Report Number: WUCSE-2003-9

2003-02-26

SRC: Stable Rate Control for Streaming Media

Cheng Huang and Lihao Xu

Rate control, in conjunction with congestion control, is important and necessary to maintain both stability of overall network and high quality of individual data transfer flows. In this paper, we study stable rate control algorithms for streaming data, based on control theory. We introduce various control rules to maintain both sending rate and receiver buffer stable. We also propose an adaptive two-state control mechanism to ensure the rate control algorithms are compatible to TCP traffics. Extensive experimental results are shown to demonstrate the effectiveness of the rate control algorithms.
... Read complete abstract on page 2.

SRC: Stable Rate Control for Streaming Media

Cheng Huang and Lihao Xu

Complete Abstract:

Rate control, in conjunction with congestion control, is important and necessary to maintain both stability of overall network and high quality of individual data transfer flows. In this paper, we study stable rate control algorithms for streaming data, based on control theory. We introduce various control rules to maintain both sending rate and receiver buffer stable. We also propose an adaptive two-state control mechanism to ensure the rate control algorithms are compatible to TCP traffics. Extensive experimental results are shown to demonstrate the effectiveness of the rate control algorithms.

2003-9

SRC: Stable Rate Control for Streaming Media

Authors: Huang, Cheng; Xu, Lihao

February 26, 2003

Abstract: Rate control, in conjunction with congestion control, is important and necessary to maintain both stability of overall network and high quality of individual data transfer flows. In this paper, we study stable rate control algorithms for streaming data, based on control theory. We introduce various control rules to maintain both sending rate and receiver bu er stable. We also propose an adaptive two-state control mechanism to ensure the rate control algorithms are compatible to TCP tra cs. Extensive experimental results are shown to demonstrate the e ectiveness of the rate control algorithms.

SRC: Stable Rate Control for Streaming Media

Cheng Huang Lihao Xu
Department of Computer Science and Engineering
Washington University in St. Louis, MO, 63130
{cheng, lihao}@cse.wustl.edu

Abstract

Rate control, in conjunction with congestion control, is important and necessary to maintain both stability of overall network and high quality of individual data transfer flows. In this paper, we study stable rate control algorithms for streaming data, based on control theory. We introduce various control rules to maintain both sending rate and receiver buffer stable. We also propose an adaptive two-state control mechanism to ensure the rate control algorithms are compatible to TCP traffics. Extensive experimental results are shown to demonstrate the effectiveness of the rate control algorithms.

1 Introduction

Streaming media is becoming increasingly important for many applications. In contrast to *bulk data* (e.g., file) transfer, streaming data is more sensitive to transmission delay, network bandwidth fluctuation and high volume data loss. Rate control for streaming media applications is thus crucial in achieving high QoS (*quality of service*). Transmission delay and data loss are usually caused by network *congestions*. Network congestions, in turn, are usually caused by 1) insufficiency of overall network bandwidth due to addition of new data flows into the network; and/or 2) high fluctuation in sending rates of existing data flows. While the first factor is unavoidable and usually addressed by various congestion control algorithms, network congestions are more often caused by the second factor. Wild swings in sending rate of each individual data flow collectively make the network unstable, leading to much harder congestion control, ineffective use of overall network bandwidth and poor transfer quality (large delay, high data loss rate, etc.) of individual data flows. Thus to mitigate and eliminate network congestions, sending rate of each individual data flow should be maintained as stable as possible at the first place. *Stable rate control* algorithms, in conjunction with efficient congestion control algorithms, are thus necessary in maintaining both the stability of overall network and the high transfer quality of individual data flows, especially for streaming flows.

Traditional approach used in data transfer protocols, such as TCP, can cause severe sending rate fluctuation and is not suitable for streaming media. In order to achieve stable transmission rate, various rate control approaches have been proposed. RAP[2] uses a simple AIMD (Additive Increase and Multiplicative Decrease) scheme and adjusts sending rate per RTT or upon packet loss. LDA+[3] controls sending rate based on RTCP feedback information in a much less frequent interval (usually several seconds). TEAR[4] calculates fair receiving rate on receiver side and accordingly adjusts sending rate. Binomial Congestion Control[5] analyzes a class of nonlinear algorithms and shows a set of TCP-compatible algorithms for streaming media. TFRC[6] adjusts sending rate based on the feedback of loss event rate and a TCP throughput equation.

