Measurement of Windows Streaming Media

by

James G. Nichols III

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

by

____

April 22, 2004

APPROVED:

____

Dr. Mark Claypool, Major Advisor

____

Dr. Robert Kinicki, Advisor

____

Dr. David Finkel, Reader

____

Dr. Michael Gennert, Head of Department

Abstract

The growth of high speed Internet connections has fueled an increase in the demand for high quality streaming video. In order to satisfy timing constraints, streaming video typically uses UDP as the default network transport protocol. Unfortunately, UDP does not have any end-to-end congestion control mechanisms, and so in the absence of higher layer congestion control can lead to unfairness and possibly congestion collapse. While there has been research done in video measurement and characterization using custom tools, to the best of our knowledge, there have been no measurement studies where the researchers had control over a commercial streaming media server and client, and control of the network conditions and content. A goal of this research is to characterize the bitrate response of Windows Streaming Media in response to network-level metrics such as capacity, loss rate, and round-trip time. We build a streaming media test bed that allows us to systematically vary network and content encoding characteristics. We analyze responsiveness by comparing streaming media flows to TCP-friendly flows under various streaming configurations and network conditions. We find Windows Streaming Media has a prominent buffering phase in which it sends data at a bitrate significantly higher than the steady-state rate. Overall, Windows Streaming Media is responsive to available capacity, but is often unfair to TCP. Knowledge of streaming media's response to congestion encountered in the network is important in building networks that better accommodate their turbulence. The additional characteristics we measure can be combined to guide emulation or simulation configurations and network traffic generators for use in further research.

Contents

1 Introduction

People are increasingly connecting to the Internet with larger capacity connections. Recent market studies show that there was a 36% increase in the number of DSL lines worldwide in the first half of 2002, and a 12% increase in the number of cable Internet subscriptions in the United States [Ltd02]. Users of these high speed connections demand high quality content such as streaming video. A well known marketing firm forecasted that worldwide streaming media market revenues will increase over $2 billion dollars by 2006 [IS02]. Coupled with the increase in availability and demand for high quality content is an increase in the impact of streaming media content on the current Internet. Particularly, research we conducted [LCKN03] showed that the resolutions of clips currently available on the Internet, and their corresponding bitrates, are currently small compared to the available pixel space on typical desktop monitors and the available capacity of inexpensive, ubiquitous, readily available high-speed broadband Internet connections. Thus, we can expect the quality, and hence the network capacity requirements of streaming media to increase in the future.

Unlike traditional network applications such as file transfer or web browsing, streaming media has specific bitrate requirements. To meet these requirements, typically streaming media uses UDP as the default network transport protocol. Unfortunately, UDP does not have any end-to-end congestion control mechanisms and thus is unresponsive to congestion without application-level controlled responsiveness. Flows going through the network which are unresponsive to congestion can cause several undesirable situations from both an end-user and network level perspective. From the former perspective, unresponsive flows can create an unfair condition when competing with responsive flows for limited capacity. The responsive flows detect congestion in the network and reduce their sending rates accordingly, allowing the unresponsive flows to use the extra capacity. This is relevant to an end-user because this unresponsive behavior, even if by only for a few flows, results in delays and increased latency due to packet loss, queuing delays, and possibly TCP or related timeouts. This impact is significant on shared Internet connections either on a household or corporate level.

At the wide area network and Internet level, flows that are unresponsive to congestion can contribute to congestion collapse of the network. Congestion collapse could occur when available capacity on the network is utilized to send packets through the network which are subsequently dropped due to congestion [FF99]. However, some streaming media applications, such as Windows Media Player use the UDP protocol but rely on the application layer to provide adaptiveness to available capacity. The performance of these application layers with respect to congestion responsiveness is unknown.

Commercial media players make up a large part of the video traffic found on the Internet. There are several different commercial products on the market for streaming video, including Microsoft's Windows Media, RealNetwork's RealPlayer, and Apple's QuickTime. Of these, market research in [IS02] predicts that Microsoft Windows Media Technologies will be the dominant streaming solution. We have completed research in [LCKN03] that characterized the content stored on the Internet in terms of metrics that hold relevance to network performance. An additional result of the work was the determination of which multimedia codec the content producer used to create content clips that we found. These results showed that RealNetwork's offerings are currently the dominate content production technology in terms of total clips produced, with Microsoft close behind, while QuickTime has a much smaller share. This knowledge and the fact that RealVideo has previously been studied by our colleagues motivated us to study the Windows Media technology in detail.

Some research has been done in the general area of streaming video, including characterization studies [VAM$^+$02, dMSK02] performed through log analysis and empirical studies using custom tools [CCZ03, WCZ01, LCK02]. Recently, we have completed research to characterize the streaming content available on the Web relevant to network performance: bitrate, duration and resolution [LCKN03].

In the course of performing several of these previous studies, the researchers did not have control of the server streaming the media. One impact this had on their results was that a limited data set of streaming media content was examined. This limitation was because they could only stream and measure content that could be found readily available on public streaming video servers on the Internet. For example, in [LCK02] IP packet fragmentation was measured during the streaming of Windows Media clips. However, the size of the packets that the server sends to the client is dependent on the encoded content and can be tuned as a parameter in the newest version of the server management software. This is just one such example of the consequences that researchers face when relying on public access content and servers. Furthermore, research in [CCZ03] was weakened by the fact that the traffic was flowing through the public network and the researchers had no way to determine the network conditions between them and the server.

Intelligent Streaming [Bir00] is the application layer mechanism used by Windows Streaming Media (WSM) to adapt to network conditions. If it is determined that the network cannot support the current bitrate, Intelligent Streaming can "thin" the stream bitrate by sending fewer video frames. If the content provider has encoded multiple bitrates into the stream, Intelligent Streaming can chose an appropriate one. [CCZ03] studied the application layer mechanism of RealPlayer called SureStream. The results of [CCZ03] suggest that technologies like Intelligent Streaming may provide responsiveness to congestion, perhaps even TCP-friendliness. However, to the best of our knowledge, there have been no published results on the performance of Intelligent Streaming.

To the best of our knowledge, no measurement studies have been completed where the researchers had control of the commercial streaming media server, content encoding parameters, and network conditions between client and server. Having the streaming media server allows control of a large number of variables including the content being streamed, compression/encoding settings, and network conditions at or close to the server.

A primary goal of this research was to characterize the bitrate response function of streaming media in response to congestion encountered in the network. In other words, we seek to examine how WSM responds in terms of bitrate when congestion is detected in the network. Again, it is important to gain understanding of WSM's response to congestion because flows which are unresponsive to congestion create unfair situations when competing with responsive flows and can lead to congestion collapse of the network. This importance is closely related to the facts that the overall use, quality, and resulting bitrates, resolutions and frame rates of streaming media continue to increase, while knowledge of its responsiveness to congestion is unknown.

To achieve our goals, we varied the network conditions and various server-side content encoding parameters including bitrates of the content as well the number of bitrates contained in the content. We measure the responsiveness under controlled capacity constraints, network packet-loss rates, and content selection. Finally, we measured other characteristics of WSM that are relevant to network performance, including packet interrarrival times and burst length distributions.

We find that WSM has a prominent buffering period and burst packet rate, meaning it cannot be modeled in simulations by simple CBR flows as is typically done by many network researchers. In terms of responsiveness, we find that WSM does respond to available capacity in many cases. When WSM does not respond to available capactiy the impact on the network is severe and high loss rates are induced. If multiple bitrates are included in the clip, WSM responds by choosing an appropriate bitrate if one is available. Otherwise, it still tries to stream at the lowest encoded rate in the clip, again resulting in high amounts of loss. For both single and multiple bitrate clips, we find that the WSM is responsive but may recieve an unfair share of capacity compared with TCP flows. Overall our work adds to the body of knowledge in the networking research area and provides a start for future research that uses emulation and simulation.

2 Background and Related Work

Analyzing the performance of streaming multimedia from several research goal-oriented perspectives is motivated by work done in the broad area of networking research. Some of this work is primarily concerned with the incorporation or modification of congestion control algorithms for networking protocols currently in use or in research and development. For completeness we will cover a seminal paper in congestion control, [FF99] and work done in [FK03], which discusses the role of modeling in network and Internet research and some of the current challenges. While our research involves no modeling, [FK03] is an example of a research area that would benefit from careful measurement and analysis of streaming multimedia.

Besides being motivated by past academic research done in the general areas of networking and congestion control, this work is motivated by a large body of market research and analysis. This market research, while commercially motivated in nature, highlights the necessity for better understanding of the behavior of commercial streaming multimedia products, particularly in response to congestion. This necessity for better understanding is in part because the pervasive use of streaming media continues to increase. This increase is driven by a rise in the number of broadband Internet connections available throughout the world, coupled with an increase in the quality and bitrate requirements of the streaming content presentation itself. We have completed research which shows that there is much room for improvement in the resolutions and corresponding bitrates of content in proportion to the available pixel space on typical desktop monitors and available capacity of broadband Internet connections.

There exists a moderately sized body of research on streaming multimedia. To understand the rationale of our approach to evaluation of streaming media there is some background material to cover. Since the main focus of our research is on the Microsoft streaming product we discuss the material from that perspective.

First, we discuss the different network transfer protocols that Windows Streaming Media uses at the network and application layers. Next, we analyze in detail Intelligent Streaming, the mechanism Windows Media uses to adjust the parameters of the stream. Then, we discuss multiple bitrate content, how it effects Intelligent Streaming as well as impacts network performance. We also present the related work on streaming media, including past streaming media empirical measurements, evaluations, and experimental research publications. We conclude the section with coverage of other types of network measurement, including key publications, to provide completeness to our work.

2.1 Motivation

When users on an educational network noticed that there was very limited available capacity on their network links, they determined that the cause was the onset of congestion collapse of the network [APS99]. To prevent this from happening they, along with other networking researchers, began the development of mechanisms to control the response of the protocols to limited resources or congestion encountered in the network. Primarily, this included the incorporation of slow start, congestion avoidance, and additive-increase multiplicative-decrease behavior to the de-facto Internet protocol TCP.

Since that time an entire area of Computer Science has developed concerning the study and understanding of congestion control mechanisms and algorithms both in theory and in practice on the Internet. At first, researchers modified TCP to respond to congestion encountered in the network, communicated to the sources from the network via packet drops. Later, entirely new protocols were proposed, along with improved ways to communicate congestion information; perhaps marking a special bit, known as Explicit Congestion Notification (ECN) [Flo94], or the addition of special headers. Much of the work done in this area is motivated by some seminal research, particularly [FF99], which attempts to motivate protocol designers and implementors to include the use of end-to-end congestion control as a means to avoid congestion collapse in the Internet.

The primary research contribution of [FF99] is to show that when flows that implement congestion control compete with flows that do not, an unfair situation, in terms of bitrate use is created. Futhermore, they show that if enough flows are unresponsive to congestion that there can be total congestion collapse of the network. This collapse results because the small amount of available capacity in the network is wasted by sending and forwarding packets that are later dropped at a congested router due to capacity or resource limitations. [FF99] argues that any new protocols or products that are developed to use the Internet or other network should implement some form of congestion control at the network, transport, or application level of abstraction.

Fundamentally, there is this issue of unresponsiveness to congestion. In [FF99], Floyd uses the term flow as an abstraction for an end-to-end connection through the network. The distinction of unresponsive flow is to differentiate the flows that do not use end-to-end congestion control. That is unresponsive flows do not reduce the rate at which they put data into the network in response to packet drops, or other indicators of congestion. A subtle but important point presented in the work is that congestion collapse cannot be prevented by active queue management techniques at the router, or any of its parameters, but can be caused by any flow which is unresponsive when packets are lost in the network.

There are several network properties of streaming multimedia that are relevant to network stability and avoidance of congestion collapse. For example, another form of congestion collapse is fragmentation-based collapse. This danger was first described in [KM87] and later in relation to TCP traffic running over ATM networks in [RF95]. In networking terminology, a fragment is an abstraction that results when a data packet is split into several smaller pieces. This can be due to hardware limitations or software settings. If the network drops a particular fragment, or piece of a packet that was broken up because it was too big at some point in its transmission through the network, all of the other fragments making up that packet are useless to the receiver. Since the sender is not the host in the network fragmenting the packet specific pieces cannot be retransmitted, instead, the receiver does not acknowledge successful transmission. In the case of TCP, the packet would be resent, while in the case of UDP, the packet wouldn't be resent by the transport layer. However, the application layer could report the packet loss and manually request a retransmission of the packet.

Packet fragmentation is a danger because the network is not aware that some fragment was dropped so it wastes what limited capacity it has by forwarding the rest of the fragments that comprise the complete, sender constructed packet. Work done to characterize streaming multimedia network turbulence was completed in [LCK02]. A key result of this work was the finding that there is a considerable amount of IP packet fragmentation caused by large application layer packets in some older versions of the Windows Streaming Media technology. Thus, we contribute a study of this phenomenon in the newest version of the technology and investigate possible causations and solutions at the network and application layer, both on the server and client end connections.

Finally, [FF99] proposes a metric for use in evaluating the performance of any network protocol. This metric is useful for comparing the bitrate used by a flow controlled by some particular protocol to that of a TCP controlled connection in the presence of network congestion. The metric is based upon an analytical model of TCP performance, simplified to give the bitrate in response to input parameters of packet size, end-to-end latency and packet loss. Thus, the TCP-friendly equation is:

$$T = \frac{1.5 * \sqrt{\frac{2}{3}} * B}{R * \sqrt{p}} \tag{1}$$

Where B is the packet size, R is the round-trip time, and p is the loss event rate. Equation (1) is an effective yardstick for comparing a particular protocol sending rate to that of TCP in terms of the important network level characteristics. If a particular flow's sending rate is greater than T, it is considered not TCP-friendly. However, Equation (1) is just a model, and as such makes some simplifications. First, there is no distinction between marking, such as ECN, and dropping types of congestion notification [Flo94]. Also, among other things, the TCP-friendly equation also does not consider retransmission timer settings, which are non-standard and implementation related. There have been more accurate models proposed, the most widely accepted might be [PFTK98], which adds the retransmit timer to the model, but has the drawback of being more complicated.

Of course, the TCP-friendly function needs to be used carefully; for some particular loss rate at a congested router the rate returned by Equation (1) may be considered a "friendly" rate while it is unfair in terms of equal capacity usage, when compared to the rates of other flows through the router. This is especially true if the round-trip times of the flows utilizing the router are heterogeneous in nature; the farther away a flow is it becomes more fragile, or sensitive to packet loss in terms of bitrate. This fragility is first caused by the fact that the flow must cut its window size in half for a packet drop. The time to increase the window is coupled with the round-trip time, so a flow traveling over a high capacity-delay product link is slowed drastically down by just a single packet drop. If this fragile flow is competing with a robust, or nearby flow, the nearby flow can have a disproportionate, unfair share of capacity while still being so-called TCP-friendly.

An additional area of active research is the study of the emergent behavior of network traffic and general activity in the network that results from the interaction of complicated congestion control algorithms, drop-tail routers, and competition or contention between users. These users themselves can also contribute to the random nature of the majority of traffic seen in the Internet. This randomness, or self-similarity of the emergent properties of the traffic is exhibited in variation of bitrate over increasing larger time scales, or perhaps long-tailed distributions of packet sizes, interrarrivals, files sizes, and multimedia content duration or bitrates. These factors contribute to make the traffic inherently difficult to manage, characterize or model, and thus understand. An active area of research is the measurement and modeling of these properties.

In [FK03], the authors discuss how challenging it is to model something as complex and evolving as the Internet. They discuss how some simulation setups are often not representative of the system they are designed to simulate. They reasoning behind this, they contend, is that there are not very many accurate measurements, or tools to even understand and analyze real Internet traffic.

The primary focus of [FK03] is to layout the foundations for measurement and modeling of Internet traffic. Of basic importance to this is that certain parameters of a model must be set; arrival rates, service time distributions, queue drop policies, protocol implementations and algorithms must be carefully thought out to highlight the phenomenon being studied. Researchers should know how the parameters affect the test-bed, model, or emulator. This includes something as simple as a maximum queue length setting, RED parameter, or something as complicated as TCP-RENO as opposed to the Linux implementation of TCP. In summary, what we need to learn from this work for application to our research is consideration of the "big picture" for our evaluation.

We build a test-bed to emulate a wide range of network conditions, but we must try keep the test-bed, which is really just a model as realistic as possible. Furthermore, we must not underestimate the importance of any of the parameters to our test-bed, as discussed in a later section. Finally, [FK03] illustrates one possible active research area that our work can be applied to: to build a better model of streaming multimedia for simulation purposes or other situations where an accurate model based on previous empirical measurements would be useful.

The work done in [FK03] motivates our research from a modeling prospective, while from a simulation point of view [PF97] motivates the work as well. [PF97] claims that the Internet is hard to simulate because it is a moving target, non-standard, and difficult to understand, if for no other reason than its huge size. From a pure science standpoint [FK03, PF97] have good presentations of how and why packet drops happen in the network and other basic network phenomenon that a researcher might try to simulate. They also describe how a lot of modeling, simulation, and test-bed studies are conducted using simple network topologies such as the dumbell, and other network research discussion.

Perhaps the biggest impact that [FK03, PF97] has to our work is that they clearly describe a fundamental limitation of simulation; it is not real. Real traffic on the Internet is controlled by the protocols that are actually implemented in current and past operating systems. Simulations can not capture this level of detail, and most scientists wouldn't want to anyways. There are many different flavors of TCP, and operating system programmers take liberties with the specifications and RFC's that define them. Perhaps to gain a little performance, either on the transport layer connection level or on the operating system level in terms of reduced overhead or computational costs efficiency.

A USENIX paper clearly articulates how the Linux implementation of TCP is non-standard and points out the key differences [SK02]. All of these factors motivate us to perform our study on a live test-bed, running the real protocols, instead of relying on simulation results. Furthermore, we use a tool called [NLA] which uses the real network stack in the operating system to generate TCP and UDP traffic in our experiments, to make our traffic sources as realistic as possible. Also, perhaps our work can be used to drive simulations to get more accurate results. Finally, from a scientific standpoint, [PF97] emphasizes that when doing network research one should vary the parameters widely, since the extremes do happen on the live Internet. These parameters are as simple as window sizes, to buffer and queue sizes, and all the way to AQM techniques at routers.

While previous research motivates study of streaming multimedia from a pure science standpoint market oriented research motivates analysis from a sales and economic perspective. For example, articles such as [Ltd02, IS02], focus researcher's attention to streaming media because it makes use of the increasingly pervasive high speed Internet connections available.

Not only are high speed Internet connections more readily available, their adoption is increasing both in the U.S. and other countries such as Korea and Canada [Ltd02]. Recent market studies show that there was a 36% increase in the number of DSL lines worldwide in the first half of 2002, and a 12% increase in the number of cable Internet subscriptions in the United States [Ltd02]. It will be important for streaming media providers to remember that while there is growth, it will not happen over night and they need to consider that some users will be on slower connections. However, some content producers might be inclined to clog the network with their traffic in an attempt to get the best quality possible. From a network-level perspective there may be performance problems if a user attempts to stream high-bitrate content over a low capacity link. It is unknown how a proprietary player, like Windows Media Player, reacts to such user behavior.

An ACM Committee in [otBLMT02] discusses from a general perspective broadband Internet and its growth. They take the position that broadband will be a platform that spawns and spurs many new applications and services. Basically, the part of the work that impacts our research is the philosophy that the link should not limit applications and that the link should have good enough performance that it would inspire the creation of newer applications to utilize it. In summary, faster links should spur more bitrate hungry applications and content. Thus, the speed of the connections and their availability continues to increase coupled with an increase in the resolutions and corresponding bitrates associated with higher quality content [LCKN03]. Many feel that these rises will be coupled with a dramatic increase in the market for streaming multimedia.

A few venture capitalists invision a future where television and radio use the Internet for delivery instead of the traditional delivery mediums. This is particularly promising for content producers because this level of connectivity allows them to not only deliver higher quality content, but charge a higher premium price to produce and deliver it. Also, some researchers predict ubiquitous computing [Wei93] where the network and its protocols will be crucial in delivering an immersed computing experience that is invisible to the user.

Market analysis done by well-known marketing firms was completed in [Ltd02, IS02] which claim that the streaming media market, worldwide, will increase over $2 billion dollars by 2006 [IS02]. Finally, an article in a popular professional magazine, [HAOS01], relates both the increase in market revenues, end user adoption of streaming media applications and TCP/IP performance.

The results of a rise in streaming multimedia technology use is for the most part unknown. Work done in [HAOS01] tried to grasp what kind of impact these applications would have, specifically on TCP/IP performance. Basically, [HAOS01] presents simulation results that show congestion collapse and unfairness is a distinct possibility in the current Internet. They have empirical measurements to back their simulation results, but the main focus of the work is to show that a particular ATM protocol enhancement might remedy some of these issues. The authors of this paper also clearly articulate the core of the issue involving streaming media technologies; they primarily use proprietary and closed-source algorithms to tune the content delivery to adapt to congestion or network conditions in general. To get the best quality, an application may not care about fairness and will try to use as much capacity as possible. This also makes business sense as one can charge more for higher quality content. As stated earlier not responding to packet drops in any way can result in congestion collapse and unfairness. Some of our early pilot studies showed that Windows Streaming Media actually *increases* its sending rate when packet loss occurrs in order to preserve quality. Clearly, this is something that requires study, as the survival of the Internet is at stake.

