Proxy Support for HTTP Adaptive Streaming

A Thesis Submitted to the

College of Graduate Studies and Research

in Partial Fulfillment of the Requirements

for the degree of Master of Science

in the Department of Computer Science

University of Saskatchewan

Saskatoon

By

Michael Chad Bullock

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

Not long ago streaming video over the Internet included only short clips of low quality video. Now the possibilities seem endless as professional productions are made available in high definition. This explosion of growth is the result of several factors, such as increasing network performance, advancements in video encoding technology, improvements to video streaming techniques, and a growing number of devices capable of handling video. However, despite the improvements to Internet video streaming this paradigm is still evolving.

HTTP adaptive streaming involves encoding a video at multiple quality levels then dividing those quality levels into small chunks. The player can then determine which quality level to retrieve the next chunk from in order to optimize video playback when considering the underlying network conditions. This thesis first presents an experimental framework that allows for adaptive streaming players to be analyzed and evaluated. Evaluation is beneficial because there are several concerns with the adaptive video streaming ecosystem such as achieving a high video playback quality while also ensuring stable playback quality.

The primary contribution of this thesis is the evaluation of prefetching by a proxy server as a means to improve streaming performance. This work considers an implementation of a proxy server that is functional with the extremely popular Netflix streaming service, and it is evaluated using two Netflix players. The results show its potential to improve video streaming performance in several scenarios. It effectively increases the buffer capacity of the player as chunks can be prefetched in advance of the player's request then stored on the proxy to be quickly delivered once requested. This allows for degradation in network conditions to be hidden from the player while the proxy serves prefetched data, preventing a reduction to the video quality as a result of an overreaction by the player. Further, the proxy can reduce the impact of the bottleneck in the network, achieving higher throughput by utilizing parallel connections to the server.

Acknowledgements

I would like to take this opportunity to give recognition to the people who have helped make this thesis possible.

With the respect and gratitude, I acknowledge my Supervisor, Dr. Derek Eager, whose wealth of knowledge, support, and guidance have made this thesis possible. His supervision has provided me with a second to none educational experience, for which I am very grateful. I would also like to thank my committee members: Dr. Daniel Teng, Dr. Dwight Makaroff, and Dr. Mark Keil, for their insightful comments. Finally, the support of my family has been invaluable throughout my life and educational endeavours. I thank my father, Kerry, my mother, Patricia, and my brothers, Bradley, Christopher, and David.

To my parents, Patricia and Kerry. Thank you for all your love and support.

Contents

List of Tables

List of Figures

List of Abbreviations

ACK	Acknowledgement
AIMD	Additive Increase Multiplicative Decrease
APV	Average Playback Version
CDN	Content Distribution Network
CSV	Comma-Separated Values
DNS	Domain Name System
GB	Gigabyte
HTTP	Hypertext Transfer Protocol
ID	Identifier
IP	Internet Protocol
KB	Kilobyte
Kbps	Kilobits Per Second
LFU	Least Frequently Used
LRU	Least Recently Used
MB	Megabyte
Mbps	Megabits Per Second
NAT	Network Address Translation
PC	Personal Computer
PO	Prefetching Overhead
PS	Playback Smoothness
QoE	Quality of Enjoyment
QoS	Quality of Service
RAM	Random Access Memory
RTCP	RTP Control Protocol
RTMP	Real Time Messaging Protocol
RTP	Real Time Transport Protocol
RTSP	Real Time Streaming Protocol
RTT	Round Trip Time
SVC	Scalable Video Coding
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VPN	Virtual Private Network
WLAN	Wireless Local Area Network
XML	Extensible Markup Language

Chapter 1

Introduction

The popularity of using the Internet to stream video has been rapidly increasing since YouTube's launch in 2005. Lately, this interest can be attributed to the release of professional productions that can be streamed in high definition as a result of increases in network performance, advances in video encoding technology and improved video streaming paradigms. Video streaming systems rely on all of these advancements for delivering their content to consumers yet there is still significant potential for improvement. The remainder of this chapter is broken down as follows. In Section 1.1, the evolution of video streaming technology is discussed. Section 1.2 presents a description of the growth of Internet video streaming. Then Sections 1.3 through 1.5 describe the motivation for the research, the thesis objectives, and the research findings.

1.1 Video Streaming Evolution

In the past TCP was never thought to be a good protocol for video delivery due to the long retransmission delays and constantly fluctuating sending rates. As a result UDP methods were investigated including RTMP or RTP with RTSP and RTCP. These server side push paradigms would receive control packets from clients, indicating their current state, including information such as buffer occupancy. Then the server would push data to the client at a rate it determines based on the control packets and the bitrate of the video. This allowed for video rate adaptation where the client can receive video encoded at multiple bitrates based on logic implemented at the server. Unfortunately this approach was not feasible for video delivery across the wide area Internet as it suffered several drawbacks. The server does not scale as well as it must maintain state for each client, the content is not cacheable without specialized servers as it is being pushed to the client and there is difficulty traversing firewalls and NATs [7]. YouTube gained popularity using a method known as progressive download. In contrast to the previously discussed push approach, progressive download is pull based, meaning that the client explicitly requests the video file or files and the server simply sends the data using TCP. The downside of this method is that if there are multiple video quality levels available it requires the viewer to select a bitrate for the video in advance. Due to the uncertainty in the Internet and the nature of this method the viewer can be left with suboptimal playback as stored video may have a quality much lower than the viewer's connection could support or the video may require more bandwidth than is available to that player and the video will frequently be freezing to rebuffer.

Scalable video coding (SVC) is another video delivery method. With SVC, the video is encoded as a hierarchical set of layers where there exists a base layer of low quality video and enhancement layers to increase the quality of video. The enhancement layers may improve frame rate, resolution or quality of the video signal. SVC video layers simply add to the video quality resulting in low redundancy, which reduces storage requirements at the servers. SVC also allows for an easy transition down in quality if the available bandwidth drops as enhancement layers can simply be dropped. However, SVC is complex and imposes codec restrictions so it has not been widely adopted [7].

HTTP adaptive streaming (or simply "stream switching") was developed to address the concerns with push-based paradigms and progressive download by encoding the video at various quality levels on the server, then dividing them into small chunks that are retrieved individually using the HTTP protocol. The client player can then dynamically and seamlessly switch between the quality levels (versions) of video at any time during playback based on its rate-adaptation logic. The end result is a video stream that is tailored to a particular player, on a particular connection, at that particular time. HTTP adaptive streaming is a pull-based method that provides rate adaptation similar to that available using server side push methods and SVC, but the logic is at the client so the server is more scalable, the content is an HTTP object so it is cacheable, and HTTP easily traverses through firewalls and NATs [7].

HTTP adaptive streaming is a complex paradigm and does not come without pitfalls. For example, determining which version of video to request next is nontrivial as the client must consider several factors such as the available bandwidth, round-trip time to the server, buffer occupancy, and the viewer's perceived quality, which involves metrics such as how frequently the quality level is changing or how large the changes are. To perform well the player must be able to quickly obtain a proper steady state, as oscillations or large changes in the quality level are annoying to viewers. Unfortunately, once a steady state is obtained there are no guarantees the underlying network conditions will not change. In fact, it is possible that the network conditions will change frequently so the player should be able to absorb short-term variation to the underlying network while adequately adjusting to long-term variation without panicking or overcompensating. There are also support systems within the network such as caches that could inject noise into the player's measurements causing instability. As such, players are still in their infancy and the systems have not been perfected.

1.2 Netflix and the Growth of Internet Video Streaming

Using the Internet to stream multimedia is immensely popular. In 2013, 68.2% of the peak period downstream traffic and 62.0% of the aggregate traffic in North America on fixed networks is attributed to real-time entertainment [68]. Netflix is chosen as the video streaming platform to be used in this research because it is the largest and most successful Internet video streaming service that utilizes HTTP adaptive streaming. Netflix was first launched in 1997 for online movie rentals. In 2007 it expanded to become a video streaming service and in 2010 the video streaming service was expanded to Canada. In 2012, Netflix finished the year

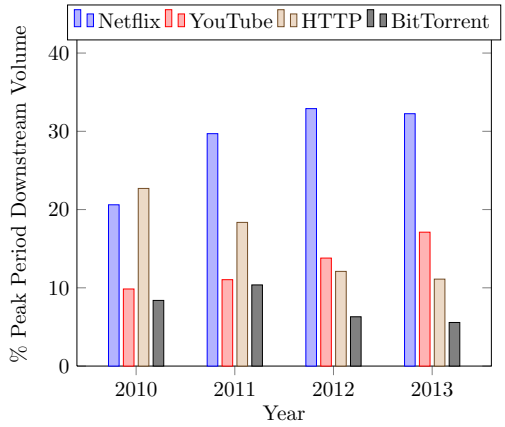

Figure 1.1: Peak Period Downstream Traffic on North American Fixed Access Networks

with 20 million members and in 2013 surpassed 30 million members, demonstrating its growth and success.[1]

Sandvine's Global Internet Phenomena Report[2] is released twice yearly to shed light on the state of the Internet and its trends. Netflix's dominance as a source for network traffic can be seen in Figure 1.1, which shows the peak period downstream volume to fixed networks in North America with Netflix utilization values starting at 20.61% in 2010 and growing to 32.25% of the traffic by volume in 2013 [64, 65, 66, 68]. As defined by Sandvine, *peak period* is a period of approximately a couple of hours in duration where traffic volume is within some level (often in the 95^{th} percentile) of the peak instantaneous daily value [66]. Comparison of Netflix to other commercial video providers such as Amazon (1.31%), HBO Go (0.34%), and Hulu (2.41%) shows that Netflix is by far the industry leader [68]. There are other alternatives to the Netflix HTTP adaptive streaming system, such as Microsoft Smooth Streaming and Adobe OSMF but they are less widely used and poorer performers respectively [2].

Netflix does not exhibit the same dominance on mobile networks. It has a comparatively low percentage of the bandwidth used, consuming only 3.98% of the peak period downstream volume in North America on mobile networks but this is still the 7^{th} largest source for mobile traffic. YouTube and web browser traffic are the most popular mobile sources at 27.33% and 19.16%, respectively. This does not indicate that wireless and mobile devices are not commonly used for streaming Netflix. Home roaming involves connecting mobile devices to a home's fixed access network. Over 25% of all real-time entertainment being delivered to homes is destined to mobile devices such as smartphones and tablets, with 10% of the total being delivered to iPads. Further, simply because mobile networks are not dominated by Netflix traffic does not mean they are not transmitting video traffic. Real-time entertainment accounts for 47.2% of the downstream traffic on mobile networks and is expected to account for over 60% by 2018 [68] and 70% by 2019 [67]. These values may be conservative, as in the past mobile video usage has been well ahead of projections [66].

[1] https://signup.netflix.com/MediaCenter/Timeline, accessed 19-October-2013
[2] https://www.sandvine.com/trends/global-internet-phenomena/, accessed 19-October-2013

Sandvine is not the only group to support claims of extensive increases to Internet video streaming. Cisco publishes the Visual Networking Index to provide networking forecasts. The 2012-2017 version makes several interesting claims [13, 14]. They estimate that in 2017, nearly a million minutes of video content will traverse the IP network globally every second and it would take 5 million years to watch the amount of video that crosses the network in one month. Much of this traffic will be destined to wireless devices so wireless traffic will eventually surpass the traffic for wired devices. It is expected that in 2017, 55% of IP traffic will be for wireless devices, up from 41% in 2012. Much of this wireless data will however remain on fixed networks as there is significant offload of mobile data; globally 33% of mobile data was sent over WiFi or femtocells. Specifically considering video, 69% of global consumer Internet traffic will be video in 2017, up from 57% in 2012. By 2017 video on demand traffic will close to triple, becoming equivalent to 6 billion DVDs per month. Further, mobile video is growing as 2012 marked the first time mobile video exceeded 50% of total mobile traffic, reaching 51%.

1.3 Motivation

With the immense popularity and growth potential of Internet video streaming it is important that providers be able to keep subscriber interest by providing the highest video quality possible. Issues arise when the network between the player and the server is not capable of sustaining the highest available video quality level for the duration of the video's playback. When a bottleneck keeps the available bandwidth level too low or too variable, a prefetching proxy can help. A prefetching proxy between the player and the server could improve player performance without requiring player modification by prefetching chunks in advance of their request so they can be quickly served to the player once requested. This could reduce the impact of long chunk delivery times resulting from a bottleneck between the proxy and the server.

The development of a prefetching proxy could allow for a service that offers clients improved video performance. This service increases throughput from the server resulting in an overall higher video quality. It also absorbs bandwidth reduction and fluctuation that allows the player to maintain a high video quality in an otherwise unreliable or slow network.

1.4 Thesis Objectives

The objectives of this thesis are as follows:

- To establish an experimental platform, which allows for the evaluation of a prefetching proxy with both wired and mobile devices.
- To characterize the Netflix rate-adaptation algorithm using two different platforms, under various network conditions.

- To develop a prefetching proxy server functional with the Netflix video streaming service.
- To evaluate the impact of the prefetching proxy on the Netflix player's streaming performance under various networking conditions.

1.5 Thesis Findings

Experiments are conducted without prefetching and with algorithms using various levels of prefetching, for both a "PC Browser" player and an iPad player. These experiments are conducted with a stable level of available bandwidth, bandwidth that frequently oscillates between two levels, long-term increases and decreases to the available bandwidth, and short-term sequential spikes and drops to the available bandwidth.

The PC Browser player and the iPad player are quite different, both in terms of their properties and their rate-adaptation behaviour. The iPad player uses chunks with a longer duration, as well as a smaller buffer. The iPad player also makes more abrupt changes to video quality during increases to the available bandwidth by often jumping multiple quality levels whereas the PC Browser player only increases by a single quality level each time. Further, for their respective versions, the bitrates for the PC Browser player are significantly higher than for the iPad player.

Both players react very quickly to available bandwidth variation, often by the next chunk request. In the case of the PC Browser, the player's buffer does have more of an impact on the behaviour of the player when its buffer occupancy increases. However, the player is not willing to sacrifice a significant portion of its buffer in favour of maintaining higher video qualities when the available bandwidth drops.

Proxy prefetching offers substantial benefits to the player's streaming performance in many cases. It is capable of utilizing parallel download to increase throughput and improve the average playback quality. Prefetching also allows for content to be stored on the proxy, effectively increasing the player's buffer capacity so reductions or fluctuations to the available bandwidth can be hidden from the player by the proxy, allowing the player to remain confident and maintain a high playback quality.

1.6 Thesis Organization

The organization of the thesis is as follows. In the next chapter the background and related work is presented. Chapter 3 covers the experimental methodology. In this chapter the details of the hardware, software, and test bed layout are discussed as well as the performance factors and metrics for evaluation. Chapter 4 presents the results of experiments characterizing the iPad and PC Browser rate-adaptation behaviour under various conditions. Chapter 5 describes experimental results concerning the prefetching proxy's impact on the performance of each player. In this chapter the experimental results using no prefetching are compared to the results using various levels of prefetching. Finally, Chapter 6 concludes the thesis.

Chapter 2

Background

This chapter presents a detailed description of previous work in the performance of Internet video streaming as well as the relevant background information needed for this research. Section 2.1 takes a look at network support for video streaming. Section 2.2 gives background on Quality of Enjoyment (QoE) which is a metric used to reflect the quality of the experience for viewers watching the video. Then Section 2.4 discusses adaptive video streaming in detail, including previous works involving bitrate adaptation that suggest improvements to advance video streaming architectures and evaluate current architectures.

2.1 Network

The network is an important part of an Internet streaming system, in fact it is among the most critical aspects of Internet video streaming as large amounts of data must successfully traverse through the network to be played by the player. Popa et al. [54] discuss the seemingly inevitable trend of HTTP becoming the defacto protocol for new services and applications in the Internet. In the mid 1990's HTTP became the dominant protocol. At the end of the 1990's audio and video delivered over RTSP and RTMP challenged HTTP and in the 2000's peer-to-peer technologies saw explosive growth. However, the cost of using CDNs has reduced drastically, by a factor of 10, between 2006 and 2010 so the need for P2P has diminished. HTTP has many benefits [54, 73]:

- It can easily traverse through routers and firewalls
- It provides simple and reliable deployment on top of TCP/IP
- It is scalable as it can be redirected to the least loaded server
- It can make use of widely available HTTP caches
- It supports both reverse and forward proxies
- It supports DNS
- CDNs have more HTTP servers than streaming servers and HTTP servers do not incur licensing costs as they are typically based on open source software
- Control can be moved to clients which decreases server complexity and increases server scalability

HTTP runs on top of TCP. There have been a number of prior studies concerning how to tune use the of TCP for video streaming. Goel et al. investigate how media streaming servers can reduce their latency by reducing the TCP send buffer size [27]. TCP has a large send buffer, but only the data in the buffer that fits within the congestion window can be sent on the TCP connection until the window moves. This means the data near the end of the buffer can suffer high delays before being transmitted. By reducing the amount of data in the send buffer to only slightly higher than the congestion window, the latency moves to the application layer. Having the latency at the application layer allows the application to have greater control over delivery of time sensitive data so the application can decide to drop stale data or send new high priority data. This strategy comes at the cost of reduced throughput because normally TCP can send data from its buffer immediately after receiving an ACK, while in this case, TCP needs to wait for the application to add new data before it can be sent. However, since the send buffer does not need to be so limited, throughput can be improved with a small increase to the latency by allowing slightly more data to be placed in the buffer than the congestion window size. The research shows that low-latency streaming such as required for live content is feasible over TCP by tuning TCP's send buffer to only keep data in flight.

Ghobadi et al. [26] look at how bursty video traffic can negatively impact the network as a whole. Their research considers YouTube and how (at the time of their study) it uses data bursts, which stress router queues, cause packet loss and result in a reduction to the congestion window due to TCP's congestion control algorithm. They found that progressive download video delivery as implemented by YouTube servers goes through two phases: the startup phase where the first 30-40 seconds of video are sent as fast as possible then the throttling phase where a token bucket algorithm is used to schedule video delivery by placing video in a token bucket at 125% of the encoding rate. Bursts followed by idle periods disrupt TCP's self clocking and disrupt other latency sensitive applications. The authors found that these bursts are responsible for 40% of YouTube's packet loss on at least one residential DSL provider. A method to combat this bursty traffic called Trickle is presented. Trickle combats these issues by dynamically setting a maximum congestion window to limit the streaming rate and reduce burst size. This change, which is made on the server side so it is easily deployable, effectively reduces retransmissions by up to 50% and reduces round trip time (RTT) by up to 28% while having negligible effect on the frequency of video freezing events.

Single TCP streams struggle in conditions of long RTT and unanticipated packet loss. The total application throughput under these conditions can be improved by using parallel streams, however this is unfair to single streams. This unfairness is addressed by inserting temporal gaps between requests to emulate long RTTs, thus enabling fairer resource sharing while achieving a more stabilized transmission rate [41]. The authors propose a system where the video is first chunked into pieces, then distributed across multiple HTTP streams; if delivery of a particular chunk stalls and it is not downloaded in a set time it is fetched by two HTTP streams to increase the probability of receiving the chunk in time. This maintains fairness because it does not open another connection but queues the chunk with an existing connection where the fairness is controlled by the temporal gap parameter.

Rao et al. study the impact of the application (Web browsers, mobile application) logic and the container (Flash, HTML5, Silverlight) on the characteristics of the network traffic generated by Netflix and YouTube streaming [56]. The authors find that different streaming strategies are used dependent on the application and the container used. They detail the network traffic characteristics of each of those strategies and present video streaming parameters that can be adapted to minimize the amount of data delivered but not used ("unused bytes") when a video is terminated by the viewer before the end. Three video streaming strategies were observed:

- No ON-OFF cycles: For this streaming strategy all data is transferred as fast as possible. An advantage of this strategy is that it requires no complex engineering at either the client or server side. However, it can overwhelm the player if there is insufficient memory on the client device and cause a large amount of unused bytes if the viewer quits the video early.
- Short ON-OFF cycles: For this streaming strategy there is a periodic transfer of blocks of size less than 2.5 MB followed by an off (idle) period. This off period is observed only when the video playback rate is lower than the capacity of the network. This allows the rate of video accumulation in the buffer to be paced so to not overwhelm the client buffer.
- Long ON-OFF cycles: This strategy is a hybrid of No ON-OFF and Short ON-OFF in that it entails long periods of downloading followed by long idle periods.