All those approaches focus on the network bandwidth bottleneck and try to provide a slow-responsive sending rate correspondingly. As network bandwidth increases dramatically, both in backbone network brought by advanced optical technologies, such as *DWDM* (Dense Wave Division Multiplexing) and on last-mile links with the widely deployment of broadband service, more and more network bandwidth is becoming available to streaming applications. Thus, although the network could be the bottleneck at sometime during a streaming session, the available bandwidth should be larger than the consumption rate

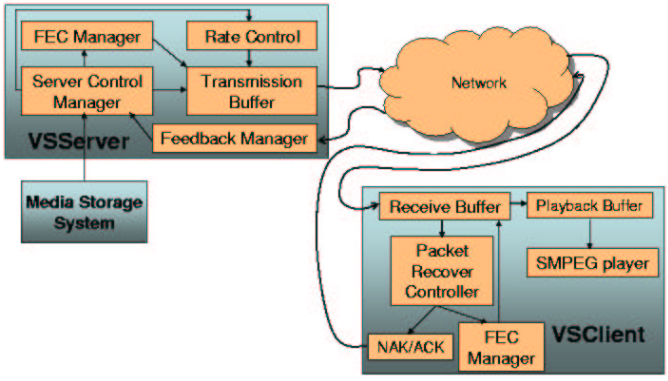


Figure 1: Streaming Media System Architecture

at most of the time. We thus need some new rules to regulate streaming flows under this circumstance. In this paper, we first show the failure of traditional TCP flow control in streaming applications. We then introduce receiver buffer level into feedback and apply control-theoretic approach to address this flow control problem. We design a set of rate control rules and the goal is to achieve relatively stable sending rate, while maintaining certain receiver buffer level. The effectiveness of these rules are demonstrated through real streaming experiments over Internet.

In an open environment like the Internet, streaming applications also need to follow the concept of fairly sharing so as not to cause starvation of other TCP traffic or congestion collapse[1]. Therefore, we combine the rate control rules with an existing TCP-friendly scheme and propose a two-state adaptive rate control mechanism: in the UNSTABLE state, sender regulates sending rate in a slowly responsive TCP-friendly fashion; in the STABLE state, sender enforces the rate control rules. State switching occurs based on both network condition (available bandwidth and packet loss rate) and receiver buffer level. Our extensive real Internet experiments show three properties of this mechanism, namely, TCP-friendliness, stable sending rate and constant receiver buffer level.

In Sec. II, we first show the ineffectiveness of TCP flow control mechanism in streaming media applications. Then, we describe the design of STABLE state rate control rules with different approaches and present experiment results. In Sec. III, we focus on the two-state adaptive rate control mechanism and also show experiment results on an architecture (Figure 1). The future directions are mentioned in Sec. IV.

2 STABLE State Rate Control Rules

2.1 Problem Statement

Figure 2 shows a simplified diagram of the streaming media system as in Figure 1, with only those components that are of interests to us for this particular problem. During a streaming session, the send component in the streaming server retrieves data constantly from the file server and sends it out to the network. The streaming client receives data and stores in its local buffer. The decoder component pulls data from the buffer as needed. The streaming client constantly feeds back buffer level information to the streaming server, which accordingly decides its sending rate. The goal of rate control mechanism is to adjust sending rate properly so as to minimize the possibility of overflowing or under-flowing receiver buffer and minimize the effect of rate adjustment on network in the meanwhile. Turning these criteria to measurable variables, we design a rate control algorithm, which keeps the client buffer level relatively constant and minimizes the fluctuation of sending rate at the same time.

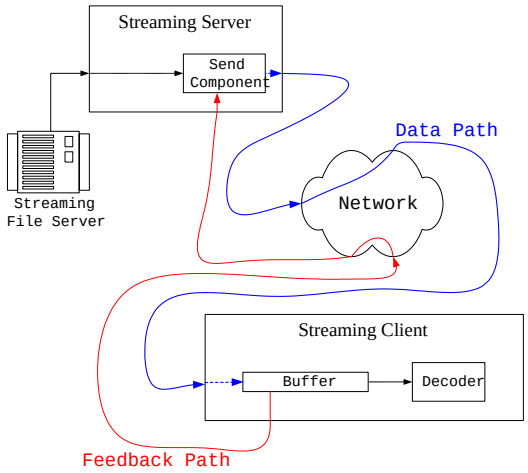
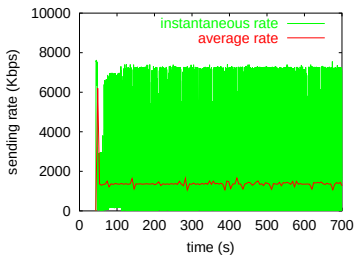
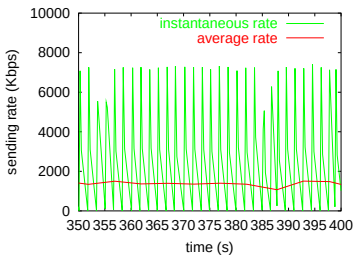


Figure 2: Abstract Streaming System Diagram



(a) Entire Streaming Session



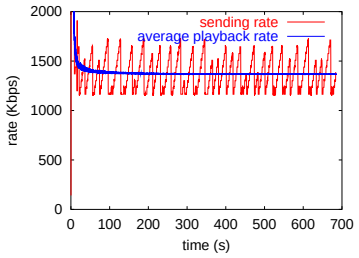
(b) Part of the Streaming Session

Figure 3: TCP Flow Control in Streaming Scenario

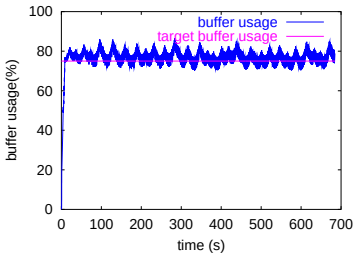
2.2 Failure of TCP Flow Control in Streaming Media

First, let us take a look at how TCP flow control fails in data streaming. Figure 3 shows an experiment result of streaming a 12 minute long video sequence from *tehran.clic.cs.columbia.edu* to *elm.cs.wustl.edu* (with an average TCP bandwidth of 7.5 Mbps and an average RTT of 32 ms) using TCP protocol. The video sequence is in MPEG-1 format and is excepted from *Terminator 2*. Its average rate is about 1.4 Mbps. The same sequence is used for all experiments in this paper.

