Fairer TCP-Friendly Congestion Control Protocol for Multimedia Streaming Applications

Soo-Hyun Choi and Mark Handley
University College London
{S.Choi, M.Handley}@cs.ucl.ac.uk

ABSTRACT

We present TFWC, a TCP-friendly window-based congestion control mechanism for real-time multimedia streaming applications. Although TFRC is regarded as a de facto standard for those types of applications, under low stat-mux conditions fairness can be an issue. We re-introduce a TCP-like ACK mechanism while retaining the TCP throughput equation to compute the sending rate. We show that our proposed protocol is fairer than TFRC when competing with TCP flows under certain circumstances.

1. INTRODUCTION

The rapid growth of consumer broadband is driving a significant increase in the use of multimedia applications. The variability of TCP's AIMD congestion control and its absolute reliability are far from ideal for these real-time applications so, where possible, they use UDP. A number of congestion control protocols have been proposed [1, 2, 3]. Among them, TCP-Friendly Rate-based Control (TFRC) [4] has emerged as a mechanism to provide smooth and predictable throughput for such applications. Recently, Datagram Congestion Control Protocol (DCCP) [7] adopted TFRC as a means to minimize abrupt changes in the sending rate.

Unfortunately, we observed that if a flow traverses a low statistically multiplexed network link (such as DSL provides) with drop-tail queuing, TFRC traffic can starve TCP traffic. Although, on average, TFRC and TCP respond similarly to loss, TFRC lacks the fine-grain congestion *avoidance* mechanism that TCP's ack-clocking provides, meaning that TFRC can briefly overshoot the available link capacity. This fills the buffer at the bottleneck link and causes TCP's ack-clock to actually reduce the sending rate. TFRC can also cause oscillatory behavior by itself due to this overshoot; for this reason TFRC builds in a short-term mechanism to reduce the transmission rate as the RTT increases. This mechanism however has a little lag which reduces its effectiveness, and while it helps significantly, it also reduces the smoothness that was the motivation for being rate-based.

Bansal and Floyd [5] observed related problems when the available capacity changes abruptly, and introduced a conservative mode to TFRC, whereby the sender is limited by the measured arrival rate at the receiver. In simulation, this works well to mitigate many problems, but accurately measuring the received rate (and indeed finely controlling the sending rate) in *real* systems is problematic. Saurin [6] discovered that even the non-conservative TFRC's limiter (which is bounded to twice the receive rate) cuts in inappropriately on real networks. Given these issues, perhaps we should simply use a congestion window in the first place?

Our main contribution is to re-introduce TCP's ack-clocking feature for limiting unacknowledged packets, while still providing equation-based characteristics similar to TFRC. By introducing the ack mechanism, we calculate sending rate based on a window size. Our protocol is noticeably fairer than TFRC when sharing lower-speed network links.

2. TFWC

TCP-Friendly Window-based Control (TFWC) uses the same TCP throughput equation [3] used by TFRC, but whereas TFRC uses the calculated sending rate in a rate-based controller, TFWC uses TCP-like ack-clocked window-based control. A TFWC sender computes the congestion window (*cwnd*) to determine its sending rate using the same equation. Like TFRC, TFWC does not retransmit lost packets as it is intended for streaming multimedia applications.

From a simplified TCP throughput equation introduced in [3], we can derive the window of packets we are allowed to send. The idea is to multiply t_{RTT}/s to both side of equation and get the number of bytes for the use of a new $cwnd$ calculation. Thus,

$$W = \frac{1}{\sqrt{\frac{2p}{3}} + \left(12\sqrt{\frac{3p}{8}}\right)p\left(1 + 32p^2\right)} \qquad (1)$$

where W is the window size in packets and p is the packet loss probability (more precisely loss event rate).

A TFWC sender behaves just like a TCP sender *until* the first lost packet is reported. From this point on, upon receipt on each ack, TFWC will compute the average loss interval to get the loss event rate. Feeding this loss event rate into Equation (1), we can calculate the new *cwnd* size. If the sequence number of new data waiting to be sent satisfies equation (2) then it can now be sent, resulting in an ack-clock.

$$\text{Seqno of New Data} \leq \textit{cwnd} + \text{Unacked Seqno} \quad (2)$$

Every time an ACK message comes into the TFWC sender, it will update the round-trip time and the "retransmit" timeout value. With the calculated timeout value, the "retransmission" timer is set every time a new packet is sent. If the timer expires, the next available packet (not in fact a retransmission) is sent, and the timer is restarted with double the previous timeout value.