The authors find that the streaming strategy that is used is dependent on the container, the application, and even the type of browser (Mozilla Firefox vs Google Chrome vs Microsoft Internet Explorer). By keeping the off periods an appropriate duration such that the buffer accumulates at a slow rate then the bytes downloaded and left unused can be minimized, which is important as many videos are quit early [19, 23, 36, 53, 69, 80].

Khemmarat et al. investigate the effectiveness of using prefetching for user-generated videos, such as those found on YouTube, in order to improve quality of service [37]. Their system works by predicting which videos may be watched in the future using search results and related video lists provided by YouTube, then prefetching the first 30% of each of those videos. The premise is that if you prefetch the start of a video then the remainder can be downloaded as the video is being watched, resulting in no freezing for rebuffering and a shorter initial buffering phase. Using the recommended list is a promising approach as it was found that approximately 45% of the time a client selects the next video from the recommended video list [69]. The prefetching agent can be implemented in the client itself or at a proxy. The benefit of hosting the prefetching agent at a proxy is that these prefixes can be cached for other users as well. This research is motivated by a user study the authors conducted that involved people watching a subset of YouTube videos in various environments. They find that without prefetching approximately 35% of all playbacks contained pauses for rebuffering and 75% of those contained more than 10 pauses. The prefetching scheme shows promise in that it can achieve a higher hit rate than caching alone by only prefetching the top 3 related

videos with greater improvement potential if more are considered. However, prefetching increases network load while caching decreases network load. It is found that by prefetching more videos there was a higher overlap between prefetched and cached videos so the two principles can be combined to keep the network load reduced while maintaining a high hit rate. Combining caching and prefetching produced up to an 81% hit rate, an increase of 5-20% over prefetching only when using a university campus' network traces as a dataset. There are drawbacks to this type of prefetching system. For example, the time required to prefetch an adequate amount of video to prevent freezing for rebuffering is too long to be effective for long movies such as those provided by Netflix.

Bhandarkar et al. present a caching framework for efficient delivery of personalized video streams [9]. Essentially, with a massive influx of many types of mobile devices such as smartphones and tablets, each device can have its own multimedia quality preferences. Thus, these devices must download video most relevant to their own constraints, which often include battery capacity, screen resolution, and video decoding/rendering capabilities. The system-wide constraints such as network bandwidth are also considered. However, this personalization may undermine the effectiveness of caching as clients are requesting different quality levels of the same video. To address this caching challenge the authors present a collaborative caching framework that gets a video chunk in a high quality then in real time creates two lower qualities to store as well resulting in each chunk using approximately 1.5x the space required for the high quality version only. The cache replacement policy used gives each video file a retention value that is the number of clients requesting that video file over the size of the file. When eviction is necessary to make room for new videos the cache starts at the file with the lowest retention value then the highest, medium and lowest qualities are evicted in that order until enough room is available for the new file. This replacement policy was shown to outperform LRU and LFU. Their proposed system utilizes a latency based collaborative caching scheme where when a miss occurs at a cache the cache looks in either another cache or a server, whichever is closer. In their experiments this method proved to reduce the client experienced latency over both a fully collaborative cache, where all caches are checked before servers and a non-collaborative cache where the request is sent directly to a server after a miss. This work gives some interesting insights into network support but the support for personalization is minimal as there are only three quality levels.

Sharma et al. discuss setting up a multimedia cache near to cellular towers to reduce latency and back-haul bandwidth in these areas with low electrical grid reliability [69]. The Internet is frequently undependable in third world countries where electrical grid reliability is poor and in these areas the fraction of users relying on smartphones to access the Internet is relatively high as they do not have access to more expensive and powerful devices. For example, in India 150,000 of the 400,000 cell towers do not have reliable access to the electrical grid. As such, network providers see these areas as a large potential market and green off-the-grid cellular towers that use renewable power sources such as solar or wind are appealing. Adding a cache server increases the energy demand of the tower so rather than adding one high powered server the authors suggest adding several (9) low powered cache servers then staggering them on and off in a load proportional manner

during a duty cycle such that all servers are active at least once during a duty cycle of 60 seconds. This strategy, called blinking allows the cache cluster to adapt to intermittent power. By using a cluster of low power machines the cache size can be scaled to match available power rather than a single high powered cache that must be shutdown completely once the available power drops below some threshold. The blinking policy determines how much time during the duty cycle each server gets by how popular the media chunks that it stores are. The authors found that the client frequently selects the next video from the recommended video list on YouTube. Based on this finding the authors implemented prefix prefetching of those YouTube videos based on [37].

Kuschnig et al. [42] show that video streaming can scale with available bandwidth by increasing the download period or the number of concurrent streams. Small chunks lead to short responses, which may experience unfairness and long RTTs, resulting in highly variable throughput for single TCP streams. Increasing the number of concurrent streams improves over the performance of one stream in the presence of packet loss by stabilizing the throughput and larger chunk sizes use the network bandwidth more effectively. The resulting system maintains fairness with single TCP streams by utilizing temporal inter-request gaps, while stabilizing quality.

2.2 QoE: Quality Of Enjoyment

In order to make money video streaming services rely on either user subscription fees or advertising revenue. To collect these fees users need to be satisfied with the content to renew subscriptions or play the video so the ads can be aired. As such, profits and the users experiencing a high quality of enjoyment are strongly correlated, which means service providers would like to maximize their QoE while limiting their costs in doing so. Plissonneau and Biersack investigate viewing behaviour of progressive download sites YouTube and Dailymotion and find that when reception quality is bad, as when the video will stops for rebuffering, the video is rarely finished and results in reduced viewing durations [53]. However, they also find that even when reception quality is good only half the videos are fully downloaded so both interest in the content and reception quality play a factor in how much of a video is viewed.

There are different methods for quantifying quality of a video stream. Objective tests that measure the spatial quality of video such as peak signal to noise ratio and subjective tests that involve users viewing video and generally ranking the video quality with a Mean Opinion Score [49]. These differ from traditional networking quality of service (QoS) metrics such as jitter, network bandwidth, packet loss rate, and round trip time in that they are directly concerned with the video application rather than the underlying network conditions. There has been some standardization for both the assessment of objective [33, 34, 35] and subjective [10, 50] quality measurements. Because concerns are with the actual enjoyment and because humans are complex in terms of perception it is hard to measure QoE with objective experiments but subjective experiments come with their own challenges. Pinson and Wolf [52] look at subjective tests and

two types of subjective testing that each have their own advantages but they find the results can be presented from one to correlate highly to the other.

Klaue et al. present a framework and toolset for evaluating the quality of video transmitted over a real or simulated network [38]. They measure QoS parameters of the underlying link as well as estimate the video quality based on frame-by-frame peak signal to noise ration (PSNR). Such a framework is important because digital quality must be based on the perceived quality by the user, reducing the jitter or improving the packet loss rate means very little if the user is experiencing what they believe to be a lower quality of video. This accents the importance of subjective testing. The issue becomes that subjective testing is very time consuming, very costly, requires special equipment and has significant manpower requirements. These costs are substantial and complex and as such objective metrics have been developed to emulate the quality impression of the human visual system. Wolf and Pinson take a detailed look at various objective tests and how they compare to subjective tests; however, the most widespread method is the PSNR image by image [78].

There has been research efforts in finding correlations between network QoS and viewer QoE. Mok et al. [47] first correlate the network QoS to the application QoS, which considers three metrics, initial buffering time, mean rebuffering time and rebuffering frequency. Then experiments are conducted to see how subjective QoE is influenced by application level QoS and this allows the authors to find a correlation between network QoS and QoE. They find that network throughput is lowered by packet loss and RTT thus increasing the rebuffering frequency and rebuffering frequency was identified to be the main factor responsible for MOS variance. Then they find that network measurements may miss important information about user dissatisfaction and subjective tests are hard to evaluate because they are subjective in nature [48]. Issues such as the viewers culture play a part as shown by Chen et al. [12] where they find that people of Asian descent do not select extremes on rating scales. The authors hypothesize that they can indirectly measure QoE because user-viewing activities such as pausing the video or shrinking the viewing area the are more likely to be triggered after the presence of rebuffering events. They first perform a survey on the participants and ask them their level of agreement with how they behave during various playback qualities. Next for evaluation the subjects, 22 people watched 3 videos at varying quality and were told to behave as they normally would. Metrics such as pausing, refreshing page, switching to higher or lower quality, screen size alterations, (in)frequent mouse movement, etcetera were analyzed and the authors found that there was a correlation between video impairments and a pause or screen size reduction shortly after whereas other metrics such as mouse movement were less predictable of QoE.

Ghinea et al. [25] investigate the effects of varying network QoS on quality of perception, which encompasses the viewer's satisfaction with the quality, an his/her ability to analyze the informational content of the multimedia. This experiment was conducted subjectively on a group of people using a mean opinion score, by varying the frame rate and colour depth of video while sending the audio in maximum quality. The results have shown that a significant reduction in frame rate does not proportionally reduce the viewer's un-

derstanding and perception of the content, people have difficulty absorbing audio, visual and textual content concurrently and tend to focus on one at a time, when the cause of the annoyance is visible (e.g. lip sync) the users will disregard it and focus on the audio message, and the link between entertainment and content understanding is not direct. Of course, perception and enjoyment are quite different; a viewer may be able to understand what is happening during a video clip but would still find it to be of poor quality and as a result low QoE.

Cranley et al. [15] investigate how users perceive changing video quality during playback. They determine that an Optimal Adaptation Trajectory exists that optimizes user's perceptions of video quality in response to changing network conditions. This is necessary because controlled video quality is needed to reduce the negative effects of congestion while providing maximum video quality. The authors find that the user is more sensitive to the frame rate when there is high temporal information (such as action) in the video content but when the video has low temporal requirement the resolution becomes more dominant. Experiments are run with both subjective tests and objective metrics and it is found that the objective metrics do not correspond to how the users perceive adapting video which suggests that measuring quality and adapting quality based on those measurements are different tasks. The authors consider two dimensions of adapting video: 1) frame rate, which is adapted by frame dropping and 2) resolution, which is adapted by stream switching. They find that there is a mixture of these two adaptations that is optimal rather to achieve a particular bitrate rather than adapting only a single dimension. They also find that users are more critical of dropping video quality than they are rewarding of increasing quality. These findings match those of Pinson and Wolf [52].

Siller and Woods [71] uses agents to control network QoS parameters to allow the users to match video quality to their own quality expectations. Doing this allows users to analyze their own cost-benefit evaluation (being that they pay for improved quality) and match their QoE expectations. By doing so users can choose a lower QoS to match their expectations, thus freeing up resources for others. A user study they found that 80% (of the 20 participants) believed that the quality adjustments were proportional to the price.

Zink et al. [81] conduct subjective tests under several scenarios of varying quality and the results show that stepwise quality decrease was rated slightly better than one single but higher decrease in quality level. Further, users ranked video with fewer quality changes as better than those with more. When given a choice between filling a gap at higher quality or lower quality it is better to fill the lower quality gap. Increasing quality at the end is perceived better than starting with higher quality then dropping. Starting high quality, dropping quality then increasing quality again is perceived as higher than starting low, increasing it and dropping again. When there is a drop in quality it is better to keep the gap small. In contrast to previous experiments where it was found that large changes in amplitude were undesirable, Zink et al. found that when spikes in quality occur, those with larger spikes are favoured. Two major takeaways from this research is that the frequency of variation should be kept small and given the option the lowest quality levels should be improved before the higher ones. An interesting outcome of these findings is that quality improvement may be achieved by transmitting less data if a transmitted chunk fills a low quality gap and it is smaller than

one that would have filled a high quality gap, and this is a case where improving the quality subjectively would possibly show a decrease in objective quality.

Dobrian et al. [18] investigated the impact of five quality metrics, namely join time (initial buffering time), buffering ratio (time buffering/session time), rate of rebuffering events (number of events / session duration), average bitrate, and rendering quality (actual frame rate/encoded frame rate) with two engagement metrics, namely play time and the total plays across a particular viewer. In this dataset a large number of views suffer from quality issues. For example, 7% of views experience buffering ratio over 10%, 5% of views have a join time higher than 10 seconds, and 37% of videos have a rendering quality lower than 90%. The dataset was collected using content providers whose sites consistently rate in the top-500 sites in overall popularity and on average over 300 million unique views and 100 million unique viewers were captured weekly. The research finds that at the view level buffering ratio is the most critical metric and join time becomes critical at the viewer level.

Balachandran et al. investigate turning independent performance metrics such as rebuffering occurrences, startup time, streaming bitrate and number of bitrate switches into a single, quantitative QoE measure [6]. This problem seems straight forward, but issues arise as these metrics often have complex relationships and implicit tradeoffs; the actual content has different viewing patterns (such as live as opposed to on demand) and the users interest effects their tolerance levels. This paper sets out a feasible roadmap towards developing a robust, unified and quantitative QoE metric to address these challenges. Such a metric is beneficial for content providers as they can evaluate cost-performance tradeoffs offered by the CDN, CDN's themselves need to know how to distribute their resources across the user population to minimize delivery cost while maximizing performance, and video player designers have to be conscious of their algorithms to maximize user enjoyment. The authors develop a basic machine learning prediction model of user engagement (play time). They believe this approach is a promising starting point that can lead to the development of a robust Internet video QoE metric.

Perkis et al. present a model for measuring QoE and validated over 4 weeks using a video on demand service on 3G mobile phones [51]. They note that there are measurable (the objective) and non-measurable (the subjective) parameters of quality. Their model determined that availability and reliability of the service are the most important factors and the mean opinion score was highly dependent on both the content and context of the multimedia. For example, news video was rated high quality and acceptable for this type of service whereas sports was rated more poorly.

2.3 Mobile and Wireless Video

Considering mobile technology increases the complexity of video delivery techniques. There are inherent differences between both mobile devices and mobile networks when compared to typical wired devices and networks. Qualities such as the instability in the network and the capability of the device are among the

biggest concerns affecting video delivery. For example, a smartphone connected to a cellular network will have both a more volatile network connection and much less physical memory for buffer space. Not only do these characteristics affect dynamic adaptive streaming over HTTP, but also progressive download. While streaming with progressive download mobile clients open their TCP connection, close it once their buffer is full, then reconnect when more data is needed; this results in a bursty traffic pattern. When HTTP adaptive streaming is considered the limited buffer space and increased volatility in the network mean the client algorithms must be more conservative in their piece selection policy to avoid emptying their buffer during decreases to the bandwidth. Not only do the characteristics of the network and hardware devices differ but the patterns of the viewers do as well. For example, a viewer on a smartphone may just be watching while on public transit or during a short break at work so they quit videos prior to completing them. Also, when a user changes position in the video, such as a rewind operation, the limited buffer at the player requires that the buffer is completely emptied so it may be filled with data from the new position; this means the amount of data transferred can actually be greater than the size of the video after all the discarded buffered data is considered [23].

Gautam et al. [24] investigate the energy and cost savings for both the user and the service provider by prefetching video to the device storage prior to it being played. Incoming [32] is an application that predicts what may be watched based on social network feeds and previous viewing history then downloads the video in advance. Its effectiveness can be measured in 1) efficiency, which is the number of downloaded videos that are viewed and 2) cache miss rate, which is the numbed of videos that are viewed but were not prefetched. Prefetching in this manner is difficult because of the sheer number of available videos, Incoming addresses this by using algorithmic prefetching to get a good initial guess and further manipulating the user to view those videos by displaying prefetched content in a more favourable way than other videos. As a result, the top 10% of users had a 63.8% efficiency ratio, based on videos watched per day over one month using a random sample of 3400 active users. This ratio declines for the users that use the application to view fewer videos per day, resulting in a 30.2% overall efficiency ratio. The advantage of this style of prefetching is that WiFi networks can be utilized and the download can occur outside of peak cellular hours, thus avoiding the congestion at that time and not contributing to it further by consuming network bandwidth to stream the video.

Streaming media applications are extremely popular however they have massive energy requirements as a result of the cellular radio being on for the duration of the stream. Given the limitations of batteries found in cellular phones the energy requirements drastically reduce battery life. The most energy efficient way to view an entire video is to download the entire video at the beginning then allow the network interface to change into a low-power state while it is being viewed. However, many videos are not watched in entirety so this method would result in wasted bandwidth as video would be downloaded but not viewed. To combat this and achieve a balance between buffering and energy efficiency content should be received in bursts. By using traffic shaping and "bursting" traffic over the cellular network Siekkinen et al. were able to save a substantial

amount of energy, up to 21% energy savings for streaming video over 3G and up to 60% energy savings streaming audio over LTE [70]. This is possible because because the time between data bursts is sufficient to switch from the high-power state to a low-power state thus reducing the amount of time the radio is kept on. However, the effectiveness of this technique is limited when content providers transmit background traffic on other connections, which are not bursted as the video content is, which keeps the radio on high power. The results show that the background traffic from YouTube can increase the average current from 2% up to 27%.

Erman et al. provide some insights into cellular video traffic [19]. A 48 hour study in 2011 consisting of approximately three million smartphones and tablets in a large tier-1 network in the United States was conducted to determine the extensiveness of video traffic on cellular networks. It was found that video accounts for 30% of downstream cellular traffic, 36% of that is adaptive video streaming while 60% can be attributed to progressive download, overall 98% of cellular multimedia is delivered over HTTP. They find that 80% of video objects are encoded at low rates, at or below 255 Kbps. Further, for progressive download 40% of videos were completely downloaded, and for 50% of the videos only 60% was downloaded, this doesn't consider actual viewing time because of the buffer. Caching is then considered on the dataset, objects are classified as one of the following, uncacheable, cacheable-local (can cache), cacheable-validate (is cacheable but the object must be validated with server). For progressive download the respective percentages are 8%, 63.2%, and 28.8%, and for adaptive video the respective percentages are 78%, 11%, and 11%. Running a simulation on an unlimited sized cache they find that 23.5% of progressive download data was served from cache, interestingly they note that 4.5% of that is the top-100 videos and 7.7% of that is for the top-1000 videos so a much smaller cache considering only the top 1000 out of the total 1.2 million unique videos would realize a third of the overall caching benefits. HTTP cache control directives set by the content providers often prevent adaptive video from being cached but doing simulation they find that of the 66000 unique adaptive videos the top 1000 and the top 2800 account for 27.4% and 50% of all the adaptive traffic respectively. Therefore, there is potential for significant caching gain provided the caches could handle the encrypted traffic. Caching effectiveness is limited by stale data so the freshness of all video traffic was considered and the results show that a substantial amount of the traffic had very short freshness duration and given that video rarely changes perhaps increasing the freshness duration would be beneficial.

Wireless transmissions experience packet loss for many reasons that are not concerns with wired connections such as encountering obstacles, atmospheric effects and multipath. These packet losses coupled with large RTT's are catastrophic to the achievable throughput. Havey et al. [29] present a client side application layer rate-adaptation mechanism for video streaming over a wireless link. They do so by employing parallel TCP streams and conducting fairness and congestion calculations at the application layer to achieve faster slow start and reduce issues with non-congestion related packet loss. This technique is successful because it is possible to differentiate between packet loss caused by congestion and packet loss caused by the wireless channel conditions thus avoiding the aggressive window reduction behaviour of TCP when unnecessary. Using experimental evaluation the results show that with 8 parallel streams they can get twice the throughput

while maintaining what they refer to as "soft-fairness" which means the parallel streams remain relatively fair to single TCP flows, given that TCP itself is not perfectly fair.

Limited bandwidth is a major obstacle in delivering high quality multimedia so Dastpak et al. use cognitive radio networks with layered encoding to optimize video streaming [16]. Cognitive radio networks allow the radio to transmit over various wireless channels / frequency bands. Therefore, the base layer or necessary low quality video is transmitted over the most stable channel, the primary channel. Then the base station can opportunistically access a secondary channel whenever that channel is idle for transmitting the enhancement layers. Through simulation the authors observe that although their solution is more computationally expensive it achieves a higher quality throughput and relatively low quality variation.

