2012-05-01

Measurement and Method for Receiver Buffer Sizing in Video Streaming

Sahel Mastoureshgh
Worcester Polytechnic Institute

Measurement and Method for Receiver Buffer Sizing in Video Streaming

by

Sahel Mastoureshgh

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

April 2012

APPROVED:

Professor Mark Claypool, Thesis Advisor
Professor Emanuel Agu, Reader

Abstract

Video streaming has become increasingly popular with commercial video streaming applications such as YouTube accounting for a large quantity of Internet traffic. While streaming video is sensitive to bandwidth jitter, a receiver buffer can ameliorate the effects of jitter by adjusting to the difference between the transmission rate and the playback rate. Unfortunately, there are few studies to determine the best size of the receiver buffer for TCP streaming. In this work, we investigate how the buffer size of video streaming applications changes with respect to variation in bandwidth. We model the video streaming system over TCP using simulation to develop our buffering algorithm. We propose using a dynamic client buffer size based on measured bandwidth variation to achieve fewer interruptions in video streaming playback. To evaluate our approach, we implement an application to run experiments comparing our algorithm with the buffer size of commercial video streaming.

Acknowledgments

I want to thank Professor Mark Claypool. His comments, suggestions and insights have helped me a lot in shaping this thesis.

Also thanks to Professor Emmanuel Agu for reading this thesis.

I would also like to thank everyone at PEDS groups and faculty of CS department.

Contents

List of Figures

List of Tables

Chapter 1. Introduction

Today, video streaming is one of the most popular services on the Internet. The increasing demand for video streaming has meant video constitutes a large portion of the total data traffic on the Internet [1].

Video data are usually encoded as frames to be displayed at fixed frequencies, for example, 1 Mbps displayed at 30 frames per second. As video data arrives at the client, data is placed into a buffer to be decoded and displayed on the screen at the right time.

Over the years a number of video streaming protocols have been developed. However, most commercial video streaming applications today use HTTP to send data to the receiver because firewalls will often only allow TCP traffic to pass through. In addition there is a great availability

of HTTP servers. In a typical streaming session, the client requests a video file from the Web server over HTTP and plays it as video data comes in from the server. An example is YouTube, which uses HTTP server for its progressive download [6].

Video streaming needs a steady data rate to have smooth playout. However, after packets are lost, the TCP congestion controller will suddenly decrease the transmission rate, causing bandwidth variation which can damage video quality.

A client buffer can be used to alleviate degradations caused by unwanted changes in data rate. Packets are temporarily stored at the client buffer in order to smooth out bandwidth variation. Figure 1.1 shows the role of the client buffer in video streaming. The fill rate is the rate data enters the client buffer and the drain rate is the rate the playout drains from the client buffer. As video data arrives at client side from the server, data put into the client buffer at the fill rate then pulled out at the drain rate and is decoded and displayed. With buffered data, the receiver is able to smooth over temporary drops in the received rate.

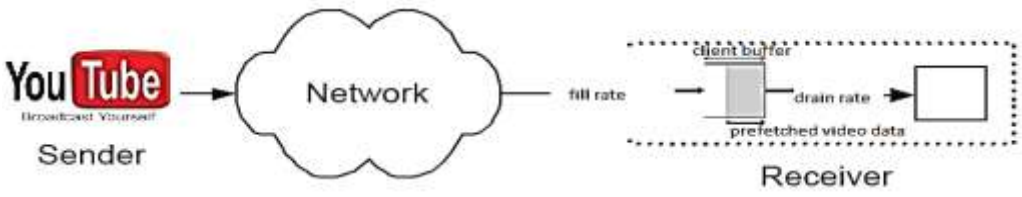

Figure1.1: video streaming system, showing video sender and video receiver, with client buffer

Choosing an appropriate size of buffer is important. If the buffer is too small, when TCP congestion control reduces the fill rate below the drain rate, it causes the buffer to empty and

results in unwanted pauses in the video playout. On the other hand, if the buffer is too large then users have to wait extra time when filling the buffer before watching the video.

Nukhet and Turhan [5] tried to tackle the problem of determining optimal receiver buffer size. Their experimental results suggest that when no bandwidth estimation is possible, a buffer with a size of 5 seconds should be chosen. Kim and Ammar [2] show that the receiver buffer requirement is determined by the network characteristics and the desired buffer underrun probability. Neither paper provides a precise algorithm to determine the right size for the client buffer. Previous work has devised some adaption algorithms [11]. A common property of these adaptation algorithms is configurable parameters to adjust the client buffer size in playout with minimal start time for streaming.

In this work, we investigate buffer sizing as a means of achieving smoother, higher quality streaming video. We describe a model of a video system that streams video data using TCP. We explain modeling a TCP friendly environment. We set different loss rates and round-trip times in Dummynet to produce variations in bandwidth but with the same average throughput. We build a video playout simulator to determine the number of interrupts and the minimum buffer size with no interrupts based on bandwidth derived from a network trace. Once filled, the buffer is drained at the encoding rate of the video and whenever empty, the number of interrupts is incremented. By increasing the buffer size and repeating the simulation, we determine the smallest buffer size in bytes when there are no interrupts, which we call the perfect buffer. We use a linear relationship between the coefficient of variation of the trace and perfect buffer size and predict the perfect buffer size from a sample of the whole trace. We study commercial client buffer

sizing by measurement experiments done for different videos from YouTube. We record start and stop times in seconds and used the video encoding rate and elapsed time to determine buffer sizes. We compare the measurement to the perfect and our predicted buffer sizes. Analysis shows that YouTube buffer sizing is independent of network variation.

Chapter 2 provides related work and background. Chapter 3 presents our methodology to gather data and setup a test environment to determine buffer size. Chapter 4 describes our methods to predict perfect buffer sizes. Chapter 5 presents an experimental study of YouTube buffer sizing and comparison to predicted and perfect buffer sizes. Chapter 6 presents the conclusions of our study and Chapter 7 provides some possible future work extensions.

Chapter 2. Related Work

This section discusses the background of the problem of client playout of video and streaming buffering systems and provides related work on the problem of determining the best buffer size.

Commercial video streaming has shifted away from custom protocols such as RTSP to HTTP/TCP because other protocols often have difficulty getting around firewalls. Outside of custom protocols for video streaming, there are few studies relating buffer sizing to video streaming.

An interrupt in video playout may occur if data is not delivered on time especially, when the transmission rate does not match the encoded rate. In practice, the system buffers a chunk of video data at the client before displaying the video, so that transient packet losses and delays do not constantly interrupt the playout of the stream. Intuitively, the more data buffered, the fewer interrupts in the future, but the startup delay induced by buffering increases, too. So, system designers must trade the reliability of uninterrupted playback against delay from the buffer size [12].

Systems that are designed mainly for synchronous delivery channels with low loss rates use transmitter-based control of the receiver buffers. On the transmitter side, the rate control algorithms usually manage the quantization scale parameters to avoid receiver buffer underflow

based on a specified receiver buffer size and fixed end-to-end delay. The clock synchronization is accomplished by transmitting special clock reference signals. However, this approach is not suitable in multi-cast scenarios [11].

A study of the buffer size for video streaming over HTTP was given by Nukhet and Turhan [5]. In their work, they collected over 1000 hours of video over LANs and WANs. Their experimental results suggest that when no bandwidth estimation is possible, a buffer of size 5 seconds should be chosen. Unlike their work, we are interested in finding out the best buffer size based on bandwidth variation.

According to Zambelli [6], in early versions of Windows Media Player and Silverlight the default buffer length was 5 seconds. Later, Microsoft used another form of delivery called progressive download over HTTP, where the video file is cut into many short segments and encoded to the desired delivery format. Chunks are typically a few seconds, with the client requesting the individual video segments from the Web server. Instead of the Microsoft streaming system, we are interested in measuring the YouTube client buffer size.

Akshabi, Begen and Dovrolis [1] did experimental evaluation to measure how Microsoft Smooth Streaming and Netflix players react to persistent and short term bandwidth variations. They used Wireshark to capture traffic and Dummynet to change available bandwidth. They find playback buffer size in Smooth Streaming decreases when the available bandwidth is less than the requested bitrate and increases when the available bandwidth increases. However, Netflix

employs a large playback buffer (up to few minutes) and sometimes changes to bitrates higher than the available bandwidth as long as the playback buffer is almost full. In our work, we set up experiments for YouTube and evaluate the reaction of YouTube to available bandwidth and video quality.