Figure 3(b) is a zoom-in version of Figure 3(a), augmenting the time period $350 \sim 400$ sec. We can see that although the average throughput is relatively constant (close to the bitrate of video sequence itself), the instantaneous sending rate fluctuates greatly. The reason is TCP flow control dominates transfer when available network bandwidth is larger than client consumption rate and thus sending rate is actually controlled by TCP receive window. To be more specific, when there is room in receiver buffer, receive window opens and sender pushes as much data as possible. And when there is no room and receive window closes, almost no data is sent out. The other reason that contributed to this phenomenon is, although receiver consumes data in relatively constant rate in large time scale, it is not true in small time scale. We will analyze this issue in detail in a later section.



(a) Sending Rate



(b) Receiver Buffer Level

Figure 4: Results of HRA

2.3 Heuristic Rate Adaptation (HRA) Algorithm

2.3.1 HRA algorithm

To address the streaming media flow control problem, we start with a heuristic approach. HRA (Heuristic Rate Adaptation) algorithm adopts similar idea to TCP AIMD (Additive Increase and Multiplicative Decrease) algorithm, which tries to approach the maximum available network bandwidth by constantly probing the network with a higher sending rate, until packet loss happens. HRA keeps increasing sending rate additively and decreases it multiplicatively when the receiver buffer level exceeds a certain threshold. The details is as follows:

Algorithm *HRA*
1. **if** (HRA timer expire)
2. **then if** $recvbuf > bufthresh$
3. **then** $sendrate = (1 - \beta) \cdot avg\_playrate$
4. **else** $sendrate = sendrate + \alpha \cdot avg\_playrate$
5. **return**

Here, *avg_playrate* is a property of the streaming session. By selecting proper α, β and HRA timer interval, we can keep sending rate within a certain range around *avg_playrate*, while maintaining the receiver buffer level around the threshold *bufthresh*.

2.3.2 Experiment

We implement HRA algorithm in our streaming media system, as shown in Figure 1. For all experiments in this paper, we disable FEC and use only retransmission as error recovery mechanism.

Figure 4 is an experiment result of HRA in the same environment as the TCP experiment in last section. Here, we set α 0.05, β 0.15, HRA timer interval 2.5 sec and *bufthresh* 1.05 times of the target receiver buffer level. By further tuning the parameters, α, β, HRA timer interval and *bufthresh*, it is possible to push sending rate and receiver buffer level into a tighter range. However, the more important thing is that HRA algorithm sheds lights on an approach of using receiver buffer level as an effective feedback information to control sending rate. Therefore, better performance is expected if we can analyze the system from theoretical aspect. We use this feedback information and apply control theory knowledge in the following approach.

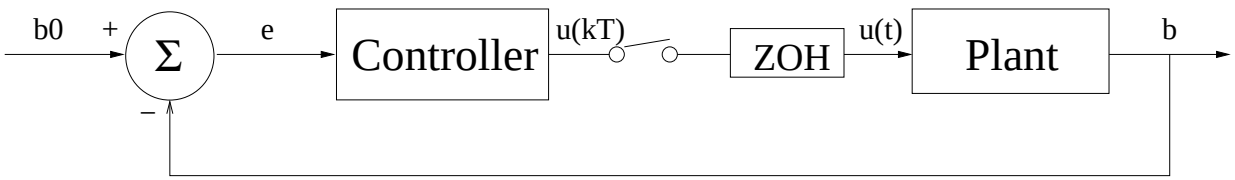


Figure 5: Feedback Control System Diagram

2.4 Feedback Control Model and Basic Design

2.4.1 Feedback Control Model

Figure 5 is a feedback control model[8] abstracted from Figure 2. Here, the **plant** consists of send component in the streaming server, the network and the client buffer. Inside the plant, a sending rate r_s is decided according to a control signal **u(t)** and data in the client buffer is consumed at a constant playback rate r_p. Both r_s and r_p contribute to actual buffer level **b**. The **controller** implements rate control rules. It calculates error **e** between the feedback value of **b** and the reference buffer level $\mathbf{b_0}$ and sets output signal **u(kT)**, at a sample rate $\frac{1}{T}$. The zero-order hold (**ZOH**) maintains the same value within the sample period **T**.