Mobile devices often have multiple interfaces available so they have the ability to use any available network (cellular, WLAN). Unfortunately unless supported by the application the video streaming will stop when there is a handover from one network to the other. Evensen et al. have developed an HTTP adaptive streaming solution for mobile roaming devices that makes use of the benefits offered by multiple links such as the ability to choose the link with the best throughput or conduct bandwidth aggregation and use both links simultaneously [21, 22]. This idea is used to investigate improving QoE of video streaming in a mobile environment by running experiments along a known and common commute tram route in Oslo, Norway [22]. Essentially the client communicates with a proxy that is populated by crowd sourcing with bandwidth estimates for a particular network at GPS locations along the commute path. The client then predicts which quality levels to request based on the data collected at the proxy to avoid issues with fluctuating bandwidth and network outages. Their solution also involved seamlessly switching between available network interfaces, such as from cellular to WLAN when multiple interfaces are available to maximize throughput. This was done on the client by creating a virtual interface controlled by a user space application that then chooses which actual network interface to use. Further, a simulation was conducted where the available interfaces were aggregated for maximum throughput. Users could imagine using this technique on a cost optimization basis, for example to utilize the links that are the least cost along the path or on a performance maximization basis to increase video quality. One down side of such a system is that it considers only fixed path commute routes and not those where a user could be in the passenger seat of a car. A similar GPS-based prediction service is presented by Riiser et al. [60].

Evensen et al. explored the concept of bandwidth aggregation for video streaming [20]. HTTP video fragments are chunked as in stream switching paradigm but the chunks are further divided from each of the chunks into subsegments. The subsegments could then be requested over multiple interfaces (e.g. WLAN and HSDPA). By allowing the subsegment size to be dynamic it is possible to overcome the link heterogeneity without a large buffer at the client. Basically if a slow interface is allocated a large portion of the chunk the client needs to wait for that interface to finish resulting in inefficiencies. Using dynamically sized subsegments (HTTP range requests) and only sending what a particular interface can handle improves over the static subsegment sizes when you have a smaller buffer, which is necessary to maintain a level of liveness.

2.4 Adaptive Video Streaming

Adaptive video streaming technique are commonly used today because there are disadvantages to progressive download such as wasted bandwidth as a result of viewers quitting the video early, not being sufficiently adaptive to network conditions, and not supporting live video [73]. As a result, much of the video delivery industry has begun to utilize HTTP adaptive streaming. There are several commercial solutions such as those made available by Netflix, Move Networks, Microsoft, Hulu, Apple, Akamai, HBO, Amazon, and Adobe. As well as open source solutions as made available by Adobe with its Open Source Media Framework. Standardization has taken place by 3GPP with the 3GP-DASH Standard [1] and MPEG with the draft MPEG-DASH solution [72]. However, these standardizations do not address the client adaptation logic, which leaves it open to industry competition [77].

A good rate-adaptation algorithm should be able to quickly achieve a proper steady state, that is, the player should choose the appropriate version rather than oscillating between quality levels, as frequent changes to the video quality are annoying to viewers [22, 36, 46, 59, 80, 81]. It should allow the player's buffer to absorb short-term bandwidth variation and it should elegantly adjust to long-term changes in available bandwidth rather than making a large quality changes [46]. Essentially, an algorithm that quickly stabilizes at the appropriate version and has smooth transitions when changing quality levels is necessary.

2.4.1 Scalable Video Coding

As discussed in the introduction chapter, scalable video coding (SVC) is another video delivery technique where the video is encoded as a hierarchical set of layers where there exists a base layer of low quality video and enhancement layers to increase the quality of video. These layers are split up by frame rate, resolution, or quality of video signal. In HTTP adaptive streaming there may be a significant amount of redundancy at the server as the video is reencoded at various bitrates. However, SVC layers simply add to the video quality, resulting in much less redundancy, which reduces storage requirements at the servers. SVC also allows for an easy transition down in quality if the available bandwidth drops as enhancement layers can simply be lost. However, SVC is still very complex and imposes codec restrictions so it has not been widely adopted [7]. The scalable video coding extension of the popular codec H.264/AVC, H.264/SVC aims to overcome its complexity shortcomings and has successfully shown improvement. Kuschnig et al. [40] evaluate three rate control algorithms and find that both bandwidth estimation algorithms and priority-deadline approaches are robust in terms of their video quality.

The concept of layered encoding video is not new. Rejaie et al. address the problem of adapting the compression without requiring servers to re-encode data while fitting the resulting stream into rapidly varying available bandwidth in their work published in the year 2000 [59]. The authors address the problem using a server side push paradigm of layered video adaptation. The presented system controls congestion at the application layer using AIMD similar to TCP to remain fair to TCP flows. Considerations of video quality

that were observed in 2000 are still major concerns today. For example, in the scheme presented in this paper a key feature is the ability to trade short-term improvement in quality level for long-term smoothing, addressing the undesirability of frequent quality changes. Two requirements must be met before a new layer is added to the video stream, first the instantaneous bandwidth must be greater than all current layers and the new layer combined, and second, there must be adequate buffering at the receiver to survive backoff and continue playing all the existing layers and the new layer. When backoff occurs the server must consider dropping a layer and it iteratively drops the highest layer when the amount of video buffered at the receiver is less than the amount estimated to be needed to recover from the backoff. Care must then be taken in bandwidth allocation for the recovery of the stream as there is dependence between the layers to play the video. This is done at the server as it can control the bandwidth share on a per layer basis meaning that if the receiver has excess of a particular layer buffered then the server can temporarily assign less of the available bandwidth to that layer. The server can dynamically adjust the distribution of buffering using the available bandwidth.

One advantage that SVC may have on stream switching is with respect to potential caching efficiency [63]. They find that SVC does offer some improvement because when there is a cache miss for layer number k, then it is possible that layers 1 to (k-1) are in the cache, and only an enhancement layer will be necessary to retrieve from the server where in the case of stream switching the client would need to get the entire chunk from the server even when lower quality versions of the same chunk number are cached. By only needing an enhancement layer fewer network resources are used on the path to the server thus limiting congestion. Mocha [58] is a similar system for layered video caching that, on a hit if the quality of cached video is lower than the available bandwidth to the client then Mocha can prefetch the missing parts of the stream from the server so they can be merged to send to the client. However, as will be discussed in Chapter 5 the potential benefits of general caching for commercial video streaming like Netflix in the Internet today is low.

Andelin et al. believe the benefits of SVC outweigh their costs so it holds merit for research with adaptive streaming [5]. The authors suggest that rather than keeping multiple versions of a particular video at the server, servers could couple SVC with the adaptive streaming paradigm. With this technique the clients decision moves from deciding which version to request next to either adding more low quality chunks to the buffer or requesting enhancement layers for the chunks currently in the buffer to increase their quality. This way rather than having the client guess about which bitrate it should download, it can incrementally increase the quality of video. However, this solution does not make the client logic trivial, as the issue then becomes whether to be aggressive and improve the quality of chunks that are already in the buffer or act more conservatively and add more base chunks to the buffer. If the bandwidth is constant a vertical scheme where the player always goes for the best quality on the current piece is optimal because the chunks are relatively the same size and the quality will be as high as possible, while remaining stable throughout playback. However, it takes a significant risk, if the network conditions degrade the player will be stuck with minimal buffered content. A completely horizontal scheme is good in the sense that it takes the least risk in

incurring pauses for rebuffering but it requires a huge buffer and takes more time to get to the appropriate quality that the network can withstand. A diagonal scheme involves alternating between prefetching and backfilling based on the slope of the diagonal, so a steeper slope will favour backfilling and improving quality while a shallower slope will favour prefetching. Choosing the optimal policy depends on the conditions of the network and hardware involved. The quality selection policy must balance the tradeoff between providing high quality video and ensuring future consistent quality so flatter slopes are required when the available bandwidth is either low or highly variable because playing it safe and prefetching ensures a minimum quality for future video.

There is extreme variation in cellular networks. Factors such as the type of cells, the users distance from the cells, and the load on the cells all contribute to this characteristic and instantly reflect on the quality of streaming video. Rath et al. investigated using a strategy for streaming video which involved both downloading content in advance and streaming content which was beneficial when the network connections were too weak or variable to sustain streaming the content [57]. A problem arises in the legal copyright and management issues which prevent content providers from using a downloading or renting service (such as iTunes) in a cost effective manner. The key feature that separates a streaming service from a downloading service is a continuous connection between the server and the user while the video is being viewed. Streamloading is a technique that allows users to potentially enjoy download quality videos while still legally being a streaming service, so provider and consumer costs can remain lower. This is accomplished using SVC by allowing users to download the enhancement layers in advance of the playback when excess bandwidth is available, while streaming the base layer. This keeps video quality up as the users available bandwidth degrades as he or she moves away from the base station because only the base layer needs to be streamed. The enhancement layers cannot be used without the base layers so a streamloading service legally qualifies as a streaming service. A potential disadvantage is that if the video is quit early all those enhancement layers are wasted bandwidth and video download is typically ten to a hundred times more expensive than a streaming service. A simulation study is done using a network with simulated cellular towers and mobile clients.

2.4.2 Client Side Improvements

Just relying on real-time bandwidth estimation is not enough for dealing with wireless networks because their volatility. The optimization by Qiu et al. [55] rely on the real-time bandwidth measurements as well as historical measurements and the length of the client buffer. Simulations of this optimization over Smooth Streaming show reductions in the stall time and number of stalls with increases in the peak signal to noise ratio.

Mok et al. presented a video system built from two components [46]. First was a bandwidth estimation component where a probing methodology is used to determine the maximum bitrate that can be supported with the current network. Second, a QoE feature was used to request an intermediate quality rather than jumping to the target bitrate to avoid sharp video quality degradation. A single intermediate quality was

used and the number of intermediate chunks that can be downloaded before switching to the target rate was determined as a function of the fragment length, the buffer capacity, and the time available to download those chunks. Subjective experiments find that by injecting a single intermediate quality level there are increases to the QoE while a longer buffer and thus a longer duration at this quality also increases QoE.

Miller et al. present a client side rate-adaptation algorithm with the following goals: (1) avoid buffer underruns, (2) maximize minimum and average video quality, (3) minimize the number of quality shifts, and (4) minimize the time after the request to view the video and the start of playback [45]. They recognize the tradeoff between the second and third goals and the second and fourth goals. An attempt is made to balance the second and third, while resolving the second and fourth by starting at the lowest quality level but ramping up fast. After the fast ramp up the algorithm is heavily dependent on the client's buffer capacity. If the buffer is in some target capacity then there are never any quality changes. This absorbs short-term bandwidth variations and limits frequent variations. Increases only occur when the buffer reaches some set capacity, and the observed throughput is large enough to warrant an increase, otherwise a delay is added to the requests to reduce the buffer capacity preventing positive spikes. This happens slowly to minimize delays and avoid fluctuations with competing players. To handle decreasing throughput the player reduces its selected quality level if the buffer capacity drops below some set threshold, and continues to drop the quality level if it remains below that threshold and the throughput is below the current bitrate.

Liu et al. present a client side rate-adaptation algorithm [44]. The purpose of the algorithm is to differentiate between short-term throughput variations and variations incurred by congestion control. This paper estimates the available bandwidth based on the chunk fetch time. It is able to determine a smoothed TCP throughput value by taking the ratio of chunk play time over the chunk fetch time, then multiplying by the current chunk bitrate. By choosing a chunk size of 10 seconds this provides adequate smoothing. The algorithm implements a step-wise increase and aggressive decrease function. If the smoothed throughput is larger than some factor, which is a function of the available bitrates, then the next highest bitrate is chosen. If the throughput is observed to be lower than capable of sustaining the current bitrate, then the quality level drops to a bitrate that would be sustainable. There is also a drop function based on the current buffer size, if the buffer goes below a threshold it will reduce quality. Through simulation the authors find that their algorithm not only quickly and accurately reaches the ideal bitrate but efficiently detects congestion and probes for spare network capacity.

Tian and Liu investigated the tradeoff between buffer occupancy stability and quality level stability, essentially the tradeoff between responsiveness of the player and smoothness of the payback [76]. This is done through the development of a client side algorithm and implementation of this algorithm both in lab simulations using dummynet and in a real world Internet experiment. To control buffer oscillations they use a Proportional-Integral controller. By precisely maintaining a buffer occupancy the viewer is exposed a potentially unstable video rate due to network variations. Thus, the authors control video rate fluctuations. To accomplish that they use a TCP throughput estimation together with an adjustment factor, which is a

function of the target buffer size, current buffer size, previous buffer size and the current video rate. A rate switching logic module is then added to smooth the playback. This module uses the buffer occupancy as an indicator of when to switch down (if at half the target occupancy then switch down), otherwise they check the target video rate as previously determined, if it is higher than the previous chunks video rate a counter is incremented, if it is not the counter is set to 0. When the counter hits a certain value (a parameter which controls the tradeoff between responsiveness and smoothness) then the player increases video quality.

2.4.3 Network Support

Benno et al. [8] conduct experiments using the Microsoft Smooth Streaming player and find that the player is particularly sensitive to the RTT variability for chunk requests. The use of caching can greatly improve the quality of video being requested but it can also introduce quality level oscillations. For their experiments a short six minute video (consisting of seven available bitrates) was played, consecutively by five players through a cache of infinite size. In this experiment a cache hit has 50 ms delay, a miss has 300 ms delay and the five players achieved the following cache hit rates 0%, 11.7%, 38.9%, 81.1%, 93.3%, respectively. However, the added delay of cache misses or a couple strategically placed cache hits was enough to cause oscillations in the quality of requested video. The prefetching proxy proposed in this thesis is an interesting solution to this problem as predicting the appropriate chunks and prefetching them at the proxy rather than relying only on content that has already passed through a cache will greatly reduce the probability of a miss and allow for a high hit efficiency without the need for the video chunks to have previously passed through the proxy.

Liu et al. investigate the use of parallel HTTP requests to fully utilize the distribution network resources in a CDN [43]. By requesting fragment n before fully receiving fragment n-1 fragment n has more time to download than it would in traditional HTTP adaptive streaming as different servers can be used to retrieve the data to cope with their inefficiency. Caution is used when determining how many requests to send in parallel as sending too many may result in wasted content if the video is quit early and a bottleneck may occur on the gateway link, which hurts the earlier requests still in the pipe. Krishnamoorthi et al. [39] conducted work in parallel with this research that focused on prefetching combined with caching at a proxy rather than a prefetching only context as considered in this thesis.

Gouache et al. look at using multiple CDN servers concurrently to improve quality and reduce quality variations for a better user QoE [28]. They propose that chunks are split up using range requests so a part of each chunk can be requested from each server, and the portion of the chunk requested from each server is determined by the estimated bandwidth on that path. This way if one server experiences difficulties then the range that was requested from that server can be redistributed to other servers prior to needing the data. The authors note that the more paths there are the easier it is to absorb the impact of a single path congestion as it represents a smaller proportion of the total bandwidth. But they note that simply having two paths is a significant improvement and adding more is likely not practical in the real world. The authors

conclude that using multiple paths increases the average bitrate and reduces the variance over a single path thus improving QoE and this is achievable without additional buffering whereas a single path would typically require a significant amount of buffering to ensure continuous playback.

2.4.4 Server Side

Summers et al. discuss designing a set of benchmarks that will evaluate an HTTP streaming server [74]. They evaluate the server as it is the most central and performance critical portion of the HTTP ecosystem and the authors find that there is a disk performance bottleneck. They also investigate how videos should be stored on the web server to maximize server throughput [75]. Essentially, should the videos be saved as many files (chunks of video) or should the video be stored as one file and accessed with range requests. An evaluation of servers is conducted using three conventional web servers (userver, nginx, and Apache), in order to be sure experiments are fair and accurate they examine the physical location of each file on the disk and place the chunked/unchunked video in the same locations. The author's experiments conclude that the performance is similar when using either approach. However, by aggressively prefetching and sequentializing disk accesses in userver double the throughput was obtained when using unchunked video when compared to chunked video. These findings do not mean that all servers should be changed to obtain this performance benefit as there are issues that must be considered outside of the performance of the server. For example, it is unclear how widespread caching is for range requests, if caches cannot handle range requests then that is an advantage of storing individual chunks which may outweigh the server throughput obtained by prefetching range requests.

De Cicco et al. [17] present a Quality Adaptation Controller for live video streaming that uses a HTTP stream switching bitrate adaptation. It is designed to maximize QoE, to have a rigorous controller by applying control theory, to have high scalability, to be CDN friendly, and to be codec agnostic. The controller employs feedback control theory to throttle video level without using any heuristics. The controller is on the server so the control loop has no delays and requires no feedback from the client. The controller takes as input the server side send buffer length and its goal is to keep that queue full. Experimental results show that the player quickly converges to its fair bandwidth against either a greedy flow or another player.

Akhshabi et al. investigate player instability that results from competing players ON-OFF behaviour in steady state causing quality level oscillations [4]. Essentially when two players are sharing a bottleneck and the ON and OFF periods that are used to maintain a target buffer occupancy do not overlap they will overestimate bandwidth and request a higher quality, then their periods will overlap and they will not be able to sustain the higher quality. To solve this issue, the server detects oscillations that are a result of ON-OFF periods and shapes the outgoing traffic to eliminate the OFF period. The player determines which level to shape at by taking the highest level in the oscillation and subtracting one, then continuing to check for oscillations. Periodically the server will turn off the shaper to see if more bandwidth has become available and set the traffic shaper to a higher level; that way the client is not stuck at a low level if available bandwidth improves while traffic is being shaped.

2.4.5 Player Evaluations

Although there have been efforts to standardize much of HTTP adaptive video streaming the adaptation logic remains open for industry competition [77]. Because of this service providers have players that behave differently in response to the underlying network. As such there has been research comparing and contrasting the players currently available.

Riiser et al. compare four HTTP adaptive streaming clients in mobile scenarios across three metrics, robustness against underruns, stability in quality and bandwidth utilization [61]. Experiments are conducted using a realistic mobile 3G environment and compare the results of several players. Their findings conclude that Apple tends to be very cautious favouring stability over higher quality while Adobe tends to favour high quality with little regard for stability or safety. Microsoft uses more available qualities than Apple but still remains fairly stable unlike Adobe, which has a small buffer making it more vulnerable to buffer underruns than Netview. Netview behaves differently at high buffer capacity than low buffer capacity; at high buffer capacity it prioritizes stability over fullness while at low capacity it favors robustness against underruns over stability. To be deterministic in their tests their server takes in a bandwidth log built from a real world test.

Akhshabi et al. [2] conducts an experimental evaluation of three HTTP adaptive streaming players. Namely, Microsoft Smooth Streaming, Adobe OSMF, and the Netflix player are evaluated. Their experiments involve characterizing the behaviour of the players to answer the following questions: 'How does the player react to persistent or short-term changes in available bandwidth?', 'Can the player quickly converge to the maximum sustainable bitrate?', 'Can two players fairly share the bottleneck link?', and 'How does adaptive streaming perform with live content?'. The authors identify several shortcomings in the rate-adaptation logic of the players for which experiments were done. For example, they found that it was often the case that when the Microsoft Smooth Streaming player adjusted to a bandwidth change that it would take too long to adjust back when the change was reverted. This resulted in the players playing a lower quality video for longer than necessary. The authors noted several of the shortcomings to Adobe OSMF player, most importantly it could not reach steady state so oscillations were prevalent. The work in this thesis similarly evaluates the Netflix player, which involves conducting similar experiments but using multiple platforms.

2.4.6 Competing Traffic

Huang et al. investigate the bandwidth estimation for three popular video streaming services, Hulu, Netflix, and Vudu [31]. However, their goal is not to compare the players but to show how the players struggle to choose the appropriate bitrate when competing with other traffic. They find that all three players play relatively well when the home network is quiet but all three players perform poorly when competing with other traffic on the link. The authors attribute this to the difficulty in estimating the fair share of throughput. The players all underestimate bandwidth because of interactions between the video playback buffer, HTTP and TCP's congestion control algorithm. The player starts downloading the video at the appropriate rate

when it is alone in the network but as soon as the competing flow begins the player picks a bitrate that is far below its fair share. The authors find that the bitrate the players choose is quite closely correlated to what they believe the players are using as their estimated throughput. Thus the problem is recognized as poorly estimating the throughput available to the client. When conducting the experiments to be sure the competing flow is fair they simply use traffic from the same CDN/Server so the RTTs are the same. Then the entire video is downloaded at its fair share bitrate rather than individual HTTP chunks, thus ensuring that they are doing analysis against competing flows in general rather than competing players. In some cases the ON-OFF periods (requests being downloaded then no network activity as the player waits to send its next request to avoid downloading too quickly) are the problem as the OFF periods put TCP back into slow start and the client does not estimate a high throughput. This is made worse when the chunk finishes downloading prior to TCP exiting slow start. TCP's trouble in achieving its fair share of bandwidth results in requests of a lower quality, which gives a smaller chunk size, so TCP ramps up even less. These ON-OFF periods can be at the HTTP level, so the player periodically paces its requests, or it can occur at the TCP level, so the player stops reading from the TCP buffer when its playback buffer is full causing the TCP socket buffer to fill up and advertise a zero window. Either way, the ON-OFF periods do not appear to be an issue in the absence of a competing flows. The issues are mostly the result of the player's conservativeness. For example, if it estimates bandwidth at X it only plays a bitrate lower than X to be safe and competing flows make it see even less. The authors find that service A requests video with a conservatism of 40% (if it perceives 2.0 Mbps it will download at at most 1.2 Mbps). The authors suggest grouping video chunk requests so TCP has a chance to reach its steady-state fair share but alternatively they propose that perhaps these players should not try to estimate bandwidth at all but instead rely only on buffer capacity. The player could increase quality when the buffer is high and decrease quality when the buffer is low. This would help avoid the feedback loop that occurs when the client starts underestimating the available bandwidth in the presence of the competing flow, then reduces it bitrate further as a result of its conservativeness, which results in the segment sizes being smaller causing the flow to become more susceptible to receiving lower throughput.