Goel, Krasic and Walpole [3] showed that a significant fraction of the latency at the application layer occurs at the sender side of TCP as a result of throughput-optimized TCP implementations. They developed an adaptive buffer-size for the sender that reduces this latency. They made a small modification to the TCP stack on the sender that can be enabled per socket. Their modification limits the send buffer size to fixed parameters, allowing a trade-off between latency and throughput. We are interested in making an adaptive receiver-side buffer algorithm, adjusting to variation in bandwidth.

Tan, Cui and Apostolopoulos [4] mentioned one way to reduce the effect of a buffer underflow when there are multiple streaming sessions is to redistribute resources. They put a label for each packet of a streaming session corresponding to the buffer occupancy on the client then use scheduling so that packets in a session with labels smaller than the buffer size transmit sooner than others. Ordering at the resource bottleneck is based on labels carried in the packets. In this way, streaming sessions with a small playback buffer receive higher throughput, therefore fewer pauses. We only consider application layer treatments in our study.

Chapter 3. Determining Perfect Buffer Size

In this chapter, we seek to determine the perfect buffer size for different network conditions. We setup an environment to gather traces in a controlled fashion, build a simulator to simulate video playout and determine buffer size. Section 3.1 provides an explanation of the system setup for gathering traces, Section 3.2 presents the pseudo code for the simulator and Section 3.3 gives an explanation of the main components in a video streaming system and analyzes correlation between measurement of variation and buffer size.

3.1 Gather Network Traces

Figure 3.1 depicts our setup for all experiments in our study. We use Dummynet [8] to control available bandwidth and Wireshark [9] to capture packets and interpret network data. The server sends data to a client.

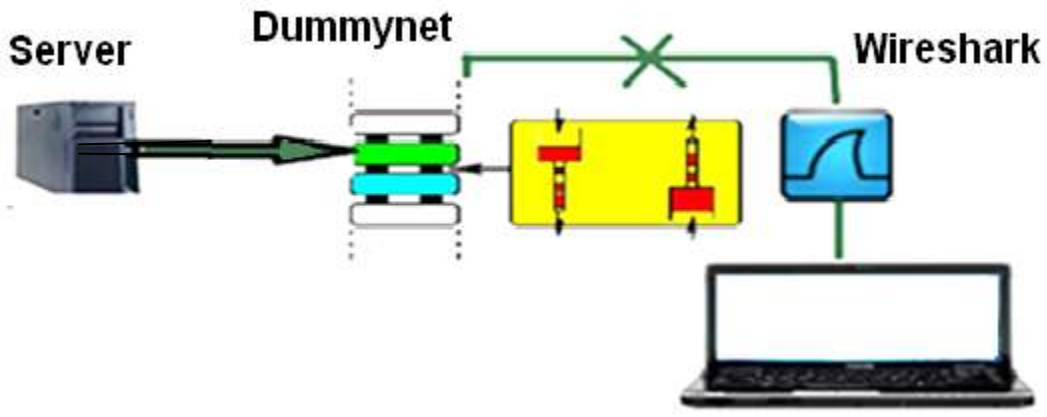

Figure 3.1: Experiment setup for streaming video with Dummynet and Wireshark

Dummynet is a tool for shaping network traffic. Dummynet works by intercepting packets as they pass through the protocol stack and passing them through one or more "pipes" which can emulate the effects of bandwidth limitation, propagation delay, bounded-size queues, packet loss, and more [8]. The example below shows how to set a 4Mbps connection sent from IP address 130.215.36.26 to IP address 208.117.209.204 with packet loss rate 0.02 and round-trip time 84 ms in Dummynet.

```
# ipfw add pipe 3 ip from 130.215.36.26 to 208.117.209.204
# ipfw pipe 3 config bw 4Mbit/s delay 84ms plr 0.02
# ipfw pipe show
```

Wireshark is a public-domain packet capture and protocol analysis tool [1]. Wireshark allows a user to capture live packets, apply filters, and display packet headers, and generate basic statistics on a collected trace. Wireshark has a straightforward graphical interface through which a user can capture live packets, apply filters to select packets with desired characteristics, and display packet headers in a convenient readable format.

To gather data, we model a TCP environment and create variation in network bandwidth. TCP-friendly flows respond to packet loss by reducing their rate of transmission. In order to have the same average bandwidth but with different bandwidth variance, we change the loss rate and round-trip time but keep the average throughput consistent based on the following formula [13]:

$$T = \frac{S}{R\sqrt{\frac{2p}{s}} + t_{RTO}\left(3\sqrt{\frac{3p}{s}}\right)p(1+32p^2)}$$

S is the packet size in bytes, R is the round-trip time in ms, p is the packet loss rate, and RTO is the TCP time out value, set to 4R in our setup.

Table 3 represents different loss rates and round-trip times used in our set up to keep the average throughput at about 125 kbytes /second.

Table 1: Configuration of different round-trip times and loss rates

Round Trip Time	Loss Rate	Throughput
23.4 ms	0.09	124032 bytes/sec
31 ms	0.07	124109 bytes/sec
43 ms	0.04	124148 bytes/sec
84 ms	0.02	124814 bytes/sec
131 ms	0.01	124194 bytes/sec
193 ms	0.004	124379 bytes/sec

We developed a client-server program in Java to send video data from a server to a client in this controlled environment, gathering bandwidth data at the client with Wireshark for 60 seconds. Figure 3.2 shows an example of the bandwidth over time for a TCP connection with round-trip time 31 ms and loss rate 0.07. The x-axis shows bandwidth in kbytes per second, and the y-axis represents time in seconds.

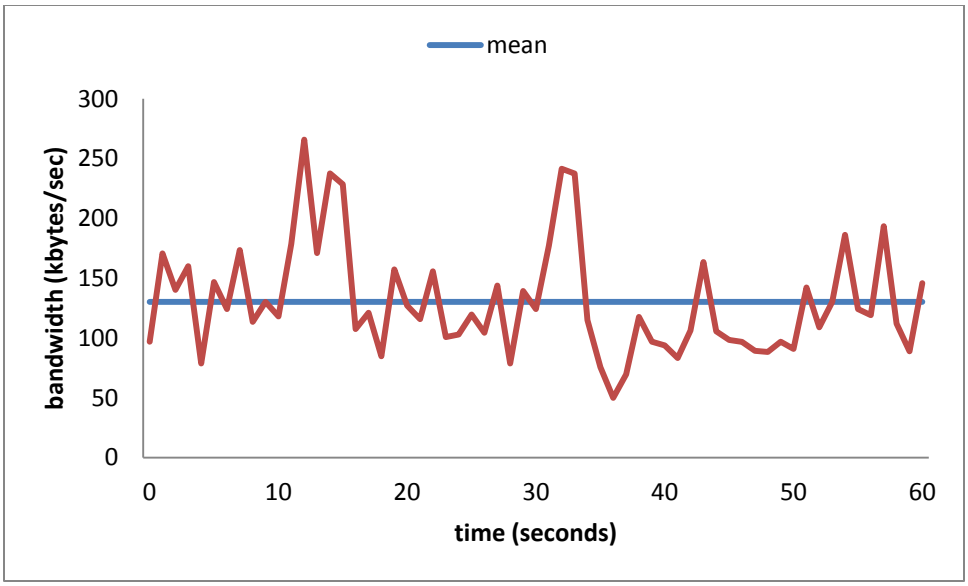

Figure 3.2: Sample Bandwidth trace

3.2 Simulate Video Playout

We built a simulator of the client video playout. In a video streaming system, there is a rate controller, client buffer and decoder. The rate controller determines the bit rate for the next video segment. Once the video is chosen, data is sent as fast as TCP will allow. We assume the rate controller always picks a data rate for the encoded video that matches the overall average bandwidth achieved by TCP.