2.4.2 Basic Design

Based on above model, a control rule is designed for the controller using root-locus method[8]. The intuition here is that if we can keep the feedback control system in steady state, then the error between receiver buffer level and the reference is small (close to 0) and the controller output is small too. Thus the sending rate is close to playback rate and relatively stable.

Define control signal u(t) = r_s - r_p, then

$$\dot{b} = u(t) \tag{1}$$

Thus, the *transfer function* of plant is

$$G(s) = \frac{1}{s} \tag{2}$$

If we let *damp ratio* ζ and *natural damp frequency* ω_n to be 0.5 and 1 respectively, the *lead compensation* for G(s) is

$$D(s) = \frac{s+1}{s+10} \tag{3}$$

Using the root-locus method, we can get control gain K = 7.22, with $\omega_n = 0.963$, $\zeta = 0.5$ and *overshoot* $M_p = 16.3\%$. Given ω_n and ζ, we can also calculate the rise time, settling time, peak time of the system, but there are of less interests to us because we focus primarily on the steady state response.

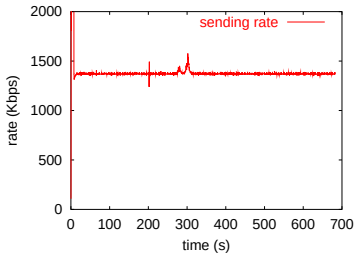
We choose sample period T to be 500 ms, then digitize D(s) using the Tustin's method[8] and get

$$D(z) = \frac{2.5786(z+0.6)}{(z+0.4286)} \tag{4}$$

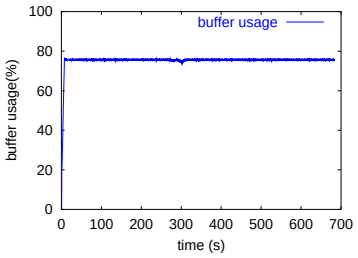
This gives a simple control rule as

$$u(k) = -0.4286u(k-1) + 2.5786(e(k) + 0.6e(k-1)) \tag{5}$$

Mapping this control rule back to our streaming system, sender constantly gets feedback information of receiver buffer level $b(k)$, compares it to the reference buffer level b_0 and calculates the error $e(k)$. It then computes the controller output $u(k)$ and adjusts sending rate accordingly.

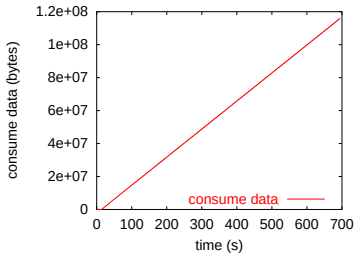


(a) Sending Rate

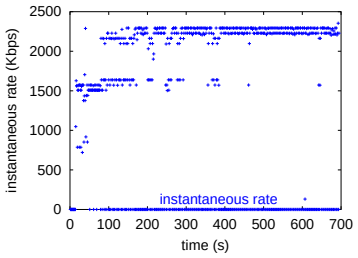


(b) Receiver Buffer Level

Figure 6: Results with a Playback Simulator



(a) Cumulative Consumed Data



(b) Instantaneous Playback Rate

Figure 7: Trace of Real Playback

2.4.3 Experiment 1

In the first experiment, we use a playback simulator in the streaming client. The purpose here is to ensure playback rate is constant without any disturbance, as the model requires. The total receiver buffer size is 4 MB and the reference buffer level $\mathbf{b_0}$ is 3 MB. The playback rate r_p is 1.4 Mbps. The result is shown in Figure 6. From the results, we can see receiver buffer is well controlled toward the reference buffer level and sending rate is not fluctuating significantly. This shows the control rule is effective.

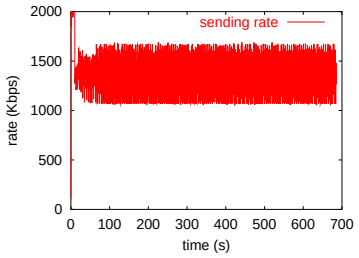
2.4.4 Experiment 2

In our second experiment, we test the control rule with a real player, which introduces disturbance to playback rate.

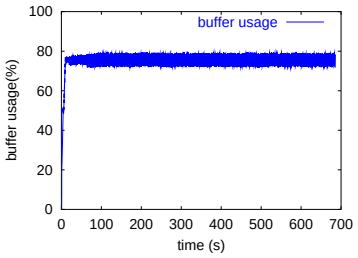
First, let's examine the property of a real playback. From Figure 7(a), we can tell that playback rate is rather constant in large time scale. However, Figure 7(b) shows that instantaneous playback rate oscillates considerably. We expect this fact will have significant effect on our feedback control system. And the great fluctuation of sending rate shown in Figure 8 confirms this.

2.4.5 Receiver Buffer Level Smoothing

The reason that the control rule doesn't work well for a real player is because the instant receiver buffer level changes significantly. This suggests if the reference buffer level is smoothed to be relatively constant even in small time scale, we can eliminate the effect of playback rate disturbance and achieve better control results.

