Oscillations and Buffer Overflows in Video Streaming under Non-Negligible Queuing Delay

Yueping Zhang and Dmitri Loguinov*
Department of Computer Science
Texas A&M University, College Station, TX 77843
{yueping, dmitri}@cs.tamu.edu

ABSTRACT

In this paper, we analyze how feedback delays affect stability, oscillations, and packet loss of several classes of congestion controllers used or proposed for video streaming in the current/future Internet (including window-based AIMD, rate-based AIMD, Scalable TCP, and TFRC). Our results indicate that window-based protocols in this list incur significantly less packet loss under delayed feedback than their rate-based counterparts, which explains their better overall performance observed in practice [12]. At the same time, we show that even TCP's congestion control is far from ideal from the control-theoretic point of view and leads to amplified oscillations when queuing delays increase. We conclude with an observation that multimedia in the future Internet is not likely to enjoy oscillation-free congestion control unless the network deploys some form of AQM.

Categories and Subject Descriptors

C.2.2 [**Communication Networks**]: Network Protocols

General Terms

Algorithms, Performance, Theory

Keywords

Buffer Overflows, Delay, Stability, Video Streaming

1. INTRODUCTION

Video streaming faces many challenges in the current Internet. One of these challenges is the design of proper congestion control that remains stable (small oscillations, low packet loss, predictable behavior) under various network conditions. While the design of control systems under immediate feedback has been extensively studied in network literature [2] and classical control-theory [9], most practical networks consist of resources that provide *delayed* feedback to individual end-flows. Feedback delays arise for a number of reasons, two of which are propagation and queuing delays. In this work, we aim to understand how feedback delays of regular end-users, induced by *queuing* at the bottleneck router, affect the behavior of streaming congestion control commonly used in previous work. We do not address the issue of delays arising due to large propagation delays, or due to queuing at non-bottleneck routers.

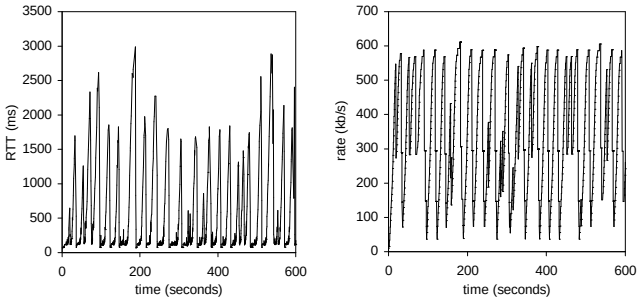


Figure 1: Video streaming over 512-kb/s residential DSL. Evolution of the RTT (left) and that of the IP-layer sending rate (right).

To better understand why delay is a significant factor in the performance of a congestion controller, consider the following illustration. We placed an MPEG-4 video server at Michigan State University and used home DSL clients to stream scalable MPEG-4 FGS video from the server. In the case reported in this paper, the client's access link supported up to 512 kb/s on the physical layer, which corresponded to approximately 450 kb/s of IP-layer throughput. As shown in Figure 1 for one representative experiment, both the RTT of the flow and the sending rate of the server exhibited significant fluctuation throughout this ten-minutes streaming session. The increase in the RTT was in response to buffering delays at the DSL link and was directly correlated with occurrences of packet loss. The server used rate-based AIMD congestion control, which frequently increased the rate to over 600 kb/s and then dropped it below 35 kb/s in response to significant packet loss. Clearly, this performance is undesirable for many reasons, including fluctuating video quality, high packet loss, low link utilization, and prohibitively large retransmission delays.

Since many current video streaming protocols are TCP-friendly in one form or another [14], [15], [17], their design

*This work was supported in part by NSF grants CCR-0306246, ANI-0312461.

NOSSDAV'04, June 16–18, 2004, Cork, Ireland.