Akhshabi et al. [3] hypothesize and evaluate three performance problems: instability, unfairness, and bandwidth underutilization, which the authors believe are impacted by the player's respective ON-OFF periods of downloading with competing players, rather than only flows [31]. Instability is the fraction of successive chunk requests for which the bitrate does not remain constant. Unfairness is difference in bitrate between the corresponding chunks requested by multiple players. Finally, underutilization is the aggregate throughput over the available bandwidth. The player's respective ON-OFF periods are believed to be a concern because players do not estimate bandwidth when they are not downloading therefore the timing between the player's ON periods has an impact on how they see the network. When the ON periods of the players do not overlap the players overestimate their fair share. They may then request higher bitrate causing congestion, then a lower bitrate again, this continues causing instability. If the ON period of one player falls completely within the ON period of another (that has a larger ON period). The player with the longer ON

period will over estimate its fair share. It is possible then that this stabilizes and the player with the longer ON period will get better quality than the player with the shorter period, causing unfairness. When the ON periods are aligned both players will estimate half the bandwidth, which is fair but it may be underutilizing the link if the available bitrates are less than and greater than half the links bandwidth capacity as now both players will request less than the links capacity. Instability can also cause underutilization when both players oscillate onto a low quality. Using lab experiments the authors found that fairness and utilization were not that bad but instability was problematic. They chose to look at three factors which contribute to instability, those being the relative duration of the OFF periods, the fair share of each player relative to the bitrates available and the number of competing players. They find that longer duration idle period, less overlap of ON periods, and a medium number of players (to give the most options for bitrates per player) all lead to a potential increase in instability.

Similar to Akhshabi et al. [3], Jiang et al. [36] investigate the issues of today's commercial players with respect to three major metrics, efficiency (the ability to choose the highest feasible bitrate to maximize QoE), fairness (the ability for multiple players to converge to an equitable allocation of network resources), and stability (the ability to avoid frequent bitrate switches as that negatively impacts QoE) [36]. In this work the authors suggest that the design of a robust adaptive video algorithm must look beyond the logic of a single player and account for the interactions across multiple streaming players that compete at bottleneck links. They observe that the ON-OFF periods that result from periodic chunk scheduling used with stateless bitrate selection (selecting only based on the previous chunks throughput) can lead to undesirable feedback loops with bandwidth estimation and cause unnecessary bitrate switches and unfairness in the choice of bitrates. The performance of Smooth Streaming worsens under all three metrics as the number of players on the bottleneck link increases and similar trends were noted for other players. The authors implement their solution known as FESTIVE using Adobe OSMF and show that it has low overhead and improves fairness by 40%, stability by 50% and efficiency by at least 10% compared to the closest alternative. This is compared against some of the existing commercial solutions, Microsoft Smooth Streaming, Akamai HD, Netflix, and Adobe OSMF. They find that their solution is more robust to bandwidth variability and the improvement with FESTIVE increases with higher variability. In a heterogeneous environment FESTIVE is more stable than the other players and spends most of the time at an efficient bitrate. The recommendations are the following: (1) randomize chunk scheduling to avoid synchronization bias between players sampling the network; (2) use a stateful bitrate selection that compensates for the biased interaction between bitrate and estimated bandwidth; (3) use a delayed update approach to tradeoff stability and efficiency; and (4) use a bandwidth estimator that uses the harmonic mean of throughput over recent chunks to be robust to outliers. To implement the four suggested recommendations the authors point out that HTTP adaptive Streaming involves three components: (1) scheduling when the next piece will be downloaded; (2) selecting a suitable bitrate for that piece; and (3) estimating the available network bandwidth. With respect to scheduling the next chunk's download there are several possible policies. The player could use an immediate greedy

download policy where the piece is downloaded immediately after the previous has finished but this is a poor choice because bandwidth is wasted if the player is closed early and it is not as adaptive because low bitrate chunks may already be downloaded by the time the network improves. Instead, periodic chunk download could be utilized when trying to maintain a buffer size; if a chunk requires 4 seconds to play, downloads will be scheduled every 4 seconds. However, with this policy may see a biased view of the network as a result of synchronization with other players. Using randomized scheduling has some random time after a chunk has been completed before the next chunk is scheduled, which allows the player to alleviate the majority of the concerns brought on by greedy and periodic downloading. The buffer stays relatively close to a target while sampling the network at various times to avoid synchronization with other players. When considering bitrate selection they use a stateful system where the rate of increase is a monotonically decreasing function of the bitrate so the current bitrate is taken into account when choosing the next bitrate; essentially, the quality ramps up faster at lower bitrates than at higher but quality can drop as soon as it is found the network cannot support its bitrate. They also only increase quality one step at a time to prevent large jumps in quality levels. Further they suggest a delayed update, which is a measured tradeoff of efficiency/fairness and stability for bandwidth estimation rather than using instantaneous throughput. A smoothed value over the last several chunks (20) is used to keep streaming robust to outliers and the harmonic mean is used rather than the arithmetic mean.

Houdaille and Gouache design a bandwidth manager in the home gateway to implement bandwidth arbitration between competing players to solve an issue regarding multiple players not fairly sharing a bottleneck link but rather "fighting" over it [30]. They determine that the home gateway is a likely place for this "fighting" to occur and it can also get a good view of the home network and identify competing video streams. They report that their implementation allows significant improvement in bandwidth sharing, stability and convergence when two players are concerned in an experimental setting with the Microsoft Smooth Streaming player and traffic shaping keeps players within two quality steps. They suggest that future work will involve scaling the research to more than two players and doing more research with different competing players (e.g. not just Microsoft Smooth Streaming). A solution such as this relies on the gateway intercepting the manifest.

2.5 Chapter Summary

This chapter has presented a detailed description of previous work in the performance of Internet video streaming as well as the relevant background information needed for this research. It considered a general look at the underlying network, QoE, mobile and wireless video as well as a specific look at adaptive video streaming.

Chapter 3

Experimental Methodology

This chapter presents the methodology for conducting the research in this thesis. First, the hardware is described, including which devices are used, as well as their specifications and roles in the research. In order for a proxy to be capable of prefetching, the Netflix video delivery methodology must be understood. The HTTP request message structure for video chunks is described, along with the construction of the player's video manifest for the prefetching proxy software. Next, a description is provided of the software portion of the research. The design of the prefetching proxy software is discussed, including a detailed description of its parts and explanations for particular design decisions. Other software is utilized during experimentation for traffic manipulation, controlling the experimental environment, as well as for data collection and analysis. The software packages used for these tasks and how they are used is described. Finally, the plan for the experimentation is described including the controlled factors and the evaluation metrics to determine the utility and impact of the prefetching proxy implementation.

3.1 Hardware

Listed below is each piece of hardware, its purpose, its specifications, and the relevant software installed.

The Proxy:

- Device: Desktop PC
- Purpose: Running the proxy software, acting as a network bridge, controlling the network conditions, capturing packet traces
- Operating System: Ubuntu Release 12.04 (Precise Pangolin) 64 bit
- Processor: Intel Core 2 6600 at 2.40 GHz x2.
- Linux kernel version 3.2.0-32-generic
- GNOME 3.4.2.
- RAM: 2 GB DDR2 800 (PC2-6400)
- VPN: OpenVPN 2.2.2

The PC Browser Client:

- Device: Desktop PC
- Purpose: Netflix Client
- Operating System: Windows 7 Professional SP1 32 bit
- Processor: AMD Athlon 64 X2 4000+ processor at 2.10 GHz
- RAM: 4 GB DDR2
- Browser: Mozilla Firefox 22.0
- VPN: OpenVpn 2.2.2

The iPad Client:

- Device: iPad 1
- Purpose: Netflix Client
- Model: MB292C
- Version: 5.1.1 (9B206)
- Netflix Version: 2.4.1 (1549684)
- Netflix SDK Version 2012.1
- VPN: GuizmOVPN 1.1.7-1

3.1.1 Experimental Platform

The individual pieces of hardware are linked together to form the test bed for the experiments as shown in Figure 3.1. The client is the device running the video, either the iPad with the dedicated Netflix application (referred to as the iPad) or the PC with the web browser at the Netflix website (referred to as the PC Browser). There is a VPN service on the client and the proxy so all traffic from the client is directed to the proxy machine. After arrival at the proxy machine the traffic is manipulated in such a way, by using IP masquerading on all incoming traffic, so that the traffic appears to have been created by the proxy and not the client. This ensures that all response traffic will come back through the proxy rather than taking an alternate path directly back to the client from the server. IP masquerading essentially allows the proxy to appear to be the client from the perspective of the server while allowing it to appear invisible from the perspective of the client. The HTTP traffic is further manipulated as it is hijacked at port 80 of the proxy so it can be redirected to the port the prefetching proxy software is listening on. After the traffic leaves the proxy machine it goes over the University of Saskatchewan Ethernet network to the Internet. To effectively control experimental conditions dummynet is used on the proxy, which allows for control of the available bandwidth levels. Packet traces are captured on the proxy for offline analysis using Wireshark. Further discussion on the software involved on the proxy machine can be found in Section 3.3.

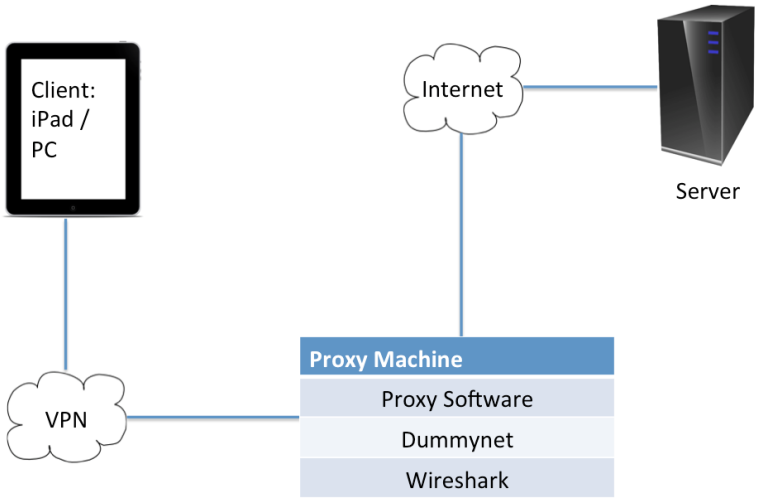

Figure 3.1: Experimental Test Bed Layout

3.2 Netflix Video Delivery

Netflix uses the HTTP protocol to allow the player to make requests for video data from the server. Below are example HTTP requests for chunks of video data from the PC Browser player and the iPad player.

The following is a PC Browser HTTP video chunk request:

```
GET /691897732.ismv/range/172073487-173312228?c=ca&n=22950&v=3&e=1372904686&t=
    YCaHOfRabGO_R1R4X5ODgGq2lBw&d=silverlight&p=5.1GoywvP0U1YDQSePaYOU-
    PVV29LRvZ29WTZVmth_nEQ&random=1669964921 HTTP/1.1
Host: 198.45.53.138
User-Agent: Mozilla/5.0 (Windows NT 6.1; rv:22.0) Gecko/20100101 Firefox/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
```

The following is an example iPad HTTP video chunk request:

```
GET /670543564.ts.prdy/range/26184864-26867503?c=ca&n=22950&v=3&e=1373171275&t=1Pp1ErE3-02
    tLiL5epUQW2vxQe0&d=ipad&p=5.GIcu00xu5UAgWk_a7CqNngx4uA0wq9EOIThtKK1_KSM HTTP/1.1
Host: 198.45.52.145
User-Agent: AppleCoreMedia/1.0.0.9B206 (iPad; U; CPU OS 5_1_1 like Mac OS X; en_us)
Accept: */*
Accept-Encoding: identity
X-Playback-Session-Id: 9C20F215-30CA-4EF8-99CC-A988BD6CE386
Connection: keep-alive
```

The formatting of the PC Browser's request and the iPad's request is similar. In both requests the GET method parameter starts with the quality level (version) identifier number to identify the video's file, *691897732* and *670543564*, respectively, followed by the file types *.ismv* and *.ts.prdy*, respectively. The version identifier and file type are followed by the byte range, which indicates the position in the file for that video chunk, *172073487-173312228* and *26184864-26867503*, respectively. The remainder of the GET method parameter provides information to the server based on the current session.

3.2.1 Building the Manifest

The manifest is a file that defines the communication between the client and the server with respect to which video chunks can be requested by a particular client's player to playback a particular video. Because the players are different, the iPad and PC Browser each require their own manifest. Having the manifest is necessary at the proxy so it is able to identify the byte ranges of the chunks for prefetching future chunks in advance of the client requesting them. The manifest for the prefetching proxy includes multiple files for each player. Each player's manifest has a single file for each quality level of video it may request and these files include the byte ranges for each video chunk as well as the approximate number of bytes expected back in response to each request. The number of bytes expected in the response could be estimated by guessing the size of the HTTP header, or by reading the header and payload individually and using the Content-Length field. The number of bytes in response to a request is not necessarily the same on subsequent requests of the same chunk, there is some variation on the order of a few bytes, possibly due to encryption. The size of the response is important for the prefetching agents to know when they have received the entire chunk from the server after requesting it, so as to avoid blocking to read more data after all the data has been read. Rather than guessing the HTTP header size or reading the header and payload separately this proxy simply includes the total number of bytes expected back in its manifest.

The manifest is not made available by Netflix so for the proxy to determine the appropriate range requests for each chunk in the video the manifest files were constructed using an initial set of experiments for a particular motion picture. Multiple steps were needed in constructing each manifest for the proxy machine. The first step was to accumulate the chunk requests from each quality level of the video for both players. This was done using brute force by playing the video several times and controlling the quality level being requested at any given time during playback by manipulating the available bandwidth with dummynet. This is non trivial as the video is variable bitrate so the version being requested can fluctuate during the video playback even when the available bandwidth is constant. Also, the players appear to have different levels of aggressiveness in choosing which quality to play at a particular available bandwidth depending on the place in the video; for example, the PC Browser player chooses higher qualities at later stages in video playback. Further, the Netflix rate-adaptation logic also plays a role in aggressiveness, potentially considering factors such as buffer occupancy. However, with careful manipulation of the available bandwidth it was possible to capture the range for each chunk. The chunks were ordered by their place in the video and it was confirmed

Table 3.1: Chunk File Excerpt

0-187
188-29515
29328-1037951
1037952-1669823
1669824-2070079
2070080-2248495
2248496-2328399
2328400-3274991
3274992-4455823

Table 3.2: Finished Manifest Excerpt

0-187,667
188-29515,29809
29328-1037951,1009105
1037952-1669823,632352
1669824-2070079,400737
2070080-2248495,178897
2248496-2328399,80385
2328400-3274991,947076
3274992-4455823,1181314

that the chunks were in the correct order for a particular quality level by ensuring that the byte ranges in the requests follow each other. An example of two chunks that appear one after the other is the chunk with the byte range *29328-1037951* and the chunk with the byte range *1037952-1669823*. An excerpt of the iPad chunk byte ranges for a single quality level can be seen in Table 3.1. After all of the byte ranges were determined the number of bytes returned in response to each request was established. To do this HTTP requests were constructed to send the video chunk requests to the Netflix server while keeping track of the total number of bytes in the response. Netflix HTTP requests become invalid after a certain time so HTTP requests were created by playing a portion of the video to capture an HTTP request for each of the quality levels, and then using those requests into the manifest builder software to be used immediately. The manifest builder software is described in Section 3.3.3. This software takes the previously gathered chunk byte ranges and determines the number of bytes to be expected in the response, completing the manifest files. See Table 3.2 for an excerpt of the beginning of the manifest for the highest quality version of the iPad player; the first section of each comma-separated line is the range of the request and the second is the actual number of bytes received from the server in response to the range request.

Note that the first two byte ranges in the manifest shown in Table 3.2 do not align with the third; it is likely that these two requests are not actual video data but instead meta information. It is believed that this is the case because the ranges in these requests are the same for each quality level and these requests are made for every quality level before the first chunk of video data can be requested regardless of the available bandwidth. Similar meta information chunks are found in the PC Browser's manifests, the difference being the PC Browser player has four options for the meta information chunk (*0-41711*, *0-41715*, *0-42143* and *0-42827*), but only one meta information chunk per quality level is requested during each playback rather than two as seen with the iPad and only a subset of quality levels are requested initially before the data chunks start to be requested. However, if the PC Browser player switches to a version that it has not already requested the meta information chunk for, it must first send the request for the meta information chunk for that version before sending the request for the data chunk.

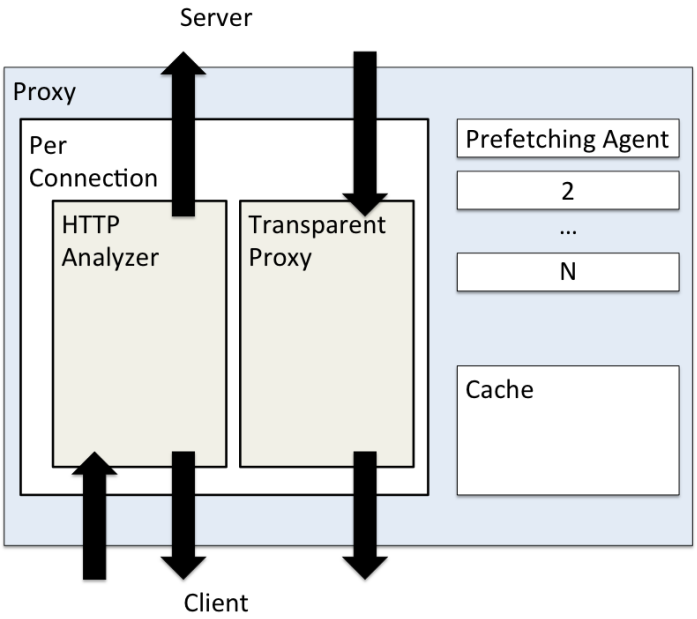

Figure 3.2: High-Level View of the Proxy Software

3.3 Software

This section describes the software used on the proxy machine. First, the actual prefetching proxy software for working with Netflix is described. Then the methods for manipulating the traffic at the proxy machine so it may to work seamlessly between the client and servers are covered. Finally, the tools that were necessary to control the experimental conditions as well as collect and evaluate the experimental results are described.

3.3.1 Prefetching Proxy Software

The proxy software is coded in the Java programming language. An abstract view of the software contains four parts: *HTTP analyzers*, *transparent proxies*, *prefetching agents*, and the *cache*. For each connection initiated from the client device there is an HTTP analyzer and a transparent proxy. The HTTP analyzer intercepts HTTP traffic from the client to the server, analyzes it to determine if it is a Netflix video chunk request, handles it appropriately if it is, and otherwise simply forwards it to its original destination. The transparent proxy forwards any response traffic back to the client that was routed through by the HTTP analyzer. Individual prefetching agents prefetch video chunks for the proxy at the request of the HTTP analyzer. Prefetching agents then place prefetched data in the cache until it is requested by the client. A high level view of the proxy can be seen in Figure 3.2. This software is discussed in more detail below.