2.2 Streaming Media Protocols

Windows Media Services can stream media over several application-layer protocols: Real-time Streaming Protocol (RTSP), Microsoft Media Server (MMS) Protocol, and Hypertext Transfer Protocol (HTTP). For RTSP and MMS the underlying transport protocol can either be the User Datagram Protocol (UDP) or the Transmission Control Protocol (TCP). The actual protocol used is chosen through a process called protocol rollover. Some clients may be unable to connect using a certain protocol for various reasons such as player version and network firewall settings. The MMS protocol is in the most recent version of the software only for compatibility reasons. The Windows Media Player and Server attempt to stream the media using the following protocols, in order:

1. RTSP over UDP
2. RTSP over TCP
3. MMS over UDP
4. MMS over TCP
5. HTTP over TCP

2.2.1 Real-time Streaming Protocol

The Real-time Streaming Protocol (RTSP) is defined in IETF RFC 2326 [SRL98]. RTSP is an application-level protocol which controls the transmission of data through the network that has real-time constraints. Perhaps the most well-known application of RTSP is in the delivery of streaming multimedia content. RTSP is primarily based on the HTTP protocol, and borrows many of the same concepts. The primary difference is that in RTSP there is explicit state, while there is none in HTTP. An example is that an RTSP connection can be playing, paused, or stopped.

RTSP provides several states, for example:

- Setup

  When in the Setup state the server does system-level allocation of resources to support content delivery to the client and to set up an RTSP session associated with the client.

- Play

  The Play state is when the server streams the data to the client.

- Pause

  The Pause state does the obvious action.

- Teardown

  The Pause state differs from the Teardown state in a fundamental way; the resources associated with the session are freed by the server, when paused the data just is not sent.

When an RTSP session starts there is capability negotiation, and any extensions to the protocol can be ignored by hosts that do not support them. These extensions could also be controlled in the presentation description file, which is usually located at the URL which the client requests, similar to an HTML Web page. Just like a Web page this file contains references to other content, usually video and audio data. This data is usually abstracted to stream form where there are multiple data streams sharing a common time axis for rendering and presentation purposes. Most URLs that specify a presentation file refer to more than one component stream. This presentation file also contains the encoding parameters, authorship/copyright information, and other parameters that the client can modify to control the media being streamed.

An example of an RTSP URL:

rtspu://saco.wpi.edu:554/graduation.wmv

Here, UDP is the transport protocol, connecting on sever port 554, the default RTSP port, on our streaming media server **saco.wpi.edu**. The content, graduation.wmv, as described by the presentation file is encoded with the Windows Media Encoder and contains a high bitrate video stream, along with a medium bitrate audio stream, and also includes a very low (56Kbps) bitrate video stream. This way, the application can seek to determine what the network conditions are and send either the high or low bitrate stream, as appropriate. Usually, this functionality is application dependent.

Unlike HTTP, RTSP sends the data on a separate channel. Thus, there is the abstraction in RTSP that there is a separation of the control of the stream and the stream itself. This allows the data to actually be sent over a wide range of transport level protocols. These protocols include UDP and TCP, but more generally the data can be sent over unicast and multicast delivery channels. The control information can also be sent over either UDP or TCP, differentiated by adding a letter to the end of the protocol identifier for UDP (rtspu). Having this separate channel for communication allows both the client and server to issue requests, which may be useful for stream control, another key difference between RTSP and HTTP.

The RFC [SRL98], also describes the session identifiers, which we see in our measurements, along with timestamp format and many options. Included is the concept of Normal Play Time, which allows for absolute time positioning in the streams for synchronization purposes, but could also be used to infer network conditions. Small stream data packets can also be bundled into larger transport layer packets to reduce packet overhead and also possibly achieve better performance.

RTSP has several message types, some trigger state changes at the server, for example a PLAY, while other are for retrieving information about the stream. The RTSP message types are as follows:

- DESCRIBE
- ANNOUNCE
- GET_PARAMETER
- OPTIONS
- PAUSE
- PLAY
- RECORD
- REDIRECT
- SETUP
- SET_PARAMETER
- TEARDOWN

Some messages simply cause a transition by the server to the state which is the same as the message type. The DESCRIBE message is sent from client to server, indicating what formats the client can accept, also requesting details about the stream, perhaps number of encoded bitrates or software required. ANNOUNCE messages are sent by the server to announce that some stream is available. REDIRECT messages are sent to direct the client to a different server for the content. GET_PARAMETER and SET_PARAMETER messages are for setting various stream options, most of which are implementation specific.

The SETUP message sent from client to server initializes the transport protocol used to send the media through the network. The SETUP can be changed midstream by sending another SETUP message, perhaps changing from UDP from TCP. Specifically, the client supplies the protocol and port numbers. This dynamic process makes streaming media hard to analyze through the use of tcpdump, a network protocol analyzer [JLM]. Some research has been focused on building a software tool based on tcpdump for analyzing the RTSP and H.261 protocols [vdMChCS00]. However, this tool isn't available for download and there hasn't been any publications using the tool for some time. We discuss this point further in the software section.

The server replies to a SETUP message with a 200 OK message, including a sequence number, session identifier, server port number, and the date. Next, the PLAY message causes the server to send data on the specified ports using the correct transport protocol specified in the SETUP message.

Since the data flow is conceptualized as a stream, these PLAY messages are handled in an interesting way. The PLAY message includes a range of time to play from some particular content. PLAY requests can be queued up on the server side, and will play them in the exact order received. The server will not stop a stream currently being sent via a PLAY message when a new one arrives, but instead it attempts to time the delivery so that the presentation on the delivery side is as seamless as possible. In other words, there should be no pausing to buffer the new stream. The RFC claims this is to allow careful editing of the stream, even to the granularity of synchronization based on UTC from different content sources, but it also serves another purpose. It allows precise control for a bitrate control algorithm implemented in the streaming server and player.

Consider, for example, a stream is being sent and the client determines that 10% of the packets being sent from the server are being lost, and those that do arrive have an average jitter of 35ms. The client assumes that this is due to congestion in the network (or insufficient capacity in general) and sends another PLAY message to the server to switch to a lower bitrate stream. Since the stream is still playing from the original PLAY the client then needs to send a TEARDOWN message to request that the streaming data stop being sent and free the resources associated with it. After this, another SETUP message is required to start the stream up again, however, since the second PLAY was issued, the presentation will switch to that one instead.

There are numerous error messages in RTSP described in [SRL98]. For example, code 453, "not enough bandwidth", is sent if the server has insufficient capacity to support the request. Also, there is a wide range of headers specified in [SRL98], including a "bandwidth" header which specifies the client-side application level estimated capacity available, in bits per second, that can change during an RTSP session.

The blocksize header is sent from the client to the server requesting a particular packet size at the application level (it doesn't include any IP, UDP, or RTP headers). However, the server could use a smaller size, or ignore the parameter completely. Finally, the Speed request header allows the client to ask that the server send the data at a particular rate. This can be ignored by the server and the default is the bitrate of the stream. The speed request is a floating point number, perhaps 2.0, which would mean stream at double the bitrate. At this point, the RFC mentions that the use of RTCP is highly recommended to track packet loss when using UDP. Lastly, [SRL98] mentions the Timestamp header and makes note of the fact that the timestamp is used at the client side as a measurement of round-trip time for use in client side retransmission timers.

2.2.2 Real-time Transport Protocol

Typically, the Real-time Transport Protocol (RTP), defined in IETF RFC 1889 [SCFJ96] is used by RTSP as the data delivery mechanism. The Real-time Transport Protocol is an end-to-end network transport-level protocol designed for delivering real-time data, particularly media content like audio and video. RTP is like UDP in that it does not respond to network conditions and does not guarantee quality of service. RTP is used in conjunction with the Real-Time Control Protocol (RTCP) protocol which provides monitoring of the traffic flow and provides a means to control RTP.

There are several features of RTP that are a result of the desire to use the protocol to send media content. Primarily, identifiers for payload types, sequence numbering, timestamping, and delivery monitoring. The protocol designers did consider network impact, noting that high-bitrate delivery over RTP needs to be careful of quickly consuming network resources. In [SCFJ96] there is a notion of a monitor, which strives to estimate the network link characteristic between sender and receiver.

The data delivery rate and other factors are controlled through analysis of reports received through RTCP. Basically, control packets are sent using the same mechanism as the data packets. Then, the quality of data distribution can be determined, and any congestion control mechanisms can be triggered. This is the RTP analog of the congestion control algorithms in TCP [APS99]. Basically, there are 5 RTCP packet types; sender reports, receiver reports, source description items, termination messages, and application specific functions. The reports are of particular relevance; they are the the transmission and receptions statistics. Since RTP uses UDP, a report is needed from both ends of the connection to gather meaningful statistics about the link properties. Thus, these reports should be sent frequently, perhaps every round-trip time. Precisely when receiver reports are sent is not specified in the standard, so we will not discuss them here.

There is a fixed format to the reports including the following fields:

- Fraction of packets lost
- Cumulative number of packet lost, not including any packets that arrive but are too late to use
- Packet interarrival jitter
- Timestamp for round-trip time calculation

- Other statistics

Certainly, some of these fields can be used to adapt the stream to network conditions, or modify its transmission characteristics. Example uses of the receiver and sender report data mentioned in the RFC include using the packet loss rate over an interval as a measure of network congestion (assuming precautions are taken to assure statistical confidence in the metric). Other uses are rate calculations, and interarrival jitter as another measure of network congestion.

The RFC discusses the fragmentation problem seen with this protocol from a high level. It notes that the maximum length of the RTP packet size is only limited by the underlying network MTU's and protocols. This could explain how huge RTP packets were sent by previous versions of Windows Media that were later fragmented into three or more fragments [LCK02].

2.3 Windows Streaming Media

2.3.1 Intelligent Streaming

Windows Media Services uses a technology called Intelligent Streaming [Bir00]. Intelligent Streaming is the process where the client and server attempt to automatically detect network conditions and adjust the properties of the video stream to maximize quality. Upon the initial connection an estimation is made of the available capacity and adjustments are made to maximize the quality of the stream. When the capacity or connection properties change, the stream is dynamically adjusted without user intervention.

The way that the streaming technology determines the capacity and connection properties differs from streaming media product to product. Windows Streaming Media uses the RTSP and RTP protocols as the application and network level protocols and uses facilities provided by them to communicate connection information. As discussed in Section 2.2.2 the RFC specifies some fields that can be filled in the application to provide some network level metrics for use in rate control, for example packet loss rate and interarrival jitter. Based upon these receiver reports the streaming media server can adjust the network requirements of the stream through this so-called intelligent streaming. Real Player has a similar mechanism, known as Sure-Stream analyzed in [CCZ03] and QuickTime also makes adjustments to the bitrate requirements of the stream by altering the quality level.

From a coarse view, all of the technologies are able to send a low bitrate stream in lieu of a high bitrate stream if it determines that the network link cannot support it. RTSP facilitates this by allowing the client to start receiving the low bitrate stream by sending SETUP and PLAY RTSP messages, as discussed in Section 2.2.1, then send a TEARDOWN message to stop transmission of the high bitrate stream. Thus, the client is able to deliver a continuous presentation of content to the user. Typically, most users prefer a lower quality movie that streams without pausing or rebuffering than a high quality one that halts frequently. Also, the server can send fewer frames or a lower quality audio stream as a means to adjust the quality and network requirements of the content.

2.3.2 Multiple Bitrates

Encoded content must contain multiple bitrates to make use of the features that intelligent streaming or related technologies provide [Bir00, CCZ03]. When content is encoded into multiple bitrates, as many as ten different video streams are encoded into one Windows Media object. This allows the client to specify a single object located on the server and through intelligent streaming the server will determine which encoded bitrate contained in the object to send. If there is only one content level, or the lowest bitrate stream has been reached, the server will start to "thin" the stream by first decreasing the video frame rate, next by stopping video frames, and lastly, if needed, only by sending audio. This intelligent streaming process is similar to the media scaling process of RealPlayer, studied in previous work.

Past work [CCZ03] measured the number of scaling levels contained in RealVideo content. We completed work in [LCKN03] that measured the actual bitrate of the content streamed from remote servers across the Internet. We found that most videos have small resolutions and subsequently small bitrates in relation to available end-host display capabilities and network capacities. In other words, increasing bitrates and resolutions of content available on the Web are likely to increase. Further, since this increase results in larger content file sizes, users will be motivated to stream the content across the Internet and view it in real-time instead of simply downloading the content to disk.

There is existing knowledge in networking research that can perhaps benefit streaming multimedia. For example, work done in [PV02] tries to actively probe the network to determine the link characteristics. They use the concept of Packet Quartets, where four packets are sent back-to-back and the time between their arrival at the reciever can be used to infer link capacity. Another technique, [Hao02] seeks to passively estimate link round-trip times by exploiting some characteristics of the TCP protocol. There certainly can be adaptations of the concepts in [PV02, Hao02] to help deliver better, more reliable streaming media content.

One of the earlier tools that sought to estimate link characteristics was the pathchar tool [Jac97]. Work done in [All99] evaluated pathchar and identified times when it does and does not measure the link properties accurately. Futhermore, the author explains the theory behind pathchar succinctly and includes ways to reduce the sample sizes needed to get accurate results. The measurements include latency, capacity, and queue delays. Finally, [All99] concludes that latency estimation is easy, but available capacity measurement is difficult to do accurately, especially on high-capacity links and that the more probes, the more accurate the measurement. Certainly, streaming multimedia over the RTSP/RTP protocol has an ample number of probes to use to determine the network link characteristics, hopefully to achieve the best quality over the network link. Examination of the network-level packet data of streaming Windows Media reveals that it does in fact make use of a capacity-estimation technique called high entropy packetpair. Apparently in this technique three large (1500 byte) UDP packets are sent via the RTP protocol back-to-back and Windows Media infers something about the link characteristics based on data gleened from these packets. It appears that this measurement only takes place just prior to buffering.

Intelligent Streaming and SureStream are not the only algorithms that attempt to adjust the quality of the stream to dynamically suit changing network conditions. Both [LOR98, GCK$^+$02] are algorithms and systems for adapting video streams to network conditions. They state that Internet streaming is controlled by variations in throughput, loss rates, and delays and that these parameters should be used in a quality controller to adapt the video stream accordingly.

The authors in [GCK$^+$02] note that rate scalable content is required to allow the server to vary the network bitrate requirements of the stream without requiring too much computation by the server. Selectively dropping frames based on priority could be an example of an expensive server side computation to scale quality. They propose that multiple redundant streams are a means to support this capability, for example, a high and low bitrate video stream along with a single audio stream. The high bitrate can be dropped while the low bitrate stream continues, with very little computation required by the server. Windows Media achieves this by encoding multiple bitrates in pre-packaged frames directly into the content object. They suggest a scheme that uses priority frame dropping and a retransmission scheme to better adapt to network conditions and evaluate performance using bitrate, loss rate, and latency as metrics.

2.4 Streaming Media Research

There is previous research in the general area of streaming media. Several publications are measurement studies trying to gain insight into the various properties of streaming media [LR01, LR02b, WCZ01, MH00, LCK02, HAOS01]. Other researchers seek to characterize streaming media traffic or workloads [dMSK02, CWVL01, VAM$^+$02, AS98]. Yet another area of active research is done to study new techniques for bitrate adaptation and streaming rate control [LR02a, GCK+02].

One of the most cited earlier works which measured streaming media is [MH00]. This work measured characteristics of Real Audio streams, a key result being that audio bitrates are usually lower than available capacity. In [WCZ01] RealVideo performance is studied empirically. They developed a custom tool which measured the properties of the stream during transmission. This tool was then bundled with a play list and distributed to geographically diverse locations; the performance was then measured from these points. The main research contribution of this work was to relate frame rate, an important quality metric, to a network metric: bitrate.

Additional streaming media research, in [LCK02] characterizes the network properties of Windows Streaming media streams and compares it that of a RealPlayer stream. They found content in both formats on publicly available streaming media servers. Then they streamed them simultaneously to compare the performance in terms of application and network level metrics. A drawback of this work is that the clip selection was limited to only 26 clips because of what was publicly available on public streaming media servers on the Internet.

Several interesting results are presented in [LCK02] including measurements of packet sizes for both streaming technologies. They found that Windows Streaming Media packet sizes can be larger than network MTU's, resulting in a great deal of IP packet fragmentation. However, streaming media server administartors can modify the packet size as a parameter during encoding or in the newest version of the Windows Media server software. By analyzing the interarrival times of the packets they found that Windows Media packets arrive at constant time intervals, while the

RealPlayer traffic is much more varied.

Of particular interest is the result that the buffering strategy is different in Windows Streaming Media (version 8) and RealPlayer. RealPlayer buffers at data rate much higher than the playback rate, while Windows Media buffers at the playback rate resulting in less bursty behavior. In our pilot studies, we found that the newest version of Windows Media Server now exhibits a buffering strategy similar to that of RealPlayer. This motivates us to examine it in detail in our work, as it is an interesting tradeoff between a quick start of playback, smooth playout, and network turbulence. Finally, an additional result they found was that Real clips had a much higher frame rate for an equivalent bitrate than Windows Media clips did. In conclusion, this work suggests areas of interest, but the limited clip selection, lack of server-side access, and uncertainty of the network conditions of the public Internet weaken the results.

In [CCZ03] the head-to-head performance of streaming RealVideo over UDP and TCP is compared. Similar to [LCK02], the clips are selected from servers on the public Internet and streamed simultaneously over both UDP and TCP. Using a token bucket filter congestion is induced on the last-hop router. They found that in general, streams over UDP receive about the same bitrate as do streams over TCP. However, during times of extreme congestion UDP can get an unfair share of the available capacity. During specific intervals the UDP streams still remain TCP-friendly, but UDP flows are getting much more of the available capacity than the TCP streams. This condition worsens as congestion levels increase. Also studied in the work was the media scaling process of RealPlayer. They find about 35% of the servers cannot scale the content, probably because they have only one encoding level contained in the content. Thus, these clips are likely unresponsive to congestion. Finally, they also measure the buffering stage of RealPlayer streaming.

Other measurement efforts have been undertaken in two very similar papers [LR01, LR02b]. [LR01, LR02b] studies how to stream video over low capacity connections, providing a good lower-bound on performance of streaming Internet applications. Basically, they dialed into ISP's from access points located throughout the U.S. Then, they stream video over the dialup connection, analyzing the performance. They discuss the start-up delay, or buffering as mentioned above.

In terms of results, [LR01, LR02b] quantified the average packet loss rates to be only 0.5%. In most cases resending lost packets was an effective means of repair and about 94% of resent packets arrived in time to be used. Roughly 40% of the connections they measured had no packet loss, with 75% less then a 0.3% loss rate, 91% less than 2%, and only about 2% of the connections had a loss rate of greater than 6%.

For round-trip time measurements, [LR01] found that the average was 700 ms, and the minimum about 120 ms over a modem. 75% of the connections had round-trip times less than 600 ms. They show that large values of delay jitter were rare and packet reordering rates were also low, only about 6.5%. A final note on this work is that they do some analysis to see if the packet burst lengths and round-trip times are heavy-tailed, but they do simple fits that lack any formal statistical test (a suitable choice would be the Downey curvature test). [LR01] also claim that the distributions of burst lengths and round-trip times are long-tailed and was in fact best modeled by a pareto distribution with an alpha parameter of 1.16. Additionally, 72% of the sessions they measured had asymmetric paths.

Characterization of streaming video traffic and workloads is another fruitful area of streaming media research. There have been several publications in this area [dMSK02, CWVL01, VAM$^+$02, AS98]. [VAM$^+$02] analyzed the logs of a popular, live, reality-show that streamed content over the Internet. They had measurements from a 1 month period with 5 *million* requests. Primarily, they analyze request arrivals, session times, bitrate, and object popularity from client, session, and application perpectives.

Further work characterizes the arrival rates and arrival process of streaming multimedia by analyzing millions of log files [dMSK02]. The best quality of this research is the fact that they processed a great deal of data points, from traces and router logs. They motivate their work by discussing the potential economic impact of streaming media. This impact again is coupled in this work with the fact that the availability of broadband connections increases. In a nutshell, they collect and analyze the data from an end-system point of view and from a network level perspective.

In summary the results of [dMSK02] impact this investigation in three main ways. First, they found that the Microsoft technology dominated the end-user choices when Real was also offered. Second, high bitrate clip popularity ratings greatly exceeded those of low bitrate content types. Finally, they found that the streaming and network transport protocol varied between a proprietary streaming protocol over UDP, TCP, and HTTP over TCP about equally for on-demand content. For live content, most (50%) of the traffic was HTTP over TCP, with the remaining split between the TCP and UDP. This is probably emergent behavior caused by the interaction between typical uses of live content, like corporate board meetings, and among other things, the fact that most reasonably-secure firewalls block incoming traffic on unknown ports using unknown protocols. Thus, HTTP over TCP dominates in the circumstance of live content as most firewalls allow Web traffic, which uses HTTP.