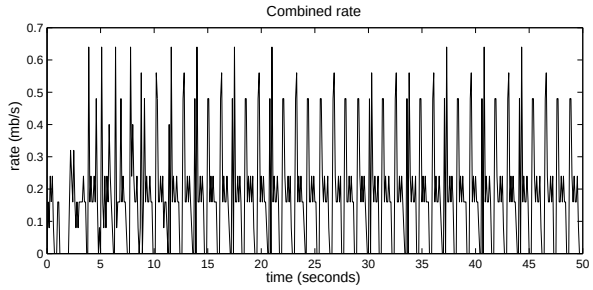


Figure 2: Combined sending rate of two competing TCP flows.

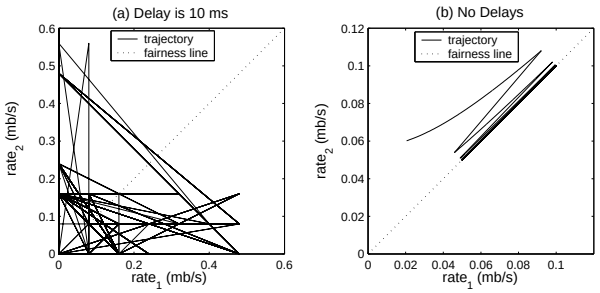


Figure 3: Trajectories of two TCP flows: (a) 10 ms delay; (b) Instantaneous feedback.

has been influenced by that of TCP, which provides a fundamental platform for the analysis of these mechanisms. In what follows in this paper, we first study TCP and demonstrate that its performance deteriorates as buffering delay becomes large and reaches levels that are clearly unacceptable for video applications (i.e., high rates of oscillations and packet loss). However, we subsequently show that TCP has the *best* delayed performance among the protocols commonly used (or proposed) for video streaming (i.e., rate-based AIMD, Scalable TCP, and TFRC). Combined, these two results paint a rather bleak picture for current best-effort streaming methods; however, they provide clear insight into a long-standing question of whether window-based protocols do in fact perform best in the current Internet.

We conclude the paper with observations that delayed instability is inherent to all AIMD-friendly classes of control methods and that better algorithms based on AQM feedback may be the only alternative for improving the quality of video streaming in the future Internet.

2. DELAYED BEHAVIOR OF AIMD

2.1 Background

Modern Internet applications desire asymptotically stable[1] flow controls that deliver packets to end users without much oscillation. In the context of Internet congestion control, stability can be interpreted as the ability of a network system to maintain the desired overall sending rate (less than or equal to the resource capacity) in its steady state and to re-establish the steady state after being disturbed. In an Internet-like network system, many disturbances are possible, including arrival and departure of flows, feedback noise, and various transient effects. In this paper, we concentrate on the ability of each flow to control its oscillations when feedback from the routers arrives with a certain non-negligible delay.

Existing congestion controls are broadly classified into two categories: rate-based and window-based, where the former is usually more desirable in video streaming. It is generally observed that controlling rate-based flows is challenging [12], [16] as their oscillations and packet loss exhibit worse performance than those of similar window-based mechanisms; however, we are not aware of any quantitative studies that support this statement in the context of streaming or outside of ATM data-link literature. While all TCP-friendly controls oscillate, these oscillations become more pronounced as the delay in the feedback increases (we call this situation "increased instability"), which happens when the flows adjust their current rates based on out-dated feedback and then overreact to packet loss caused by the overshoot.

2.2 Delay-Related Oscillations in TCP

We first consider delayed behavior of TCP. Recall that TCP follows the AIMD policy [2], [6] in its window adjustment, where the sender additively increases its congestion window per positive ACK and multiplicatively decreases it upon each packet loss. When buffers sizes are large (such as in the DSL example shown in the introduction), propagation delay is less of an issue and contributes negligibly to the delayed feedback and/or the dynamics of the whole system. Thus, in the rest of the paper, we assume a network configuration with delays arising only due to buffering at the most congested (bottleneck) router of a given path.

In such a model, when the combined sending rate of all flows exceeds the bottleneck capacity, TCP continues increasing its window and queuing the extra data at the bottleneck until congestion is detected. The delay needed for TCP to realize that the buffer is full is directly related to the amount of *bursty* packet loss the flow will suffer and the amount of window-size reduction following the loss.