Manifest Storage and Chunk Design

On startup the proxy first reads in the manifest files. It creates an object for each quality level of the video and stores these version objects in an ordered array. Inside each version object is another array that houses an object for each of the video chunks that belong to that version, ordered by their position in the video. By ordering the chunks by their position in the video array accesses can be very quick even during a version change by keeping a variable that indexes the current chunk. Each version object contains the start and end of the HTTP requests, which are initialized on the clients first request of each version. The start and end of the HTTP request are separated by the range of the request. Each video chunk object contains the ID for the version to which the chunk belongs, the byte range of the request, and an uninitialized array to store video payload data. The data array only has memory allocated to it when the chunk is to be prefetched. The range of the request can be filled in by the specific chunk to be prefetched and there will be a complete HTTP request for that video chunk. The ability to create an HTTP request for any given chunk is important because it allows for any chunk to be prefetched in advance of the client device requesting it.

Cache Design

Once a chunk is chosen to be prefetched it is placed in a local cache object inside the proxy software. The cache is stored as a HashMap with the key to access a particular object being the version ID for that version, with the range request appended to the end. For example, the highest quality version for the iPad payer is identified by the ID Number *670543564*. Supposing that a chunk from that version is associated with the range request *26867504-27627407*, to store or retrieve this chunk from the cache the following key would be used: *67054356426867504-27627407*. This is satisfactory for caching videos where it is guaranteed that each version has a unique version identifier.

When a video chunk is served from the cache its video payload data is nullified and the object is removed from the cache. For video chunks that are never served their data remains in the cache. The low storage overhead of the prefetching scheme allows this technique to be reasonable for experimentation as chunks are only prefetched and not requested by the client player when a version change occurs during playback. Therefore, when playing a single video one time prior to resetting the proxy for the next experiment this is satisfactory.

HTTP Analyzer and Transparent Proxy

For each TCP connection request that comes into the proxy machine two threads are created, an HTTP analyzer and a transparent proxy. These are created to handle that specific connection between the client and the server. The HTTP analyzer accepts all HTTP traffic from the client. It is responsible for handling video chunk requests by either serving them from cache or forwarding them to the server if they have not yet been prefetched and for setting up the prefetching of future chunks. HTTP traffic that is not Netflix video is simply routed through to its original destination. The transparent proxy forwards the response to

any request that was routed through the HTTP analyzer back to the client; this includes audio data, video chunks that were not prefetched, and all port 80 traffic unrelated to Netflix.

There is a setting that determines if all traffic will go through the HTTP analyzer and transparent proxy without being analyzed as if the proxy was just a router, or if prefetching will be utilized. This way characterization experiments can be conducted with no prefetching as HTTP traffic is read in and passed through the proxy with very little overhead. When the prefetching setting is set the HTTP GET request messages are analyzed after they are read from the client and potentially served from cache by the HTTP analyzer rather than being passed through to the server. Once an HTTP request message is received at the HTTP analyzer the GET method is compared to a regular expression so it can be determined if it is a Netflix video chunk request. If it does not match the regular expression it is forwarded to its destination. This allows the proxy to be robust to traffic that is not destined for Netflix. When it is determined that the HTTP request is a video chunk the first step is setting the prefetching agents to prefetch necessary chunks for future requests; this reduces the latency of the prefetching at very little extra cost to serving the current request as the prefetching agents run in their own threads. The transparent proxy thread is necessary as opposed to simply waiting for the response in the HTTP analyzer thread because the length of the response from much of the traffic is unknown and a blocking read from the server is not capable of determining when to exit the read from the server to accept another request from the client.

Based on the settings of the proxy one or more chunks may be prefetched when a video request comes into the proxy. When it is determined that prefetching is required the HTTP analyzer cycles through the prefetching agents, utilizing the least recently used and available prefetching agent. The HTTP analyzer sets the version of video to be prefetched and the chunk number from that version then notifies the waiting prefetching agent to begin prefetching the chunk. This technique is necessary rather than keeping a variable to indicate which chunks have been prefetched in the version object because the iPad player does backfilling. Backfilling occurs when the player goes back into its buffer and clears all the data after some portion and requests another quality level to replace those chunks. When this occurs it is possible for anywhere from the last chunk to be removed and replaced to the entire buffer being emptied. This causes the player to go backward and request chunks earlier in the video than have recently been requested. This means the same chunk number can be requested from multiple versions or from the same version that has already been requested. For example, if the iPad player has buffered up to chunk number 50 in version 1 then experiences an increase in available bandwidth, the buffer can be emptied and the player can request version 2-5 with chunk numbers less than 50 to replace lower quality chunks in the buffer. Note that chunks are not selectively removed and replaced; if backfilling occurs the player must re-request all chunks from the point the backfilling takes place.

When a video chunk request is received and after the prefetching for future chunks has been started it is determined if the currently requested chunk can be served from the cache. When checking the cache the current HTTP GET request is analyzed to determine if the chunk is available in the cache, using the version

ID and range as the key. If it is cached, the video data associated with that chunk is sent to the client. It is possible for the chunk to be in the cache but not be completely finished its download. This is because as soon as a chunk has begun being prefetched it is placed in the cache. In this case any data that is in the cache is sent to the client and the HTTP analyzer thread serving the chunk goes into a wait state until the prefetching agent has notified it indicating more data has been made available, at which time it wakes up and transmits the newly cached data. As the video data is stored in a byte array, this type of producer-consumer solution is capable of streaming data to the client while handling synchronization and deadlock avoidance. This strategy of storing the chunk in the cache as soon as it has started being prefetched also allows a streaming-like download experience for the client, imposing very little extra latency as opposed to storing the chunk only after it has been completely downloaded. This is beneficial because that extra latency incurred when waiting for the download to complete at the proxy before sending the chunk is enough to cause the client to timeout the connection and reduce the requested quality level.

Prefetching Agents

The proxy runs with multiple prefetching agent threads that act independently of each other. It is necessary to have more prefetching agents than would minimally be necessary to prefetch N chunks ahead because if all prefetching agents are busy and the version changes an available prefetching agent is needed to begin prefetching the new version. While not actively prefetching the prefetching agents stay in a wait state. While in this wait state the HTTP analyzer thread can update which version and chunk number a prefetching agent should prefetch then notify the specific prefetching agent to begin prefetching that chunk.

Each chunk is given its own object when the manifest is built so the first thing the prefetching agent does is retrieve the object and designate memory within that object for the video payload data. This memory is cleared when the chunk is served so memory usage is only a slight overhead.

The chunk object is then placed in the cache immediately and the chunk request is sent to the Netflix server. By caching the chunk immediately it is possible for the HTTP analyzer to start serving a video chunk that has not been completely downloaded to the client in a streaming-like fashion. This prevents the client from seeing any delay at the proxy if a request comes in for a chunk that has not been completely cached yet. To do this the prefetching agent notifies the HTTP analyzer serving this connection after every read once the bytes have been added to the cache. If the chunk has not been requested by the client yet there will be no thread waiting and the resulting notify has no effect. If the HTTP analyzer needs this data and it has already sent all the currently cached data to the client it transitions to a wait state, awaiting a notification from the prefetching agent currently downloading that chunk to indicate more data is available.

When downloading a chunk to cache it is critical to serve exactly the correct number of bytes to the client otherwise it will assume there is an issue in the network or with the chunk and time out. Therefore to manage this issue the prefetching agent checks the input stream for data between reads on the stream, if it responds indicating there is no data available the prefetching agent is put to sleep for 300 milliseconds but

only if the total quantity prefetched data is within 200 bytes of the expected data size in the manifest. If still no bytes are available in the input stream the chunk is marked as a complete chunk, the HTTP analyzer is notified of its completion and the prefetching agent moves back into the wait state. This process allows the server to send its next congestion window to ensure all the bytes are received while avoiding the problem of entering a blocking read to the input stream when the entire chunk has already been received.

3.3.2 Traffic Manipulation

This section describes how traffic manipulation is used to allow client traffic to be pushed to the proxy. In doing so the proxy appears invisible from the perspective of the client and to be the client from the perspective of the server.

In order for the proxy to function, the Netflix traffic must be returned to the proxy from the server before being sent to the client otherwise server responses would be sent directly to the client resulting in error. To accommodate this concern IP masquerading is used. What IP masquerading does is set the source IP in the TCP packet to be that of the proxy rather than the client. Upon response the packet is forwarded back to the client as the destination IP address is accordingly set. Masquerading utilizes iptables, the command is as follows:

```
iptables -t nat -A POSTROUTING -o eth2 -j MASQUERADE
```

In order for the HTTP traffic to go through the proxy software the original port 80 traffic must be redirected to the port the proxy software is listening on rather than moving directly from the client player device to the server. One possibility is to use a proxy setting in the browser player but this would not allow use of the iPad player as it does not have a proxy setting for the Netflix application. Instead, a method was developed for diverting traffic through the proxy on any device capable of using OpenVPN. All the traffic from the client device is pushed through to the proxy hardware using the VPN. The proxy machine hijacks all traffic coming in on port 80 and sends it to the port the proxy software is listening on (port 13131). By hijacking port 80 traffic only the HTTP traffic is passing through the prefetching proxy software and all other protocols just go straight through the proxy bypassing the prefetching software. Like masquerading, hijacking also uses iptables and the command is as follows.

```
iptables -t nat -A PREROUTING -i tap0 -p tcp --dport 80 -j REDIRECT --to-port 13131
```

3.3.3 Tools

The Manifest Builder

The manifest builder is coded in the Java programming language. This application takes a file containing the byte ranges for each individual chunk in a quality level as well as an HTTP request functional with that quality level with the byte range omitted. This application then takes a byte range from the file, inserts it

into the HTTP request and sends the completed HTTP request to the server. The application then reads all the bytes from the server in response to that request for a minimum time of 1.5 times the chunk's play duration. This extended read duration ensures all response bytes for that chunk are read from the server. Once this time period has ended the byte range and the number of bytes read in the response are added to the manifest for the appropriate quality level and the next byte range is sent. Once all byte ranges are sent to the server, and the number of bytes read in the response have been recorded a completed manifest file for that quality level is produced.

Dummynet / IPFW

Dummynet [11, 62] is a network traffic emulation tool that allows fine-grained control of the characteristics of the network within the limitations of the actual network. It was used in this research to create multiple "pipes" connected to specific ports, and then to limit the available bandwidth to the desired level on each of those pipes.

By giving each prefetching agent and the transparent proxy connection their own pipe they each have their own bandwidth share rather than sharing a single pipe and competing with each other. This is realistic if the bottleneck is in the network after the prefetching proxy machine, for example, in a router, or if delivery is throttled by per-connection pacing at the server. In particular if the server is under load and giving each connection an equal share of its outgoing bandwidth then by using multiple connections to the server the prefetching proxy gets multiple shares of bandwidth for one client. For example, if there are 5 connections and 1 of them belongs to the proxy then the proxy gets $1/5^{th}$ of the resources; if the proxy opens two more connections then the proxy gets $3/7^{th}$ of the resources.

To use dummynet while avoiding other traffic going through the dummynet pipes, the hostnames of the servers for serving Netflix content were determined and reverse DNS lookups done to establish the IP addresses associated with those hostnames. Those IP addresses were then added to a file that was imported into an ipfirewall (ipfw) table to be used as the sources for dummynet to throttle traffic. After compiling a list of hostnames in a text file, the IP addresses were collected with the command that follows:

```
ips=dig -f ./netflixHostnames.txt +short
```

Those IP addresses are then added to table 1 and the pipes for the prefetching agents are built with commands such as the following:

```
ipfw add $(($c + 100)) pipe $c tcp from "table(1)" to 128.233.173.180 $(($c-1+20000)) in
```

where *c* is the ID of the prefetching agent, ordered starting at 1, *20000* indicates the starting port number of the designated prefetching agent, *128.233.173.180* is the IP address of the proxy, *table(1)* stores the Netflix IP addresses, and the option *in* indicates that only incoming traffic is of concern, and not the outgoing chunk requests, ACKs, or other control traffic. Only the traffic incoming from Netflix to the proxy box is limited because by limiting all traffic the ACKs and content requests from the client player will be competing with the traffic being downloaded by the proxy and that traffic should not be added to the bottleneck.

To build a pipe for the traffic that is not going through the prefetching agents, the following command was used:

```
ipfw add $(($1+100)) pipe $1 tcp from "table(1)" to 128.233.173.180 in
```

where the variable *$1* is *c+1*.

Finally, the available bandwidth for each of the pipes was manipulated to emulate the desired networking conditions by looping though the pipes where d is the current pipe number and e is the bandwidth for that pipe:

```
ipfw pipe $d config bw $(($e))KByte/s
```

Wireshark

Wireshark is a network protocol analyzer that makes it possible to capture and log the network traffic between the player and/or proxy and the Netflix content distribution network.[1]

To support use of wireshark packet traces, a parsing tool was built to analyze traces for a higher level look at the performance of the player or proxy. To accomplish this packet traces were captured on all interfaces, and then any non-HTTP traffic was filtered out. The packet summary was then exported to XML as a wireshark PSML file. The XML for a sample packet looks like the following:

```
<packet>
<section>9963</section>
<section>15.142846</section>
<section>10.8.0.8</section>
<section>198.45.53.138</section>
<section>HTTP</section>
<section>511</section>
<section>GET /691897732.ismv/range/3750723-4016329?c=ca&amp;n=22950&amp;v=3&amp;e
    =1374551797&amp;t=RTDjPdovxabHU4xUqboflIFUpyU&amp;d=silverlight&amp;p=5.
    AHZkpRZ17ch0cPFZO_lqAKDnu4FKX80auvLtbN_fFe8&amp;random=549067536 HTTP/1.1 </section>
</packet>
```

These packets contain important information such as the time, source IP address, destination IP address, protocol, and the GET method. The PSML file was then parsed into a CSV file, considering only XML packets containing video chunk requests, taking the time, and determining the version and chunk number from the GET method. Further, the source IP address was used to determine which chunks were requested by the proxy and which subset of those were requested by the client as every request that the client sends is also sent by the proxy. These CSV files were then parsed into graphs for visual analysis by running them through a Matlab script and examined based on the performance metrics discussed in the next section.

[1] http://www.wireshark.org, accessed 26-September-2011

3.4 Experimental Evaluation

In order to determine the effectiveness of the prefetching proxy it must first be determined how each of the respective players performs under various conditions without prefetching. For this reason, a performance characterization is done on both the iPad player and the PC Browser player. This characterization includes observations as to how the player reaches steady state at various available bandwidth levels and how the player reacts when experiencing various network conditions while considering the players state by manipulating the network conditions at various times during playback. Each type of experiment as described in Section 3.4.1 is conducted once for each unique combination of the performance factors as described in Section 3.4.2. An initial subset of experiments was conducted multiple times with similar behaviour to provide confidence in the repeatability of the results.

3.4.1 Experiments

Four types of experiments were conducted in this research to both characterize the Netflix rate-adaptation algorithm as well as determine the utility of the prefetching proxy. The types of experiments are outlined as follows:

- Stable Available Bandwidth: Experiments with stable available bandwidth involved setting the available bandwidth using dummynet prior to clicking play on the video and allowing this bandwidth to remain constant throughout the entire experiment. This type of experiment shows which quality level the player is capable of playing at that available bandwidth level and how the player achieves its steady state.
- Oscillation: Experiments with oscillation involve the available bandwidth starting at a high level and alternating between the high and a low setting. The duration spent at each level between oscillations is common between both available bandwidth levels. This type of experiment shows how the player performs when it experiences highly variable network conditions.
- Single Long-Term Increase or Decrease: During a single increase or decrease the available bandwidth starts at X Mbps and changes to Y Mbps after a certain number of seconds of video playback. This shows how the player responds to a single change in the available bandwidth. A realistic example of this would be large competing flows coming or going from the bottleneck.
- Consecutive Short-Term Spikes or Drops: During consecutive spikes (increases) or drops (decreases) the bandwidth starts at X Mbps and changes to Y Mbps then back to X Mbps three consecutive times for a fixed duration and interval between changes. This type of experiment allows for observation of how the player performs when there are small short duration changes to the available bandwidth. A realistic example of this would be short flows coming or going from the bottleneck.

3.4.2 Performance Factors

The factors that were controlled during experimentation include variation of the duration, bandwidth region, and time during playback the bandwidth variation occurs, as described below:

- Duration: With respect to the duration of a bandwidth variation two types of experiments were conducted. For the single increase or decrease experiments the changed bandwidth was in effect for the remainder of the experiment. For the consecutive increases or decreases experiments the duration of the change was 1, 2, 5, 10, or 30 seconds. There were three change events during each experiment in this case and between each of those change events was a recovery interval where the bandwidth returns to its original level. These recovery intervals lasted for either 2 or 10 seconds to simulate quicker or slower bandwidth variations. For the oscillation experiments the duration between oscillations was 0.25, 1, 5, 10, or 30 seconds.
- Time of Variation: The time of variation determines when the first change in available bandwidth takes place. During the oscillations the time the variation begins the moment the play button is pressed or clicked, with the oscillations starting at the higher of the two available bandwidth levels. For the single long-term increase and decrease and the consecutive short-term spikes or drops experiments the changes start at 10, 30, 60, 240, and 360 seconds of playback, meaning the video must start and play for x seconds before the change event takes place.
- Bandwidth Region: The bandwidth region considers which bandwidth levels are used in the experiment. This includes both the actual level for stable available bandwidth experiments and the magnitude of the variation for experiments which include bandwidth variation. The magnitude of variation considers large magnitude changes such as 0.5 Mbps to 4 Mbps as well as low magnitude changes of 1 Mbps to 2 Mbps. This allows for observation of the behaviour of the players during modest and more extreme conditions. Bandwidth levels used for oscillations, single and consecutive increases and decreases consist of 0.5, 1, 2.375, and 4 Mbps for the PC Browser player and 0.25, 0.5, 1, 2, and 4 Mbps for the iPad player. When doing the experiments with stable available bandwidth, the experiments were done using 0.125 Mbps buckets starting at 0.5 Mbps (the minimum available bandwidth to play without freezing) and ending at 2.375 Mbps (the minimum to steady state the highest quality version inside the experiment time) for the PC Browser. Similar parameters were used when doing experiments for the iPad player except the experiment buckets started at 0.25 Mbps and ended 2.0 Mbps. An experiment was also included for both players at 4 Mbps to see how they performed under excess available bandwidth.
- Prefetching Algorithms: When prefetching is considered the algorithm the proxy uses to determine which chunks to prefetch will play a role in the performance potential of the player. Experiments are run with *N-ahead* prefetching with N=1 to 5. In *N-ahead* prefetching, when a request is made for chunk n from version m then prefetching should have started or will start for chunks $n+1$ though

n+N of version *m*. If $N > 1$ and version *m* was also requested for chunk *n-1* then prefetching will have been started for chunks up to *n+N-1* so the new request should only require chunk *n+N* to be initiated. When the version changes from version *m* to any other version on the request of chunk *n* then prefetching agents send requests for all chunks *n+1* to *n+N*. Note that the backfilling that takes place with the iPad player could introduce gaps into the chunks that are to be prefetched, so it cannot be assumed that just because a chunk *n+N* has already been prefetched that *n+N-1* does not still need to be prefetched again. See Section 4.1 for more discussion on iPad backfilling.