3. SIMULATION RESULTS

We implemented TFWC in *ns* [9]; the results here use a dumbbell topology. We vary the number of sources from 1 to 100 and used the same number of competing TCP and TFRC sources and similarly competing TCP and TFWC sources. The bottleneck bandwidth varies from 0.1 Mb/s to 20 Mb/s with a 10ms RTT delay, and the access link delay was chosen randomly between 0.2 ms to 2.0 ms. There are reverse path TCP traffic in the simulation in order to break simulation phase effects and exacerbate any issues due to TFWC ack-compression. The drop-tail buffer size at the bottleneck was set to a delay-bandwidth product to allow TCP to perform correctly. In this scenario, the approximate average round-trip time is 12ms - a fairly typical DSL RTT to the local ISP. We draw the protocol fairness graphs as follows. Let the protocol fairness indicator θ be:

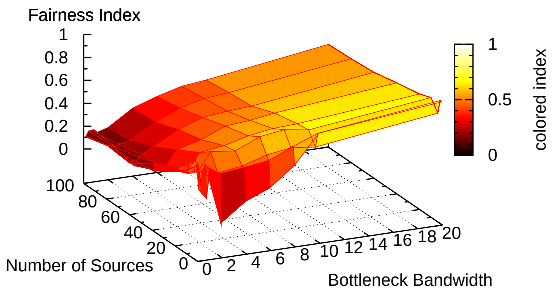
$$\theta = \frac{\sum(\text{TCP Rate})}{\sum(\text{ TCP Rate + TFWC Rate})} \quad (3)$$

If θ is 0.5, then it indicates TCP perfectly shares the link with TFWC (or TFRC). As θ approaches to 1, TFWC (or TFRC) traffic is starved by TCP traffic.

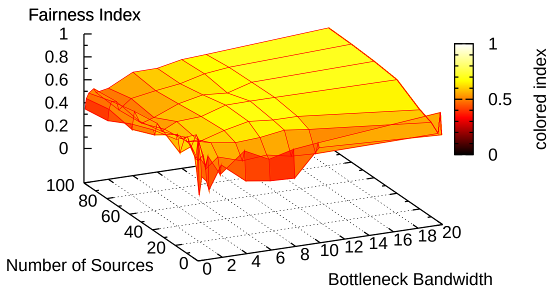
Figure 1 shows the protocol fairness for (a) TFRC vs TCP and (b) TFWC vs TCP. The x-axis shows the bottleneck bandwidth, and he y-axis gives the number of flows sharing the bottleneck, and the z-axis stands for the fairness indicator (θ). The TFRC protocol fairness is largely dependent upon the bottleneck bandwidth. If the bottleneck bandwidth is less than 1 Mb/s, then we can hardly say the TFRC is fair with TCP traffic no matter what the number of sources: $\theta \cong 0.1$. Note that the one flow case also shows little bit of unfairness. TFWC does significantly better. However if the bottleneck bandwidth is large (e.g., over 5 Mb/s), then both TFWC and TFRC are reasonably fair.

4. CONCLUSION

In this paper, we introduced TCP-Friendly Window-based Control (TFWC), a simple but novel congestion control mechanism for streaming media applications. TFWC outperforms



(a) Fairness (TCP and TFRC)



(b) Fairness (TCP and TFWC)

Figure 1: Fairness Comparison, $t_{RTT} \cong 12$ ms

TFRC in that it provided fairer network resource share over ns-2 simulation, especially under DSL-like environments that most many multimedia users are likely to use. Future work includes an implementation and evaluation in a real-world environment using real applications.

5. REFERENCES

[1] L. Brakmo, S.O'Malley and L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," ACM SIGCOMM '94.
[2] Reza Rejaie, Mark Handley, Deborah Estrin, "RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet," IEEE INFOCOM '99.
[3] S. Floyd, M. Handley, J. Padhye, and J. Widmer, " Equation-Based Congestion Control for Unicast Applications," ACM SIGCOMM '00.
[4] M. Handley, S. Floyd, J. Padhye, and J. Widmer, "TCP Friendly Rate Control (TFRC): Protocol Specification," RFC 3448, January 2003.
[5] D. Bansal, H. Balakrishnan, S. Flyod, and S. Shenker, "Dynamic Behavior of Slowly-Responsive Congestion Control Algorithms," SIGCOMM '01.
[6] Alvaro Saurin, "Congestion Control for Video-conferencing Applications," MSc Thesis, University of Glasgow, December 2006.
[7] E. Kohler, M. Handley, and S. Floyd, " Profile for Datagram Congestion Control Protocol (DCCP)," RFC 4340, March 2006.
[8] S. Floyd, E. Kohler, and J. Padhye, " Profile for Datagram Congestion Control Protocol (DCCP) CCID 3," RFC 4342, March 2006.
[9] The Netework Simulator, `http://www.isi.edu/nsnam/ns/`