We use the bandwidth trace in kbytes/sec, as one of the two inputs to our simulator. The other input is the encoding rate of video in kbytes/sec. All traces are assumed to be 60 seconds long. The simulator fills the initial buffer with incoming bandwidth trace until the incoming chunk from the trace is greater than the buffer that needs to be filled. Then, playing the video is simulated by subtracting the amount of the filled buffer from the encoding rate. When the buffer

is less than the encoding rate, there is an interrupt in the playing of the video. The interrupt count is incremented and the buffer is re-filled. This process repeats until the video ends. In each run, the simulator produces the number of interrupts for a given buffer size. Figure 3.3 gives the pseudo code for the simulator.

```
int countInterrupt (B[],b,E){
1. buffer=0;
   num_interrupt=0;
   i=0;
   pause=false;
2. while(buffer<=b){
         need=b-buffer;
         if(B[i])>need){
                buffer=buffer+need;
                B[i]=B[i]-need;
         }else{
                buffer=buffer+B[i];
                i++;
         }
   }
3. while(i<60){
         if(buffer>E){
                buffer=buffer-E;
                pause=false;
         }else{
                if(pause==false){
                        num_interrupt++;
                        pause=true;
                }
         }
         buffer=buffer+B[i];
         i++;
   }
4. return num_interrupt;
```

Figure 3.3: Pseudo code for video playout simulator

In Figure 3.3, B [] is an array of bandwidth in kbytes/sec, buffer is the amount of the buffer filled in kbytes, b is a buffer size in kbytes, and E is encoding rate in kbytes. Block 1 initializes the values for buffer, num_interrupt, and i to zero, and pause to false. Block 2 fills the initial buffer. In block 3, if the filled buffer is greater than the encoding rate, the simulator drains the buffer. Otherwise, it increments the number of interrupts and pauses the playout.

The simulator can determine when interrupts happen. In Figure 3.4, the y-axis is bandwidth trace in kbytes/sec, and the x-axis is time in seconds. The red squares are where interrupts happen when the buffer size is 130 kbytes.

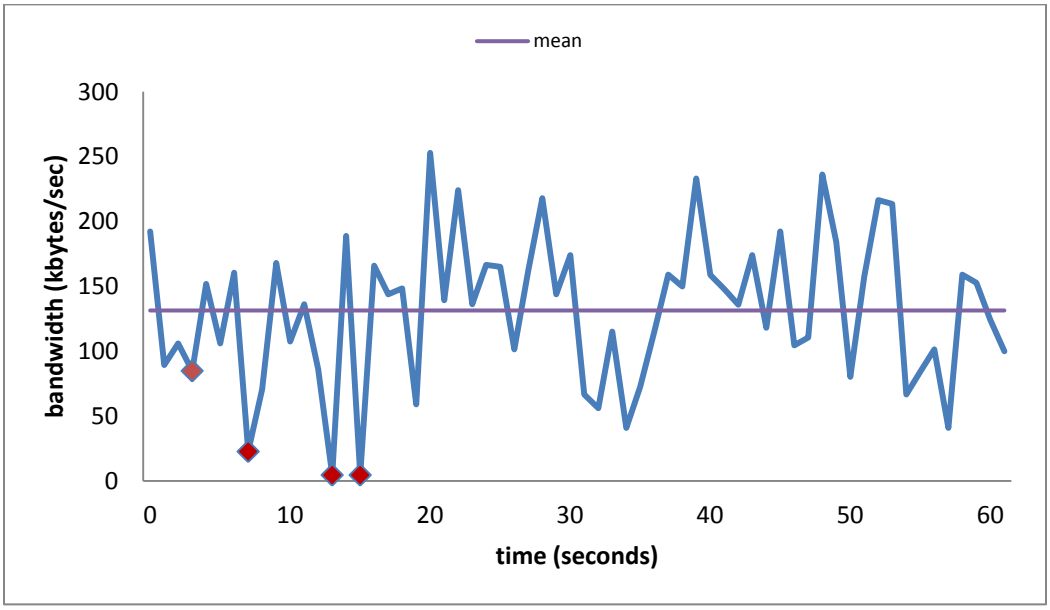

Figure 3.4: Bandwidth versus time, where red squares are interrupts in playout

The simulator allows analysis of the buffer occupancy, too. For the same trace, in Figure 3.5, the y-axis is buffer occupancy (buffer in the simulator code) in kbytes, and the x-axis is time in seconds. Whenever the buffer falls below the encoding rate, 130 kbytes/sec, there is an interrupt in playout.

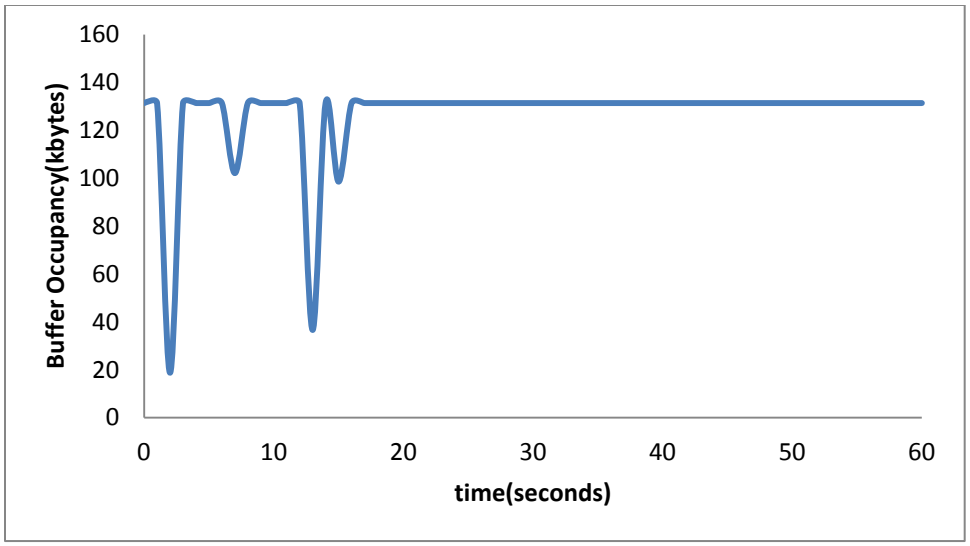

Figure 3.5: Buffer occupancy versus time

In order to find minimum client buffer size with no interrupts for a given bandwidth trace and encoding rate, the simulator is started with buffer size 1 byte and increased by 100 bytes in each run until there are no interrupts.

Figure 3.6 represents the relationship between the number of interrupts and buffer size for an encoding rate of 130 kbytes/sec for a trace with round-trip time 23 ms and loss rate 0.09. The x-axis is the buffer size in kbytes/sec and the y-axis is the number of pauses. As the buffer size increases, there are fewer pauses. At a 130 kbytes buffer, there are 4 interrupts for the trace in Figure 3.4. At 470 kbytes, there are zero interrupts for the same trace.

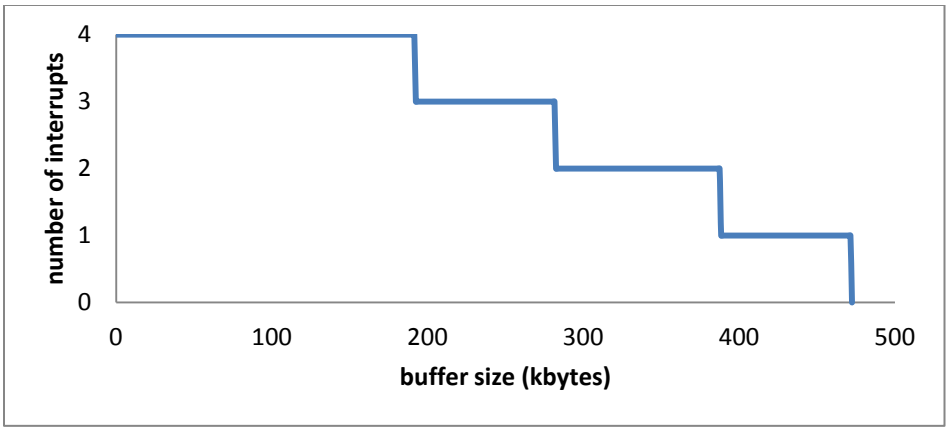

Figure 3.6: Number of interrupts versus buffer size

3.3 Perfect Buffer Size versus Variation in Network Bandwidth

The perfect buffer size is the minimum buffer size with no interrupts. While our simulator is able to determine the perfect buffer size if the entire bandwidth trace is known ahead of time, we seek a heuristic to predict buffer size on the fly based on network variance.

Figure 3.7 presents a graph of traces with the same average bandwidth but with different variance. The x-axis is time in seconds and y-axis is bandwidth trace in kbytes. Each color represents one adjustment to the trace which is a result of compressing or stretching the original trace around the mean. We created this graph to provide samples with a broader range of variance.