3.4.3 Performance Metrics

In Section 2.2 several of the related works discussed metrics used to evaluate video performance and rate-adaptation behaviours that are favourable [6, 18, 80, 81]. General consensus is that pauses for rebuffering are very undesirable. This metric is not used in this research because any significant playback interruptions are deemed to be unacceptable when considering modern commercial video players. Therefore, in determining the impact of the prefetching proxy the following metrics are used:

- Average Playback Version (APV): A "video chunk" is defined here as a Netflix HTTP chunk request for a video media data. The initial meta chunks such as *0-41711* that are requested before a version's video data is requested or audio requests, are not considered video chunks. Each video chunk belongs to a version, and these versions are indexed in an order determined by their relative quality levels. Because the video is variable bitrate this does not mean that all video chunks from version 3 are lower bitrate than version 4, however for any particular video chunk number it would hold true. Experiments are run for either 3, 6, 8, 10 or 12 minutes of actual playback time. Note that, a 3 minute experiment could include more or fewer video chunks than other 3 minute experiments dependent on how much buffer occupancy has been accumulated at the end of 3 minutes of playback. The APV is the average version number of all requested video chunks. The average version number is used rather than the average bitrate of the requested chunks because the video quality does not vary linearly with the bitrate, for example, doubling the bitrate does not necessarily double the perceived video quality. Further, doubling the bitrate at low bitrates likely has a larger impact on the video quality than the same increase to the bitrate at higher bitrates. Quality likely does not vary linearly with the average version number either, however using version numbers yields more easily interpreted results. If n_i is the version number requested for chunk number i and there are N chunks requested in the experiment, then the APV is defined as:

$$APV = \sum_{i=1}^{N} (n_i)/N \tag{3.1}$$

• Playback Smoothness (PS): Earlier research on perceived video quality has found that when video frequently changes quality level it is annoying to viewers [22, 59, 81]. Therefore long durations without version changes are preferred. A "run" is defined as a sequence of consecutively played chunks from the same version. Denote the total number of chunks in the experiment by N, the version index of the r^{th} chunk by I_n, the number of runs by M and the number of chunks in the r^{th} run by n_r. Xiang et al. define PS as shown in Equation 3.2 [80]. This equation does not take into account the fact that large changes in quality level sustained by skipping intermediate versions are less desirable than including intermediate steps [36, 46]. Equation 3.2 is modified slightly so that it remains the same if changes are only by a single quality level but it now takes into account both the number of consecutive chunks at a particular version as well as quality level changes where intermediate levels are skipped to give a single value for how smooth the video is for the viewer, as shown in Equation 3.3. Note that with this modified equation, unlike PS as described in Equation 3.2, a single large change is not smoother than stepwise "one at a time" changes between the same initial and final quality levels. The experiments in this research also involve different play lengths. With the original definition a longer experiment will appear to have higher playback smoothness than a shorter experiment, even if they both play only the highest quality, due to there being more chunks. To account for this playback smoothness is adjusted by dividing by the total number of played chunks to get a per chunk playback smoothness.

$$PS_{original\_definition} = \sqrt{\frac{\sum_{r=1}^{M}(n_r)^2}{M}} \tag{3.2}$$

$$PS = \frac{\sqrt{\sum_{r=1}^{M}(n_r)^2/(1+\sum_{r=2}^{N}(|I_n - I_{n-1}|))}}{N} \tag{3.3}$$

• Prefetching Overhead (PO): Prefetching overhead involves chunks that are prefetched but never requested by the player. It is calculated as the fraction of the total proxy-requested data volume that is overhead by dividing the total number of bytes that belonged to chunks that were prefetched but never requested by the player divided by the total number of bytes for all the requested chunks (Equation 3.4).

$$PO = \frac{PrefetchedBytesNeverRequested}{PlayerRequestedBytes} \tag{3.4}$$

3.5 Chapter Summary

This chapter discussed the methodology for the thesis experimentation. All pieces of hardware have been detailed, including the purpose and specifications of each machine. The Netflix video delivery methodology

was explained. The software used in the proxy was detailed including functionality and the rationale for design decisions. Finally, the experimental plan was laid out including which experiments were run, the performance factors and the performance metrics.

Chapter 4

Netflix Characterization Results

This chapter presents a characterization of the behaviour of the Netflix rate-adaptation algorithms. It is not possible to give an exact description of the rate-adaptation algorithms since access to the Netflix source code is unavailable and there are unknown and potentially complex interactions between factors such as buffer occupancy, current available bandwidth, smoothed available bandwidth, latency, and even the current place during playback of the video. However, thorough experimentation was conducted that allows for a high level description of the rate-adaptation behaviour under various conditions similar to the research conducted by Akhshabi et al. [2] and in mobile conditions by Riiser et al. [61]. First, the high level differences between the PC Browser and iPad Netflix players are laid out and the properties of the chosen video, *Knight and Day* are presented. Then each player's behaviour during playback is examined at a high level, including experiments under several conditions such as during a stable level of available bandwidth, during long-term changes to available bandwidth, during consecutive short-term changes to the available bandwidth, and during oscillations to the available bandwidth. These experiments are done in multiple bandwidth regions to include extreme changes from very low and very high available bandwidth levels as well as more moderate levels of change all for various durations and at various times during the experiment.

4.1 Player Comparisons

The PC Browser player and the iPad player have several notable differences:

Chunk Durations: The playback durations for individual chunks are much larger for the iPad player than the PC Browser player. The PC Browser player chunk durations are 4 seconds of playback and the iPad player chunk durations are 10 seconds of playback. It is possible that longer chunk durations were chosen for the iPad player because its video versions have a relatively low bitrate. With short chunk durations the player would not be able to fully utilize the available bandwidth. The lower bitrates would result in a smaller data transfer, which is not suitable for ramping up the TCP congestion window to maximize throughput.

Buffer Capacity: The maximum buffer capacity of the players also differs. The iPad player has a much smaller buffer for downloading chunks in advance. The iPad player's buffer is 50 seconds. Coupled with the long chunk durations, this means that its buffer holds a maximum of 5 chunks whereas the PC Browser

player has a buffer capable of storing up to 4 minutes of content, which allows for storage of 60 chunks of video.

Quality Level Changes: When changing quality levels the iPad player and the PC Browser player differ in multiple ways. When increasing quality levels the PC Browser player always requests at least a single chunk from each intermediate quality level whereas the iPad player may either request some intermediate level chunks or be as extreme as jumping directly from the lowest quality level to the highest quality level. This behaviour is similar for decreasing quality levels although the PC Browser player will on occasion skip intermediate qualities. It is out of the scope of this research to determine if skipping intermediate versions has a positive or negative impact on the QoE for these players. However, some prior work has been done on this topic, mentioned in Section 2.2.

Backfilling: The iPad player also uses a form of buffer replacement that will be referred to as *backfilling*. When increasing quality levels the player may go back and empty the buffer to a certain point and begin downloading the chunks of the new version at that point. This allows the player to play a higher bitrate of video after it has already downloaded lower quality chunks. Figure 4.1 shows the backfilling behaviour during an experiment in which the available bandwidth was increased at 60 seconds of playback. In this graph time 0 would be associated with the time the play button was pressed, but video playback begins at a later point which explains why the quality level changes closer to 90 seconds as opposed to 60 seconds. When increasing the quality level the backfilling attempts to go back as far as possible and replace the chunks with the new version. This behaviour has a potential pitfall with respect to depleting the buffer as discussed later in Section 4.4.1. Backfilling also occurs when the iPad player decreases the quality of video it is requesting, as is seen in Figure 4.1. In every case encountered during this research not only does the player cancel the current request when decreasing versions but it re-requests the chunk prior, which is already completely buffered. This results in a higher quality chunk being replaced with a lower quality chunk. For example, if the iPad player requests chunk number 15 from version 5 and then chooses to drop quality to version 1 the player will start by requesting chunk 14 from version 1 rather than simply sending the request for chunk 15 of version 1. The PC Browser player does no such backfilling.

4.2 Determining the Average and Peak Video Bitrate

While conducting initial experiments at various available bandwidth levels it was determined that the PC Browser player has six distinct video quality levels that it may request, the iPad player has five distinct video quality levels, and each player has a single audio version. In this research the packet headers are included in the bitrate computations. This is done because video data has to traverse through the Internet rather than being played directly from a DVD or blue-ray disc, as such those packet headers become a necessary part of the video traffic. However, the packet headers are small in comparison to the video payload, so it would not

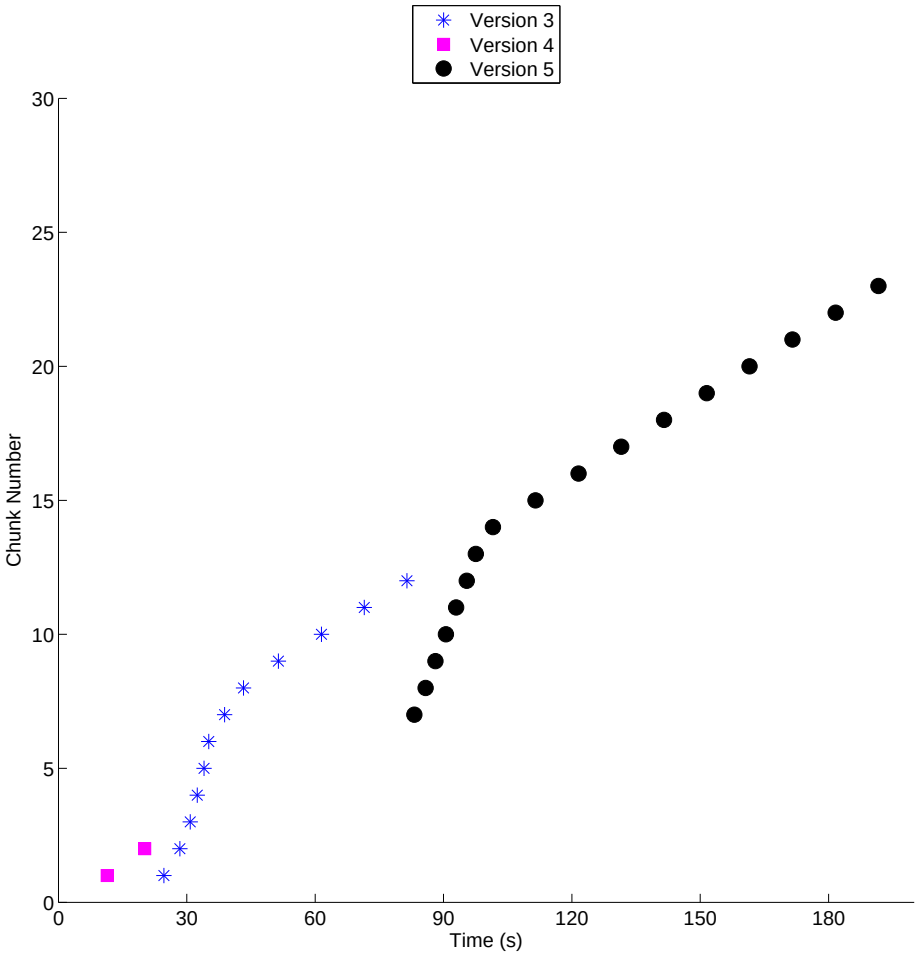

Figure 4.1: iPad: Long-Term Increase from 1 Mbps to 4 Mbps at 60 s Playback - Backfilling

Table 4.1: PC Browser: Bitrates

Version	Approximate Average Bitrate (Mb/s)	Peak Chunk Bitrate (Mb/s)
1	0.231	0.481
2	0.364	0.703
3	0.539	1.106
4	0.719	1.141
5	1.000	1.980
6	1.670	3.351

Table 4.2: iPad: Bitrates

Version	Approximate Average Bitrate (Mb/s)	Peak Chunk Bitrate (Mb/s)
1	0.118	0.180
2	0.191	0.299
3	0.249	0.410
4	0.385	0.638
5	0.565	0.948

have a significant impact on the results if they were omitted. The *Approximate Average Bitrate* is defined as the bitrate of a version of video averaged over the entire playback duration. The *Peak Chunk Bitrate* is the bitrate of the largest chunk averaged over the chunk's duration. The approximate average bitrate was determined by sending a request for each of the chunks for a particular version, summing the total number of bytes received in response then dividing by the duration of the video. Determining the peak rate by chunk can be done easily as the chunks all have the same play duration. To determine the peak chunk bitrate, the chunk of maximum size was taken from each version and its size was divided by the play duration of the chunk. See Table 4.1 for the breakdown of the versions for the PC Browser player and Table 4.2 for the breakdown of the versions for the iPad player. It is observed that the calculated bitrates for the PC Browser player versions are considerably larger than those of the corresponding iPad versions.

4.3 Ramp Up

Steady State is achieved when the playback version that the player requests has stabilized. The behaviour from the time the user initiates playback until the time steady state is achieved, assuming it is achievable under the current networking conditions, will be referred to as *Ramp Up*. This section examines the behaviour of the players during ramp up.

The time required for playback to begin after hitting the play button, termed *Startup Time*, varies greatly depending on the player's observed available bandwidth. Table 4.3 and Table 4.4 give the arithmetic mean startup time as well as the standard deviation at various levels of available bandwidth over a minimum of 10 experiments, generally closer to 25 experiments for each available bandwidth, for the PC Browser player and the iPad player, respectively. As expected a higher available bandwidth results in a shorter startup time. However, the benefits to the startup time by increasing the available bandwidth diminish as the available bandwidth increases. This may be partially due to the player requesting higher quality chunks, which have a higher bitrate and as a result require more bandwidth to download for the same play duration. For the

Table 4.3: PC Browser: Startup Time

Bandwidth (Mbps)	**Mean Startup Time (s)**	**Std. Dev. (s)**
0.50	29.93	0.76
1.00	11.57	0.83
2.375	10.18	0.72
4.00	9.29	0.85

Table 4.4: iPad: Startup Time

Bandwidth (Mbps)	**Mean Startup Time (s)**	**Std. Dev. (s)**
0.25	41.53	2.90
0.50	33.84	1.73
1.00	22.24	1.72
2.00	12.26	0.50
4.00	7.93	1.36

lower available bandwidth levels the iPad takes a very long time to begin playback. This is partially due to the behaviour of the player when it observes very low available bandwidth. When play is pressed the player first downloads a meta audio chunk, when this chunk is received too slowly the player resends the request to various servers trying to obtain better throughput. This behaviour occurred at 0.25 Mbps, 0.5 Mbps and at 1.0 Mbps but stopped once the available bandwidth got to 2.0 Mbps. The PC Browser player did not demonstrate similar startup behaviour with respect to changing servers.

The players cannot play back a version with the available bandwidth set at or only slightly above the version's approximate average bitrate. This is due to having some level of conservativeness to guard against potentially changing network conditions, the video's variable bitrate nature, and other traffic consuming bandwidth. Therefore, the available download bandwidth necessary for each version to achieve its steady state from the beginning of the video is determined using a bucket approach. To do this available bandwidth values are considered in increments of size 0.125 Mbps. The minimum required available bandwidth to stream the lowest quality version is determined by setting the available bandwidth to 0.5 Mbps and conducting experiments while decreasing the level by 0.125 Mbps from the previous level until an experiment is encountered where the video freezes after playback has begun. Once a freezing event occurs the available bandwidth has been dropped too low and video playback cannot be supported. The bandwidth level above the value that freezes is set as the minimum available bandwidth to achieve steady state for the lowest quality of video. Next, the experiments are conducted with increasing levels of available bandwidth until they have been run for a few bandwidth levels after the player has begun streaming steady state at the highest quality level. Extra experiments are also conducted to see how the player performs when it observes a significant excess of available bandwidth at 4 Mbps. These experiments enable the observation of the behaviour during *Ramp Up* of a video under available bandwidth levels across the entire bandwidth spectrum. Table 4.5 and Table 4.6 show the minimum bandwidths for each version to achieve steady state for the PC Browser player and the iPad player, respectively. It must be noted that the available bandwidth necessary to maintain steady state for a particular version appears to change throughout the playback of the video. For example, it was found during the building of the manifest file that the player is more aggressive in playing higher quality versions

Table 4.5: PC Browser: Minimum Required Bandwidth

Version	Approximate Minimum Required Bandwidth (Mbps)
1	0.500
2	0.750
3	1.000
4	1.375
5	1.750
6	2.375

Table 4.6: iPad: Minimum Required Bandwidth

Version	Approximate Minimum Required Bandwidth (Mbps)
1	0.250
2	0.750
3	1.000
4	1.375
5	2.000

later in the video. Thus for the purposes of this section the definition of the necessary bandwidth to obtain steady state considers only the first 10 minutes of playback. The player must begin downloading the version for which it achieves steady state within 3 minutes from the beginning of the experiment and if the player maintains it for the duration of the experiment then the player is considered to be playing at steady state. During these experiments a situation where the player requests a version before 3 minutes and moves up to a higher version within the 10 minute experiment when it had not already accomplished steady state for the lower version at a lower available bandwidth was never encountered. This lends credibility to this definition for the required bandwidth to achieve steady state. These available bandwidth values are used for the remainder of the experiments are reasonable for the portion of video being used in this research.

When extending the definition of the time to achieve steady state from 3 minutes to refer to any stability inside 10 minutes of playback it is observed that the time from the beginning of playback to achieving version stability is very different when comparing the PC Browser player and the iPad player. The iPad player determines its steady state at every available bandwidth level used in the experiments almost immediately. The iPad player was observed to request a maximum of two video chunks prior to moving directly to the player's determined steady state, and only in one case were those chunks not backfilled. The PC Browser player takes much longer to stabilize and this is dependent on how much the available bandwidth exceeds the minimum required bandwidth to steady state at a particular version while also not getting high enough to request the next higher version. Figures 4.2, 4.3, and 4.4 show the gradual reduction in the time required to achieve stability for the PC Browser player at 1.125 Mbps, 1.25 Mbps, and 1.375 Mbps, respectively.

4.4 Rate-Adaptation Characterization

This section presents the rate-adaptation behaviour while the players experience various network conditions. Experiments are conducted using long-term variation, short-term variation and oscillations as described in

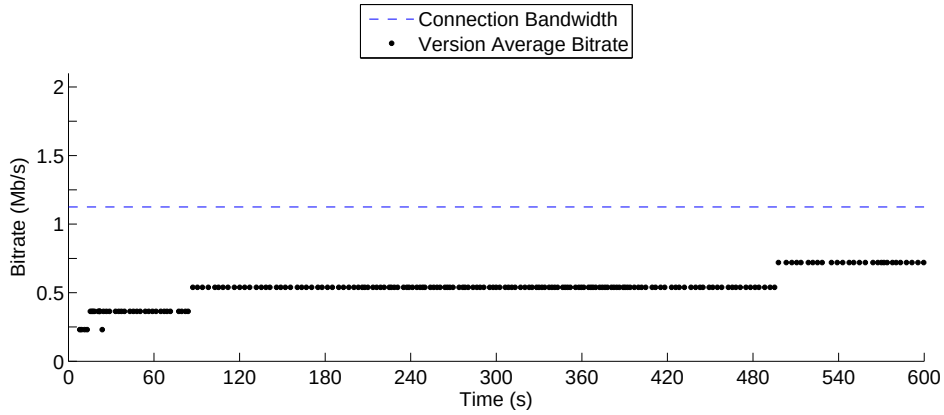

Figure 4.2: PC Browser: 1.125 Mbps Stable Available Bandwidth

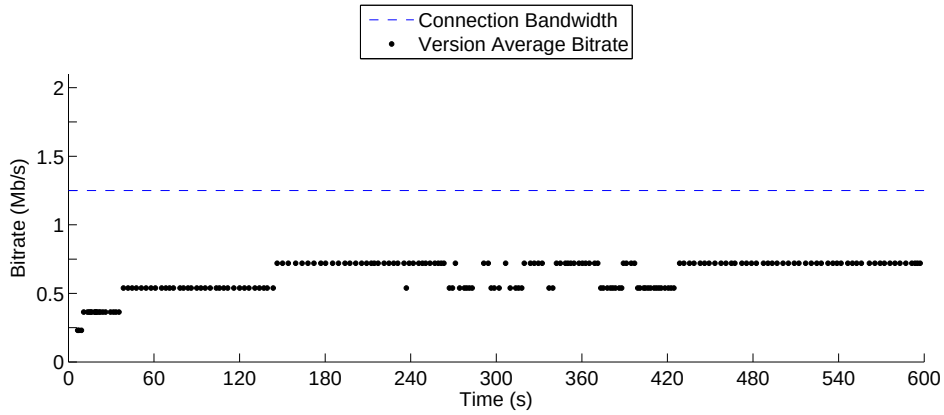

Figure 4.3: PC Browser: 1.250 Mbps Stable Available Bandwidth

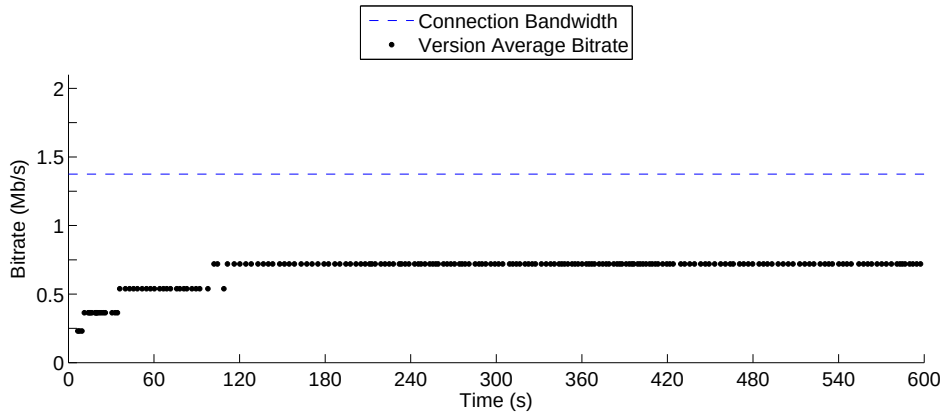

Figure 4.4: PC Browser: 1.375 Mbps Stable Available Bandwidth

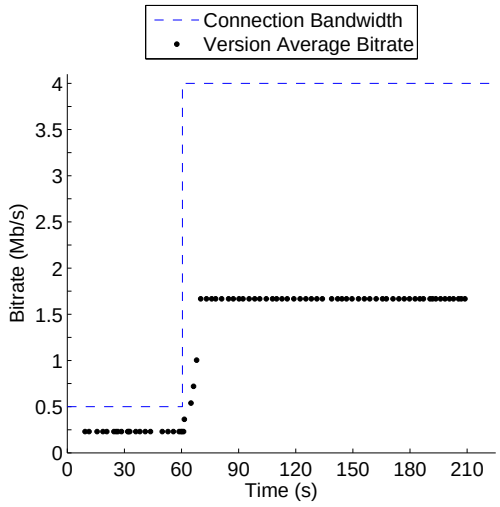

Figure 4.5: PC Browser: Long-Term Increase from 0.25 Mbps to 4 Mbps at 30 s of Playback

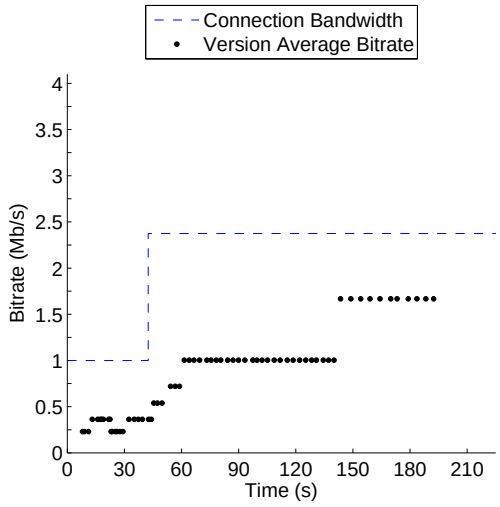

Figure 4.6: PC Browser: Long-Term Increase from 1 Mbps to 2.375 Mbps at 30 s of Playback

Section 3.4.1. The performance factors used are described in Section 3.4.2 and they allow for evaluation of change at various bandwidth regions, for various durations, and at various times during playback. The results in this section provide a description of the behaviour of the players so as to understand the basics of the Netflix rate-adaptation algorithms rather than in-depth statistical analysis. In each subsection both players are evaluated, starting with the PC Browser player then the iPad player.