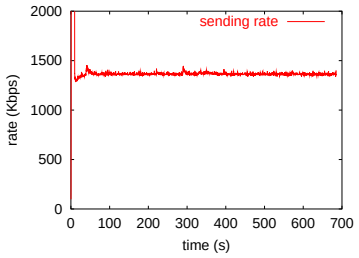


(a) Sending Rate

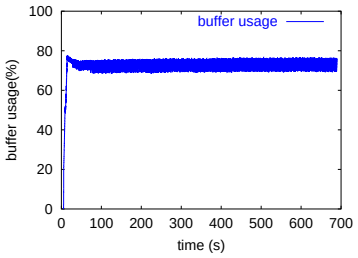


(b) Receiver Buffer Level

Figure 8: Results with a Real Player



(a) Sending Rate



(b) Receiver Buffer Level

Figure 9: Results with a Real Player after Receiver Buffer Smoothing

From Figure 7(a), we can see that consumed data lies on a straight line with a certain slope. So we design a heuristic smoothing approach here, where is to calculate the slope with a measured value and predict a smoothed value with that slope. To be specific, define t_{now} as current measurement time, t_{last} as last measurement time, C_m as the measurement value of cumulative consumed data at t_{now}, C_p as the predict value of cumulative consumed data at t_{now} and C_{last} as the measurement value of cumulative consumed data at t_{last}. C_0 and t_0 represent the measurement value of consumed data and time when playback starts. Thus, we can predict C_p by

$$slope = \frac{C_m - C_0}{t_m - t_0} \tag{6}$$

$$C_p = C_{last} + slope(t_m - t_{last}) \tag{7}$$

If we let R_m be the total data received at time t_m, then the smoothed buffer level $\hat{\mathrm{b}}$ is

$$\hat{\mathbf{b}} = R_m - C_p \tag{8}$$

We implement this receiver buffer level smoothing mechanism and the results in Figure 9 show significant improvement in terms of sending rate fluctuation.

2.5 Refined Model and Advanced Design

2.5.1 Refined Model

In above section, we see that our heuristic approach of smoothing buffer level helps to decrease sending rate fluctuation significantly. More improvement in controlling the stability of sending rate is achieved by

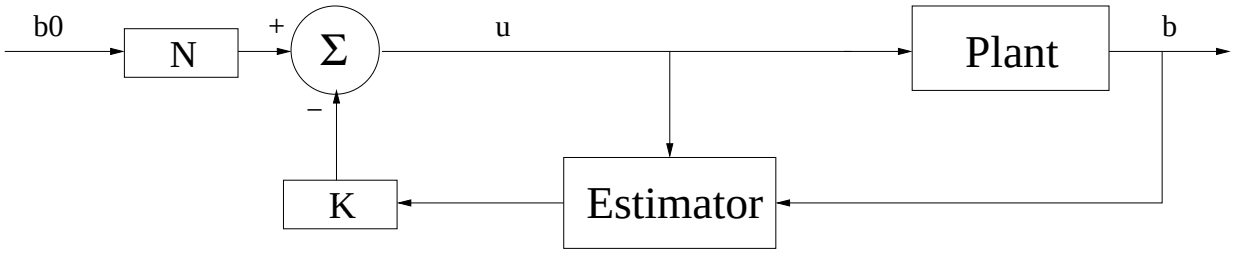


Figure 10: Feedback Control System Diagram with Estimator

introducing **estimator** into the feedback control system. As described in [9], an estimator is used as an approximation when it is not possible to measure the state variable. In our case, however, we introduce estimator to decrease the effect of receiver buffer level fluctuation.

Figure 10 is a feedback control model with an estimator, where we use state-space method[9] to do design.

2.5.2 Advanced Design

First, the state-space description of the plant is:

$$\dot{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{G}u \tag{9}$$

$$b = \mathbf{H}\mathbf{x} + Ju \tag{10}$$

Here, $\mathbf{F}$, $\mathbf{G}$, $\mathbf{H}$ and J are the *system matrix*, *input matrix*, *output matrix* and *transmission term*, respectively. $\mathbf{x}$ is the *system state vector*. u is control input and b is receiver buffer level, as shown in Figure 10. Given a sample time period T, we can obtain a discrete state-space representation as

$$\dot{\mathbf{x}}(k+1) = \mathbf{\Phi}\mathbf{x}(k) + \mathbf{\Gamma}u(k) \tag{11}$$

$$b(k) = \mathbf{H}\mathbf{x}(k) + Ju(k) \tag{12}$$

After introducing $\hat{\mathbf{x}}$, $\mathbf{K}$ and $\bar{N}$ as the *estimate matrix* of $\mathbf{x}$, *control gain* and *reference input gain multiplication*, the control rule of the complete feedback control system is

$$\hat{\mathbf{x}}(k) = [\mathbf{\Phi} - \mathbf{\Gamma}\mathbf{K} - \mathbf{L_c}\mathbf{H}\mathbf{\Phi} + \mathbf{L_c}\mathbf{H}\mathbf{\Gamma}\mathbf{K}]\hat{\mathbf{x}}(k-1) + \mathbf{L_c}b(k) \tag{13}$$

$$u(k) = -\mathbf{K}\hat{\mathbf{x}}(k) + \bar{N}b_0 \tag{14}$$

$\mathbf{L_c}$ is the *gain matrix* of the current estimator and $\mathbf{K}$ is the *feedback gain matrix* of the close-loop system.