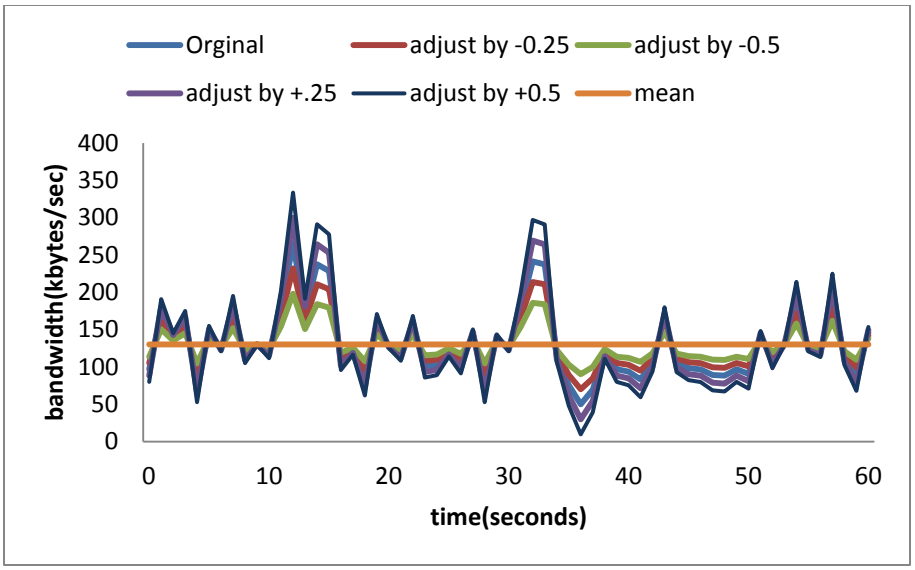

Figure 3.7: Traces represent different bandwidth variations for the same bandwidth average

We created 6 tests with different bandwidth variations based on Table 3 and ran each trace 5 times. Then, we scale each trace by factor of 0.25 and 0.50 of original data. In total, we have 150 traces.

Figure 3.8 represents the relationship between buffer size and coefficient of variation for the 5 traces in Figure 3.7. The buffer size increases as the coefficient of variation increases. The x-axis is coefficient of variation of bandwidth and the y-axis is buffer size in kbytes.

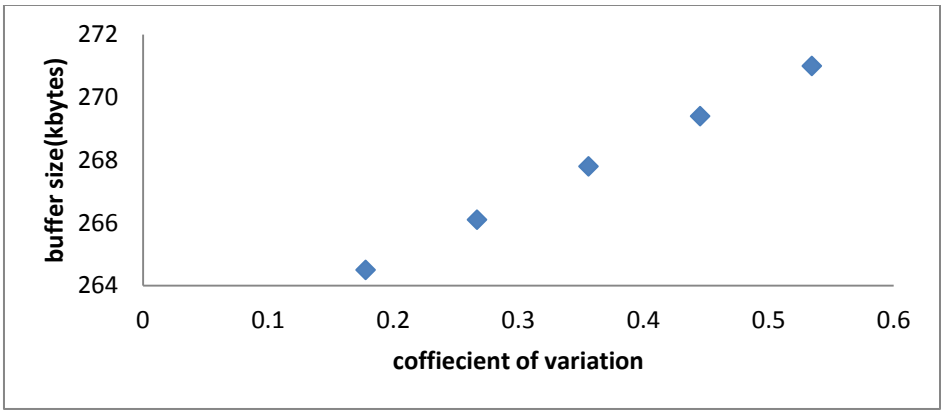

Figure 3.8: Buffer size versus coefficient of variation of bandwidth

For all 150 traces, we find out the perfect buffer size versus coefficient of variation, range, standard deviation, and inter quartile range (IQR). Figure 3.9, Figure 3.10, Figure 3.11, Figure 3.12, and Figure 3.13 represent the relation between bandwidth traces and perfect buffer size. In Figure 3.9, for each trace, we calculated the inter quartile range (IQR), being equal to the difference between the upper and lower quartiles. The x-axis is the IQR and y-axis is the perfect buffer size. Each point in Figure 3.9 represents one trace given to the simulator. Figure 3.10-13 are the same, except the x-axis is standard deviation, coefficient of variation, range, and 90-10 percent, respectively.

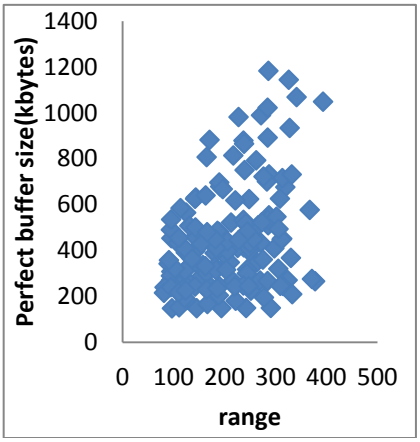

Figure 3.10: Perfect buffer size versus range

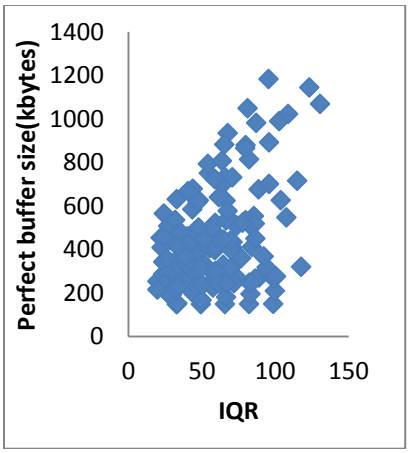

Figure 3.9: Perfect buffer size versus IQR

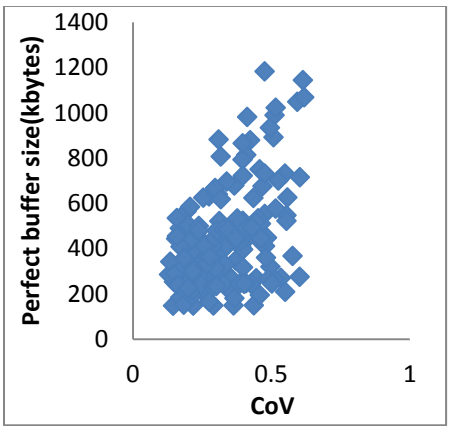

Figure 3.11: Perfect buffer size versus coefficient of variation

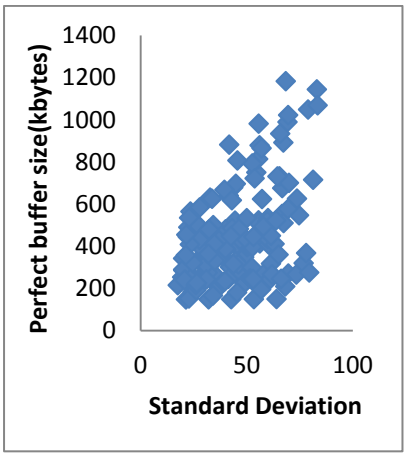

Figure 3.12: Perfect buffer size versus standard deviation

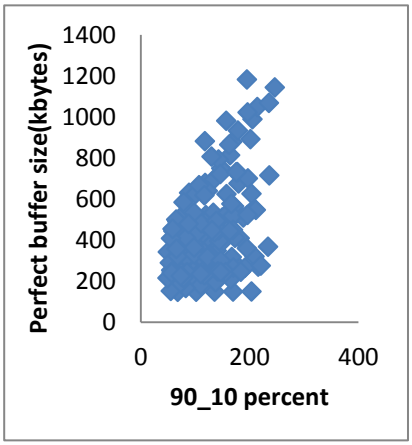

Figure 3.13: Perfect buffer size versus 90-10

Chapter 4. Predicting Perfect Buffer Size

This chapter describes our approach to predict the perfect buffer size based on coefficient of variation of bandwidth. Section 4.1 provides a formula to calculate the predicted buffer size and calculates the predicted buffer for all traces based on the formula. Section 4.2 compares the predicted buffer size to the perfect buffer size and analyzes the number of interrupts for all traces with our predicted buffer size.