Also of interest in [dMSK02] is the statement of the fact that both Real and Microsoft recommend streaming over their proprietary protocols instead of HTTP. The authors allude to the fact that this could be motivated by the notion that they can ignore some packet loss and get better quality, thus more users and money. In conclusion, additional results of this work, not described here, about modeling the end-user behavior, such as requests, could also be useful to researchers modeling streaming media.

Design and analysis of new streaming media protocols and congestion control mechanisms is the focus of past work. [LR02a] discusses a new mechanism for increasing and decreasing the rate and quality, and thus network requirements of streaming. They claim their technique is a scalable form of rate-based flow control that performs well in two new performance metrics they devised. First, they measure monotonicity of convergence to fairness, and second, packet-scalability, equated with constant packet-loss. From a network performance evaluation perspective, their link size is a T1 at 1,544 Kbps.

[LR02a] introduces a notion of a difference between congestion control algorithms that are implemented in protocols with packet acknowledgement and without packet acknowledgement. Typically, streaming Windows Media uses UDP and thus falls into the second category of NACK-based protocols. A finding of their work is that to achieve true scalability flows must be able to accurately determine bottleneck capacity. This finding begs the question how to make this measurement. An interesting angle to TCP-friendliness is that DiffServ and related technologies will eliminate the need for protocols to be TCP-friendly. This fact remains to be seen.

2.5 Measurement of Non-Streaming Traffic

To provide complete background to our work we cover several papers that are in the area of network measurement, but not specific to streaming media traffic measurement or streaming performance evaluation. The measurement research [Pax99, All00, FA99, Wil01, PF97, PF95] gives our methodology justification and gives perspective to our research goals.

One of the largest measurement efforts done by Paxson in [Pax99] characterized network performance using a wide-range of techniques. They found that out of order packets are fairly prevalent, while duplicated packets are rare. As mentioned earlier, this work found average loss rates between 2.7% and 5.2%. They found that loss events were not well-modeled as an independent process for TCP traffic, as a result of drop-tail queue management in most Internet routers. The author makes an interesting distinction between bottleneck and available capacity, the former being analogous to possible bitrate, the latter to what bitrate the connection should use. Finally, the author makes a point that measurement scientists should be careful that any losses measured are genuine loss and not measurement drops.

Allman presents the results of instrumenting a particular WWW server over a long period of time [All00]. Using tcpdump [JLM] he measures packet loss for the incoming

TCP connections as well as round-trip times using the Karn algorithm and tcptrace tool [KP87]. The empirically measured distribution of round-trip times is a key result of the work. They found that 85% of the round-trip times were between 15 and 500 ms, the median was only about 100 ms, with 75% of the round-trip times greater than 100 ms while only 40% were greater than 200 ms. These results are valuable to us in guiding our test bed parameters, but this work also is a good introduction to protocol measurement, especially instrumenting the server side of the connection.

In [FA99], they discuss the proper way to evaluate the TCP protocol and the various modifications that have been proposed. However, the methodology they present is applicable to other types of protocols. First, they describe how test-beds, simulation, and emulation are really the only way to evaluate new TCP versions, testing on the live Internet or other network is usually out of the question. Second, they discuss what some researchers forget: the scientific method. They talk about how knowing and providing the details are crucial for the effective evaluation of TCP or any other protocol. Third, they talk about choosing a TCP implementation, perhaps Windows or Linux, and some of the possible effects. In our experiments we use TCP traffic as comparison for Windows Streaming Media traffic, so these effects are perhaps relevant to us. Since we are using Windows, it is hard to determine these effects. However, Windows TCP probably dominates Internet traffic, due to the dominance of the Windows family of operating systems.

Another perspective is when to just simulate the whole protocol instead of test-beds doing network emulation or live Internet tests. We chose to do the live testing over real networks running the real TCP implementations in use. However, if we wanted to evaluate some high-speed networks or satellite links we may have trouble with our limited equipment and the packet capturing boxes cannot handle much more than a few megabits per second. An additional drawback to using a test-bed is that we use a commercial operating system, so the TCP implementation is black-boxed and we have no way to modify it, although we probably would not want to if we could. Also, there are some bugs or quirks of the implementation that may be hard to discover [Pax99, FA99].

Our approach is actually a combination of a test-bed and network simulation. We insert a machine between our client and server which emulates live Internet or other network conditions, artificially and controllably by modifying the traffic passing through it. This allows us to evaluate protocols over any type of link that our hardware can fundamentally imitate, usually 100 MBs Ethernet being sufficient. An advantage of using emulation is that it is not necessary to use the public Internet, thus making our results more reproducible. In [FA99] they mention that their emulation abstracts some of the real behavior of the network, but they do not provide an example.

Finally, [FA99] discusses some other aspects of protocol evaluation that are of relevance to us. We will not discuss them here in detail, but merely enumerate them for later reference. First, a proper TCP window size must be chosen that is appropriate to network capacity, otherwise under utilization will occur. Second, the actual data sent needs to be chosen carefully; however for us we encode high quality content and let the Windows Media encoder take care of the packetization. Third, they discuss some possible metrics for evaluation: loss, fairness, and router queue length. From a measurement standpoint, they discuss the fact that application and network level measurements will often be different. Finally, they discuss further network parameters that need to be considered: segment size and maximum queue length, and suggest it be set to at least the capacity-delay product.

In a tutorial, [Wil01], Williamson discuss the various approaches to Internet traffic measurement and the role that this area has in networking research. Basically, measurement can be helpful for troubleshooting, protocol debugging, workload characterization, and general performance evaluation. The work covers the distinction between hardware and software monitoring and the different advantages. Also compared is passive vs. active measurement, on-line vs. off-line traffic analysis, LAN vs. WAN measurement, and the appropriate level in the networking stack to take measurements. The tcpdump [JLM] tool is covered. Finally, the work concludes with the ten biggest recent results from traffic measurement, a few of them include: TCP traffic dominates on the Internet; TCP connections are usually short; packet interrarrival times are not exponentially distributed; session interrarrivals also aren't exponentially distributed; packet size are bimodally distributed; network traffic exhibits "locality" properties; aggregate traffic is multi-fractal and self-similar.

3 Methodology

To evaluate performance, characterize the response to congestion, and measure the relevant emergent network-level behavior of streaming media, we built a test bed and instrumented it with network emulation and measurement software. Then, based on some pilot studies we developed some hypotheses, encoded some content, and designed experiments to test the hypotheses. This section covers our research methodology including test bed and software design, and the content encoding process and parameters.

3.1 Streaming Media Test Bed

In order to effectively and precisely characterize the response of streaming media to congestion we built a test environment that allows for network level parameters to be adjusted such as packet loss rate, round-trip time, and maximum capacity. Additionally, the test bed facilitates accurate measurements of network performance metrics, including achieved bitrate, interrarrival times, and burst lengths, along with streaming media related measurements, such as content buffering period length.

3.1.1 Network Topology

Figure (1) depicts the network topology of our streaming media test bed. The streaming media server is connected to the Proxy-ARP router via Ethernet. On this segment we placed a hub and a measurement machine. The router does Proxy ARP forwarding from the external network interface to the two machines located on the internal Ethernet segment. This allows us to control the last hop-between the server and the

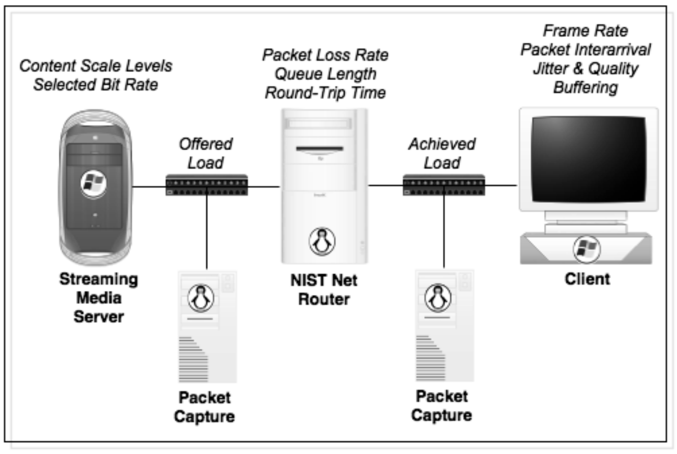

Figure 1: Streaming Media Test Bed Topology

rest of the world. Between the router and the external network we place another hub and measurement machine. This configuration provides a means to measure the offered traffic going in both directions and a way to compare offered load with the achieved throughput passing through the router.

The baseline network overhead for using the network topology and routing strategy is low. Pings through the router from a machine on the external network to the server are only a few milliseconds longer than pings to the server when the router is removed. The client streaming media from the server was located on the WPI local area network. Since the WPI LAN consists of fiber optic Gigabit uplinks from the academic buildings to the WPI backbone [Net], it is unlikely that any congestion or interference was seen between the clients and the streaming media server. Thus, our network bottleneck greatly exceeds the typical end-user's network configuration. We ran simple tests to evaluate the performance of the entire test bed using traceroute and ping from various locations on the network. The traceroute tests showed all machines on the WPI LAN were within one IP hop of the Proxy-ARP router. We were unable to measure or determine any switches between hosts that worked below the IP layer. When we measured round-trip times using ping we found that they were less than 1 ms for small 64 byte ping requests and generally less than 5 ms for larger 1400 byte packets.

This network configuration provides a high-performance baseline for measuring streaming media. We also needed to use open-source and custom-built software tools to facilitate the experiments. Having a high performance LAN and test-bed as the underlying technology allowed us to limit the performance of the network for experimental purposes. Using special software installed on the router we were able to emulate the network conditions experienced by typical broadband end users with connection speeds well below that of our LAN. This allowed us to accurately and succintly measure the performance of streaming media under realistic situations.

3.1.2 Hardware

The streaming media server is a dual processor AMD machine running at 1.6 Ghz. It is equiped with a fast hard drive (7200 rpm and 8MB Cache), high-quality network interface card (100 Mbps), 1 GB of RAM, and is running the Windows operating system suporting the version 9 of Windows Media Services. The client machine is a Celeron machine running at 433 Mhz with 128 MB of RAM running Windows 2000. The routing and measurement machines are all 233 Mhz Pentium MMX machines with 128-256 MB of RAM. Each runs the Gentoo distribution of the Linux operating system, kernel version 2.4.20, and all the binaries on the system are compiled with

Pentium MMX specific compiler optimizations for better performance. A packet capture machine is equiped with several ultra-wide SCSI drives and and off-board SCSI controller card. This machine acts as an NFS server for data and log file storage, but during measurements each machine caches any data locally and transfers the data after the experiment is completed.

3.2 Software

This section discusses the software used in this research. We used NIST Net [NIS], a popular network emulation tool, standard network performance analysis tools, and custom built tools, constructed for our measurements and experiments.

3.2.1 NIST Net emulator

As discussed in the Background section we decided to build a test bed and use emulation to measure the performance of streaming media under a wide range of network conditions. This method of experimentation was selected for several reasons. First, we wanted to examine the performance of real traffic, partly because streaming media traffic generation tools do not exist. Second, we wanted to examine Windows Streaming Media (WSM) specifically, thus we had to have a real version of the server and operating system running. Currently, there exists no way to simulate the behavior of real WSM traffic. Thus we chose to emulate the network to provide fine-grained control over the experimental parameters.

There were several choices of network emulation software packages. Among these, [Lui97] and a wireless emulator [KGM+02], have been used in successful research publications. NIST Net is Linux based, simple and open source. NIST Net was previously used with success in other research [SGB$^+$03]. At a fundamental level, NIST Net is a kernel module that intercepts packets that match user-defined rules and does some special processing on them, including random packet loss, delay, and duplication. Also capacity constraints and RED like AQM disciplines can be applied, to facilitate the emulation of a wide range of network conditions. Random loss applied to a particular host pair can emulate channel loss on a wireless link, while bitrate limitations could mimic the effect of congestion caused by contention for capacity with other flows that do not actually exist. Thus, a level of realism is lost by emulation. However the level of control afforded to researchers would not be attainable by other means.

NIST Net, by default, uses a mechanism for limiting capacity called Derivative Random Drop (DRD). DRD works in a way very similar to the active queue management technique RED but is simplified considerably to make it easier to implement. DRD effectively limits bitrate use, as would a simple drop-tail scheme, but DRD was chosen because it would not drop bursts of packets. From an emulation standpoint perhaps this is the correct approach. On a real link, if the available capacity is exceeded, packets would get dropped by the congested router in a drop-tail fashion. Since there probably are many flows through the router, the effect is that some packets are dropped from one flow and some from another, an effect that DRD emulates.

However, initial pilot studies showed that DRD, and its accompanying three parameters, were causing odd behavior including under utilization. In our research we did not want to tune these three parameters, since we were not interested in performance of DRD. Thus we chose to implement drop tail in NIST Net. This is more representative of last-hop routers where congestion is likely to occur on real end-user Internet connections.

At a very low level, the way that NIST Net works is as follows. On every packet arrival NIST Net checks to see if the packet matches any of the rules that user entered into the interface. If not, the packet is sent through the normal kernel routing table. If there is a match, NIST Net marks a portion of the sk_buff kernel data structure which represents a packet. Next, the random delay, duplication and drop is applied. In the case of delay the packet is placed inside a kernel data structure which schedules the transmission of packet delayed by the emulator, due to a random delay, or queuing delay due to a link capacity limitation. Drops can also occur due to exceeding the queue length, but the random drop probability is applied under all conditions. When the time in the scheduler has passed, the packet is placed back on the interface it originally was received upon. Finally, NIST Net sees that it already has processed the packet and forwards it though the normal kernel routing table. This operation results in duplicate captures of each packet when packet capturing software is running on the router itself. This was part of the motivation for having two separate measurment machines in the test bed to capture the traffic flowing in and out of the router. NIST Net also provides performance data and we wrote a script which collects the data at discrete intervals for analysis purposes.

3.2.2 Measurement Tools

When the content is streaming from client to server, the program tcpdump [JLM] was originally run on the routing machine to log the packets being transferred. However, we found that the routing machine was not able to keep up with the packet capturing load and subsequently added the separate packet capture machines. Capturing and saving packet data allowed later analysis using the open-source program ethereal [Com] to measure packet size and throughput. Client side measurements also were made using the program MediaTracker developed in previous work [LCK02]. MediaTracker allowed several performance statistics to be measured with respect to time: bitrate of the clip being received, buffering progress, and the reception quality - a percentage of packets received in the last 30 seconds. NIST Net has facilities for capturing per-connection statistics such as the number of dropped packets, bitrate, and total bytes sent. Finally, we used a set of custom analysis tools to measure characteristics of streaming media that have an impact on network performance, for example, packet interarrival time and burst length distributions.

3.2.3 Experimental Scripts, Parsing, and Analysis Tools

We created several custom software tools to help us in our experiments. First, we wrote a perl script which processed the packet capture output. This perl script measured bitrate over time and summary statistics, such as average packet size, and interarrival time. Futhermore, the tool parsed the output from MediaTracker to aid in determination of the buffering stage of the content. The packet capture parsing tool examined the output file and found the length of time taken by MediaTracker to report that the buffer was 100% full. Next, the tool found the RTSP PLAY message in the packet capture file and considered the time of this message plus the length of buffering to be the buffering period and reported some statistics before and after the buffering stage was completed.

Next, we created another perl tool which automated the experimental setup. Basically, this tool allowed a range of parameters to be input and the number of experiments to repeat to obtain statistical confidence. With these tools in place we were able to hypothesize based upon results of simple measurements using basic test bed configurations, then repeat for statistical significance and vary parameters to understand the performance under a range of conditions.

Once the experimental setup was in place we ran several pilot studies and examined the preliminary results. As is often encountered in network measurements, we were initially overwhelmed by the shear amount of the measurement data collected. Using Microsoft Excel we visualized the results in some straight-forward ways. However, we found that having a separate perl script and external visualization tool was cumbersome. Thus, the perl script was combined with the open source program Gnuplot to allow rapid processing and visualization of the measurement data.

3.3 Encoding

In order for operating systems to play multimedia content, a player typically must be installed. The player is software that can process certain types of specially formatted multimedia files. The format of these files is created by the content encoding software. This encoding software, typically provided for free with the streaming media server, processes existing multimedia files or raw data streams from rendering software or digital video cameras and outputs a multimedia file in the particular format specified by the codec. The codec also includes specifications of compression techniques and parameters, that are not discussed in detail here. This section presents content types used in prior work, the process of encoding Windows Media content, and the selection of bitrates.

3.3.1 Content Types

The content to stream from the server can be varied in a number of ways. These variations include size of video, audio quality, and amount of motion. Content types in previous studies [CWVL01, VAM^{+}02, AS98, dMSK02, CCZ03, WCZ01, MH00, LCK02] included live video of a reality show in Brazil, sports clips, commercials, music videos, news, and movie trailers. In order to be streamed from a Windows Media Server, the content must be encoded using the Windows Media Encoder into the Windows Media file format. During the encoding process a large number of parameters can be tuned, including content bitrate, frame rate, audio settings, and frame size.

3.3.2 Windows Media Encoding Process

Upon initially starting the Windows Media Encoder the user can choose to run one of the provided “Wizards”. The types of Wizards include converting an existing media file to the Windows Media file format, capturing content from a device, etc. Choosing a Wizard automates setting up the encoding process by first requiring the user to choose from several distribution methods. The option that is at the center of this research is labeled *Windows Media Server (streaming).* Once this option has been selected, the user can choose how to encode the content. However, the default selections are multiple bitrates video and multiple bitrates audio. Additionally, the Wizard provides several default bitrates to choose from for the encoded content, ranging from 1128 Kbps to 28 Kbps. The difference between bitrates is the adjustment of several parameters such as audio sampling rate, video output size, and frame rate.

3.3.3 Selected Bitrates

According to the documentation on intelligent streaming, content should be encoded into several bitrates. This may play a significant role in how responsive the streaming media is to congestion. The stream can only be responsive to congestion if the content contains appropriate bitrates to choose from. Otherwise the server will thin the stream when congestion is encountered. To test this hypothesis, multiple encoded versions of the same original content were created and placed on the media server, using the default encoder settings:

- Ten single bitrate clips with encoded rates of 1128, 764, 548, 340, 282, 148, 109, 58, 43, 28 Kbps.
- One multiple bitrate clip with all ten of the default encoded rates.
- Starting with the 1128 Kbps clip we added the next highest bitrate to create a multiple bitrate clip containing 2 bitrates (1128, 764). Next, we created a clip with the next highest bitrate for a total of 3 bitrates (1128, 764, 548). This

process was continued for all of the remaining bitrates, resulting in a set of 10 clips where the number of bitrates increased by the next highest bitrate until all 10 bitrates were included in the clip.

- Next, we repeated this process but started with the 28 Kbps clip and iteratively added the next lowest bitrate, i.e. (28, 42). This process resulted in another set of 10 clips where the number of bitrates increased by the next lowest bitrate until all 10 bitrates were included in the clip.
- Several additional multiple bitrate clips containing two or three bitrates.

3.4 Independent Variables and Network Parameters

With the streaming media test bed in place, we were able to setup finely controlled and reproducible experiments. These experiments, were designed to control network level characteristics such as capacity contstraints and loss rates. We also were able to control the content on the server and the network and application protocols used to stream the media. This coupled with our measurement methods allowed us to characterize the responsiveness of streaming Windows Media under a variety of scenarios.

First, the round-trip time in the testbed was fixed at 45 ms, a setting reasonable in light of prior measurment results [LR01, Pax99, All00]. We vary the loss-rates in our experiments in small increments over the range of rates that are typically found in the Internet (less than 5%), but increase them to very high loss rates (20%), (again, in light of previous measurement results [LR01, Pax99, All00]). Next, capacity constraints were chosen to reflect typical end-user Internet connections: 250 Kbps (low-speed broadband), 725 Kbps (Cable/DSL), 1.5 Mbps (High-end DSL).

In order to enable the capacity constraints on the NIST Net router we also had to choose the maximum queue length parameter for use in the drop-tail queue. We examined some previous work to find some justification for the setting of this parameter. In [FA99], they stated that the maximum queue length should be set to be at least the capacity-delay product. In [CJOS00] they compare RED and drop tail queues. Besides motivating a setting between 7-135 ms for round-trip time they further strengthen the fact that the queue length should be at least the capacity-delay product. The authors of [CJOS00] also point out that the queue length parameter setting has been discussed many times on networking research mailing lists, which we can support from our own personal experience as subscribers on the IETF end-to-end mailing list. In [CJOS00] they discuss how varying this parameter balances a tradeoff between latency, utilization, and throughput. They find that if the queue length is too long the latency increases sharply, while when it is too low the flows may timeout. From our pilot studies, we supported these claims. However, we found that setting the maximum queue length to only the capacity-delay product resulted in frequent oscillations in throughput and unpredictable behavior during a 60 second run. Thus, we iteratively increased the queue length parameter until we found the throughput stabilized and kept this queue length parameter for all of the experiments. The experiment results are reported in the next section.