2.5.3 Experiment

Given the transfer function of the plant as

$$G(s) = \frac{1}{s} \tag{15}$$

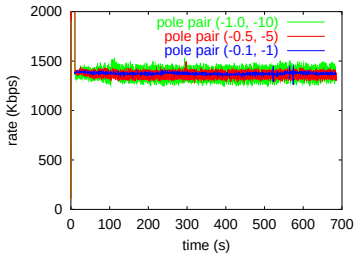
We can get

$$\mathbf{F} = [0] \qquad \mathbf{G} = [1]$$

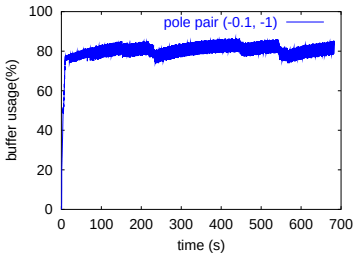
$$\mathbf{H} = [1] \qquad J = 0$$

If we choose the same sample period $T = 500$ ms, the same as previous design, then

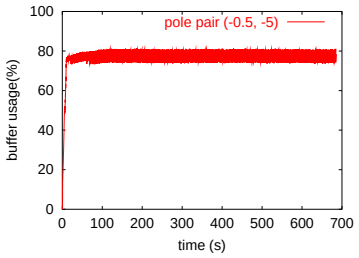
$$\mathbf{\Phi} = [1] \qquad \mathbf{\Gamma} = [0.5]$$



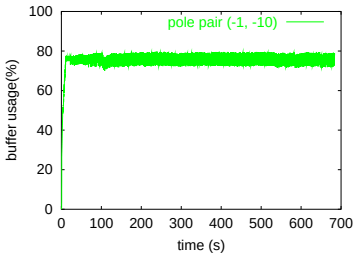
(a) Sending Rate



(b) Buffer Level at Pole Pair$(-0.1, -1)$



(c) Buffer Level at Pole Pair$(-0.5, -5)$



(d) Buffer Level at Pole Pair$(-1, -10)$

Figure 11: Feedback Control with Estimator at Different Pole Pairs

To complete the control system design, we need to select poles for both the close-loop and the estimator. The trade-off here is between the responsiveness to feedback and the fluctuation of sending rate. If we choose poles far away from the origin, the response of the feedback control system is quick. It could have very stable receiver buffer level, but results in great fluctuation of sending rate. On the other hand, if we select poles too close to the origin, the response could be too slow. We would have much more stable sending rate at the price of receiver buffer level fluctuation.

The experiment shown in Figure 11 is carried out in the same environment as previous ones, with different pole pairs and thus different sets of rate control rules. We can see that the smallest pole pair $(-0.1, -1)$, which has the longest settling time, results in the least fluctuated sending rate, however the most fluctuation in buffer level. Vice versa, the largest pole pair $(-1, -10)$, which has the smallest settling time, produces the most fluctuated sending rate, while maintaining the smoothest buffer level.

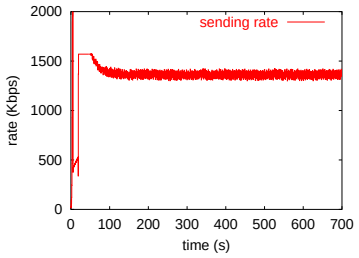
Finally, we decide pole pair $(-0.5, -5)$ to be a good balance, with which both sending rate and receiver buffer level are well controlled within a small range. Correspondingly, the control parameters are:

$$\mathbf{K} = [0.4424]$$
$$\mathbf{L_c} = [0.9179]$$
$$\bar{N} = 0.4424$$

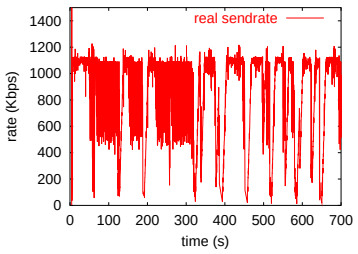
And the control rule is

$$\hat{\mathbf{x}}(k) = [0.0639]\hat{\mathbf{x}}(k-1) + 0.9179b(k) \quad (16)$$
$$u(k) = [-0.4424]\hat{\mathbf{x}}(k) + 0.4424b_0 \quad (17)$$

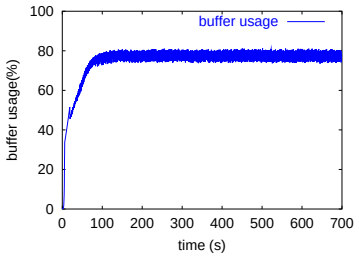
With this rate control rule, we also run experiments through paths with large RTT. Figure 12 shows the result of streaming from *wombat.ee.mu.oz.au* to *elm.cs.wustl.edu* (with an average TCP bandwidth of