It is well known that under ideal assumption of instantaneous feedback, all AIMD congestion controls converge to the fair operating point with minimal packet loss [2]. However, when large buffers in the network create delays, TCP becomes more "unstable" and exhibits complex behavior on small time-scales. Consider one such example. We use the `ns2` simulator and examine two competing TCP flows sharing a bottleneck link with 0.2 mb/s bandwidth and 10 ms propagation delay. The buffer size is 20 packets and the sampling rate is 100 ms. Figure 2 depicts the combined rate of these two flows. As seen in the figure, the *average* combined rate is just below the link's capacity; however, the instantaneous combined rate reaches as high as 0.6 mb/s, which overflows the link and leads to large bursts of packet loss. In response to this loss, the combined rate periodically drops almost to zero, which leads to under-utilization of network resources and amplified oscillations compared to the non-delayed case.

We next examine how fairness between the two flows in Figure 2 is affected by delays. As shown in Figure 3 (a),

[1]In this paper, we generally assume a control-theoretic notion of stability, which is the ability of a flow $x(t)$ to converge to a stationary point x^* as $t \to \infty$. In the context of AIMD, "delayed stability" usually refers to the behavior of the flows in comparison to the non-delayed case.

when the link becomes congested, the flows persistently oscillate around the fairness line and do not maintain fairness on small time-scales. Moreover, the trajectory exhibits no noticeable regularity and appears unpredictable. In contrast, Figure 3 (b) depicts the behavior of two TCP flows under immediate feedback, where the oscillations along the fairness line are relatively small and fairness is maintained at all times.

To understand the delayed behavior of TCP in analytical terms, we next investigate its control equation and derive the amount of lost data as a function of queuing delay D.

2.3 Delayed TCP

It is well known that AIMD mechanisms buffer excess data when the bottleneck link is saturated since feedback delays prevent the sources from adjusting their rates in a timely manner. For sufficiently large feedback delays (i.e., large buffers), end-users eventually overshoot the bottleneck link and experience bursty packet losses due to the excess data sent into the network before congestion is detected. We call this behavior "excessive buffering" and examine its extent in AIMD and other protocols in the rest of this paper.

We start our investigation with window-based AIMD (e.g., TCP) schemes. Our next result shows that as soon as the bottleneck link is saturated, TCP automatically switches the growth rate of its window $W(t)$ from linear to $\sqrt{t}$. This naturally leads to a more "conservative" behavior of TCP under non-negligible buffering delays and explains its advantage over the other methods studied in this paper.

LEMMA 1. *After the bottleneck link is saturated, the size of TCP's congestion window grows proportionally to* $\sqrt{t}$.

PROOF. Assume discrete time and recall that TCP's congestion window $W(t)$ is increased by $MTU^2/W(t-1)$ upon each positive (non-duplicate) ACK [1]. For clarity of presentation, we express window size in units of packets instead of bytes. Using this notation, $W(t)$ is given by:

$$W(t) = \begin{cases} W(t-1) + 1/W(t-1) & \text{per ACK} \\ \beta W(t-1) & \text{per loss} \end{cases}, \quad (1)$$

where β is the factor of multiplicative decrease and $W(t)$ is congestion window in units of packets. After the link is saturated, TCP increments its window in response to a stream of ACKs coming from the receiver at fixed average rate C pkts/sec, where C is the bottleneck link capacity. This happens because the bottleneck continues buffering some incoming data, while transmitting the remaining packets to the receiver exactly at the rate of C pkts/sec. Once these packets arrive at the receiver, they generate a stream of ACKs at the same packet-rate C. It is common to approximate (1) with a fluid model, in which the ACKs arrive as a fluid at the exact rate C. Then, we can re-write (1) using a differential equation:

$$\frac{dW}{dt} = \frac{C}{W}. \quad (2)$$

Re-organizing (2) and integrating both sides, we get:

$$W(t) = \sqrt{2Ct + \delta}, \quad (3)$$

where $\delta = W(0)^2$ is the integration constant. □

We next consider an `ns2` simulation to validate the conclusion of Lemma 1. In this simulation, we run a single TCP flow over a bottleneck link with an infinitely large buffer. The packet size is 1,040 bytes, the bottleneck link capacity $C = 2$ mb/s (244 pkts/sec), and the round-trip propagation delay is 60 ms. Using $C = 244$ pkts/sec in (3), observe from Figure 4 that `ns2` simulations match the model very well.