The authors in [CJOS00] discuss how they let the flows "warm-up" before starting measurement, usually about 20 minutes of each run. We found that we only needed to let TCP warm-up for 10 seconds for it to achieve full link utilization. To obtain

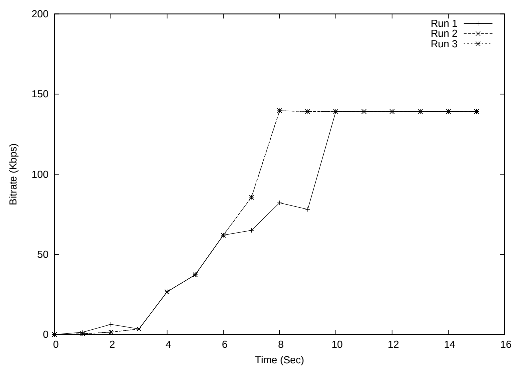

Figure 2: Bitrate versus time for TCP flow.

this result, we initiated a TCP flow with iperf [NLA] over a link with a very high latency, 1000 ms, and a very low capacity, 140 Kbps. These network settings have the effect of making the TCP flow more fragile then any of the TCP flows in our experiments. Thus, these results give us the worst-case performance. We separately ran three TCP flows over the link, Figure 2 shows the bitrate versus time for the begining of each of the three flows. We see that in each case, after 10 seconds has elapsed the TCP flow has warmed-up and is using the full link capacity. Thus, in our experiments allowing the TCP flows to warm-up for this long is sufficient.

4 Results

The first set of experiments is designed to illustrate the fundamental behavior of Windows Streaming Media (WSM). We stream single bitrate clips and examine their performance in detail. Next, we experiment with the effects of different content encoding parameters, specifically, the number and selection of bitrates contained in multiple bitrate clips. Finally, we examine the behavior of WSM over a range of loss and latency settings.

4.1 Single Bitrate Clips

Our first experiment was to examine the basic behavior of WSM. First, we establish the network characteristics for this experiment. The bottleneck capacity is constrained using the NIST Net router to that of a typical cable Internet connection, 725 Kbps, and the latency is set to a reasonable value that might be found on the Internet, 45 ms (as discussed in Section 3.4) and [LR01, Pax99, All00]. In these experiments no packet loss is induced. The WSM clip contains a single bitrate, 340 Kbps, roughly half of the fair share of the available capacity. We use iperf [NLA] to initiate a TCP transfer over the link which is allowed to "warm-up" for 10 seconds before a 60 second WSM clip is streamed concurrently. The TCP transfer is not meant to induce congestion on the link, but rather it serves a basis for comparison to the WSM flow.

The bitrate over time for both the TCP and WSM flows are shown in Figure 3. After the RTSP stream is initialized by the sending of the SETUP and DESCRIBE messages, as discussed in Section 2.2.1, there is a short period (about 5 seconds)

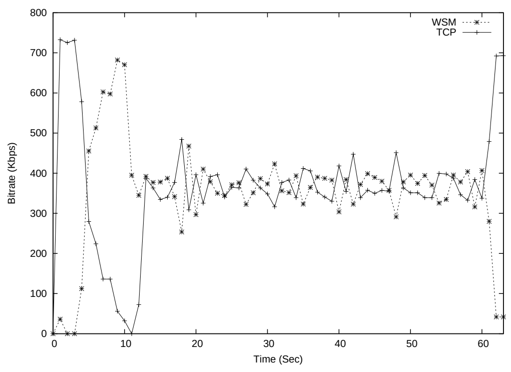

Figure 3: Bitrate versus Time for a 340 Kbps Clip.

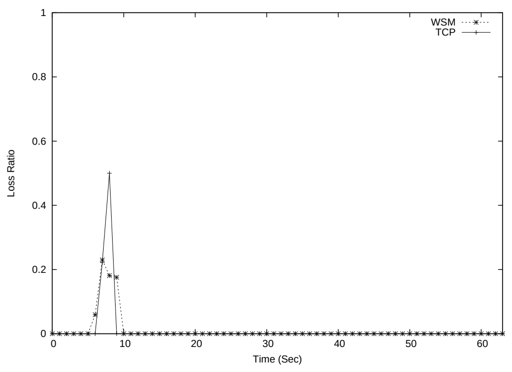

Figure 4: Loss Ratio versus Time for a 340 Kbps Clip.

where no data is sent. Then, the WSM clip streams at 500 to 600 Kbps for about 10 seconds while building up the initial content playout buffer. Finally, the WSM clip streams at roughly the fair-share bitrate of 340 Kbps. This result shows that WSM flows cannot be modeled as simple “fire-hose” CBR flows, as is done by many researchers. Instead, an accurate bitrate distribution characterizing streaming media should include a prominent buffering period, followed by a steady-state playout phase, with possibly markedly different bitrates. In later experiments, this result motivates separate examination of each of these phases.

The loss rate experienced by each of the flows is shown in Figure 4. Here, the loss rate experienced by both flows during the buffering period is very high, reaching almost 40%. Initially when we first saw these results we were concerned that our network setup was somehow influencing the results. A loss rate of 40% seemed much to high, we hypothesized that the queue length parameter of 80 packets at the NIST Net router might be causing such a high rate. To quantify the effects of the queue length parameter we devised the following experiment. With the same network configuration of the previous experiment we stream a 548 Kbps clip concurrently with a TCP flow.

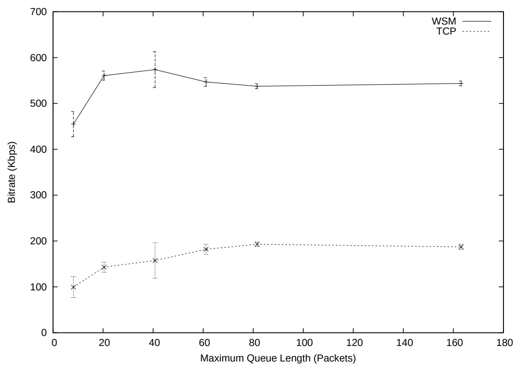

Figure 5: Average Bitrate During Buffering Period - 548 Kbps Clip.

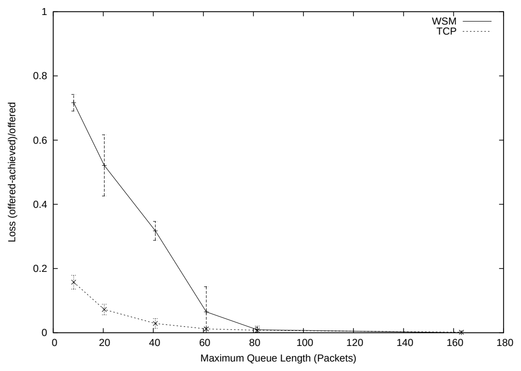

Figure 6: Aggregate Loss Ratio During Buffering Period - 548 Kbps Clip.

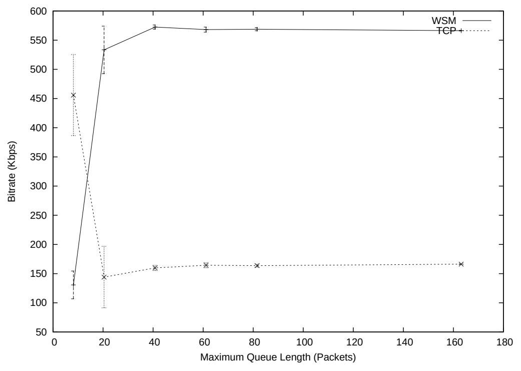

Figure 7: Average Bitrate During Post Buffering Period - 548 Kbps Clip.

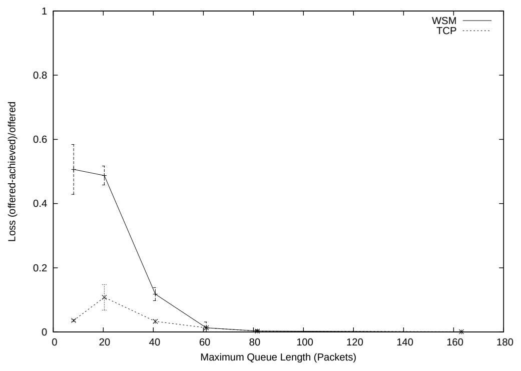

Figure 8: Aggregate Loss Ratio During Post Buffering Period - 548 Kbps Clip.

We quantify the buffering period using the process described in Section 3.2.3, allowing measurement of the average bitrate and loss rates during both the buffering and post-buffering period. We measure these quantities as we iteratively increase the queue length parameter to deterime any possible correlation between maximum queue length and flow performance.

The average bitrate and aggregate loss rate during the buffering period versus the maximum queue length parameter are shown in Figures 5 and 6, and during the post-buffering period in Figures 7 and 8. Each point represents the average of three runs and the bars show the standard deviation of these three samples. The results show

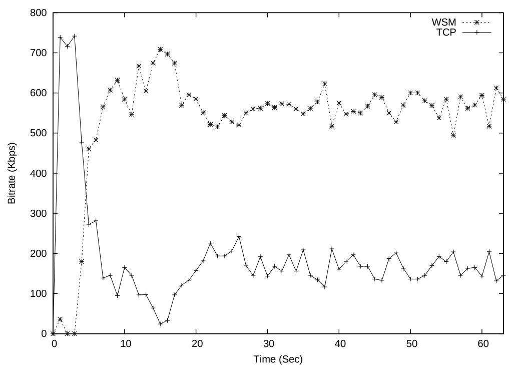

Figure 9: Bitrate versus Time for a 548 Kbps Clip.

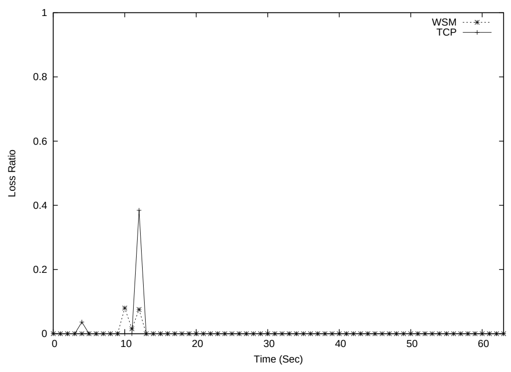

Figure 10: Loss Ratio versus Time for a 548 Kbps Clip.

that performance is similar for all queue lengths above 60 packets. Results remain the same for larger queue sizes while the variation of the samples is lower than for queue sizes less than 60 packets. This is an effect of the bursty nature of the WSM traffic which induces the loss at the router, which is somewhat random at lower queue sizes resulting in unpredictable behavior. Thus, our setting of 80 packets is appropriate for our experiments as it is beyond the "knee" of this queue size curve.

The results shown in Figure 3 suggest that after the buffering period, WSM is in fact TCP-friendly as it consumes roughly half of the available capacity. However, this is not facilitated by Intelligent Streaming or any protocol desicion, as described in Section 2.3. Instead, it is a result of the fact that the clip is not encoded at a higher bitrate. To illustrate this fact, we repeated the experiment but with a higher bitrate clip, 540 Kbps, higher then the fair-share of capacity. The streaming of a clip higher then the fair-share could occur if there was heavy congestion present in the network, the user made the wrong choice from the selection of bitrates available, or the content producer failed to encode and provide the video at a suitable bitrate.

The results with this 540 Kbps bitrate clip are shown in Figures 9 and 10. In

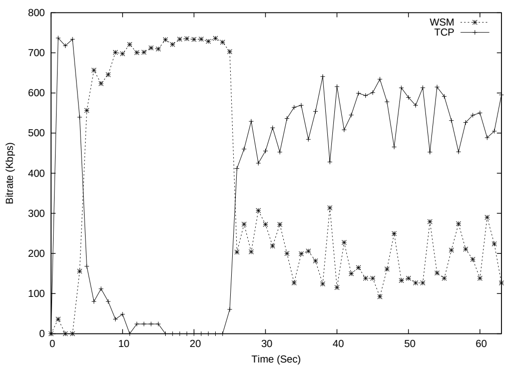

Figure 11: Bitrate versus Time for a 1128 Kbps Clip.

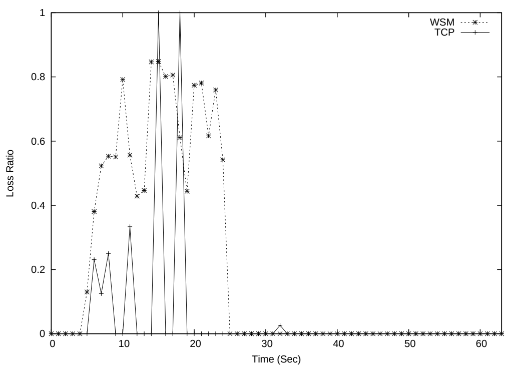

Figure 12: Loss Ratio versus Time for a 1128 Kbps Clip.

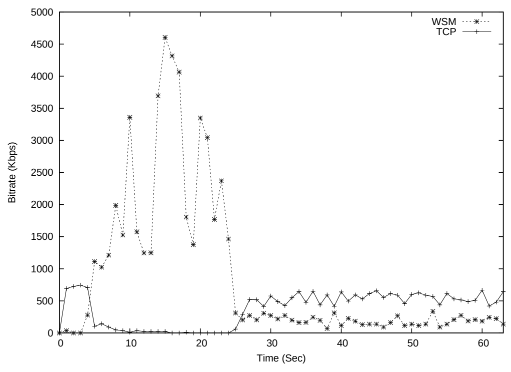

Figure 13: Bitrate versus Time for a 1128 Kbps Clip.

this experiment the streaming flow clearly consumes more than its fair share of the available capacity. In this case, the ideal streaming media server would have made use of the Intelligent Streaming feature to "thin" the stream and send fewer video frames to lower the bitrate in response to limited network resources. However, this is not the case since, the TCP flow does not receive its fair share of the available capacity. Furthermore, during the buffering period there are times when both flows experience heavy packet loss.

If we stream an even *higher* bitrate clip, say 1128 Kbps, Figures 11 and 12 show that there is some response by WSM to severe network congestion it imposes. Some time into the streaming of the 1128 Kbps clip the bitrate is changed to a much lower one, but the fair-share bitrate is undershot in this case. Examination of the offered load during this experiment, shown in Figure 13, reveals that WSM initially attempts to respond to the packet loss by offering more traffic, resulting in a bitrate much higher then the content encoded rate, and packet loss rates of over 80%!

Next, we seek to quantify the relationship of the response of WSM to network conditions in the case of single bitrate clips further. To achieve this 10 single bitrate clips of increasing bitrates are streamed one at a time over a network link with varying capacities. First, we set the link capacity to that of a typical low-speed broadband connection (250Kbps), then to a cable connection (725 Kbps), and finally a high-end DSL or T1 conneciton (1.5 Mbps). In all experiments there is no loss induced by the NIST Net router and the base network latency is fixed at 45 ms.

We report the results of these experiments from two perspectives: during the buffering period and after the buffering period. The results are shown in terms of bitrates and loss rates versus content encoded bitrate in Figures 14 - 19 for the buffering period, and in Figures 20 - 25 for the post-buffering period. Figures 26 - 31 show the lengths in seconds of the buffering and post-buffering periods. Note that the lengths of these periods are how long the traffic actually flowed, not the playback duration. These lengths may be different because when traffic stops flowing from the streaming server to the client the player has content stored in the buffer that it must playback. Thus, the playback duration may be slightly longer then the post-buffering period length.

The results in Figures 14 - 19 illustrate that during the buffering period WSM has little regard for the network. The buffering rate increases along with the content encoding rate until the encoding rate exceeds the bottleneck capacity. After this point the loss rate grows, almost reaching 80% in some cases. If the network has sufficient capacity, for example in Figure 16, the loss rate does not grow as large, but is still significant.

There is an interesting trend in Figure 17. For the single bitrate clip encoding as you go from left to right, the loss rate is increasing up until the content reaches the 548 Kbps encoding rate, where there is a "dip" in the measured loss rates, then it increases again. Clearly, the behavior of WSM during buffering changes between the 340 Kbps and 548 Kbps encoding rates. The achieved bitrates during buffering for the 340 Kbps clip and 548 Kbps clip are similar. This is not what we would expect, as the trend for lower bitrates indicates that the buffering rate increases with encoding rate. For the clips encoded at rates below 548 Kbps, WSM buffers them at 2 - 4 times the encoding rate. However, for the 548 Kbps clip, it appears to buffer it at the encoding rate. Perhaps, WSM has detected that the network cannot support a higher rate. We leave further exploration of this detection as future work.

Further evidence that there is different behavior during buffering when streaming at these two bitrates is that the packet burst rate distributions are different. Figures 32 and 33 show the CDF of interarrival times during and after buffering for a particular experiment when streaming the 340 Kbps and 548 Kbps clip. We measure these interarrival times before the traffic reaches the NIST Net router, so there are no effects of queueing delay on these distributions. From these figures, we see that the packet interarrivals during buffering of both clips is somewhat bursty, which is indicated by more than 90% being sent within a few milliseconds of each other. However, during buffering it appears that the packets comprising the 340 Kbps clip are sent about 0.5 ms closer together than for the 540 Kbps clip.

The CDF's shown in Figures 32 and 33 fail to convey the entire range of interarrival times. Therefore, we graph the CCDF of interarrival times, shown in Figures 34 and 35. These distributions show us approximately how far apart the bursts arrive. In Figure 35 we see that during post-buffering, the bursts of packets arrive about the same distance apart for both the 340 and 548 Kbps clip. However, in Figure 34 we see that the bursts of packets from the 340 Kbps clip arrive somewhat closer together during buffering, about 300 ms apart, while for the 548 Kbps clip the bursts of packets are slightly more spread out, about 475 ms apart. These results can be confirmed visually by looking at the arriving packet sequence numbers versus time.

We graph the RTP packet sequence number versus time for the 340 Kbps and 548 Kbps clip in Figures 36 and 37. Here we see that the bursts for the 548 Kbps clip are somewhat larger, but the time between them is larger than for the 340 Kbps clip. The “spikes” in Figure 36 are out-of-order received sequence numbers that are a result of retransmission of lost packets. The result of this traffic pattern for the 340 Kbps clip are losses at the NIST Net router because of the per-packet drop-tail queue. A possible explanation of this behavior is that at the lower bitrate WSM simply shortens the transmission time between packets bursts to buffer more quickly.

Thus, during the buffering period WSM is unresponsive to network conditions and congestion. However, the behavior of WSM is different durring the period after buffering. The results during the post-buffering period are shown in Figures 14 - 19.

Here we see that there is some response of WSM to limited network resources. In particular, in the case of a low-speed broadband connection, 250 Kbps, shown in Figures 14 and 15, reveal that the bitrate of WSM during the post-buffering period increases linearly until the encoded bitrate reaches the bottleneck capacity. After this point, as the content bitrate increases the loss-rate also increases to almost 40%, but the content is eventually thinned and uses less than its fair share of the capacity. In Figure 14 the data points for WSM and TCP show a different relationship than the preceeding bitrates. The explanation for this phenomenon is that the thinning of the stream does not take place directly after the buffering period. Thus, in the case of the highest encoded bitrate clip, the average over the entire post-buffering period is correspondingly higher, since the clip encoding is so much higher then the other clips. It is at this rate that WSM attempts to stream the clip before thinning.

Thus, measurement of a simple average over the entire post-buffering period misses some of the behavior of the WSM flow, particularly *when* this thinning occurs. Earlier, in Figures 9 and 10 we looked at the behavior of a 540 Kbps bitrate clip when the bottleneck capacity was 725 Kbps. Figures 38, 39, and 40 show the bitrate, loss rates, and offered load when the same clip is streamed over a 250 Kbps link. Here we see that the clip streams for almost 40 seconds at the 548 Kbps bitrate, inducing massive amounts of packet loss (over 80%), but eventually the stream is thinned, only requiring about 50 Kbps of capacity. The graph of the offered load shows that WSM initially responds to the packet loss by resending packets, resulting in even more offered traffic. These graphs tell us two things. First, by looking at an average of bitrate over time, WSM may be appear to be TCP-friendly, but there are periods where it is very unfriendly followed by periods where WSM is very friendly. Second, the bitrate of the WSM flow must be examined along with the corresponding loss rate over the same period to seek insight into the true behavior of WSM. Thus, in the case of Figures 14 and 15, we found that the while the bitrate was low and "TCP-friendly" there were high amounts of packet loss, meaning that there were times when WSM was very "un-friendly."

For a higher bottleneck capacity, such as 725 Kbps, Figures 16 and 17 show the behavior is similar to the lower capacity case that is discussed above. We find that the bitrate during the post-buffering period increases linearly as the encoded bitrate increases until the bottleneck capacity is exceeded, after which the loss rate increases and eventually the content is thinned in response to the network congestion. When the link capacity is even larger, 1.5 Mbps, shown in Figures 18 and 19, there are not any clips in the experiments that exceed the bottleneck capacity. Thus, the post-buffering rate increases linearly with the content encoding rate.