4.1 Formula to Predict a Buffer Size

We determine a relationship between measured coefficient of variation of bandwidth and smallest buffer that has no interrupts. Based on the simulator results, use a linear relation between coefficient of variation of bandwidth and client buffer size, a line that goes above all the data points, but as low as possible so as not to have any interrupts. The line minimizes error while still having no interrupts. In Figure 4.1, this line is shown in red. x is the coefficient of variation of bandwidth and y is the predicted buffer size. The formula (1) is:

$$\text{Predicted buffer} = 1.8 \text{ Mbytes} \times \text{CoV} + 0.32 \text{ Mbytes} \quad (1)$$

A client can use the formula to predict the buffer size needed based on the measured bandwidth variation, then buffer that much data, before starting to play a video.

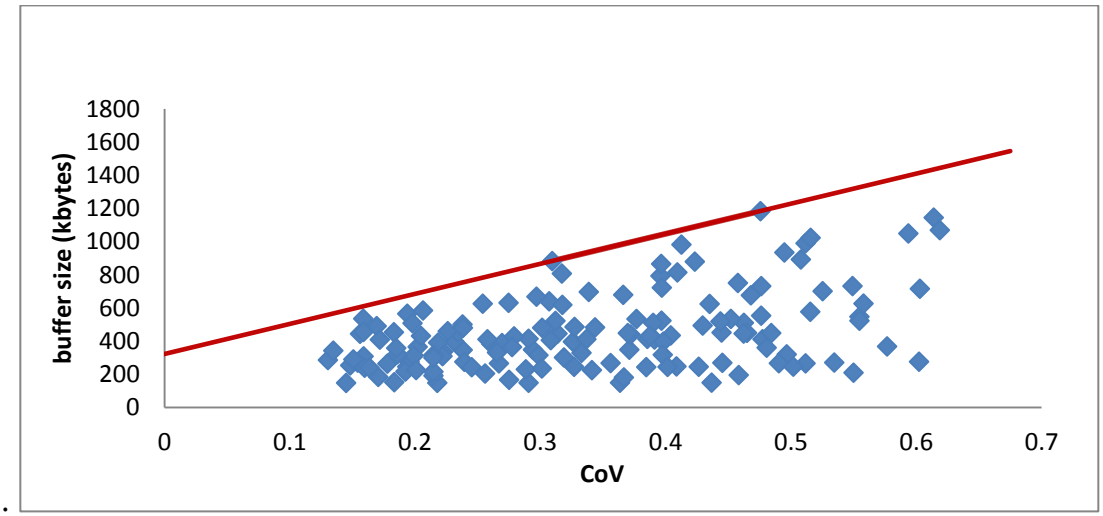

Figure 4.1: Linear formula for predicted buffer size in red

We use formula (1) to predict the buffer size needed for 60 seconds of a video. We calculate the coefficient of variation of bandwidth using the first Δt seconds of a trace to predict the needed buffer size. Figure 4.2 shows the predicted buffer size for all 150 traces for Δt seconds. The x-axis is Δt seconds, where Δt is between 2 and 60 seconds of video and the y-axis is the predicted buffer size.

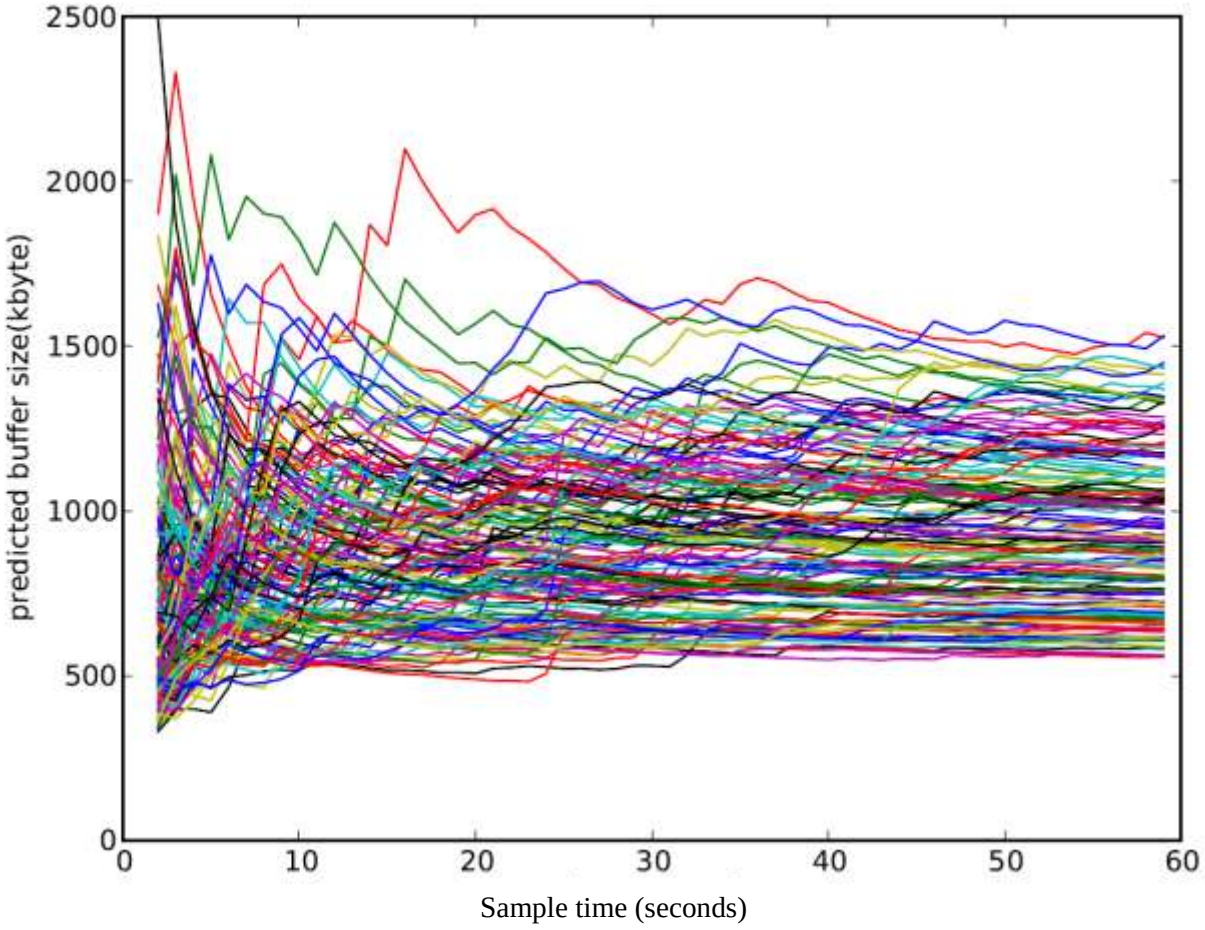

Figure 4.2: Predicted buffer versus time for different network traces

4.2 Predicted Buffer Size Compared to Perfect Buffer Size

We also compare the predicted buffer size for Δt seconds of the video trace with the perfect buffer size. Figure 4.3 represents this difference for all 150 traces. As Δt approaches 7 seconds, the predicted buffer is generally above the perfect buffer size. As we can see in the Figure, as Δt increases, the difference between predicted and perfect buffer size reaches zero for all traces.