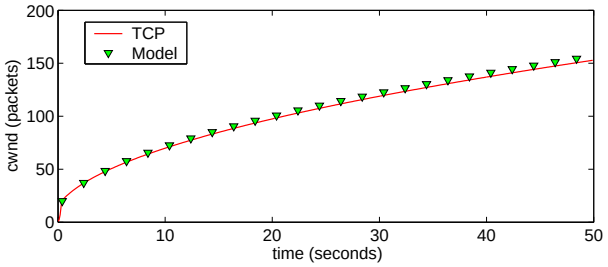


Figure 4: The congestion window of a single TCP flows under delay.

As demonstrated above, TCP overshoots the bottleneck link after saturation and its congestion window tends to infinity provided that the bottleneck queuing delay allows so. Note, however, that the sending rate $r(t)$ of TCP does not grow to infinity (and in fact converges to link capacity C as we show below) since the RTT also increases when packets start being queued at the bottleneck link. Nevertheless, the amount of extra data sent into the link (all of which gets lost *prior* to TCP's reaction to the actual losses) is non-negligible as we show in the next result.

LEMMA 2. *The aggregated amount of lost data in window-based AIMD during each overshoot is proportional to the square root of the buffering delay* $\sqrt{D}$.

PROOF. Assume the time starts at $t = 0$ when the bottleneck link is about to overflow. For each received ACK after $t = 0$, a TCP source sends out $1 + 1/W(t)$ packets, which means that the amount of "extra" data injected into the network is $1/W(t)$ packets per ACK. As discussed in the proof of Lemma 1, the ACKs are fed back to the sender at rate C pkts/sec. Then, the amount of extra data $S(t)$ sent into the link during the segment $[0, t]$ can be modeled by a simple recurrence:

$$S(t) = S(t-1) + 1/W(t-1), \quad (4)$$

where discrete time t is given in ACKs and starts from the point of bottleneck saturation. Converting the above into a differential equation and shifting time to seconds:

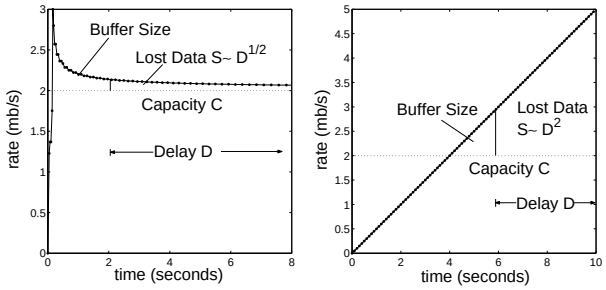
$$\frac{dS(t)}{dt} = \frac{C}{W(t)}, \quad (5)$$

where congestion window $W(t)$ is given in (3). Expanding $W(t)$:

$$\frac{dS(t)}{dt} = \frac{C}{\sqrt{2Ct + \delta}}, \quad (6)$$

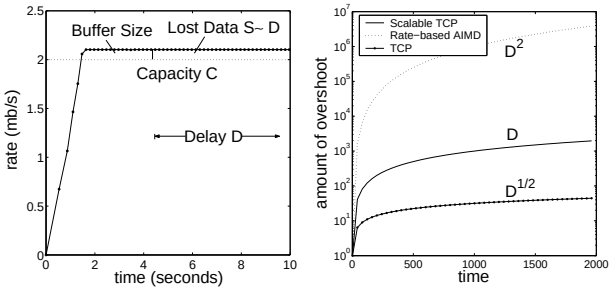
where $\delta = W(0)^2$ is again the square of the congestion window just before the link overflows at time $t = 0$. Re-organizing the terms in (6) and integrating both sides, we have:

$$S(D) = \int_0^D \frac{C}{\sqrt{2Ct + \delta}} dt. \quad (7)$$



(a) (b)

Figure 5: (a) Rate adjustment of window-based protocols under delays. (b) Rate adjustment of rate-based protocols under delays.