4.4.1 Increasing Bandwidth

Increasing bandwidth experiments includes experiments starting at a relatively low available bandwidth then increasing to a higher available bandwidth. These increases are either long-term increases, where only a single change is made, or spikes, where the available bandwidth increases and returns to the original level three consecutive times.

Long-Term Increases

PC Browser: The PC Browser player reacts quickly to bandwidth increases. When a large increase occurs the PC Browser player begins requesting higher quality chunks immediately, requesting at least one chunk per quality level until it reaches its new steady state at the highest quality level. With a small increase the player still reacts immediately although it takes slightly longer for the player to become confident and move up through the versions all the way to its new steady state. Figure 4.5 shows the behaviour of the player when the available bandwidth is increased from 0.5 Mbps to 4 Mbps at 30 seconds of video playback. This can be compared to the impact of a smaller increase from 1 Mbps to 2.375 Mbps at 30 seconds of playback in Figure 4.6.

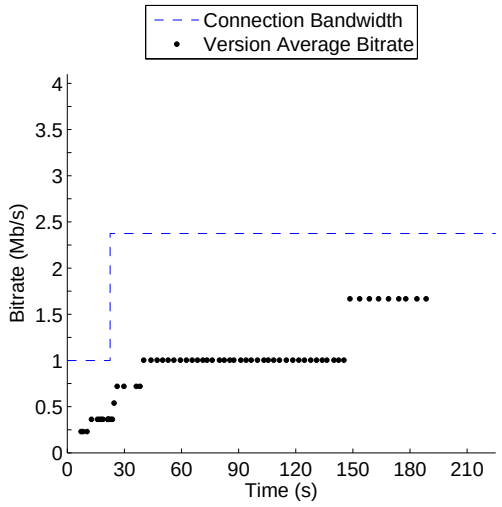

Figure 4.7: PC Browser: Long-Term Increase from 1 Mbps to 2 Mbps at 10 s of Playback

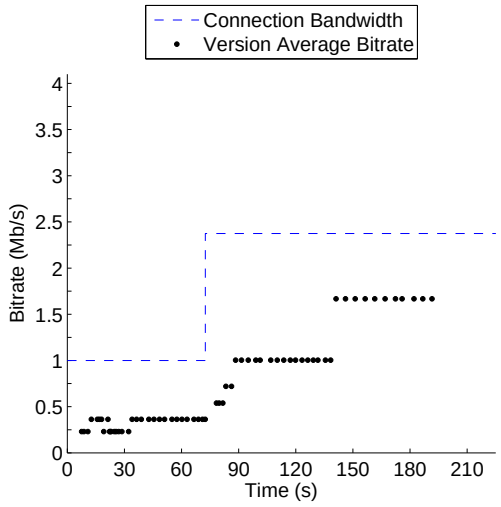

Figure 4.8: PC Browser: Long-Term Increase from 1 Mbps to 2 Mbps at 60 s of Playback

When considering smaller bandwidth increases, the player takes roughly the same amount of playback time to reach its new steady state regardless of whether the increase is at 10, 30, or 60 seconds. This could indicate that the player's concern is with buffer occupancy and current available bandwidth more than the level of smoothing or perhaps the player determines there is too much risk of freezing for rebuffering if the player is too aggressive too early. Figure 4.7 and Figure 4.8 show the behaviour when increasing the available bandwidth from 1 Mbps to 2.375 Mbps at 10 seconds of playback and 60 seconds of playback, respectively. Note that in both instances the player takes just short of 150 seconds from the time play is pressed to begin requesting the highest quality level despite the 50 second difference in the time of the increase.

iPad: The iPad player is very aggressive in the presence of large increases to the available bandwidth. In such cases it will generally immediately jump to the highest quality, skipping all intermediate qualities. If the increase is more modest in magnitude it will more often take some chunks from intermediate quality levels prior to moving to the highest quality but is not as diligent in doing so as the PC Browser. Figure 4.9 shows the behaviour of a large increase and Figure 4.10 shows the results of a modest magnitude increase both occurring at 60 seconds of video playback.

The time at which the increase takes place seems to have only a slight impact on the iPad player's behaviour. The player reacts quickly when the increase occurs at 10 seconds or 60 seconds of playback and the behaviour is very similar, however it seems that the increases happening at 10 seconds of playback request more intermediate chunks. This difference is very small as generally the player requests only one extra chunk.

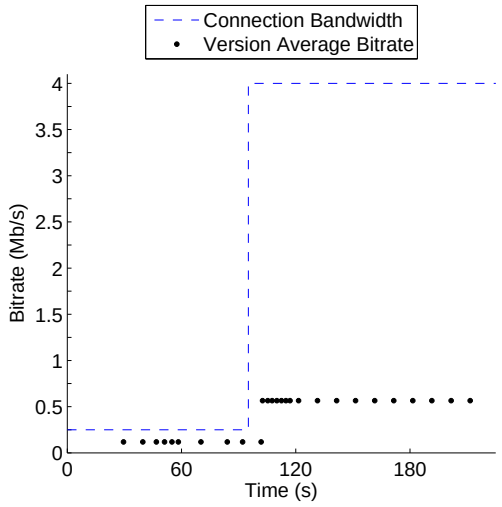

Figure 4.9: iPad: Long-Term Increase from 0.25 Mbps to 4 Mbps at 60 s of Playback

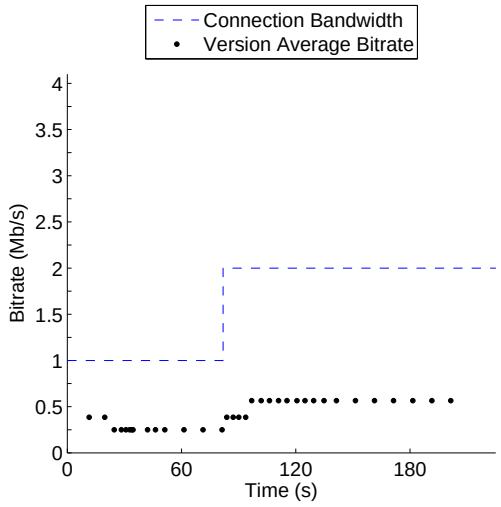

Figure 4.10: iPad: Long-Term Increase from 1 Mbps to 2 Mbps at 60 s of Playback

Short-Term Spikes

PC Browser: In general the PC Browser player reacted to the bandwidth changes in most of the short-term spike experiments except in a number of cases where the increase was only 1 second in duration. When considering the bandwidth region those spikes of lower magnitude seem to yield smoother playback quality during the spikes as the player reacts so quickly to a significant excess in available bandwidth. This can be seen with a larger magnitude spike in Figure 4.11 where there are fewer consecutive requests at a particular bitrate during the spikes and a lower magnitude spike in Figure 4.12 where there is still some instability but in general appears much smoother than with the large magnitude spike. Previous research indicates that smoothness is an important factor in the QoE [22, 36, 46, 59, 80, 81].

The time during playback when the variation takes place has an impact with variations at 4 minutes resulting in more aggressive player behaviour (hitting a higher quality level or maintaining the increased quality from a spike for more chunks) than with variations happening earlier. Also, the player seems to be more responsive to spikes at 60 seconds than spikes at 10 seconds which implies that the player's buffer occupancy has an impact on the player's aggressiveness during short-term available bandwidth increases. The player's response is more aggressive with spikes at 60 seconds of playback than with those of the same duration and magnitude occurring at 10 seconds of playback as can be seen when comparing the earlier spikes in Figure 4.12 with the later spikes in Figure 4.13.

Having short intervals between the spikes yields more aggressive player behaviour than when the intervals are long, with the player frequently reaching a higher quality level. This can be seen by the difference in behaviour between experiments with 2 second intervals (Figure 4.14) and 10 second intervals (Figure 4.15). Short intervals also seem to reduce the impact that the time the variations occur has on the player's behaviour.

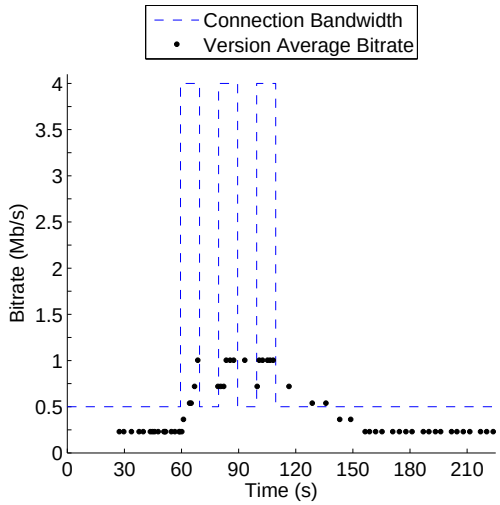

Figure 4.11: PC Browser: Spikes from 0.5 Mbps to 4 Mbps at 10 s of Playback for 10 s Durations and 10 s Intervals

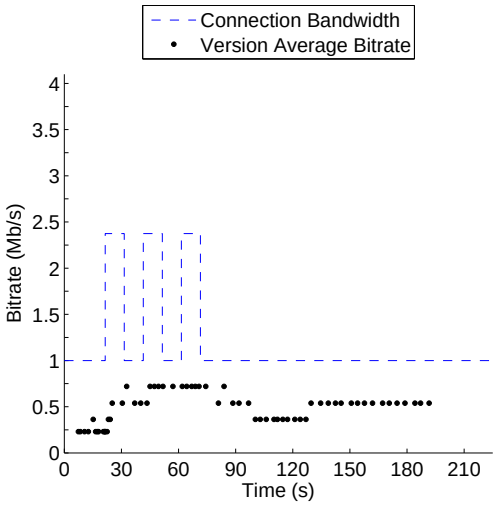

Figure 4.12: PC Browser: Spikes from 1 Mbps to 2.375 Mbps at 10 s of Playback for 10 s Durations and 10 s Intervals

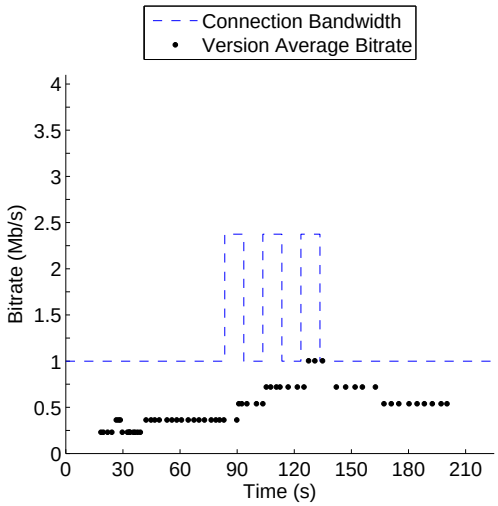

Figure 4.13: PC Browser: Spikes from 1 Mbps to 2.375 Mbps at 60 s of Playback for 10 s Durations and 10 s Intervals

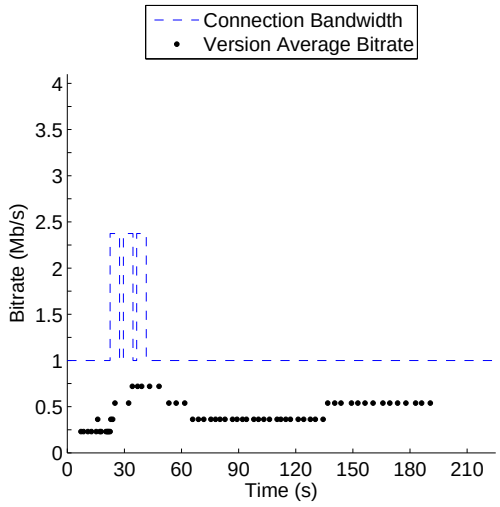

Figure 4.14: PC Browser: Spikes from 1 Mbps to 2.375 Mbps at 10 s of Playback for 5 s Durations and 2 s Intervals

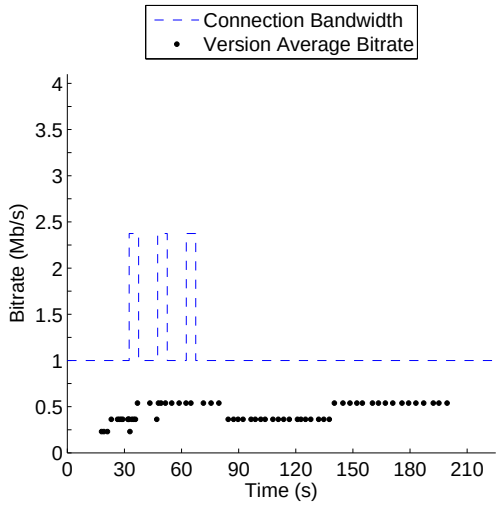

Figure 4.15: PC Browser: Spikes from 1 Mbps to 2 Mbps at 10 s of Playback for 5 s Durations and 10 s Intervals

This means the 2 second intervals appear to have similar aggressiveness when the variations occur at 10 seconds, 60 seconds or 4 minutes of playback.

iPad: When running experiments with the iPad player, a large magnitude available bandwidth spike from 0.5 Mbps to 4 Mbps almost always causes a reaction but more aggressively for spikes that have a longer duration. Figure 4.16 and Figure 4.17 show this behaviour. The spikes of low magnitude do not cause version jumps to the same degree as the large spikes.

When comparing the version average bitrate and the available bandwidth necessary for steady state (Tables 4.1, 4.2, 4.5, 4.6), the iPad player is more cautious than the PC Browser player to increase its quality level when the available bandwidth remains stable despite its likely ability to play back the higher quality smoothly. Figure 4.18 shows the player behaviour with the available bandwidth set at 1 Mbps. The player quickly reaches its steady state and plays the third quality level. However, if there are spikes to 2 Mbps for even very short durations the player increases its requested quality level and achieves a new steady state, as in Figure 4.19.

There are cases where the video playback freezes for rebuffering after spikes when the initial available bandwidth is at the lowest playable setting of 0.25 Mbps and it temporarily spikes to a high available bandwidth such as 4 Mbps. This may be a result of the player being too aggressive and not being tuned to handle significant decreases after significant spikes. One possible explanation for the freezing after large consecutive spikes concerns the backfilling done by the iPad player. When a large spike occurs backfilling empties the buffer of low quality chunks, and the player begins requesting high quality chunks. The problem arises when the spikes end prior to replenishing the buffer. When the available bandwidth drops back down

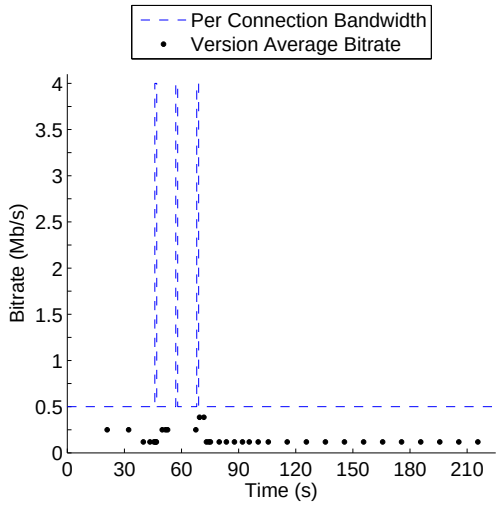

Figure 4.16: iPad: Spikes from 0.5 Mbps to 4 Mbps at 10 s of Playback for 1 s Durations and 10 s Intervals

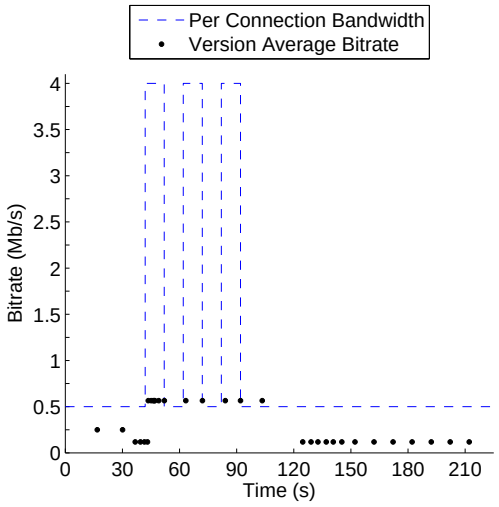

Figure 4.17: iPad: Spikes from 0.5 Mbps to 4 Mbps at 10 s of Playback for 10 s Durations and 10 s Intervals

the buffer is depleted causing a freezing event as a high quality level is being downloaded with insufficient available bandwidth. Figure 4.20 and Figure 4.21 show the behaviour while doing large spikes from 0.25 Mbps to 4 Mbps. Freezing for rebuffering occurs in all of the experiments involving large spikes at 60 seconds of playback when starting at 0.25 Mbps. Interestingly, this behaviour is less common when the spikes occur at 10 seconds of playback despite the player's request behaviour being similar. If the low level of bandwidth is increased from 0.25 Mbps to 0.5 Mbps the freezing no longer occurs as the player can recover quickly enough to avoid a buffer underrun.

In general, the behaviour of the iPad player under spikes is very similar when the spikes occur at 10 seconds, 60 seconds, or at 4 minutes of playback. The player may be slightly more aggressive later in the video, but this could not be concluded definitively through these experiments. The interval between the spikes seems to be of little significance as the player can download an entire chunk during the spike in bandwidth and the chunks have long durations. If the spikes are very short duration then the short interval between spikes may result in the player being more aggressive as the player's observations of the recent available bandwidth are higher.

4.4.2 Decreasing Bandwidth

When considering decreases in available bandwidth similar experiments to increasing available bandwidth are conducted. Single decreases in available bandwidth to analyze how the player reduces its quality level and short-term drops, where the available bandwidth is decreased three consecutive times.