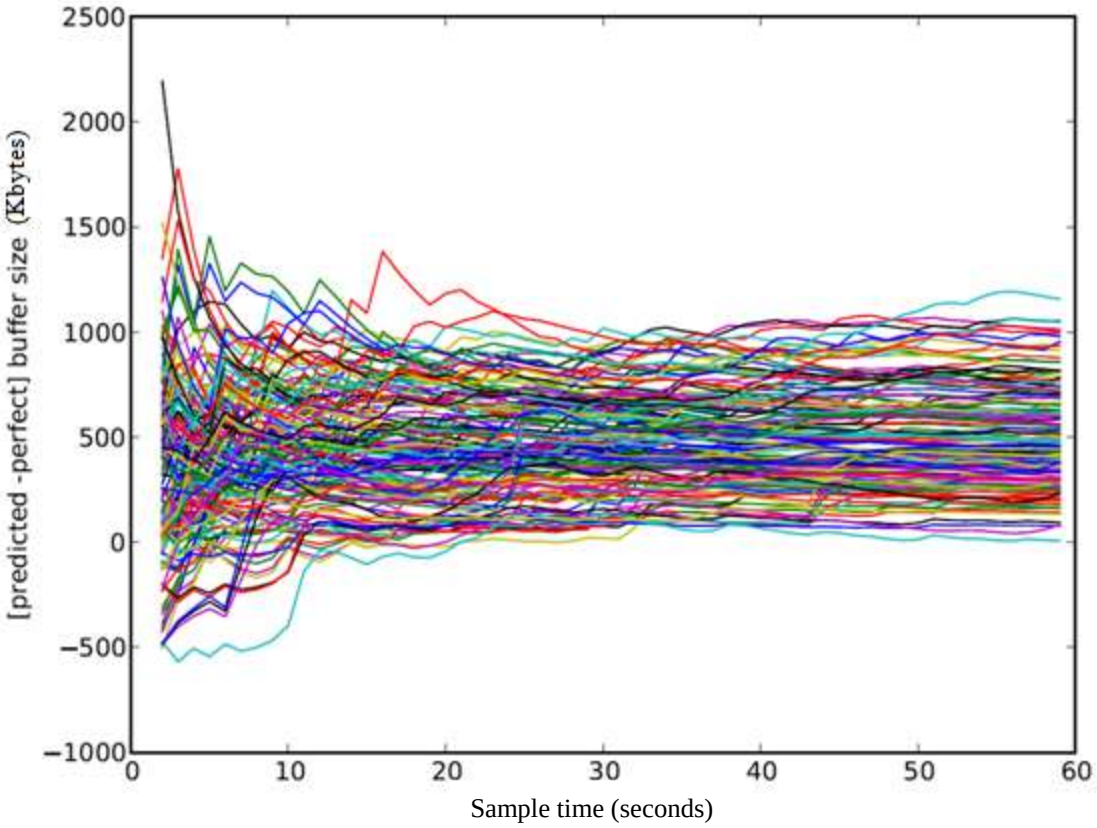

Figure 4.3: Difference between predicted buffer size and perfect buffer size

Figure 4.4 summarizes how the predicted buffer size compares to the perfect buffer size. The blue line is the average of all traces for the predicted buffer minus the perfect buffer. The red line is the average plus the standard deviation and the green line is average minus the standard deviation for all traces. By increasing the sample interval, we have a smaller difference between the predicted and perfect buffer sizes. As we see, at around 5 seconds sample size the line becomes mostly flat.

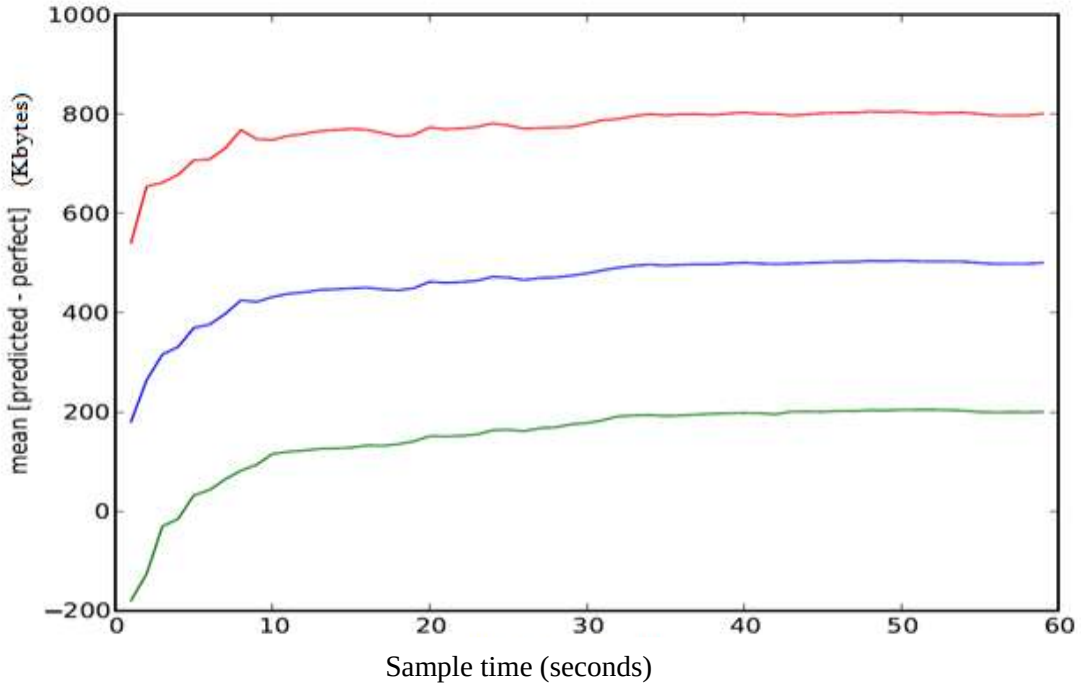

Figure 4.4: Difference between predicted and perfect buffer size

We modify our simulator to use Δt seconds of the bandwidth trace to compute number of interrupts for the predicted buffer size. We measure the predicted buffer and number of interrupts for all traces. Table 4 shows the average, medium, and maximum number of interrupts for all traces for samples up to 8 seconds. After 8 seconds, the average of interrupts goes to zero for all traces by using our predicted buffer size formula.

Table 2: Statistics results for interrupts number

time	median	max	mean	Percentage with no interrupts
2	0	19	2.84	59%
3	0	19	2.46	73%
4	0	17	1.19	88%
5	0	13	0.74	91%
6	0	6	0.14	93%
7	0	2	0.075	96%
8	0	2	0.054	97%
9	0	2	0.054	97%
10	0	2	0.054	97%
11	0	1	0.040	97%
12	0	1	0.040	97%
13	0	1	0.040	97%
14	0	1	0.040	97%
15	0	1	0.040	97%
16	0	1	0.040	97%
17	0	1	0.040	97%
18	0	1	0.020	98%
19	0	1	0.020	98%
20	0	1	0.020	98%
21	0	1	0.020	98%
22	0	1	0.020	98%
23	0	1	0.020	98%
24	0	1	0.020	99%
25	0	1	0.010	99%
26	0	1	0.010	99%
27	0	1	0.010	99%
28	0	1	0.010	99%
29	0	1	0.010	99%
30	0	1	0.010	99%
31	0	1	0.010	99%
32	0	1	0.010	99%
33	0	1	0.010	99%
34	0	0	0.0	100%

The CDF of this data is shown in Figure 4.5. The x-axis is the number of interrupts in each second and the y-axis is the cumulative distribution. The graph shows that more than 96 percent of traces after 8 seconds have zero interrupts.

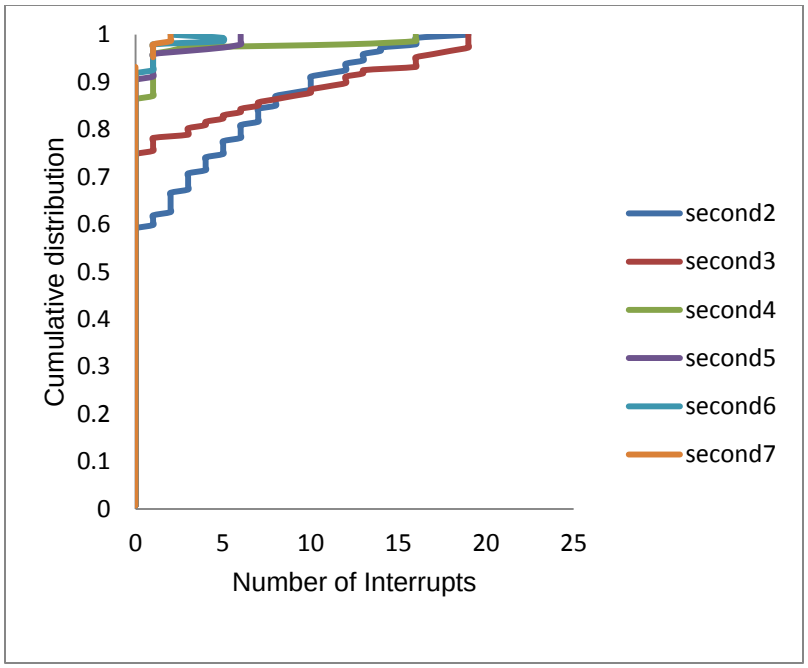

Figure 4.5: CDF of number of interrupts for predicted buffer size

Figure 4.6 represents the average number of interrupts for Δt seconds of the video trace.