(a) (b)

Figure 6: (a) Rate adjustment of Scalable TCP under delay. (b) Comparison of lost data per overshoot in TCP, Scalable TCP, and rate-based AIMD.

Solving this integral, we get $S(D) \sim \sqrt{2CD+\delta}$. □

We now return to the sending rate $r(t)$ of TCP. Recall that after the bottleneck link is saturated, TCP sends out $1 + 1/W(t)$ packets per ACK and ACKs arrive at rate C. According to (3), congestion window W tends to infinity as t becomes large. Thus, $1+1/W(t) \to 1$ and TCP eventually sends out exactly one packet per ACK and converges its rate $r(t)$ to C as time progresses.

Consider an `ns2` illustration in Figure 5 (a), where we use the same configuration as in Figure 4 except that we now disable slow start to better simulate TCP's behavior in congestion avoidance. Notice in the figure that, after the bottleneck link becomes full, the increase in TCP's rate is slowed down and its $r(t)$ eventually converges to capacity C.

2.4 Delayed Rate-based AIMD

We next examine a class of *rate-base* AIMD flows, in which the control actions take places once per RTT instead of once per ACK. Note that both window-based and rate-based AIMD perform essentially the same until the point at which their sending rates start to exceed capacity C; after that, their performance becomes drastically different. The explanation of this phenomenon is simple as rate-based methods must battle the various difficulties in *accurately* and *timely* estimating the RTT and noticing its increase in response to a growing buffer at the bottleneck (this information is automatically supplied to window-based methods through positive ACKs). Unfortunately, a closed-form solution to the exact queuing model coupled with end-flow control equations does not exist for both rate-based AIMD and TFRC, even when we assume that the delay in obtaining RTT samples is negligible. The case becomes more complicated when the RTT feedback is delayed and/or smoothed with an exponential filter.

In this paper, we only solve the simpler case of rate-based AIMD and TFRC in which the RTT is not accurately tracked by the source (i.e., is perceived to remain more or less constant until packet loss is detected) and leave the more extensive analysis of the variable RTT for the full version of the paper.

LEMMA 3. *Under constant RTT, the amount of lost data in rate-based AIMD during each overshoot is proportional to* D^2.

PROOF. The equation for rate-based AIMD is given by:

$$r(t) = \begin{cases} r(t-RTT)+\alpha & \text{per RTT} \\ \beta r(t-RTT) & \text{per loss} \end{cases}, \qquad (8)$$

where $r(t)$ is the sending rate at time t. The source increases its sending rate by constant α per RTT and reduces it by a factor of β upon each packet loss. Taking feedback delays into account, the increase phase of (8) can be written in the fluid sense as following:

$$\frac{dr(t)}{dt} = \frac{\alpha}{RTT(t-\Delta)}, \qquad (9)$$

where Δ is the delay needed to drain parts of the buffer for the source to recognize the increase in the RTT and $RTT(t) = q(t)/C$ is the queue size at time t divided by capacity C. Closed-form solution to this model does not exist even when the feedback delay is zero (i.e., $\Delta = 0$). We abandon this direction and study a model with constant RTT, which often happens in practice when $\Delta \geq D$ and the flow cannot realize that its RTT has increased *until after* it has drained the bottleneck queue (by which time packet loss has occurred and any attempts to adjust the rate are too late from the control-theoretic view).

Solving (9) with constant RTT, we obtain that the sender's rate is a linear function of time (i.e., $r(t) \sim t$) and the amount of lost data by time t is:

$$S(t) = \int_0^t (r(u) - C)du = \int_0^t r(u)du - Ct, \qquad (10)$$

where time $t = 0$ is again the instant when the buffer is about to overflow. From (10), we obtain that $S(t) \sim t^2$ and $S(D) \sim D^2$. □

This situation is illustrated in Figure 5 (b), in which the amount of overshoot S grows quadratically as a function of time needed for the source to react to packet loss. Taking into account *variable* RTT, numerical solutions to the exact queuing model (9) show that for a certain amount of time immediately following the overshoot of C, $r(t)$ behaves as a linear function; however, the remaining increase in $r(t)$ is only logarithmic. Nevertheless, both cases show that rate-based AIMD grows its sending rate to infinity under a sufficiently large delay.