Figures 26 - 31 show the the length in seconds of the buffering and post-buffering periods. In all cases, the length of the buffering period increases approximately linearly with the content encoding rate. Due to thinning, the length of the post-buffering period decreases when the content encoding rate exceeds the bottleneck capacity. When the encoding rate is less than capacity, the post-buffering period length is about the same, regardless of encoding rate.

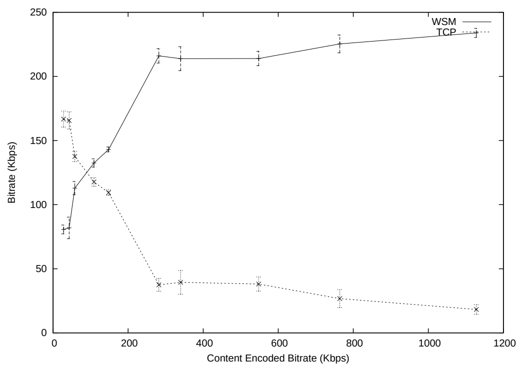

Figure 14: Average Bitrate During Buffering Period - 250 Kbps Bottleneck Capacity.

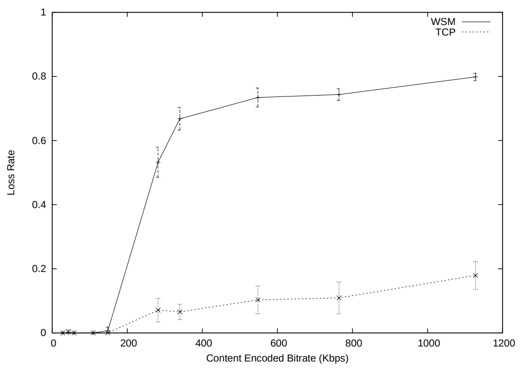

Figure 15: Aggregate Loss Ratio During Buffering Period - 250 Kbps Bottleneck Capacity.

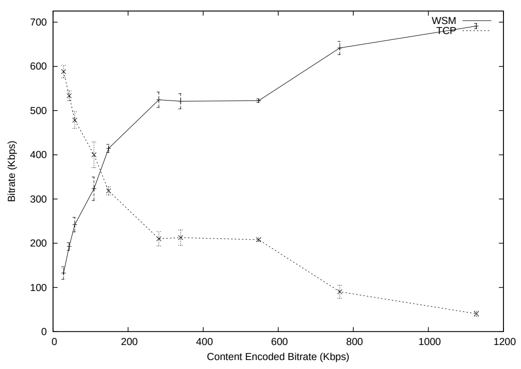

Figure 16: Average Bitrate During Buffering Period - 725 Kbps Bottleneck Capacity.

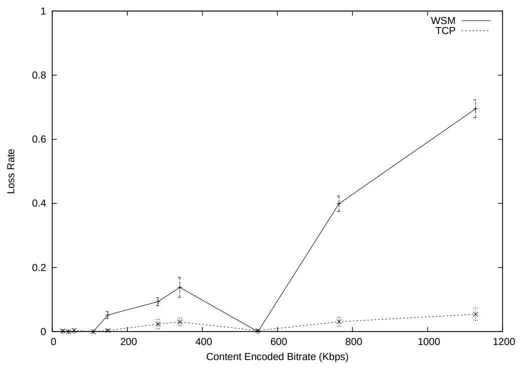

Figure 17: Aggregate Loss Ratio During Buffering Period - 725 Kbps Bottleneck Capacity.

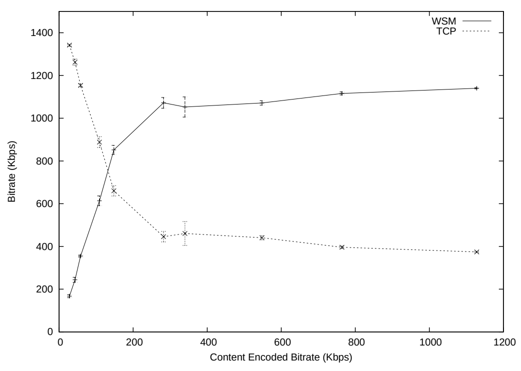

Figure 18: Average Bitrate During Buffering Period - 1.5 Mbps Bottleneck Capacity.

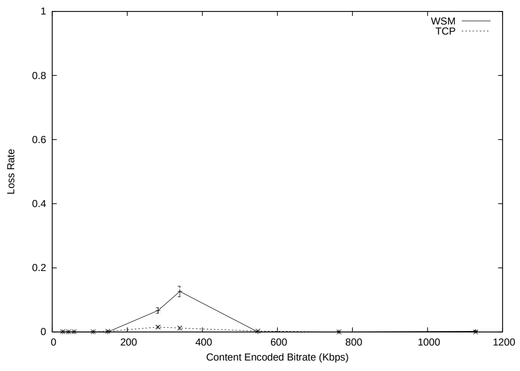

Figure 19: Aggregate Loss Ratio During Buffering Period - 1.5 Mbps Bottleneck Capacity.

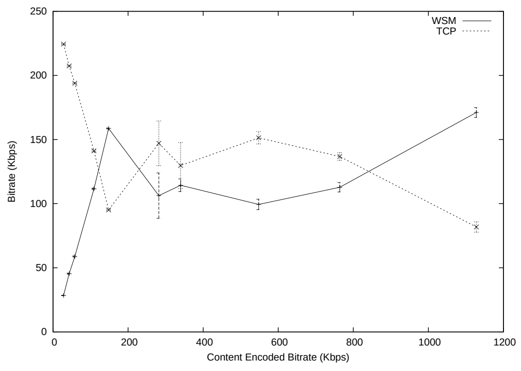

Figure 20: Average Bitrate After Buffering Period - 250 Kbps Bottleneck Capactity.

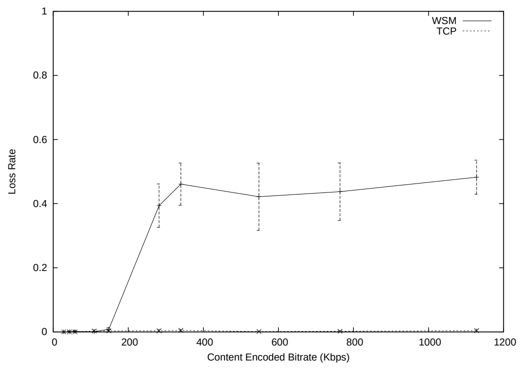

Figure 21: Aggregate Loss Ratio After Buffering Period - 250 Kbps Bottleneck Capactity.

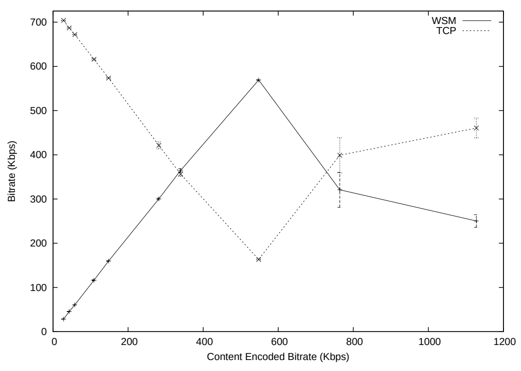

Figure 22: Average Bitrate After Buffering Period - 725 Kbps Bottleneck Capactiy.

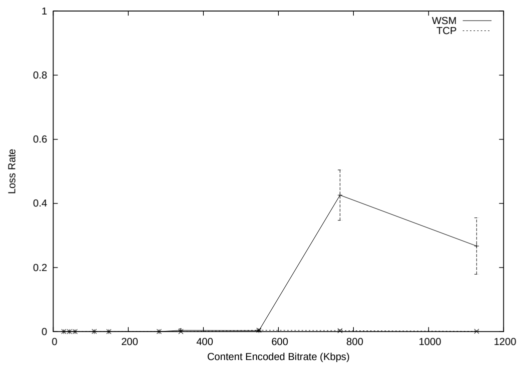

Figure 23: Aggregate Loss Ratio After Buffering Period - 725 Kbps Bottleneck Capacity.

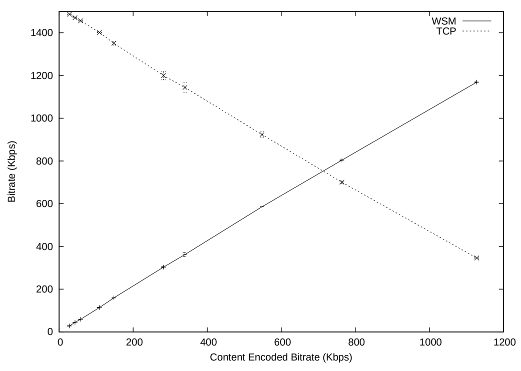

Figure 24: Average Bitrate After Buffering Period - 1.5 Mbps Bottleneck Capacity.

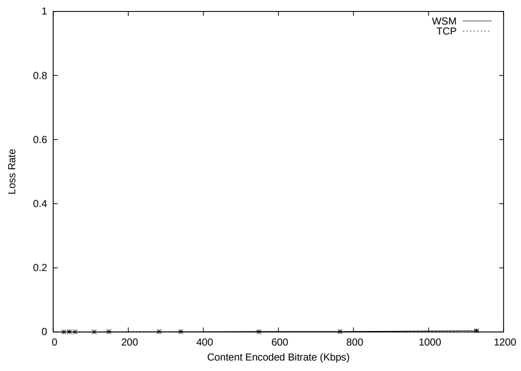

Figure 25: Aggregate Loss Ratio After Buffering Period - 1.5 Mbps Bottleneck Capacity.

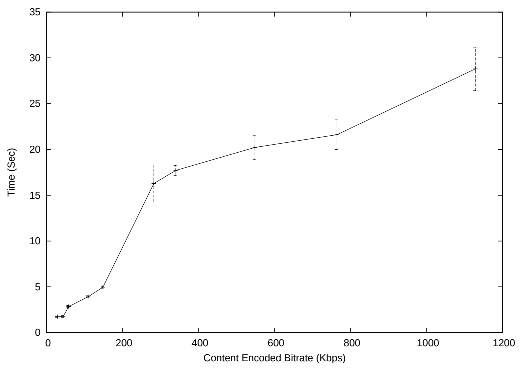

Figure 26: Length of Buffering Period - 250 Kbps Bottleneck Capacity.

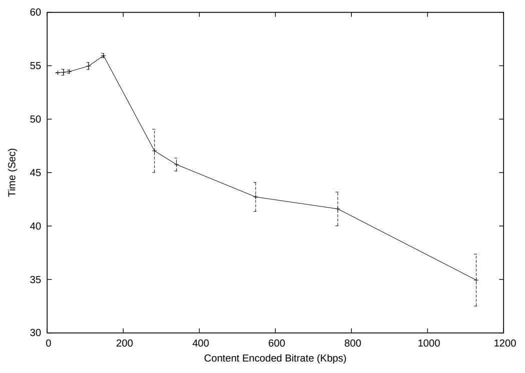

Figure 27: Length of Post Buffering Period - 250 Kbps Bottleneck Capacity.

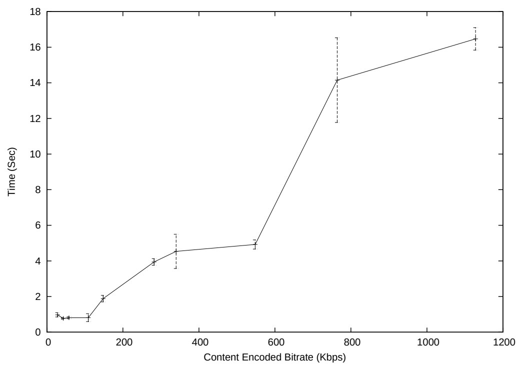

Figure 28: Length of Buffering Period - 725 Kbps Bottleneck Capacity.

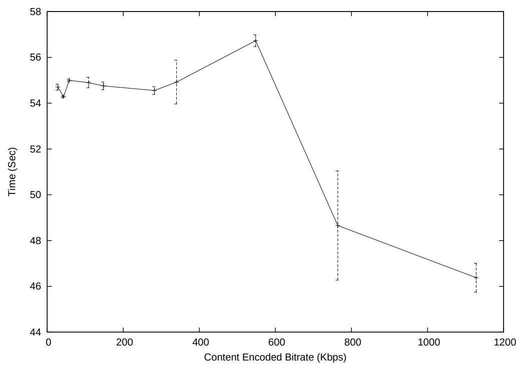

Figure 29: Length of Post Buffering Period - 725 Kbps Bottleneck Capacity.

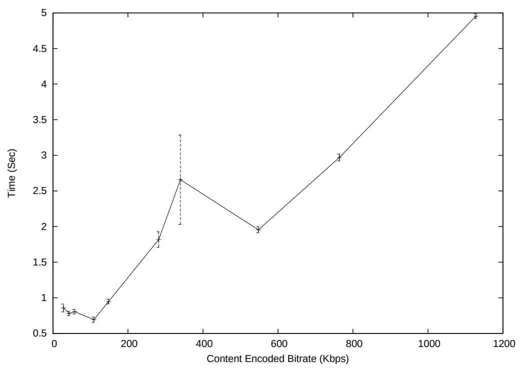

Figure 30: Length of Buffering Period - 1.5 Mbps Bottleneck Capacity.

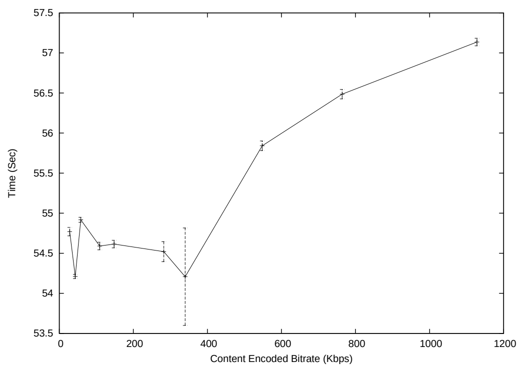

Figure 31: Length of Post Buffering Period - 1.5 Mbps Bottleneck Capacity.

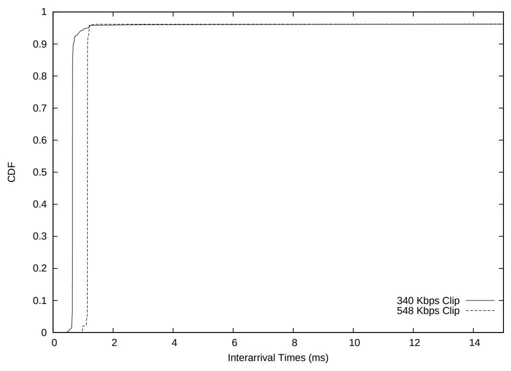

Figure 32: CDF of Interarrival Times During Buffering.

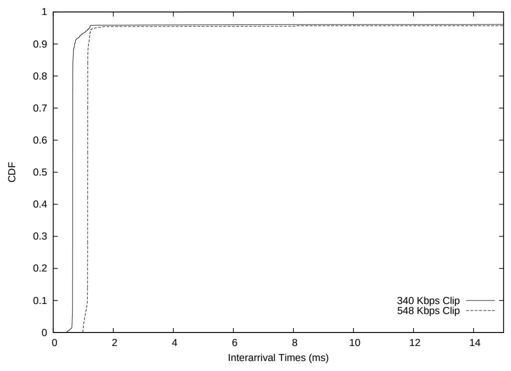

Figure 33: CDF of Interarrival Times After Buffering.

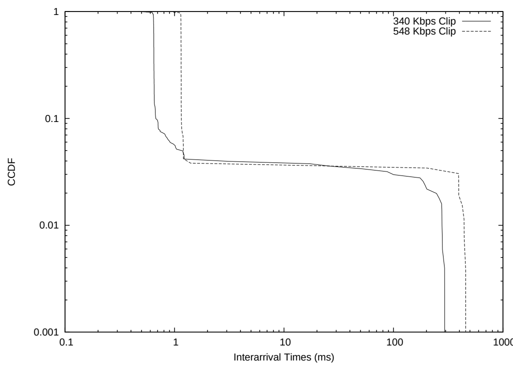

Figure 34: CCDF of Interarrival Times During Buffering.

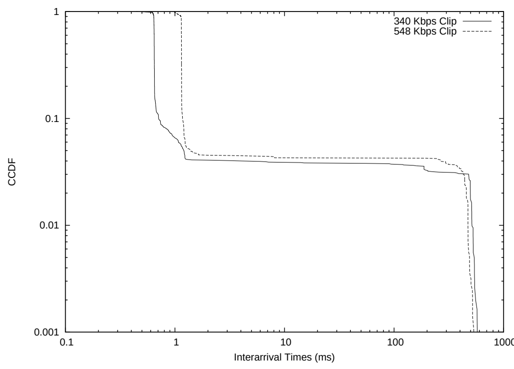

Figure 35: CCDF of Interarrival Times After Buffering.

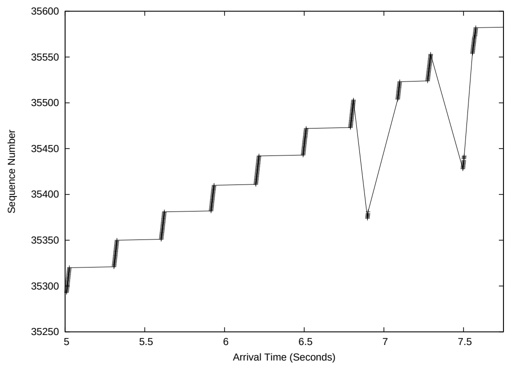

Figure 36: Packet Sequence Number Versus Time for 340 Kbps Clip.

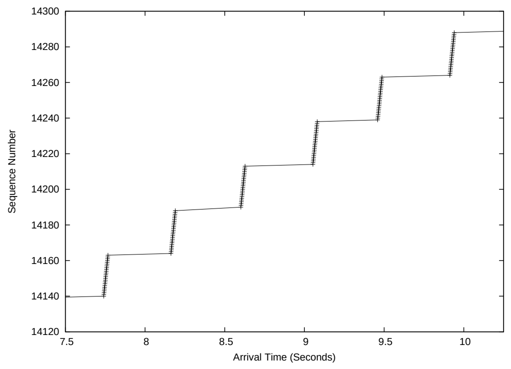

Figure 37: Packet Sequence Number Versus Time for 548 Kbps Clip.

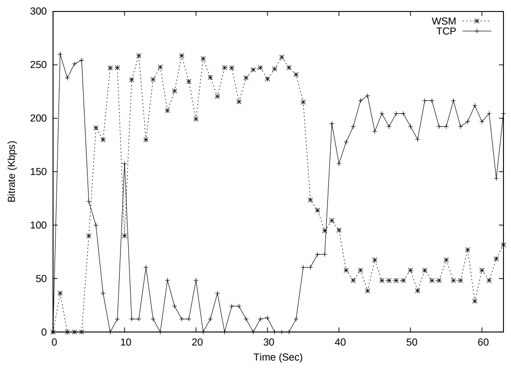

Figure 38: Bitrate versus Time for a 548 Kbps Clip.

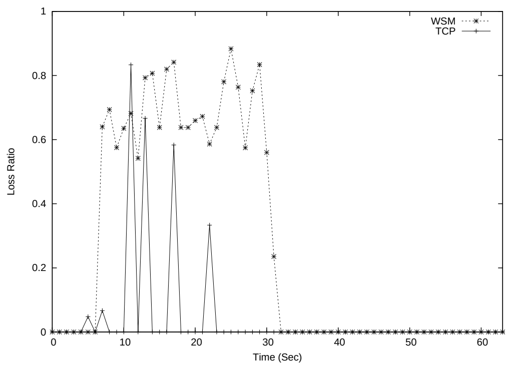

Figure 39: Loss Ratio versus Time for a 548 Kbps Clip.

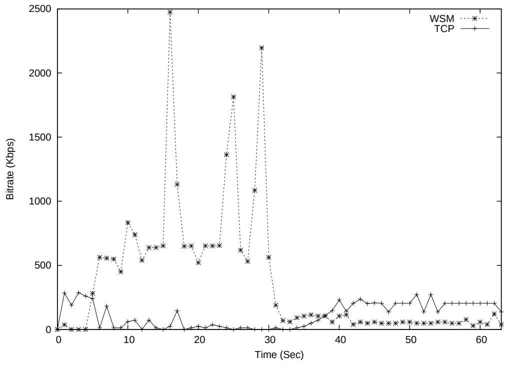

Figure 40: Bitrate versus Time for a 548 Kbps Clip.

4.2 Multiple Bitrate Clips

We ran our next set of experiments to determine the effects of multiple encoded bitrates, or scaling levels contained in the clip. The documentation related to Intelligent Streaming [Bir00] and prior work [CCZ03] suggests that the responsiveness of streaming media may be coupled with the number of encoded bitrates contained in the content. First, we ran an experiment with the bottleneck capacity set to 725 Kbps, the latency to 45 ms, and no induced loss. We stream two sets of multiple-bitrate clips with one set containing clips with 3 encoding levels and the other with only 2.