Our predicted buffer method can estimate a perfect buffer size after 8 seconds of video for the average video trace.

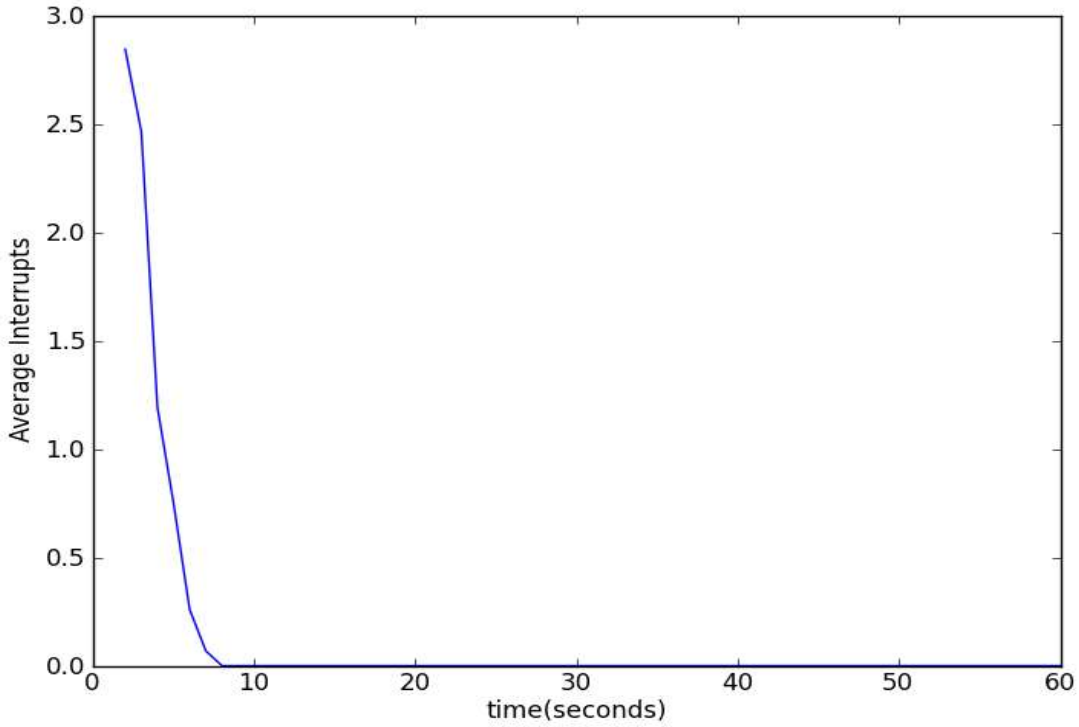

Figure 4.6: Average number of interrupts for predicted buffer size

A graph of the percentage of traces that have an interrupt versus sample interval is shown in Figure 4.7. The y-axis is the percentage of interrupts in each second and the x-axis is time in seconds. The graph shows that more than 96 percent of traces after 8 seconds have zero interrupts.

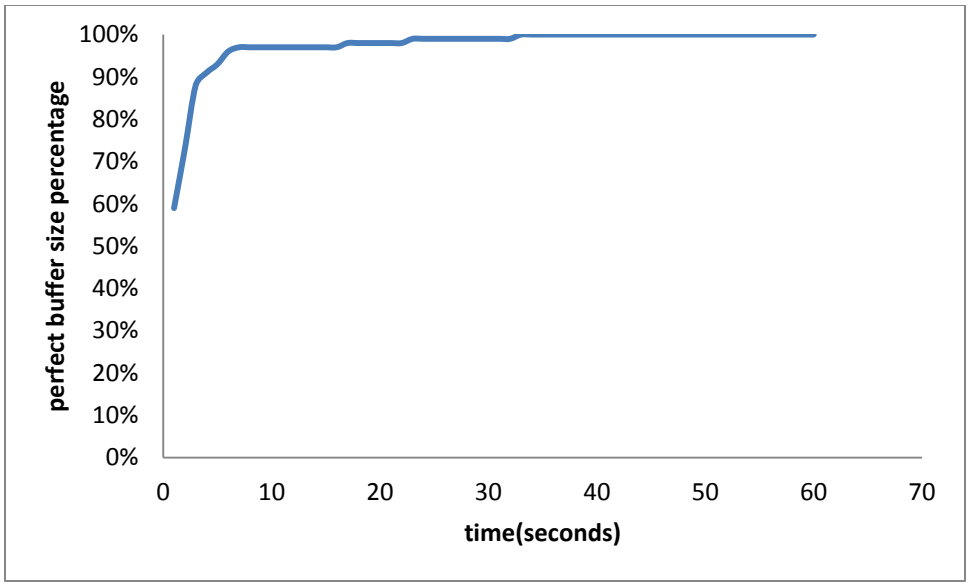

Figure 4.7: percentage of buffer with no interrupts for predicted buffer size versus time

Chapter 5. Commercial Video Streaming

This chapter describes our approach to measure the buffer sizing of YouTube as a representative of commercial video streaming.

We select a video[1] from YouTube with encoding rate with 1 Mbit/sec. We created variation in the bandwidth by changing the loss rate and round-trip time in Dummynet as in Figure 3.1. Then, we played the video while capturing network traffic with Wireshark. We recorded the start and stop time for each interrupt. We computed the amount buffered in bytes by $\int_{t1}^{t2} f(t)$ where f(t) is packet size at time t, time t1 is when we see the video stop and t2 is when we see the video start. We verified our data by finding the sum of the captured packets from the YouTube server and comparing this to the actual size of the downloaded video file stored on the disk.

We made 5 tests (5 runs each) from 60 seconds of the video with loss rates and round-trip times and number of interrupts shown in Table 5.

---

1 http://www.youtube.com/watch?v=xPJS8TJBSZc&feature=player_embedded

Table 3: Loss rate and round-trip time for YouTube

Test	RTT	Loss Rate	Number Interrupts
1	64 ms	0.03	10
2	43 ms	0.05	12
3	85 ms	0.02	13
4	131 ms	0.01	12
5	448 ms	0.001	14

We calculate the amount of buffering for each test. In Figure 5.1, the x-axis is the coefficient of variation of bandwidth and the y-axis is buffer size in kilobytes for each interrupt.

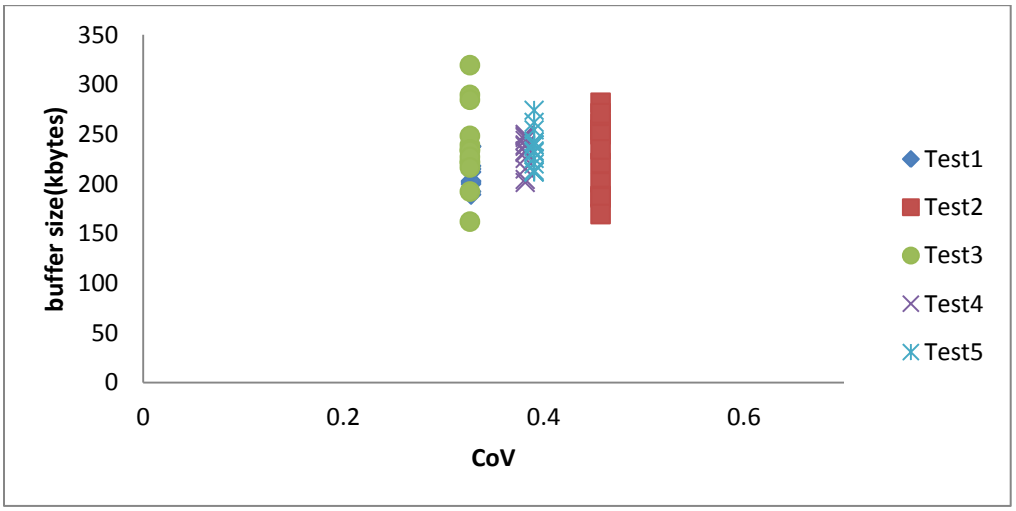

Figure 5.1: YouTube client buffer size versus network variance

We run our simulator with the bandwidth traces gathered from YouTube. We determine the perfect buffer size based on the simulator output then predict the perfect buffer size by calculating the CoV of the trace and put it in Formula (1). In Figure 5.2, the x-axis is the test number and the y-axis is the buffer size in kilobytes. We compare the predicted and perfect

buffers with the YouTube buffer. We can see that the average YouTube buffer remains almost the same for each test independent of variations in bandwidth, unlike the predicted and perfect buffer. The predicted buffer is larger than the perfect buffer size, it guarantees no interrupts.