Compared to window-based AIMD methods, this growth in $r(t)$ is clearly a problem and supports the observation that rate-based AIMD flows experience more packet loss and higher oscillations than their window-based counterparts [12], [16].

3. NEXT-GENERATION TCP

Recent research efforts to design better congestion controls for high-bandwidth networks have led to the development of next-generation TCPs – High-speed TCP (HSTCP) [4] and Scalable TCP [10] – which incorporate simple and easily deployable changes to classical TCP. Since HSTCP is similar to Scalable TCP, we only consider the latter and examine its behavior under delayed feedback.

LEMMA 4. *The amount of lost data in Scalable TCP during each overshoot is proportional to D.*

PROOF. Recall that Scalable TCP relies on the following binary-feedback controller [10]:

$$W(t) = \begin{cases} W(t-1) + 0.01 & \text{per ACK} \\ .875W(t-1) & \text{per loss} \end{cases}, \quad (11)$$

where $W(t)$ is the size of congestion window at time t. Notice that after each RTT, congestion window $W(t)$ is increased by a factor of 1.01 since for each RTT, the number of arriving ACKs is at most equal to the window size (i.e., Scalable TCP is an MIMD controller).

Similar to TCP, the sending rate of Scalable TCP does not grow at the same pace as the window size because of the increased RTT. After the bottleneck link is fully utilized, the source sends out 1.01 packets per ACK (i.e., its rate $r(t) = 1.01C$ exceeds bottleneck capacity by a fixed fraction). Thus, the amount of excess data sent per ACK is fixed, i.e., 0.1 packets, and the total overshoot S grows linearly with time, i.e., $S \sim D$. □

Figure 6 (a) demonstrates the buffering behavior of Scalable TCP in `ns2`. For convenience of visualization, we change STCP's window increase step size from 0.01 to 0.05 such that the difference between the steady-state flow rate and capacity C is easy to identify. As the figure shows, simulations match the discussion in Lemma 4 very well.

To summarize the results, Figure 6 (b) shows a comparison of the amount of lost data among all three methods. Notice that window-based self-clocking in TCP and Scalable TCP are powerful mechanisms that prevent the sending rates of these flows from growing to infinity when the bottleneck link becomes saturated.

4. TFRC

TFRC (TCP-Friendly Rate Control) (e.g., [5], [14]) has become a de-facto standard for multimedia applications. Instead of immediately responding to congestion in a manner like TCP, TFRC gradually adjusts its rate if the congestion persists. Recall that TFRC uses a discrete TCP-friendly equation and directly adjusts its rate based on the latest measurement of packet loss and RTT:

$$r(n) = \frac{MTU}{\sqrt{p(n-\Delta_1)}RTT(n-\Delta_2)}, \quad (12)$$

where MTU is the maximum transmission unit, $p(n)$ is the long-term average packet loss computed at time n, and $RTT(n) = q(n)/C$ is the round trip delay seen inside the router at time n. Since both packet loss and RTT are fed back from the network, their values are retarded by the corresponding feedback delays Δ_1 and Δ_2.

Again assuming that the source maintains a constant RTT until the first loss is detected (i.e., either because $\Delta_2 > D$ or the smoothing filter on RTT exhibits slow convergence), TFRC's behavior is stated in the following lemma.

LEMMA 5. *Under constant RTT, the amount of lost data in TFRC during each overshoot is proportional to D^2.*