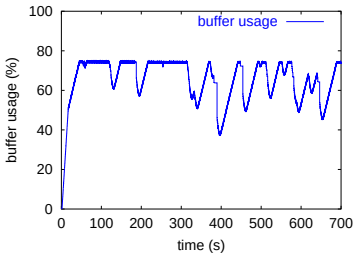
(a) Sending Rate with Control Rule



(b) Sending Rate with TCP



(c) Buffer Level with Control Rule



(d) Buffer Level with TCP

Figure 12: Experiment over Path with Large RTT

1.12 Mbps and an average RTT of 228 ms). We can see that sending rate and receiver buffer level is well controlled from Figure 12(a) and Figure 12(c). The result also shows that the rate control rule is not sensitive to the value of RTT, which is a desirable property since a control rule can be applied for different network paths. In Figure 12(b) and Figure 12(d), we show the result of streaming using TCP protocol. Since the average TCP bandwidth is not enough, we can't streaming the real video sequence of 1.4 Mbps. Instead, we use a playback simulator to consume data at 0.8 Mbps. We can see when buffer level is stable, which means the available network bandwidth is greater than the playback rate and TCP flow control takes effect, sending rate fluctuates severely.

Note although our rate control rule relies on the feedback information in adjusting sending rate, it does not depend on ACK clock, which is used in TCP. So if an acknowledgment packet is lost, rate control uses old value in current time period, which might introduce some disturbance, but certainly does not cause severe problem.

3 Two-State Adaptive Rate Control Mechanism

3.1 Introduction

In last section, the primary goal of applying feedback rate control rule is to keep relatively stable sending rate, as well as maintaining certain receiver buffer level. However, in a rapidly changing network environment, we might not always get enough network bandwidth as the feedback control rule demands. Thus, although it's reasonable to assume that during the entire streaming session, the average available network bandwidth is larger than the playback rate, it makes less sense to assume that we will always have enough network bandwidth at any instant time point. If we force sending rate to be as high as the feedback control

rule requires, then it is not much different from using encoding rate as sending rate, which actually steals network bandwidth share from cross traffic. To be fair to other traffic in the network, a streaming session should be compatible to TCP traffic, which is the most common traffic in the Internet nowadays.

3.2 TCP-Friendly Rate Control(TFRC) Protocol

TFRC[6] is an equation-based rate control mechanism, proposed by M. Handley et al., and is now under standardization process. It uses a TCP response formula[10] as following:

$$T = \frac{s}{R\sqrt{\frac{2p}{3}} + t_{RTO}(3\sqrt{\frac{3p}{8}})p(1+32p^2)} \tag{18}$$

This gives an upper bound on the sending rate T, as a function of packet size s, round-trip time R, steady-state loss event rate p, and TCP retransmit timeout value t_{RTO}.

TFRC is responsive to network congestion over longer time period and changes sending rate in a slowly-responsive manner. This is desirable to streaming media, as the purpose of its design. It also gives the application knowledge of sending rate upper bound. As long as the application is sending within the bound, it is considered to be compatible to TCP flows, i.e., *TCP-friendly*.

These are exactly features that we need in our rate control scheme to achieve TCP-friendly feature. By integrating TFRC with our feedback rate control rule, we propose a two-state adaptive rate control mechanism.

3.3 Two-State Adaptive Rate Control Mechanism

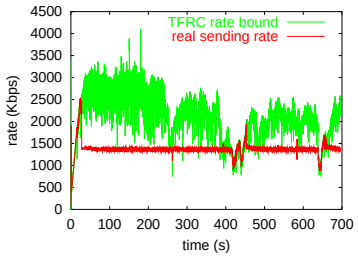
Rate control operates in tow states: the STABLE state and the UNSTABLE state. Normally, it operates in the STABLE state and switches to the UNSTABLE state in two cases: 1) available network bandwidth is not enough; 2) the receiver buffer level is low. In the UNSTABLE state, sending rate is set to an upper bound given by TFRC. In case 1, the draining rate of client buffer is minimized so that it can last longer network congestion period. In case 2, receiver buffer level is pushed up quickly to the control level by the sender sending more data than consumed by the receiver. When the available network bandwidth is enough and the receiver buffer level is high, rate control switches to the STABLE state, where the feedback rate control rule takes over.

3.4 Experiment

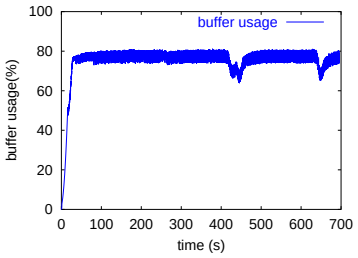
We implement TFRC in our streaming system, integrate it with our feedback rate control rule to get the two-state adaptive rate control mechanism and run experiments over the Internet. Figure 13 is the result of a typical streaming session. This experiment is streaming from *hip.caltech.edu* to *elm.cs.wustl.edu* (with an average RTT of 66 ms), using the same video sequence. We can see from Figure 13(a) that, during three time periods, rate control operates in the UNSTABLE state, where sending rate matches the upper bound given by TFRC. The three time periods are the startup period, 400∼500 sec and 600∼700 sec. In other times, the feedback rate control rule takes effect. From Figure 13(b), we can see that receiver buffer builds up at the startup period quickly. And during two other UNSTABLE period, buffer drains first and returns to target level later. During the STABLE periods, sending rate and receiver buffer level are well controlled within a small range.

