it causes significant additional delay. Several possible delay values exist. First, there are a few packets with delays of around 200 *ms*. This is because a TCP timeout occurred in these cases. Second, there are several packets with delays around 90 *ms*. In this case, the dropped packet is re-sent due to a fast retransmission, incurring an extra RTT (60 *ms*) delay (in addition to the original OTT). Note that also, there are nearby packets that may have about a 90 *ms* delay because they arrived after the dropped packet, but cannot be received by the application. Third, there are many packets with delays around 60 *ms*. These delays correspond to the packets with 90 *ms*; they arrive after a dropped packet, but due to a small congestion window size, they are sent 30 *ms* later than the dropped packet. As expected, as the loss rate increases, there are more packets with delay. However, the number of timeouts does not change significantly with loss rate.

On the other hand, the bottom of Figure 2 shows the network delay for TCP-RC under the same conditions. The delays for TCP-RC are between 30 *ms* and 60 *ms*. As with regular TCP, most of the delays are 30 *ms*, when packets are not dropped. However, there are no large delays. This is because TCP-RC does not wait for retransmissions; instead, it forges packets.

Overall, the results, show the viability of TCP-RC for delay-sensitive applications. This is because of two reasons. First, the worst case delay for TCP-RC is always close to the RTT, rather than based on a a TCP timeout. Second, the average delay for TCP-RC is always close to the OTT, rather than the RTT.

3.3. Friendliness

While TCP-RC is effective in reducing end-to-end latency, it is important that it does not do so at the expense of other applications. In particular, TCP-RC could become *more* efficient than regular TCP by simply forging packets yet never adjusting to their loss. Regular TCP adjusts to loss by multiplicatively decreasing the congestion window.

Figure 3 shows the playback time of both regular TCP and TCP-RC when varying RTT (left) and loss rate (right). We observe that the performance of TCP-RC is slightly better than regular TCP under relatively low loss (less than 1%) and small to moderate RTTs (less than 60 *ms*), primarily because TCP-RC does not wait as long for timeouts; however,

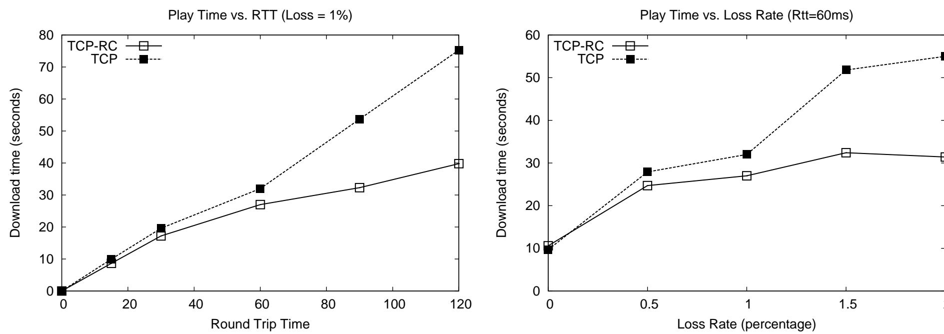

Figure 3. Playback time for TCP and TCP-RC. The left-hand side is a fixed 1% loss, and the right-hand side is a fixed RTT of 60 *ms*.

TCP-RC does react to loss by reducing the window size. For larger loss rates and RTTs, we will need to further slow down TCP-RC by delaying acks, once we detect that a timeout would otherwise happen. We are currently working on this rate control to maintain TCP-friendliness in all cases.

4. FUTURE WORK

This work provides a basis for low-latency multimedia transfer over TCP. Our primary future direction is to use TCP-RC to save energy on wireless clients by transitioning their network interface (WNIC) to *sleep* mode when no packets are expected. To carry this out, we will leverage off of our prior work that saves WNIC energy for TCP downloads.[14] The key issue will be trading off smooth play with energy saving.

REFERENCES

1. "The TCP-Friendly website, http://www.psc.edu/networking/tcp_friendly.html."
2. S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *ACM SIGCOMM*, pp. 43–56, (Stockholm, Sweden), 2000.
3. I. Rhee, V. Ozdemir, and Y. Yi, "TEAR: TCP emulation at receivers – flow control for multimedia streaming," tech. rep., NC State, 2000.
4. S. Liang and D. Cheriton, "TCP-RTM: Using TCP for real time multimedia applications," in *International Conference on Network Protocols*, Nov. 2002.
5. S. Cen, C. Pu, and J. Walpole, "Flow and congestion control for internet streaming applications," in *Multimedia Computing and Networking*, 1998.
6. R. Stewart and Q. Xie. et al., "Rfc 2960: Stream control transmission protocol," Dec 2000.
7. J.-R. Li, S. Ha, and V. Bharghavan, "HPF: A transport protocol for heterogeneous packet flows in the internet," in *INFOCOM (2)*, pp. 543–550, 1999.
8. C. Krasic, K. Li, and J. Walpole, "The case for streaming multimedia with TCP," *Lecture Notes in Computer Science* **2158**, 2001.
9. B. Mukherjee and T. Brecht, "Time-lined TCP: a transport protocol for delivery of streaming media over the internet," in *International Conference on Network Protocols*, 2000.
10. C. Zhang and V. Tsaoussidis, "TCP-Real: improving real-time capabilities of TCP over heterogeneous networks," in *Workshop on Network and Operating Systems Support for Digital Audio and Video*, June 2001.
11. J. Semke, J. Mahdavi, and M. Mathis, "Automatic TCP buffer tuning," in *ACM SIGCOMM*, pp. 315–323, 1998.
12. A. Goel, C. Krasic, K. Li, and J. Walpole, "Supporting low latency tcp-based media streams," in *The Tenth International Workshop on Quality of Service (IWQoS)*, 2002.
13. "Nist net network emulator, http://snad.ncsl.nist.gov/itg/nistnet/," 2000.
14. H. Yan, R. Krishnan, S. A. Watterson, D. K. Lowenthal, K. Li, and L. L. Peterson, "Client-centered energy and delay analysis for TCP downloads," in *Proceedings of the 14th IEEE Int'l Wkshop on Quality of Service*, June 2004.