The first set of clips contains content encoded with 3 scaling levels. To choose the bitrates we started with the three highest bitrates, (1128,764,548) Kbps, then we removed the highest and added the next lowest, thus the remaining three are: (764,548,340) Kbps, (548,340,282) Kbps, (340,282,148) Kbps. Figures 41 - 44 show the bitrates and loss rates for both the buffering and post-buffering periods versus the content encodings. These results show that during the buffering period, if the clip contains the 548 Kbps bitrate it is the one chosen by WSM to stream. If the 1128 or 764 Kbps bitrate is present and was chosen then we would expect the bitrate during the buffering period to be higher. The rates for the 548 and 340 Kbps clip during buffering are similar, which explains the similarity in when the 548 Kbps bitrate is not present in the clip. During the post-buffering period, the 548 Kbps rate is always chosen if present. This result makes sense because according to Figure 22 WSM thinks that this is a suitable bitrate for the network conditions as it does not attempt to thin it in the previoius experiment. If the 540 Kbps bitrate is not encoded in the clip, the next highest bitrate, 340 Kbps, is chosen.

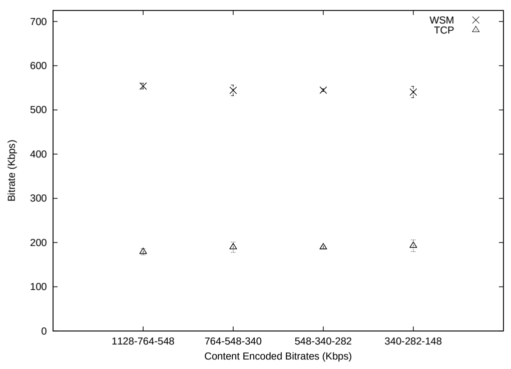

Figure 41: Average Bitrate During Buffering Period - Clips containing 3 bitrates.

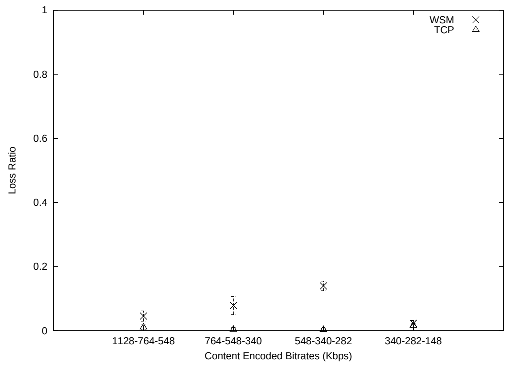

Figure 42: Aggregate Loss Ratio During Buffering Period - Clips containing 3 bitrates.

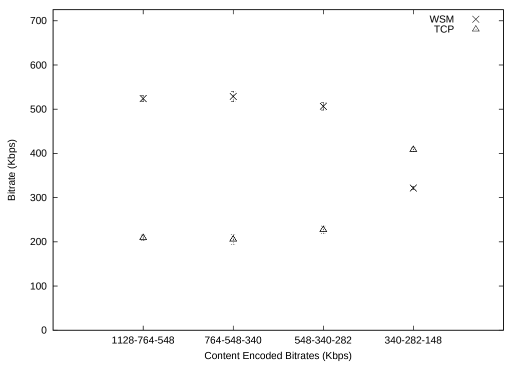

Figure 43: Average Bitrate After Buffering Period - Clips containing 3 bitrates.

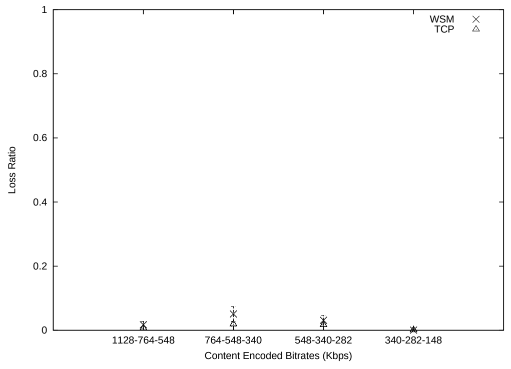

Figure 44: Aggregate Loss Ratio After Buffering Period - Clips containing 3 bitrates.

We exercised more creativity in the selection of bitrates to encode into the content used in the next set of experiments. We could not run an experiment for every possible combination of bitrates so we had to pick only a few. Thus, we chose two bitrates for each clip, one rate that was above the bottleneck capacity and the other below it. This was to determine if WSM would choose the higher bitrate and thin it, or just stream the lower bitrate. Figures 45 - 48 show the results. We see that in all cases WSM chooses the lower bitrate. This is confirmed by comparing the measured bitrates with those of the single bitrate clips encoded at the same rate, which correlate well. After the buffering period, Figure 47 shows that the achieved bitrate of the WSM flow is slightly higher then that of the single bitrate clip. For example, the clip with the 1128 and 340 Kbps encoding rates achieves a bitrate of approximately 425 Kbps, while as shown in Figure 22, a single bitrate clip encoded at 340 Kbps achieves about 340 Kbps. Examination of the Windows Media Encoder settings revealed that the audio portion of the stream at the 1128 Kbps bitrate was 128 Kbps while at the 340 Kbps rate the audio bitrate was 40 Kbps. Thus, WSM chooses the audio stream belonging to the 1128 Kbps rate, but the video stream belonging to the 340 Kbps rate, for a total bitrate of 428 Kbps.

Once we had investigated some simple cases of multiple bitrate clips we sought to thoroughly expolore the relationship between the number of bitrates contained in the content and responsiveness. To achieve this we created two sets of multiple bitrate clips. In the first we started with a clip that contained the highest bitrate (1128 Kbps). Next, we added the next highest bitrate to create a clip with two bitrates (1128, 764 Kbps). We iterated this process to produce a total of 10 clips, where each succesive clip had the next highest bitrate not in the clip. Thus, the last clip contained all 10 bitrates (1128, 764, 548, 340, 282, 148, 108, 58, 43, 28 Kbps). The other set of clips were created in a similar manner, except we began with the lowest bitrate and iteratively added the next lowest bitrate.

The network latency was set at 45 ms each way and there was no induced loss in all of the experiments. We varied the bottleneck capacity between 250 Kbps, 725 Kbps, and 1500 Kbps. Again, we report our results in relation to the buffering and

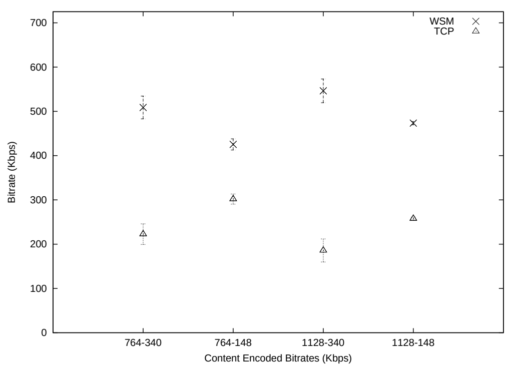

Figure 45: Average Bitrate During Buffering Period - Clips containing 2 bitrates.

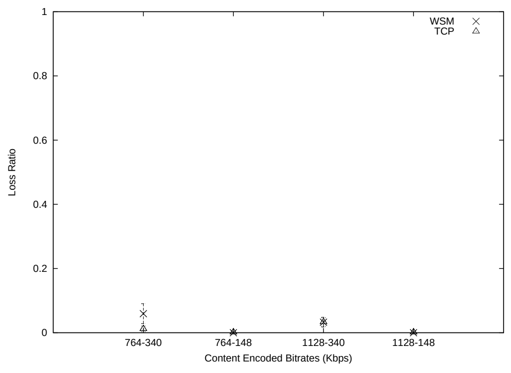

Figure 46: Aggregate Loss Ratio During Buffering Period - Clips containing 2 bitrates.

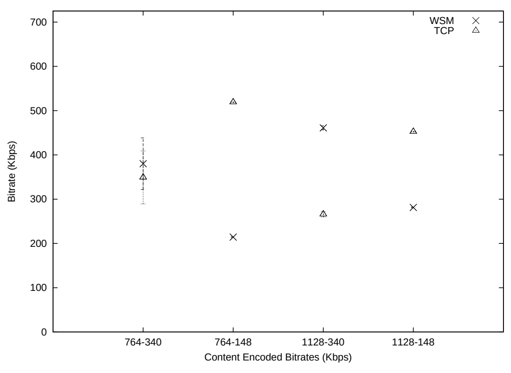

Figure 47: Average Bitrate After Buffering Period - Clips containing 2 bitrates.

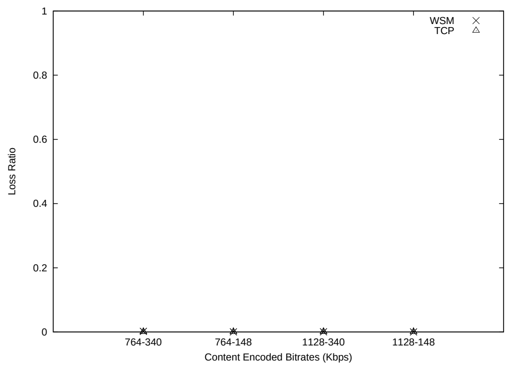

Figure 48: Aggregate Loss Ratio After Buffering Period - Clips containing 2 bitrates.

post-buffering periods. For the set of clips with the decreasing lowest encoded rate, Figures 49 - 54 show the results for the buffering period, and Figures 55 - 60 show the results for the post-buffering period. In the experiments with a low bottleneck capacity, WSM chooses a bitrate that is lower then the capacity if it is available, otherwise it chooses the lowest one available. This is shown by the decrease in loss rate as the clip contains increasing lower bitrates.

Since this decision to send a lower available bitrate is made before streaming is initiated, RTCP reports are not used to convery network conditions back to the server. We looked at the packet traces during these experiments and found that after the intial RTSP SETUP message was sent, a SET_PARAMETER message was sent from the client to the server with the type field specifying "high-entropy-packetpair." Then, a few hundred milliseconds later, three RTP packets were sent very closely together. Finally, a few seconds later the actual video and audio streams were initialized with SETUP messages. We conclude that WSM is using two packet-pair estimates to determine available network capacity. We examined all of the packet traces and found that these packet-pairs are only sent out just prior to streaming, not during steady state playout, thus WSM uses RTCP reports during the actual session. Examination of the packet traces reveal that these RTCP reports are normally sent infrequently. However, during periods of packet loss the reports are very frequent, sometimes more than ten per second. Perhaps this is the mechanism the client uses to request retransmission of lost packets.

Similar results are seen for the other bottleneck capacities. In the 725 Kbps bottleneck capacity case WSM chooses the 548 Kbps rate, even if a lower bitrate is included in the clip, which is in accord with the results of previous experiments. Figure 53 shows interesting behavior when the lowest bitrates are added into the content, the sending rate increases as they are added and the corresponding loss rate is higher. Examination of the packet traces reveals no insights, but perhaps WSM sends additional data in the form of an extra bitrate to facilitate faster changes to a lower bitrate if appropriate. After buffering WSM again receives slightly more capacity for a particular encoding rate then the corresponsing single bitrate clip. For example, in Figure 57 WSM chooses the 548 Kbps encoding rate when it is contained in the clip and receives slightly more than 600 Kbps of capacity. The 548 Kbps bitrate contains a 500 Kbps video stream and a 48 Kbps audio stream. WSM chooses the 128 Kbps audio stream contained in the 1128 Kbps clip and the 500 Kbps video stream from the 548 Kbps clip for a total of about 628 Kbps.

For the set of clips with the increasing highest encoded rate, Figures 61 - 66 show the results for the buffering period, and Figures 67 - 72 show the results for the post-buffering period. The results for these experiments are similar to those for the other set of MBR clips. During the buffering period, WSM chooses the largest bitrate that will fit the bottleneck capacity, thus ignoring other competing traffic. In the post-buffering period WSM chooses the bitrate that was the last before thinning was performed in the single bitrate clip experiments. For example, at the 750 Kbps bottleneck capacity, shown in Figure 69, once the clip contains the 548 Kbps bitrate it is always chosen, otherwise it selects the next highest available. Unfortunately, as we saw earlier, when streaming the 548 Kbps clip, WSM recieves an unfair share of capacity.

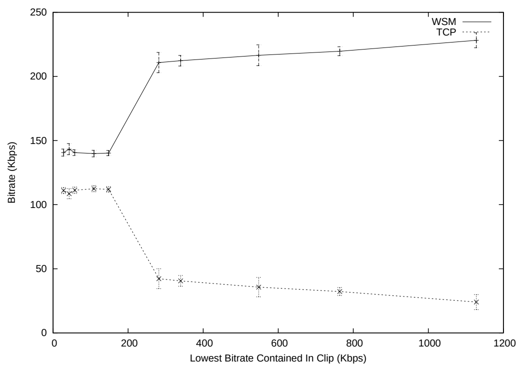

Figure 49: Average Bitrate During Buffering Period - 250 Kbps Bottleneck Capacity.

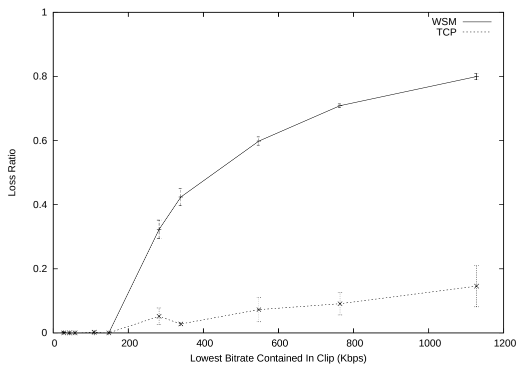

Figure 50: Aggregate Loss Ratio During Buffering Period - 250 Kbps Bottleneck Capacity.

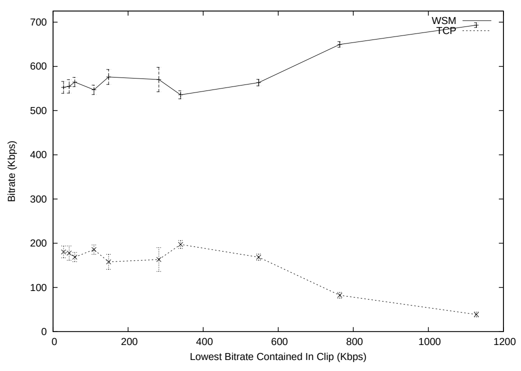

Figure 51: Average Bitrate During Buffering Period - 725 Kbps Bottleneck Capacity.

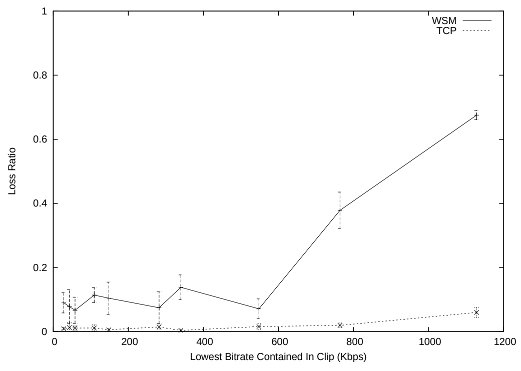

Figure 52: Aggregate Loss Ratio During Buffering Period - 725 Kbps Bottleneck Capacity.

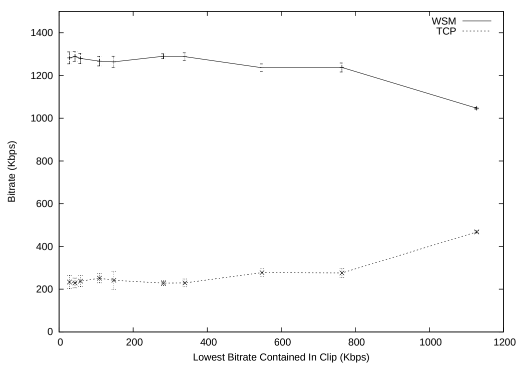

Figure 53: Average Bitrate During Buffering Period - 1.5 Mbps Bottleneck Capacity.

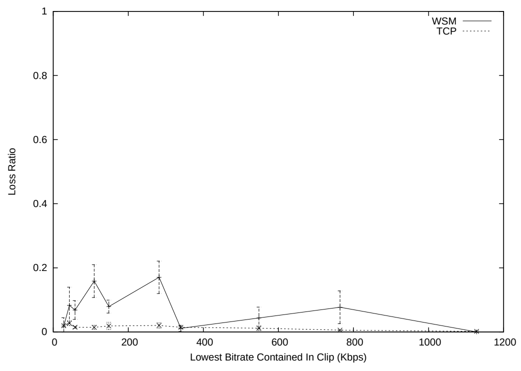

Figure 54: Aggregate Loss Ratio During Buffering Period - 1.5 Mbps Bottleneck Capacity.

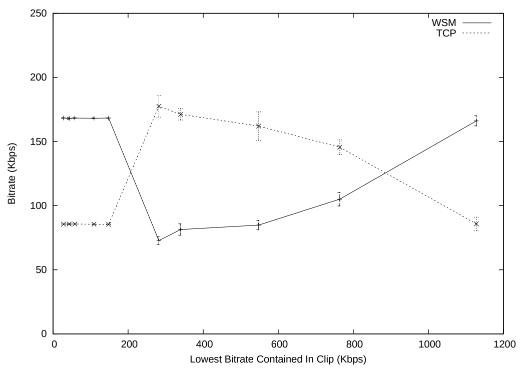

Figure 55: Average Bitrate After Buffering Period - 250 Kbps Bottleneck Capacity.

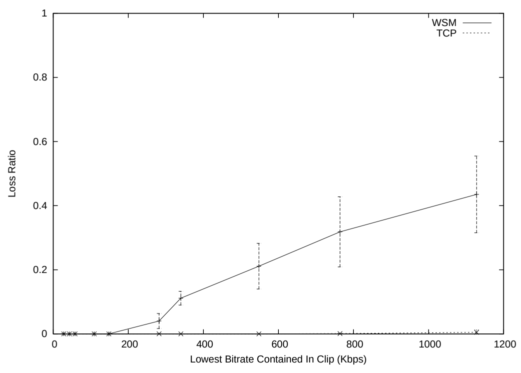

Figure 56: Aggregate Loss Ratio After Buffering Period - 250 Kbps Bottleneck Capacity.

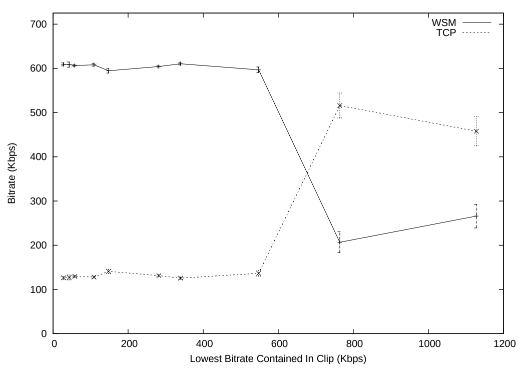

Figure 57: Average Bitrate After Buffering Period - 725 Kbps Bottleneck Capacity.

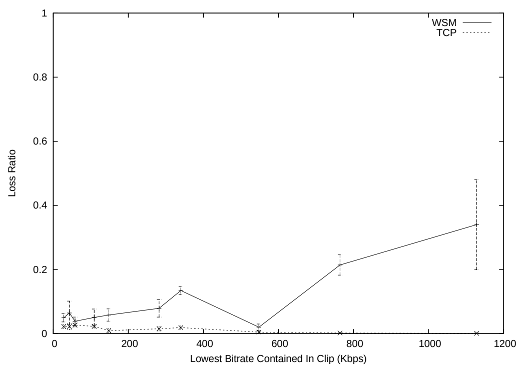

Figure 58: Aggregate Loss Ratio After Buffering Period - 725 Kbps Bottleneck Capacity.

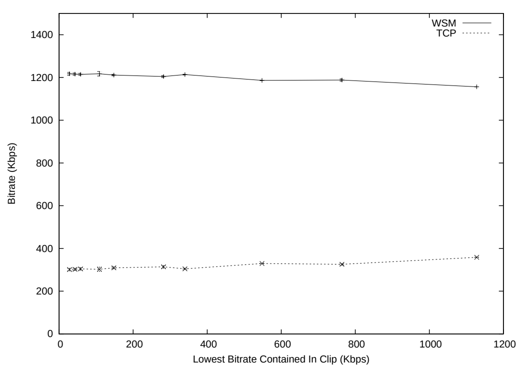

Figure 59: Average Bitrate After Buffering Period - 1.5 Mbps Bottleneck Capacity.

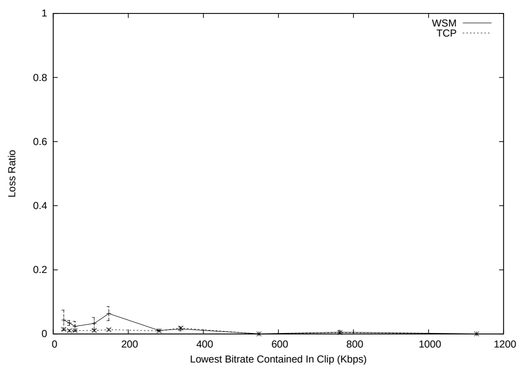

Figure 60: Aggregate Loss Ratio After Buffering Period - 1.5 Mbps Bottleneck Capacity.