Figure 14(a) shows the loss event rate p, which is driven by the actual packet loss pattern shown in Figure 14(b). We can see that the value of p is not directly associated with packet loss number, but with the density of loss event. This is the essential reason of TFRC's slow responsiveness and desirable feature for streaming.

Notice here in our experiment, there are some jitters during 300∼400 sec, where the TFRC sending rate bound is actually below real sending rate. This happens because in our real implementation, when rate control is in the STABLE state, it doesn't immediately switch to the UNSTABLE state when the sending

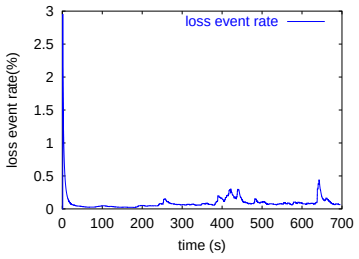


(a) Sending Rate with TCP-friendly Bound

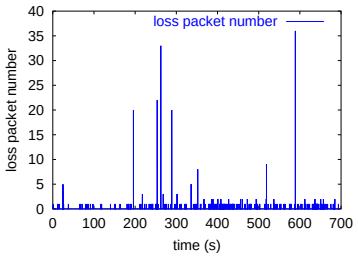


(b) Buffer Level with Bounded Rate

Figure 13: Two-state Rate Control



(a) Loss Event Pattern



(b) Loss Packet Number Pattern

Figure 14: Loss Pattern

rate upper bound is lower than real sending rate. The sender waits to see packet loss before it changes state. One reason of doing this is to stay in the STABLE state whenever possible so as to avoid unnecessary sending rate fluctuation. Another reason is because the sending rate upper bound is limited to two times of receiving rate reported by the receiver. On systems (such as Linux used in our experiment) with clock tick about 10 ms, the feedback receiving rate fluctuates considerably due to measurement disturbance. By filtering the feedback receiving rate through certain anti-aliasing filter and not switching state too rapidly, we can decrease this effect.

4 Conclusions

In this paper, we describe a control-theoretic approach to regulate sending rate of streaming media and also present a two-state adaptive rate control mechanism. We show their effectiveness by real Internet experiments.

There remain several issues to be addressed. Our feedback control model does *not* take transmission delay into account. When a sample time period is significantly larger than RTT, our experiments show the effect of RTT is minor. For long distance data streaming, however, we need to investigate how a large RTT affects rate control rules. It is also interesting to explore how to extend our rate control algorithms to multi-layered data streaming, when there is at least one more state (rate switching) to consider.

Acknowledgements

The authors would like to thank Parallel and Distributed Systems Group at Caltech, Internet Real-Time Lab at Columbia University, Internet Research Lab at UCLA, Postel Center at USC/UCLA and CUBINlab at University of Melbourne for providing resources for our experiments. The first author would also like to thank Prof. Chengyang Lu and Prof. William Murphy at Washington University in St. Louis for helpful discussion about the feedback control system design.

References

[1] S. Floyd and K. Fall, "Promoting the Use of End-to-end Congestion Control in the Internet," IEEE/ACM Transactions on Networking, vol. 7, no. 4, Aug. 1999, pp. 458-72.

[2] R. Rejaie, M. Handley, and D. Estrin, "RAP: An End-to-end Rate-based Congestion Control Mechanism for Realtime Streams in the Internet," Proc. IEEE INFOCOM, Mar. 1999.

[3] D. Sisalm and A. Wolisz, "LDA+ TCP-Friendly Adaptation: A Measurement and Comparison Study," Proc. NOSSDAV, Jun. 2000.

[4] I. Rhee, V. Ozdemir, and Y. Yi, "TEAR: TCP Emulation at Receivers - Flow Control for Multimedia Streaming," Technical report, Dept. of Computer Science, NCSU, Apr. 2000.

[5] D. Bansal and H. Balakrishnam, "Binomial Congestion Control Algorithms," Proc. IEEE INFOCOM, Apr. 2001.

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

[7] J. Mahdavi and S. Floyd, "TCP-Friendly Unicast Rate-based Flow Control", Technical Note to End2end-interest Mailing List, Jan. 1997.

[8] G. Franklin, J. Powell, and A. Emami-Naeini, "Feedback Control of Dynamic Systems", 3rd ed., Addison-Wesley Publishing Company, 1993.

[9] G. Franklin, J. Powell, and M. Workman, "Digital Control of Dynamic Systems", 3rd ed., Addison-Wesley Longman, Inc. 1997.

[10] J. Padhye, J. Kurose, D. Towdley, and R. Koodli, A Model Based TCP-Friendly Rate Control Protocol. Proc. of NOSSDAV, 1999.