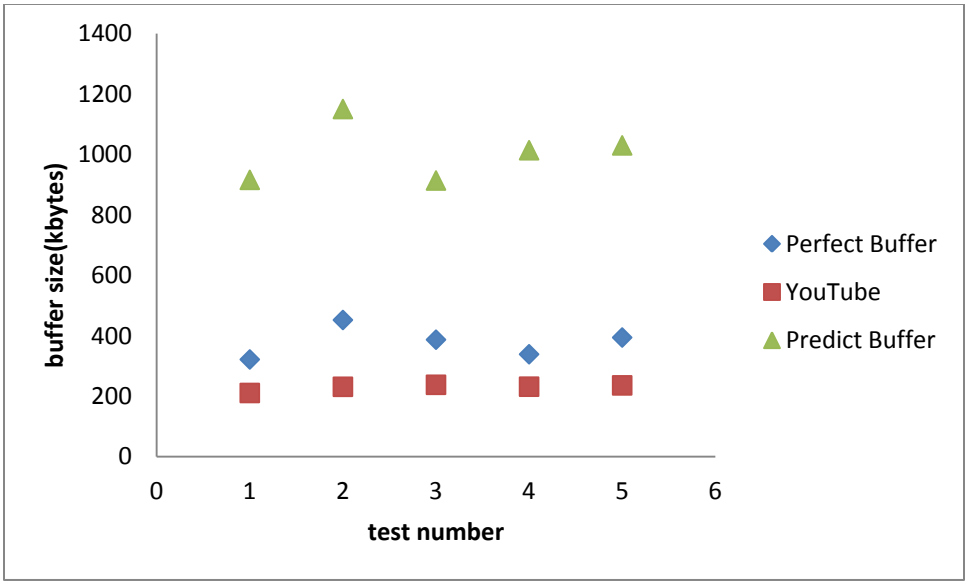

Figure 5.2: YouTube buffer to predicted and perfect buffer

Figure 5.3 presents a graph comparing YouTube to predicted and perfect buffer sizes with different coefficients of variation in bandwidth. The x-axis is the coefficient of variation in the bandwidth trace and y-axis is the buffer size. Our predicted buffer size increases with an increasing coefficient of variation in the y-axis. However, YouTube does not increase its buffer size when we have more variance in the bandwidth.

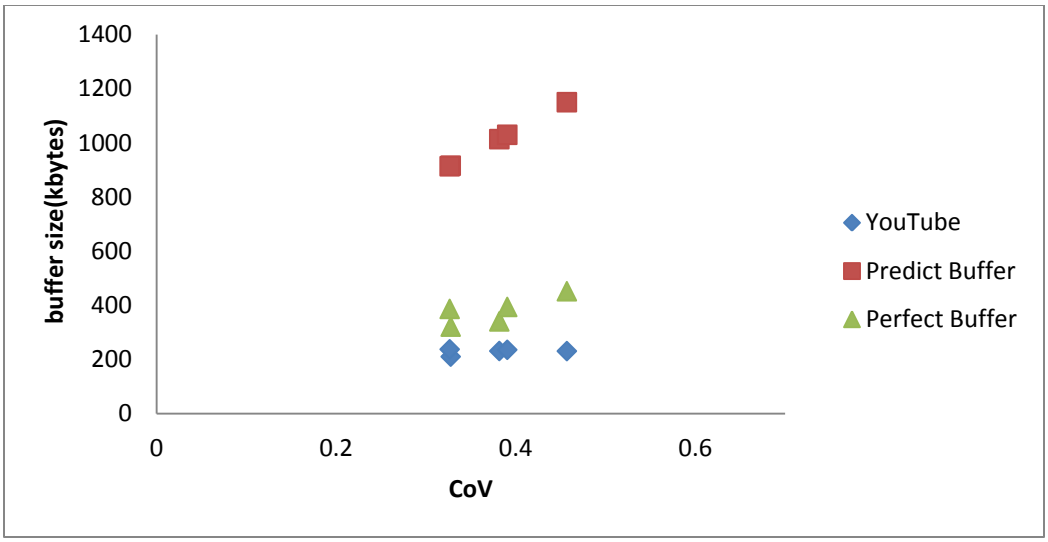

Figure 5.3: YouTube predicted and perfect buffer sizes versus network variance

Chapter 6. Conclusion

The importance of studying buffer size is to improve video smoothness. Our experimental evaluation shows buffer size should change based on variation in network bandwidth.

We built simulator to determine the perfect buffer size by increasing buffer size till there are zero interrupts. Based on simulation with 150 traces, we propose that the buffer size should increase linearly with the coefficient of variation in bandwidth.

Using our formula for a perfect buffer, we analyze the sample interval to find out how accurate predictions can be for small time intervals. We note that after 8 seconds, the average interrupts go to zero using our predicted buffer. Therefore by calculating 8 seconds of the coefficient of variation of the bandwidth trace we can predict the buffer size which has no interrupts for the rest of video. We suggest client buffer size should change with bandwidth variation in order to prevent interrupts.

We made experiments to measure how YouTube buffer reacts to conditions with different variation in bandwidth. We find YouTube buffer sizing is independent of network variance.

Chapter 7. Future Work

This chapter presents possible future work as an extension to our study. As mentioned in the introduction, there are few studies measuring buffer size. More trace gathering can help elaborate on the results. We can gather more traces with different encoding rates, different content and different resolutions for video. We can use the same analysis but with different network bottlenecks.

Our study does not include other commercial video streaming systems, such as Microsoft Silverlight, Netflix, and Apple video streaming. So, finding effective methods to gather more information about buffering for other commercial video streaming systems can be interesting for extending of this work.

Implement our predicted buffer heuristics in a streaming video system and evaluate by comparing to commercial video streaming such as YouTube. Also, our simulator can be implemented in the client to measure variation in bandwidth and calculate the predicted buffer size.

References

1. S.Akshabi, A.Begen , and C.Dovrolis, “An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP ”, in Proceedings of the second annual ACM conference on Multimedia systems, ACM New York, NY, USA 2011.

2. T. Kim and M. H. Ammar, “Receiver Buffer Requirement for Video Streaming Over TCP,” in Proceedings of Visual Communications and Image Processing, San Jose, CA, USA 2006.

3. A.Goel, C.Krasic, and J.Walpole, “Low-Latency Adaptive Streaming over TCP”, ACM Transactions on Multimedia Computing, Communications and Applications (TOMCCAP), Volume 4, Number 3, August 2008.

4.W.Tan, W.Cui, and J.G.Apostolopoulos, “An Experimental Evaluation of Rate-Adaptation Algorithms in Adaptive Streaming over HTTP”, Hewlett-Packard Laboratories, Palo Alto, CA, USA 2009.

5. O.Nukhet and T.Turhan, “On optimal receiver buffer size in adaptive Internet video streaming”, Journal of Zhejiang University; ISSN 1862-1775 (Online) , Izmir, Turkey, February 2006

6. A.Zambelli, “IIS Smooth Streaming Technical Overview”. Microsoft Corporation, Technical report, 2009.

7. Floyd and Fall, “Promoting the Use of End-to-End Congestion Control in the Internet”, Journal IEEE/ACM Transactions on Networking (TON) Volume 7, Issue 4, August 1999

8. L. Rizzo, “Dummynet: a Simple Approach to the Evaluation of Network Protocols”. SIGCOMM CCR, 27(1):31–41, 1997.

9. Wireshark: The World’s Most Popular Network Protocol Analyzer http://www.wireshark.org/

10. M.Sun and A.Reibman, “Compressed Video Over Networks” , ISBN:0-8247-9423-0, 2001.

11.W.Feng, “Buffering Techniques For Delivery Of Compressed Video In Video-On-Demand Systems”, 1997.

12. G. Liang and B. Liang, “Balancing interruption frequency and buffering penalties in vbr video streaming,” in Proc. of IEEE INFOCOM, May 2007.

13. J.Padhye, J.Widmer, “TCP Friendly Rate Control (TFRC): Protocol Specification”, University College London, September 2008. http://www.ietf.org/rfc/rfc5348.txt