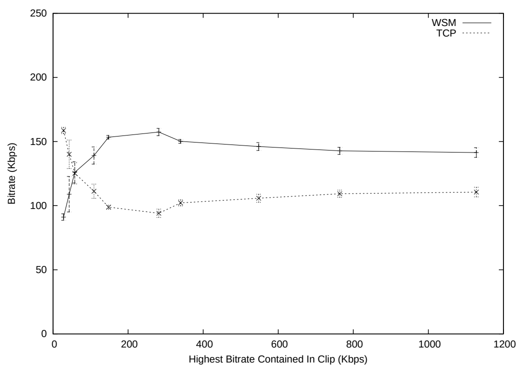

Figure 61: Average Bitrate During Buffering Period - 250 Kbps Bottleneck Capacity.

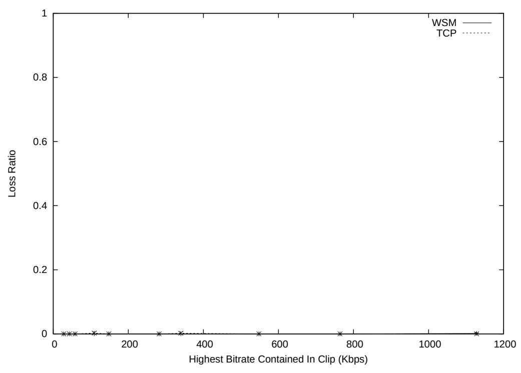

Figure 62: Aggregate Loss Ratio During Buffering Period - 250 Kbps Bottleneck Capacity.

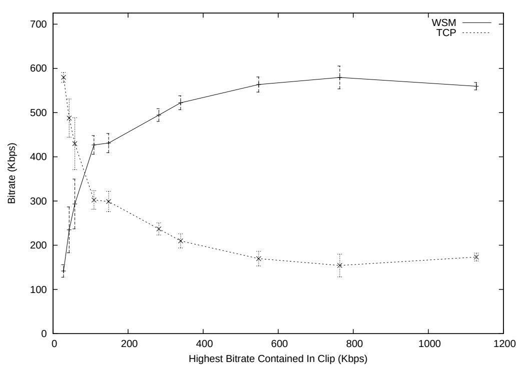

Figure 63: Average Bitrate During Buffering Period - 725 Kbps Bottleneck Capacity.

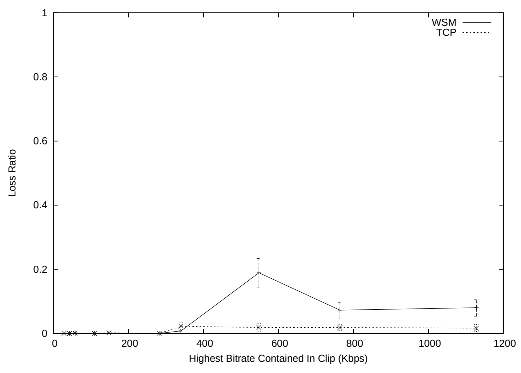

Figure 64: Aggregate Loss Ratio During Buffering Period - 725 Kbps Bottleneck Capacity.

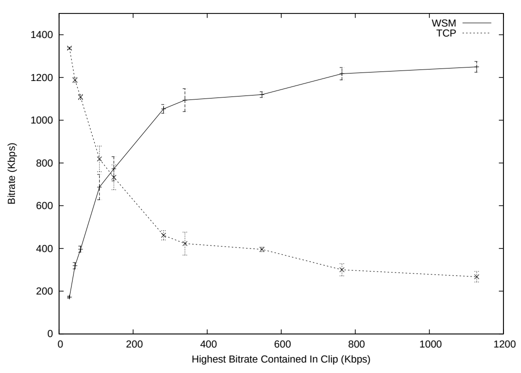

Figure 65: Average Bitrate During Buffering Period - 1.5 Mbps Bottleneck Capacity.

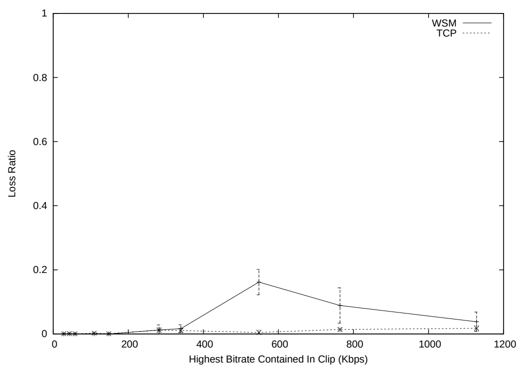

Figure 66: Aggregate Loss Ratio During Buffering Period - 1.5 Mbps Bottleneck Capacity.

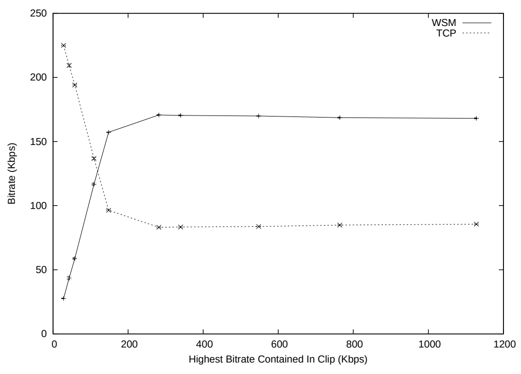

Figure 67: Average Bitrate After Buffering Period - 250 Kbps Bottleneck Capacity.

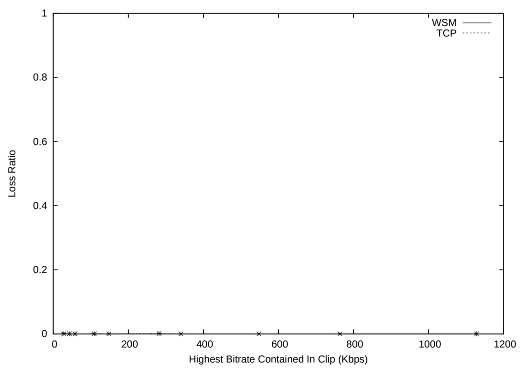

Figure 68: Aggregate Loss Ratio After Buffering Period - 250 Kbps Bottleneck Capacity.

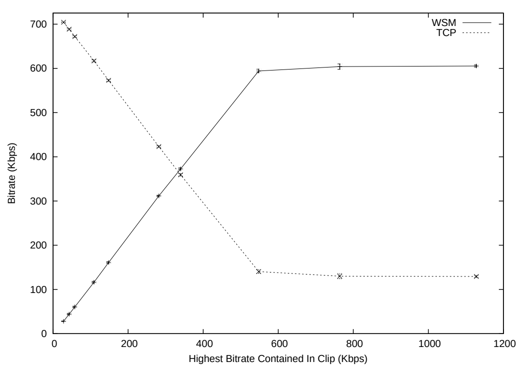

Figure 69: Average Bitrate After Buffering Period - 725 Kbps Bottleneck Capacity.

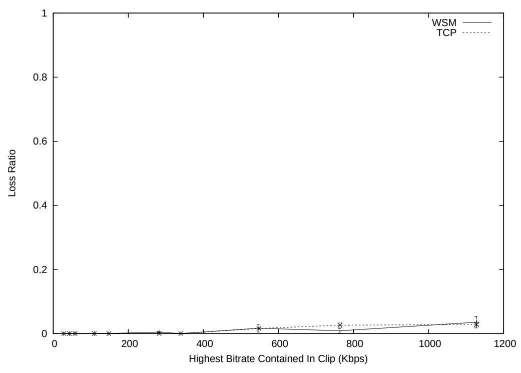

Figure 70: Aggregate Loss Ratio After Buffering Period - 725 Kbps Bottleneck Capacity.

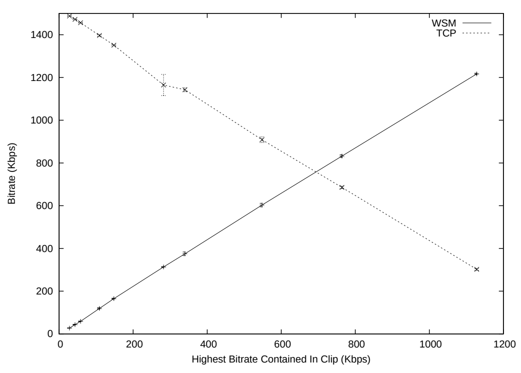

Figure 71: Average Bitrate After Buffering Period - 1.5 Mbps Bottleneck Capacity.

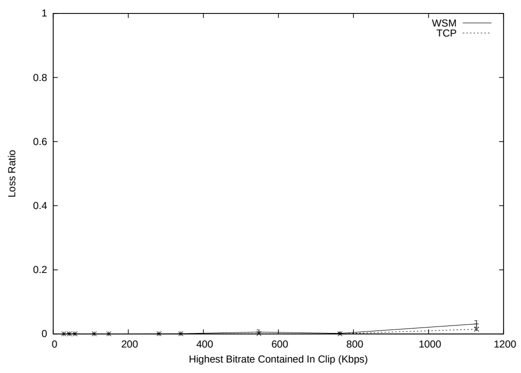

Figure 72: Aggregate Loss Ratio After Buffering Period - 1.5 Mbps Bottleneck Capacity.

4.3 Induced Loss

In these experiment we use the NIST Net router to induce loss on the network link, emulating the effect of congestion in the network. We fix the bottleneck capacity at 725 Kbps and latency at 45 ms. First, we stream the 548 Kbps clip because it is the rate just before WSM will begin to thin a single bitrate clip when the capacity is 725 Kbps. We vary the loss from 0% to 20% which includes the range of loss rates seen on the Internet [LR01, Pax99, All00]. Figures 73 - 76 show the results. We see that as the loss rate increases, the TCP flow backs off, while the WSM flow uses the additional available capacity during buffering and increases its sending rate to compensate for the high loss rate. The packet loss rates during buffering are very high because they are the cummulative effect of WSM's self-induced drops in addition to the loss rate that NIST Net is imposing. For the post-buffering stage we see that loss rates between 3% and 5% cause WSM to thin the stream in response to the higher loss rate. After 5% additional losses do not effect the behavior of WSM.

Next, we stream a multiple bitrate clip, containing the following bitrates: 548, 340, 282, 148, 106, 58, 43, and 28 Kbps. We include these lower bitrates so that instead of thinning, perhaps WSM can choose one of these lower bitrates when the loss rate increases. Again, we vary the loss from 0% to 20%. Figures 77 - 80 show the results. The bitrate during the buffering stage is the same as in the prior experiment. However, during the post-buffering period WSM chooses a lower bitrate for loss rates greater then 5% instead of thinning the 540 Kbps rate. We see that for a lower loss rate of 3% WSM sometimes keeps streaming the higher bitrate, but other times

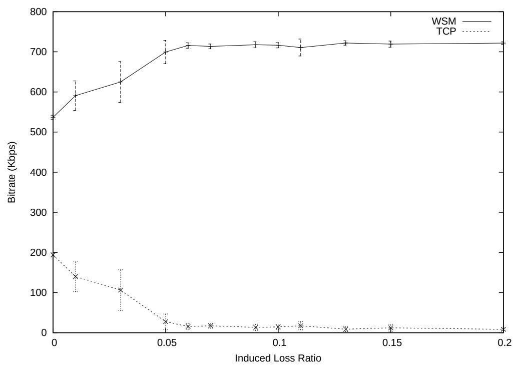

Figure 73: Average Bitrate During Buffering Period - Single Bitrate Clip - Induced Loss.

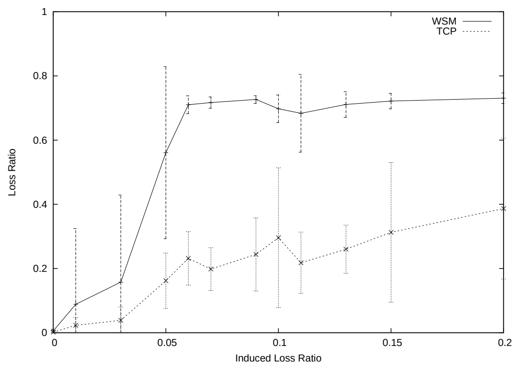

Figure 74: Aggregate Loss Ratio During Buffering Period - Single Bitrate Clip - Induced Loss.

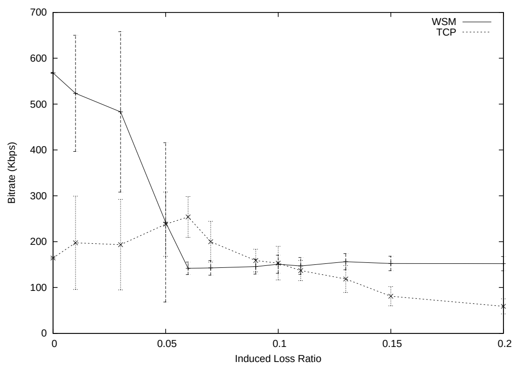

Figure 75: Average Bitrate After Buffering Period - Single Bitrate Clip - Induced Loss.

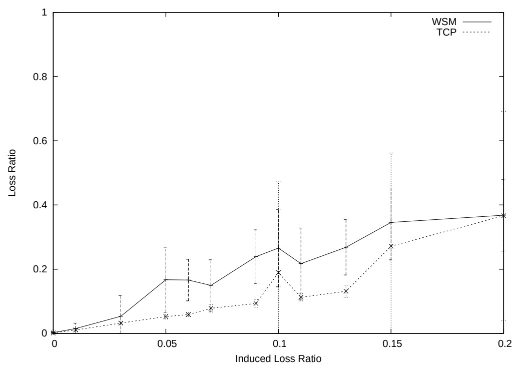

Figure 76: Aggregate Loss Ratio After Buffering Period - Single Bitrate Clip - Induced Loss.

chooses to send the lower rate.

For the final set of induced loss experiments we wanted to examine the behavior of WSM if the loss rate suddenly changes during the playout of the clip. We ran a similiar experiment as before by streaming a 548 Kbps clip. However, instead of setting the loss rate at the begining of the experiment, we waited 15 seconds after the start of streaming before setting the NIST Net router to induce loss. Figures 81 - 82 show the bitrates during and after buffering. We see that WSM does not react to the sudden increase in loss rate.

If we induce loss during buffering and for a short time into the playback of the clip and then stop inducing loss we see that WSM also does not respond to this decrease in loss rate. The results are shown in Figures 83 - 84.

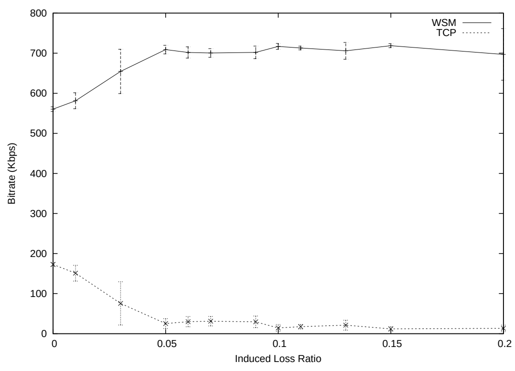

Figure 77: Average Bitrate During Buffering Period - Multiple Bitrate Clip - Induced Loss.

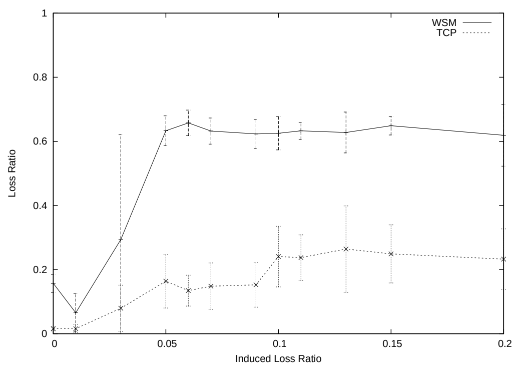

Figure 78: Aggregate Loss Ratio During Buffering Period - Multiple Bitrate Clip - Induced Loss.

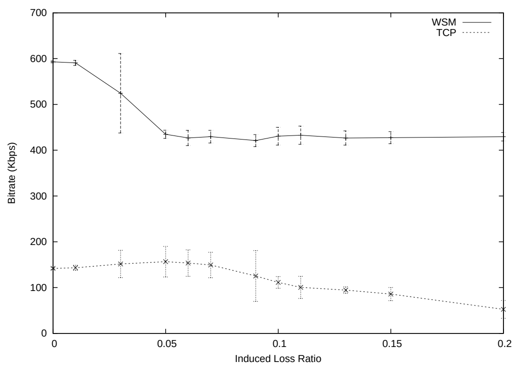

Figure 79: Average Bitrate After Buffering Period - Multiple Bitrate Clip - Induced Loss.

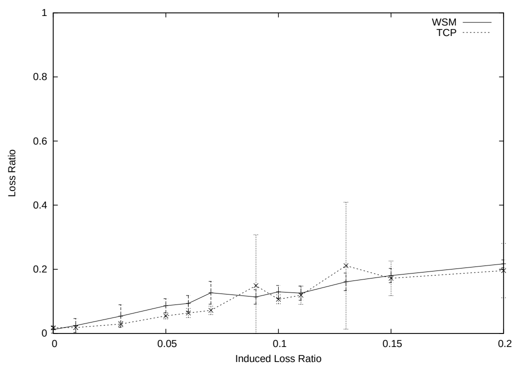

Figure 80: Aggregate Loss Ratio After Buffering Period - Multiple Bitrate Clip - Induced Loss.

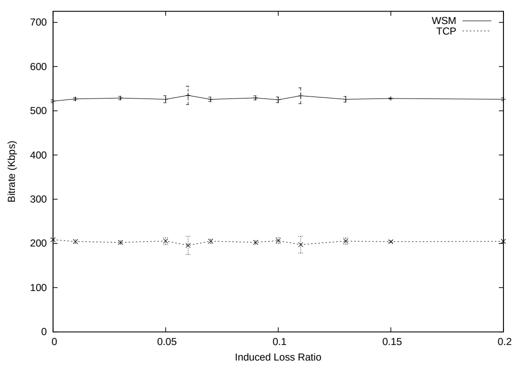

Figure 81: Average Bitrate During Buffering Period Single Bitrate Clip - Induced Loss Started At 15 Seconds.

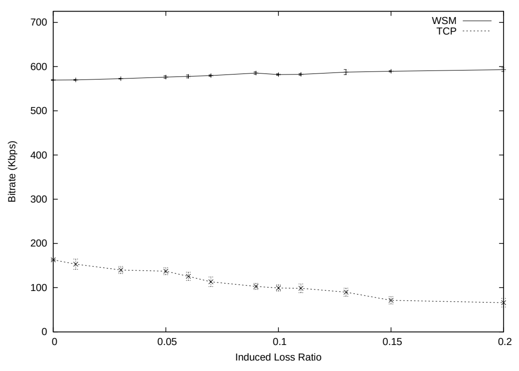

Figure 82: Average Bitrate After Buffering Period Single Bitrate Clip - Induced Loss Started At 15 Seconds.

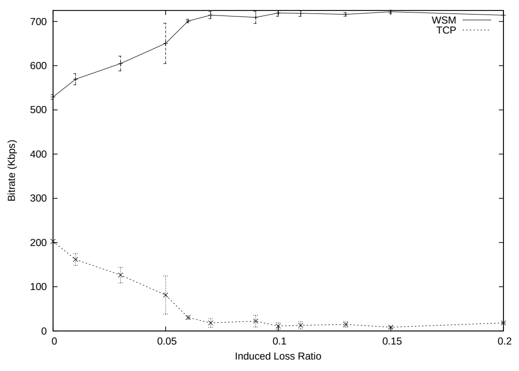

Figure 83: Average Bitrate During Buffering Period Single Bitrate Clip - Induced Loss Stopped At 15 Seconds.

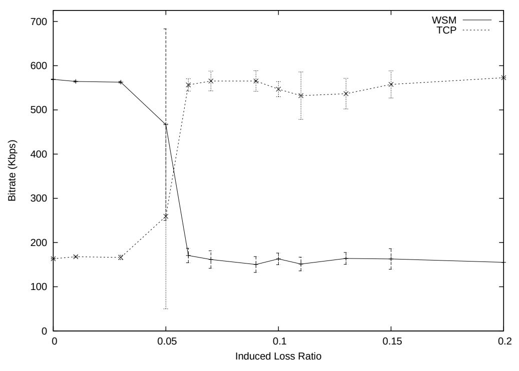

Figure 84: Average Bitrate After Buffering Period Single Bitrate Clip - Induced Loss Stopped At 15 Seconds.

4.4 Induced Latency

In the final set of experiments we seek to determine the effects of increasing latencies on the performance of WSM. We fix the bottleneck capacity at 725 Kbps and latency at 45 ms. While streaming a single bitrate 548 Kbps clip we vary the latency on the network link between the streaming server and client from 150 ms to 1350 ms. This wide range of round-trip times includes those experienced on end-user Internet connections [LR01, Pax99, All00]. Figures 85 - 88 show the results of these experiments. We see that as the round-trip time increases behavior of the WSM flow changes. We see that the loss rate increases dramatically during the buffering period, and as a result the post-buffering bandwith is decreases as WSM responds to the loss rate. However, these measurements are a result of the static queue length parameter at the NIST Net router. When the latency is increased, more packets must be enqueued at the router to impose the artificial delay, thus, the average number of packets in the queue is higher then at lower latencies.