PROOF. Assume at time $t = 0$ the buffer is about to overflow and the average long-term packet loss up to that point is strictly positive. Let $M(n) > 0$ be the number of lost packets up to time n and $T(n)$ be the total number of transmitted packets up to time n. Modeling $p(n)$ as the long-term average loss[2], we get:

$$p(n) = \frac{M(n)}{T(n)}, \ t \geq 0, \quad (13)$$

where $M(n) = M$ is constant since the source has not detected any *new* buffer overflows and continues to perceive the network as uncongested. Since the change in $T(n)$ is simply rate $r(n)$, we can write the following continuous-time model:

$$\frac{dT(t)}{dt} = r(t), \quad (14)$$

where dT/dt represents the number of packets sent per time unit. Combining this with (12)-(13) and assuming instantaneous feedback, we have:

$$\frac{dT(t)}{dt} = \frac{\omega\sqrt{T(t)}}{\sqrt{M}}, \quad (15)$$

where $\omega = MTU/RTT$ is a constant. Reorganizing (15) and integrating both sides, we get $T(t) \sim t^2$ and $r(t) \sim t$. Thus, the amount of lost data also increases quadratically with delay D. □

We next explore control-theoretic stability of TFRC under the assumption that an AQM-enabled router can feed back the exact value of the most-recent packet loss $p(n)$. Such AQM support no longer requires TFRC to smooth the long-term packet loss obtained from the receiver and may potentially improve TFRC's performance. However, as our next lemma shows, this is not the case.

LEMMA 6. *Under AQM feedback and constant RTT, TFRC can only be stabilized at points r^* that incur no less than 33% packet loss. For other values of packet loss, TFRC cannot be stabilized and oscillates.*

PROOF. Write a TFRC control equation for a single flow and immediate AQM feedback $p(n) = (r(n) - C)^+/r(n)$:

$$r(n) = \frac{\omega\sqrt{r(n-1)}}{\sqrt{r(n-1) - C}}, \quad (16)$$

where $\omega = MTU/RTT$ is a constant and we assume that $r(n-1) > C$. The only non-negative stationary point of this system is given by:

$$r^* = \frac{C + \sqrt{C^2 + 4\omega^2}}{2}. \quad (17)$$

[2]To keep the problem tractable, we simplify several aspects in TFRC's computation of packet loss.

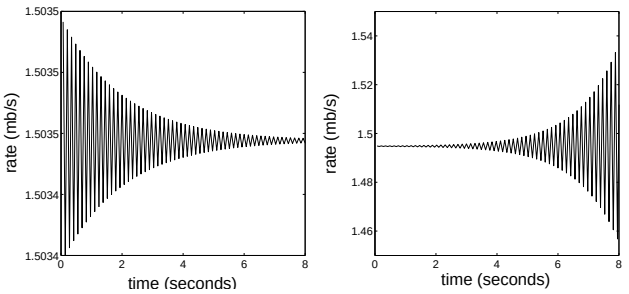


Figure 7: (a) Behavior of TFRC for different stationary points: (a) $r^* = 1.503C$; (b) $r^* = 1.494C$.

We next check local stability of TFRC at r^* and derive its average loss in the stationary state. Linearizing (16) in r^*, we get:

$$\left.\frac{\partial r(n)}{\partial r(n-1)}\right|_{r^*} = \left.\frac{-\omega C}{2\sqrt{r}(r-C)^{3/2}}\right|_{r^*} = \frac{-C}{2(r^*-C)}. \quad (18)$$

Recall that system (16) is stable at r^* if the absolute value of (18) is less than one, which translates into $|\frac{C}{2(r^*-C)}| < 1$ where r^* is in (17). Solving this inequality, we have:

$$r^* > \frac{3C}{2}. \quad (19)$$

This means that system (16) may be stable in the stationary point only at the expense of at least 33% of the packets being dropped. □

According to (17), the stationary point r^* can be tuned by properly choosing constant ω. Consider a simulation of (16) in Figure 7 with two different stationary points. In Figure 7(a), we set capacity C to 1 mb/s, the MTU to 1,500 bytes, and the RTT to 13.8 ms ($\omega = 870$ kb/s). Under these conditions, the stationary point r^* is 1,503 kb/s and the flow clearly converges to r^* after decaying oscillations. In Figure 7(b), we adjust the RTT to 14 ms such that $r^* = 1,494$ kb/s. As the figure shows, the system diverges even when started in a close vicinity of the stationary point. Thus, within the operating range of most applications (i.e., packet loss below 33%), AQM-TFRC cannot be stabilized.