Therefore, we ran an additional set of experiments where the queue length parameter was increased as a function of the latency on the link. Initially, for 45 ms of one-way latency we set the maximum queue length to be 20 times the capacity-delay product (80 packets). We use this same formula for the higher latencies instead of a statically set queue length. Figures 89 - 92 show the results for a multiple bitrate clip with 548, 340, 282, 148, 106, 58, 43, and 28 Kbps encoded bitrates. The results for a single bitrate clip are the same. The WSM flow does not react differently at varying levels of latency, except to achieve a higher bitrate since TCP makes some capacity available at very high latencies.

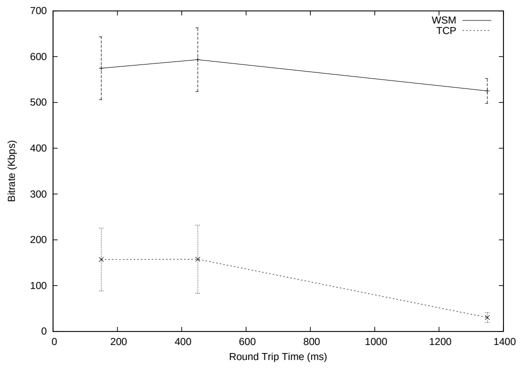

Figure 85: Average Bitrate During Buffering Period - Single Bitrate Clip - Induced Latency.

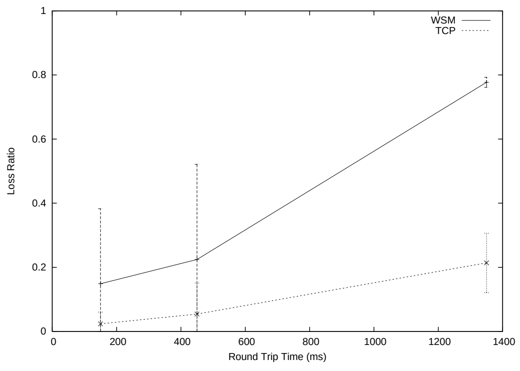

Figure 86: Aggregate Loss Ratio During Buffering Period - Single Bitrate Clip - Induced Latency.

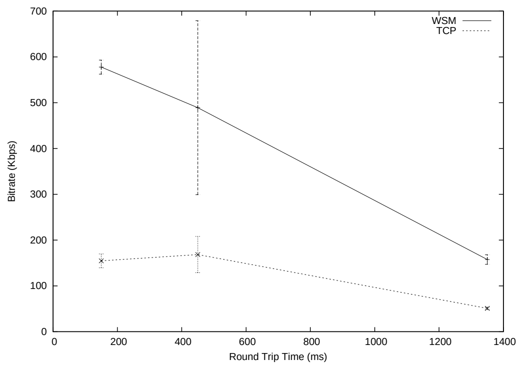

Figure 87: Average Bitrate After Buffering Period - Single Bitrate Clip - Induced Latency.

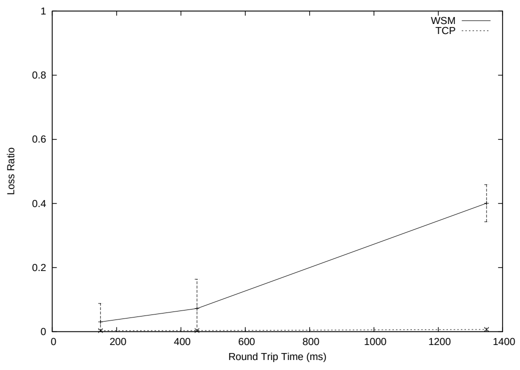

Figure 88: Aggregate Loss Ratio After Buffering Period - Single Bitrate Clip - Induced Latency.

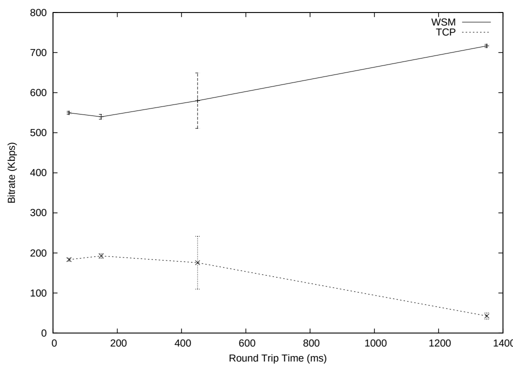

Figure 89: Average Bitrate During Buffering Period - Multiple Bitrate Clip - Induced Latency - Dynamic Queue Length.

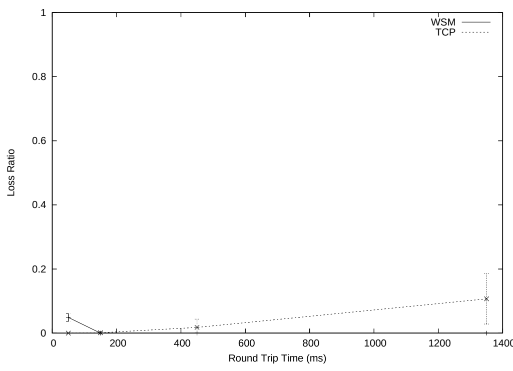

Figure 90: Aggregate Loss Ratio During Buffering Period - Multiple Bitrate Clip - Induced Latency - Dynamic Queue Length.

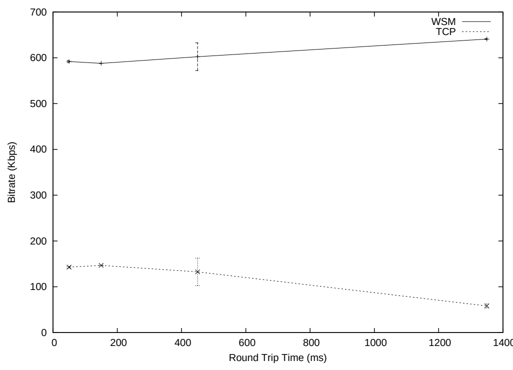

Figure 91: Average Bitrate After Buffering Period - Multiple Bitrate Clip - Induced Latency - Dynamic Queue Length.

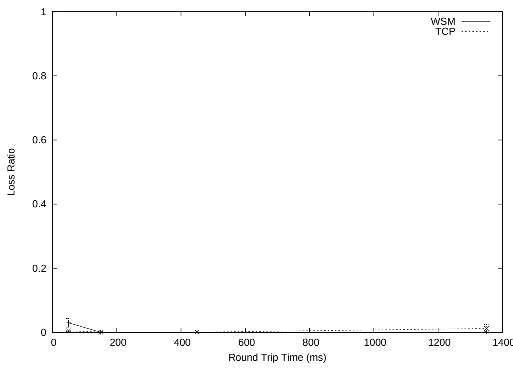

Figure 92: Aggregate Loss Ratio After Buffering Period - Multiple Bitrate Clip - Induced Latency - Dynamic Queue Length.

5 Conclusions

The results of our experiments lead us to make some conclusions about the overall performance or Windows Streaming Media (WSM) and its responsiveness to congestion. First, due to the prominent buffering period WSM cannot be modeled as a simple CBR flow. An accurate bitrate distribution for WSM includes a buffering stage and post-buffering stage where the actual bitrate is dependent on the encoding bitrate of the content and the network conditions. Moreover, an accurate model of WSM must include the bursty nature of packet transmission, especially during buffering for some encoded bitrates. The buffering behavior also leads us to conclude that there are periods when WSM can be very TCP-unfriendly, but there are times when WSM is more than TCP-friendly, consuming less then the fair-share of capacity. Looking at a simple average bitrate over time over the length of the entire clip may not reveal the true nature of WSM and may miss the buffering period where WSM can induce loss the network.

A main goal of this research was to examine the effects of the content encoding parameters on the responsivenss of WSM in order to understand what streaming server administrators and content producers themselves can do to influence the responsivneness of WSM. Since our previous work [LCKN03] found that most of the media clips currently on the Web contain just a single bitrate we make some conclusions based on our initial set of experiments. When streaming single bitrate clips, WSM responds to capacity only when the encoding rate is less than estimated capacity. Otherwise, WSM still attempts to buffer at the encoding rate and high loss rates are induced. During playout, WSM responds to available capacity by thinning and discarding frames if necessary. Furthermore, if the encoded bitrate is less than capacity, WSM still responds to high loss rates (5%) as long as the loss is present at the start of streaming. However, there some encoding rates where WSM will receive more than a fair share of the available capacity.

For multiple bitrate clips, WSM responds to capacity during buffering only when the content contains a suitable bitrate to choose. This chosen bitrate is the largest that capacity allows, which may mean WSM is unfair to TCP. Otherwise, WSM still tries to buffer at the smallest encoding rate available, again resulting in high amounts of loss. During playout, WSM is responsive to available capacity, either because it chose the proper encoding rate, or because it thins if the proper rate is not encoded in the clip. Again, if there was a bitrate included in the clip that was less than the capacity WSM chooses that rate, which may be unfair to TCP. Our results show that content producers can help WSM be more responsive to congestion by encoding several bitrates into their content. However, multiple encoded bitrates are not a panacea for making WSM TCP-friendly and ensuring fairness. But during the buffering period in particular, multiple bitrates do help WSM behave in a more reasonable fashion.

From a network level standpoint, WSM exhibits some undersirable characteristics. First, WSM traffic is somewhat burtsy, leading to bursty packet losses at the NIST Net router queue. This burstiness could be avoided by using a rate-based transport protocol instead of RTP/UDP and application layer rate control. One way to better accomodate WSM traffic would be to over-provision router queues to handle the bursty nature of the traffic, however this may be problematic because of the increased delay larger queues would cause. The use of packet-pair capacity estimates to determine the inital bitrate to stream can lead to incorrect results. An Internet router that uses a token-bucket filter to shape traffic and impose bitrate limitations could cause problems with this technique. If enough tokens are available when the packet probes arrive, they may be quickly let out of the queue, resulting in an inaccurate capacity estimate. Also, this packet-pair estimate may miss any competing traffic on the link.

The results of this thesis are useful because they give a better understanding of Windows Streaming Media. This will facilitate the construction of networks to better accomodate or detect WSM flows. Also, future researchers can conduct experiments using better simulation models of streaming media based upon the results of our careful measurements. Finally, we can uses our results to improve current streaming media products or design future ones.

6 Future Work

Future work that could be performed includes running the experiments with another streaming technology, such as RealVideo or Quicktime. It would be interesting to compare the responsiveness of each of the different technologies available under network conditions similar to the ones studied here. Quicktime in particular would be interesting to study because the application layer congestion control mechanisms are parameterized, such as when to thin the stream, allowing additional tuning by the content provider. Due to the closed-source nature of Windows Streaming Media, we are forced to leave it as a "black box" in our current work. However, the Quicktime streaming server, Darwin, is open-source allowing us to compare our insights reached through measurement with the implementation code.

Another avenue of future work would be to more fully explore the effects of the content on the behavior of WSM. Primarily, one could examine the effect (if any) of content type, for example a sports clip compared to a news clip. The encoded properties of each content type might be different, and this may play a role in performance while streaming.

Finally, future research into the creation of an analytical model for use in simulation could be very useful. A model for use in a network simulator where a researcher could input the content encoding parameters and the model would calculate the appropriate bitrate for the current simulated network conditions would be more accurate than the CBR bitrate distributions typically used. This model would have to incorporate the buffering and post-buffering stages seen during streaming and would need to respond to excessive loss rates.

References

[All99] Allen B. Downey. Using Pathchar to Estimate Internet Link Characteristics. In *Measurement and Modeling of Computer Systems*, pages 222–223, 1999.

[All00] Mark Allman. A Web Server's View of the Transport Layer. *ACM Computer Communication Review*, 30(4), October 2000.

[APS99] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. *IETF Request for Comments (RFC) 2581*, April 1999.

[AS98] Soam Acharya and Brian Smith. An Experiment to Characterize Videos Stored on the Web. In *Proceedings of the ACM/SPIE Multimedia Computing and Networking (MMCN)*, January 1998.

[Bir00] Bill Birney. Microsoft Corporation - Intelligent Streaming, October 2000. Online at: http://msdn.microsoft.com/library/en-us/dnwmt/html/intstreaming.asp.

[CCZ03] Jae Chung, Mark Claypool, and Yali Zhu. Measurement of the Congestion Responsiveness of RealPlayer Streaming Video Over UDP. In *Proceedings of the Packet Video Workshop (PV)*, April 2003.

[CJOS00] M. Christiansen, K. Jeffay, D. Ott, and F.D. Smith. Tuning RED for Web Traffic. In *Proceedings of ACM SIGCOMM Conference*, August 2000.

[Com] Gerald Combs. The Ethereal Network Analyzer. Online at: http://www.ethereal.com.

[CWVL01] M. Chesire, A. Wolman, G. Voelker, and H. Levy. Measurement and Analysis of a Streaming Media Workload. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems (USITS)*, March 2001.

[dMSK02] Jacobus Van der Merwe, Subhabrata Sen, and Charles Kalmanek. Streaming Video Traffic: Characterization and Network Impact. In *Proceedings of the 7th International Workshop on Web Content Caching and Distribution*, Boulder, CO, USA, August 2002.

[FA99] Aaron Falk and Mark Allman. On the Effective Evaluation of TCP. *ACM Computer Communication Review*, 5(29), 1999.

[FF99] Sally Floyd and Kevin Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, 1999.

[FK03] Sally Floyd and Eddie Kohler. Internet Research Needs Better Models. *ACM SIGCOMM Computer Communications Review*, 33(1), January 2003. HOTNETS-1 Workshop Issue.

[Flo94] S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24(5):10–23, 1994.

[GCK$^+$02] Bernd Girod, Jacob Chakareski, Mark Kalman, Yi J. Liang, Eric Setton, and Rui Zhang. Advances in Network-adaptive Video Streaming. In *Proceedings of 2002 Tyrrhenian International Workshop on Digital Communications (IWDC 2002)*, pages 1–8, Capri, Italy, September 2002.

[Hao02] Hao Jiang and Constantinos Dovrolis. Passive Estimation of TCP Round-Trip Times. *ACM SIGCOMM Computer Communications Review*, 32(3), July 2002.

[HAOS01] Duke P. Hong, Celio Albuquerque, Carlos Oliveira, and Tatsuya Suda. Evaluating the Impact of Emerging Streaming Media Applications on TCP/IP Performance. *IEEE Communications*, April 2001.

[IS02] In-Stat/MDR. Big Game Safari: Strategies For Business Streaming Media, September 2002. Press Release. Online at: http://www.instat.com/press.asp?ID=338&sku=IN020029MB.

[Jac97] Van Jacobson. Pathchar - a Tool to Infer Characteristics of Internet Paths, April 1997. Presented at Mathematical Sciences Research Institute (MSRI).

[JLM] Van Jacobson, Craig Lares, and Steven McCanne. TCPDUMP public repository. Online at: http://www.tcpdump.org/.

[KGM$^+$02] Markku Kojo, Andrei Gurtov, Jukka Manner, Pasi Sarolahti, Timo Alanko, and Kimmo Raatikainen. Seawind: a Wireless Network Emulator. In *Proceedings of 11th GI/ITG Conference on Measuring, Modeling and Evaluation of Computer and Communication Systems (MMB 2001)*, RWTH Aachen, Germany, September 2002.

[KM87] C. Kent and J. Mogul. Fragmentation Considered Harmful. *SIGCOMM Symposium on Communications Architectures and Protocols*, pages 390–401, August 1987.

[KP87] Phil Karn and Craig Partridge. Improving Round-Trip Time Estimates in Reliable Transport Protocols. *ACM SIGCOMM*, August 1987.

[LCK02] Mingzhe Li, Mark Claypool, and Robert Kinicki. MediaPlayer versus RealPlayer - A Comparison of Network Turbulence. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

[LCKN03] Mingzhe Li, Mark Claypool, Robert Kinicki, and James Nichols. Characteristics of Streaming Media Stored on the Internet. Technical Report WPI-CS-TR-03-18, CS Department, Worcester Polytechnic Institute, May 2003.

[LOR98] T. V. Lakshman, Antonio Ortega, and Amy R. Reibman. VBR Video: Tradeoffs and Potentials. *Proceedings of the IEEE*, 86(5), May 1998.

[LR01] D. Loguinov and H. Radha. Measurement Study of Low-bitrate Internet Video Streaming. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, November 2001.

[LR02a] D. Loguinov and H. Radha. Increase-Decrease Congestion Control for Real-time Streaming: Scalability. In *IEEE Infocom 2002*, New York, New York, June 2002.

[LR02b] D. Loguinov and H. Radha. Large Scale Experimental Study of Internet Performance Using Video Traffic. *ACM SIGCOMM Computer Communication Review*, 32(1):7–19, January 2002.

[Ltd02] Point Topic Ltd. DSL Benchmarking Report, September 2002. Press Release. Online at: http://www.point-topic.com.

[Lui97] Luigi Rizzo. Dummynet: A Simple Approach to the Evaluation of Network Protocols. *ACM Computer Communication Review*, 27(1):31–41, 1997. Online at: http://info.iet.unipi.it/~luigi/ip_dummynet/.

[MH00] Art Mena and John Heidemann. An Empirical Study of Real Audio Traffic. In *Proceedings of the IEEE Infocom*, pages 101 – 110, March 2000.

[MPL] MPLAYER. Mplayer, the Movie Player for Linux Official Site. Online at: http://www.mplayerhq.hu/.

[Net] WPI Netops. WPI Network Operations - Network Infrastructure. Online at: http://www.wpi.edu/Admin/Netops/infrastructure.html.

[NIS] NIST. NIST Net Home Page. Online at: http://www.antd.nist.gov/itg/nistnet/.

[NLA] NLANR. NLANR/DAST : Iperf 1.7.0 - The TCP/UDP Bandwidth Measurement Tool. Online at: http://dast.nlanr.net/Projects/Iperf/.

[otBLMT02] Committee on the Broadband Last Mile Technology. Broadband: Bringing Home the Bits. *ACM SIGCOMM Computer Communications Review*, 32(2), April 2002.

[Pax99] Vern Paxson. End-to-End Internet Packet Dynamics. *IEEE/ACM Transactions on Networking*, 7(3):277–292, 1999.

[PF95] Vern Paxson and Sally Floyd. Wide-Area Traffic: the Failure of Poisson Modeling. *IEEE/ACM Transactions on Networking*, 3:226 – 244, 1995.

[PF97] Vern Paxson and Sally Floyd. Why We Don't Know How to Simulate the Internet. In *Winter Simulation Conference*, pages 1037–1044, 1997.

[PFTK98] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and Its Empirical Validation. In *Proceedings of ACM SIGCOMM*, 1998.

[PV02] Attila Pasztor and Darryl Veitch. Active Probing using Packet Quartets. In *Proceedings of ACM SIGCOMM Internet Measurement Workshop*, Marseille, France, November 2002.

[RF95] A. Romanow and S. Floyd. Dynamics of TCP Traffic over ATM Networks. *IEEE Journal on Selected Areas in Communications*, 13, 1995. Available online at: http:www.nrg.ee.lbl.gov/nrg-papers.html.

[SCFJ96] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. *IETF Request for Comments (RFC) 1889*, January 1996.

[SGB$^+$03] Nathan Sheldon, Eric Girard, Seth Borg, Mark Claypool, and Emmanuel Agu. The Effect of Latency on User Performance in Warcraft III. In *Proceedings of ACM NetGames*, 2003.

[SK02] Pasi Sarolahti and Alexey Kuznetsov. Congestion Control in Linux TCP. In *Proceedings of USENIX Conference*, 2002.

[SRL98] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). *IETF Request for Comments (RFC) 2326*, April 1998.

[VAM$^+$02] Eveline Veloso, Virgilio Almeida, Wagner Meira, Azer Bestavros, and Shudong Jin. A Hierarchical Characterization of a Live Streaming Media Workload. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, November 2002.

[vdMChCS00] Jacobus van der Merwe, Ramon Caceres, Yang hua Chu, and Cormac Sreenan. mmdump - A Tool for Monitoring Internet Multimedia Traffic. *ACM Computer Communication Review*, 30(4), October 2000.

[WBJ03] Z. Wang, S. Banerjee, and S. Jamin. Studying Streaming Video Quality: From An Application Point of View. In *Proceedings of the ACM Multimedia Conference*, November 2003.

[WCZ01] Yubing Wang, Mark Claypool, and Zheng Zuo. An Empirical Study of RealVideo Performance Across the Internet. In *Proceedings of the ACM SIGCOMM Internet Measurement Workshop*, San Francisco, California, USA, November 2001.

[Wei93] Mark Weiser. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 36(7), July 1993.

[Wil01] Carey Williamson. A Tutorial on Internet Traffic Measurement. *IEEE Internet Computing*, 5(6):70–74, November 2001.