5. DISCUSSION AND CONCLUSION

It is possible that eventually the Internet will adopt a class of AQM-based mechanisms that are capable of providing asymptotically stable congestion control to end-flows. In such a case, we find that a wide variety of algorithms, including XCP [8] and Kelly controls [9], [11], [13], will be able to supply video flows with rate-based, oscillation-free virtual channels. While the issue of designing oscillation-free, rate-based congestion control for best-effort networks remains open, we find that window-based protocols are expected to become more popular as they offer better (albeit far from ideal) performance under delay and an easy-to-implement platform being part of the TCP/IP protocol stack.

On a bigger scale, we believe that more effort should be put into scalable extensions to control methods that are *provably* stable under arbitrary delay in the control-theoretic sense and that their presentation to the streaming community should become more "digestible." Our initial work in this direction assuming AQM support is reported in [3], [7], [18], which demonstrates that *rate-based* control methods can not only be oscillation-free, but also provably stable and fair under *heterogeneous* end-user feedback delays.

6. REFERENCES

[1] M. Allman, V. Paxson, and W. Stevens, "TCP Congestion Control," *RFC 2581*, April 1999.

[2] D.-M. Chiu and R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN Systems*, 17(1):1–14, June 1989.

[3] M. Dai and D. Loguinov, "Analysis of Rate-Distortion Functions and Congestion Control in Scalable Internet Video Streaming," *ACM NOSSDAV*, June 2003.

[4] S. Floyd, "High-speed TCP for Large Congestion Windows," *RFC 3649*, December 2003.

[5] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-Based Congestion Control for Unicast Applications," *ACM SIGCOMM*, August 2000.

[6] V. Jacobson, "Congestion Avoidance and Control," *ACM SIGCOMM*, August 1988.

[7] S.-R. Kang, Y. Zhang, M. Dai, and D. Loguinov, "Multi-Layer Active Queue Management and Congestion Control for Scalable Video Streaming," *IEEE ICDCS*, March 2004.

[8] D. Katabi, M. Handley, and C. Rohrs, "Congestion Control for High Bandwidth Delay Product Networks," *ACM SIGCOMM*, August 2002.

[9] F. Kelly, A. Maulloo, and D. Tan, "Rate Control for Communication Networks: Shadow Prices, Proportional Fairness and Stability," *Journal of the Operational Research Society*, 49(3):237–252, March 1998.

[10] T. Kelly, "Scalable TCP: Improving Performance in High-speed Wide Area Networks," *First International Workshop on Protocols for Fast Long-Distance Networks*, February 2003.

[11] S. Kunniyur and R. Srikant, "Analysis and Design of an Adaptive Virtual Queue (AVQ) Algorithm for Active Queue Management," *ACM SIGCOMM Computer Communication Review*, 31(4):123 – 134, August 2001.

[12] D. Loguinov and H. Radha, "End-to-End Rate-Based Congestion Control: Convergence Properties and Salability Analysis," *IEEE/ACM Transactions on Networking*, 11(5):564–577, August 2003.

[13] S. H. Low and D. E. Lapsley, "Optimization Flow Control I: Basic Algorithm and Convergence," *IEEE/ACM Transactions on Networking*, 7(6):861–874, December 1999.

[14] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose, "Modeling TCP Throughput: A Simple Model and Its Empirical Validation," *ACM SIGCOMM*, September 1998.

[15] R. Rejaie, M. Handley, and D. Estrin, "RAP: An End-to-End Rate-Based Congestion Control Mechanism for Realtime Streams in the Internet," *IEEE INFOCOM*, March 1999.

[16] Y. Xiong, J.-C. Liu, K. Shin, and W. Zhao, "On the Modeling and Optimization of Discontinuous Network Congestion Control Systems," *IEEE INFOCOM*, March 2004.

[17] Y. Yang, M. Kim, and S. Lam, "Transient Behaviors of TCP-friendly Congestion Control Protocols," *IEEE INFOCOM*, April 2001.

[18] Y. Zhang, S.-R. Kang, and D. Loguinov, "Delayed Stability and Performance of Distributed Congestion Control," *To Appear in ACM SIGCOMM*, August 2004.