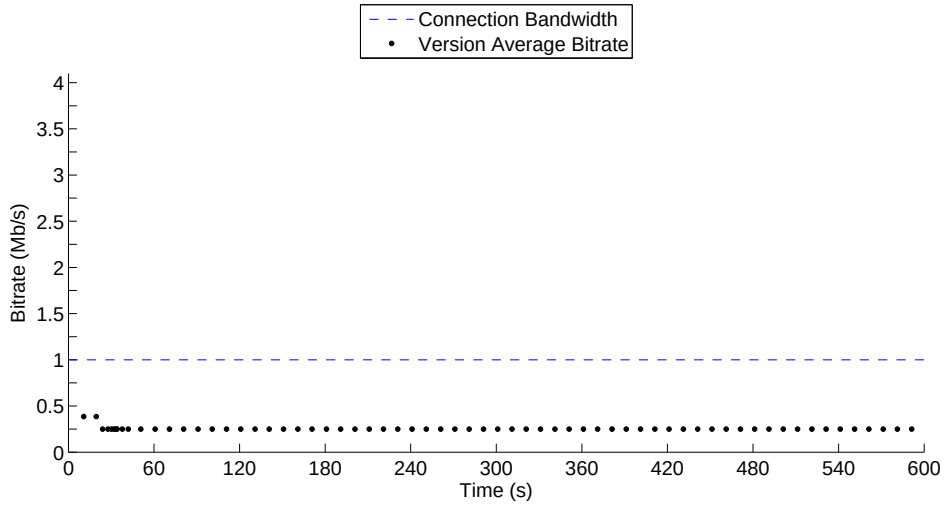

Figure 4.18: iPad: 1 Mbps Stable Available Bandwidth

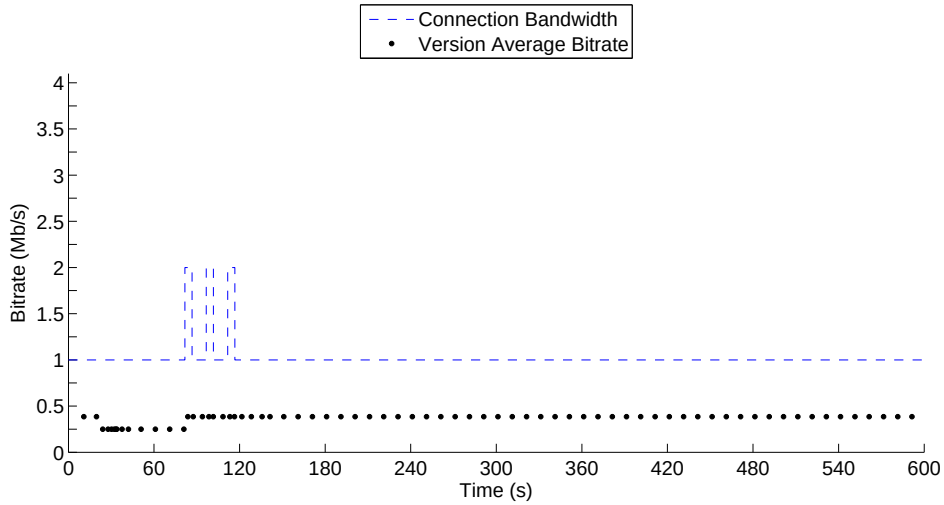

Figure 4.19: iPad: Spikes from 1 Mbps to 2 Mbps at 60 s of Playback for 5 s Durations and 10 s Intervals

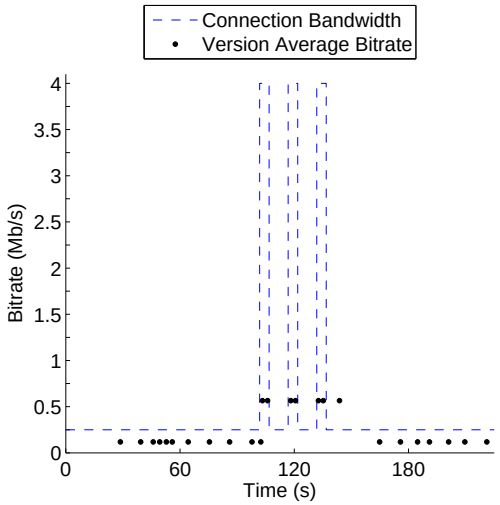

Figure 4.20: iPad: Spikes from 0.25 Mbps to 4 Mbps at 60 s of Playback for 5 s Durations and 10 s Intervals - Rebuffering

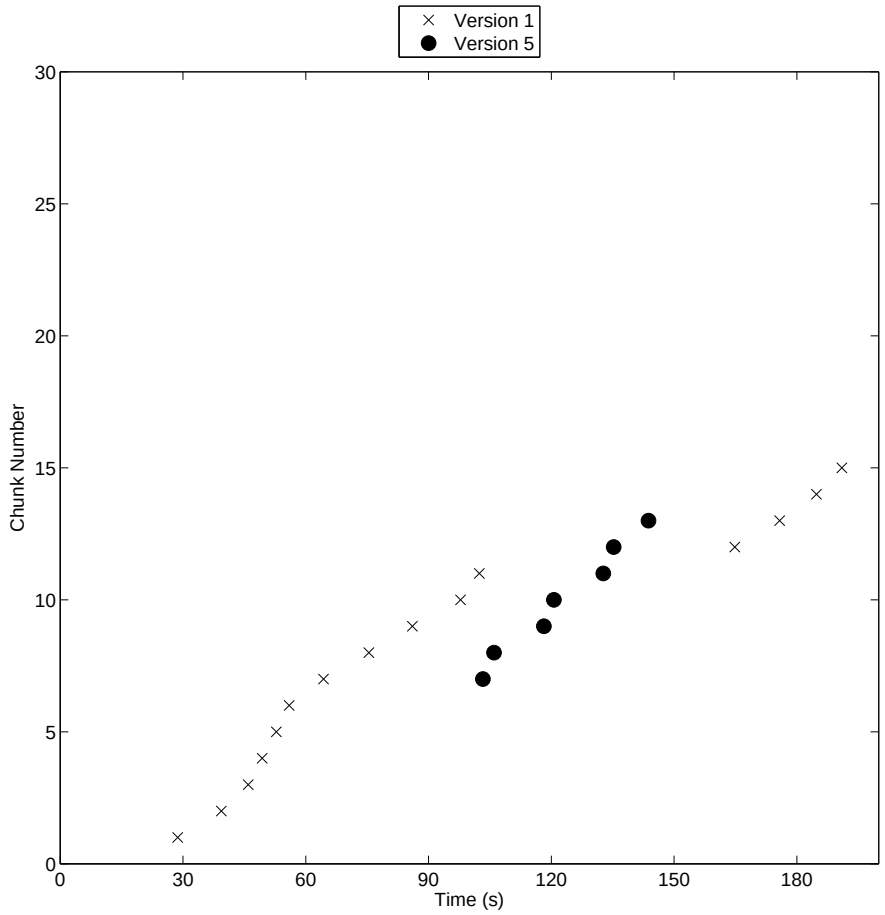

Figure 4.21: iPad: Backfilling Spikes from 0.25 Mbps to 4 Mbps at 60 s of Playback for 5 s Durations and 10 s Intervals - Rebuffering

Long-Term Decreases

PC Browser: For the PC Browser player the magnitude of the change seems to have little effect on how long it takes the player to react to the available bandwidth change with the player generally responding quickly. However, with changes of lower magnitude, the player generally requests two or three chunks at each intermediate quality level whereas with substantial decreases in available bandwidth only one or two chunks are requested at chosen intermediate levels. With the large magnitude decreases, the player is also prone to skipping versions on the way down.

The PC Browser player reacts very quickly to bandwidth variation in general, often by the next chunk request, and generally requests several intermediate quality level chunks at various bitrates to smooth the quality level reduction as shown in Figure 4.22. When available bandwidth variation happens very early in video playback, at 10 seconds for example, the player tends to change to the lowest quality more quickly and conduct little smoothing as seen in Figure 4.23. However, the behaviour changes when the available bandwidth variation is at 6 minutes, if the player originally has a substantial amount of bandwidth. In this case the player may hold onto the highest quality level for longer before abruptly reducing quality despite plenty of buffer occupancy still remaining as can be seen in Figure 4.24. This behaviour suggests that even with a large buffer the player is not willing to allow its buffer to substantially drain in the hope that the available bandwidth will recover. It instead focuses on a more instant measure of the available bandwidth and imposes minimum levels on buffer occupancy. This abrupt behaviour is noted for both decreases from 4 Mbps to 1 Mbps and 4 Mbps to 0.5 Mbps; however, when decreasing from 2.375 Mbps to 0.5 Mbps the same behaviour is not observed. In this case the player begins to reduce quality at the chunk following the bandwidth decrease and progressively reduces quality rather than moving directly to the lowest quality level.

iPad: The iPad player responds very quickly to large decrease from 4 Mbps to 0.25 Mbps as seen in Figure 4.25, and within a few chunk requests for decreases of lower magnitude such as 4 Mbps to 1 Mbps as seen in Figure 4.26. However, there are cases where the decrease does not result in any immediate reduction in quality level, but instead the player continues to request a high quality level, even though it would not have streamed that quality if it had started playback at that level of available bandwidth. For example, when decreasing from 4 Mbps to 1.25 Mbps after 60 seconds of playback the player maintained the highest quality level for the duration of the 10 minute experiment when it would have played a lower quality version if the player had started with 1.25 Mbps available bandwidth.

Whether the available bandwidth decrease happens at 10 seconds, 60 seconds or 6 minutes of playback does not seem to have a significant impact on the behaviour which indicates that the player is mostly concerned with current available bandwidth rather than its history of available bandwidth or buffer occupancy.

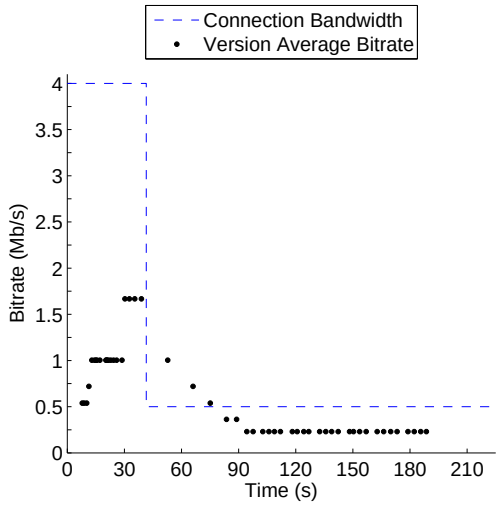

Figure 4.22: PC Browser: Long-Term Decrease from 4 Mbps to 0.5 Mbps at 30 s of Playback

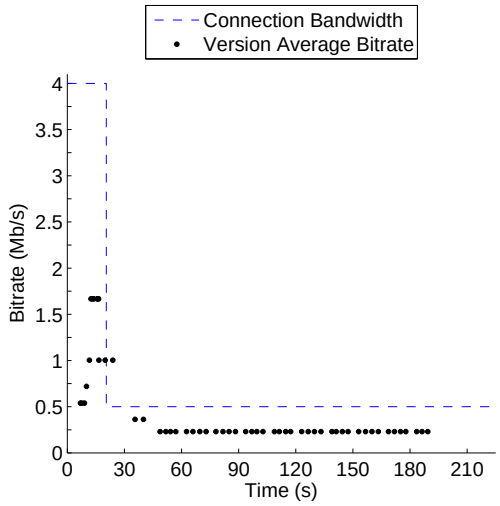

Figure 4.23: PC Browser: Long-Term Decrease from 4 Mbps to 0.5 Mbps at 10 s of Playback

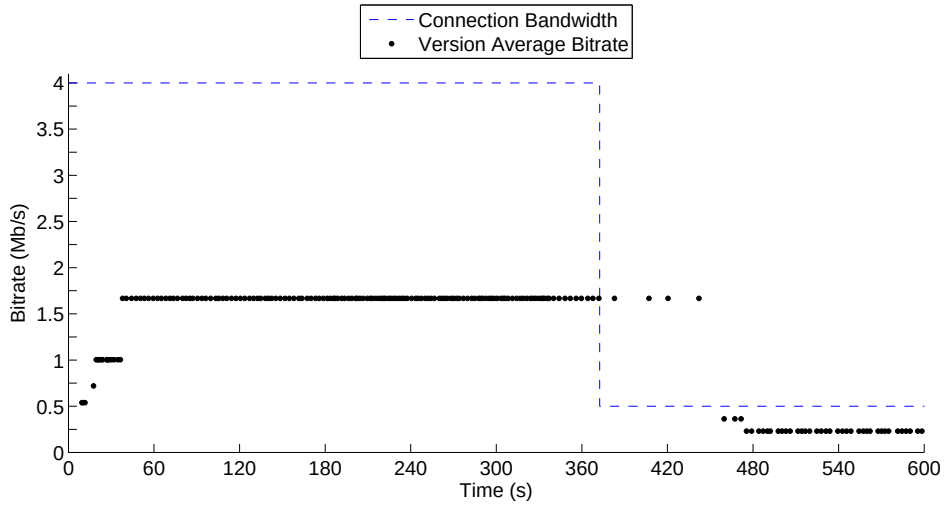

Figure 4.24: PC Browser: Long-Term Decrease from 4 Mbps to 0.5 Mbps at 6 min of Playback

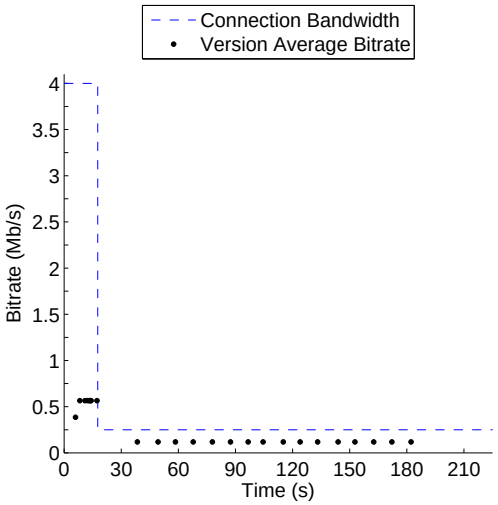

Figure 4.25: iPad: Long-Term Decrease from 4 Mbps to 0.25 Mbps at 10 s of Playback

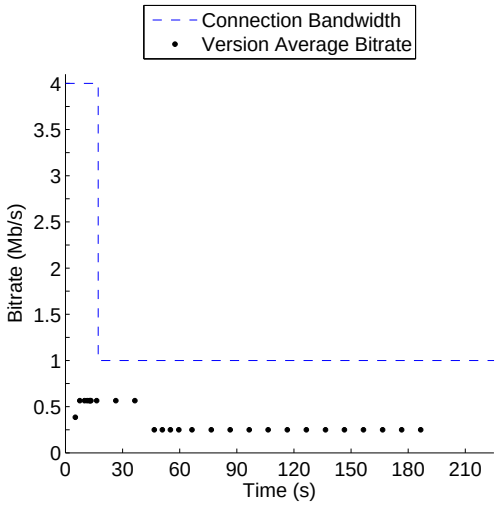

Figure 4.26: iPad: Long-Term Decrease from 4 Mbps to 1 Mbps at 10 s of Playback

Short-Term Drops

PC Browser: When considering short-term drops with the PC Browser player the time during playback that the drop occurs and the amount of excess bandwidth have a significant effect on the ability of the player to maintain a high quality level. For example, when doing experiments with drops at 4 minutes of playback and starting with an available bandwidth of 4 Mbps the player does a good job of sustaining the highest quality version, reducing the quality level for only a couple of chunks during 10 second drops to 0.5 Mbps and maintaining the highest quality for those with shorter durations. However, when the player starts with 2.375 Mbps of available bandwidth (the lowest available bandwidth sufficient to obtain steady state at the highest quality level), the player reacts significantly, reducing the requested quality even when only 1 second drops take place, and even when considering a less substantial drop to 1 Mbps. Figure 4.27 and Figure 4.28 show these behaviours. When drops occur inside 60 seconds of playback the behaviour is harder to define for the less substantial drops as the player has not reached steady state before the bandwidth variations occur.

The player is often, but not always, capable of absorbing 1 or 2 second duration drops when they occur at 10 second intervals, and more likely to absorb the variation if it occurs at 60 seconds of playback rather than 10 seconds where there will likely be some reduction in the video quality level. When the drops are very early during playback the drops with 10 second durations and 2 second intervals between drops result in a reduced the quality level whereas drops with long intervals between them often do not cause a quality level reduction but instead just increase the time it takes to ramp up the quality of video and achieve steady state. Figure 4.29 and Figure 4.30 show the behaviour for long drops with long intervals and Figure 4.31 and Figure 4.32 show the behaviour for short drops with short intervals. The results with lower magnitude drops exhibit many of the same trends.

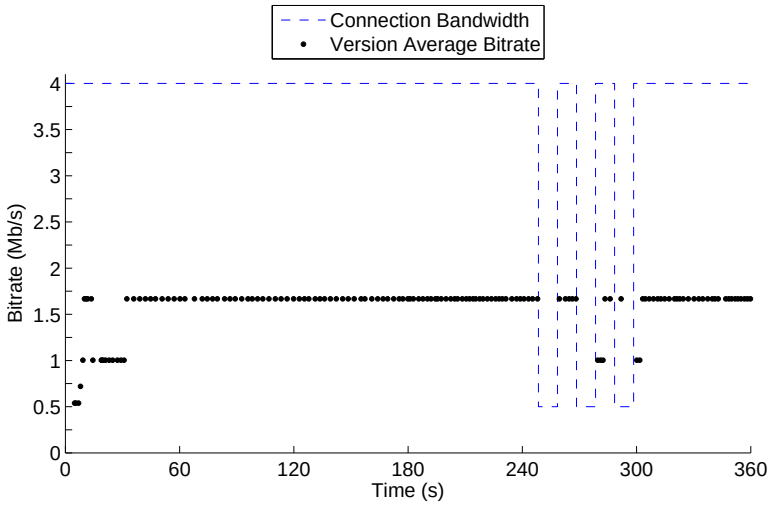

Figure 4.27: PC Browser: Drops from 4 Mbps to 0.5 Mbps at 4 min of Playback for 10 s Durations and 10 s Intervals

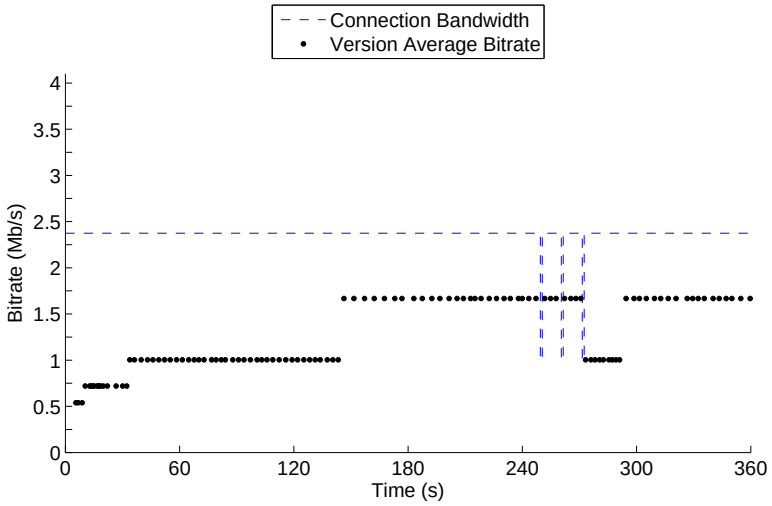

Figure 4.28: PC Browser: Drops from 4 Mbps to 0.5 Mbps at 4 min of Playback for 1 s Durations and 10 s Intervals

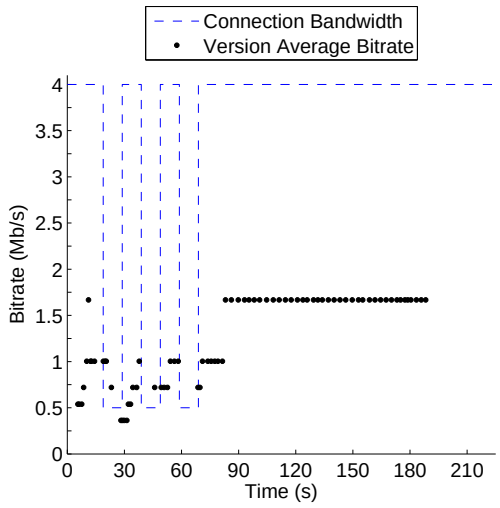

Figure 4.29: PC Browser: Drops from 4 Mbps to 0.5 Mbps at 10 s of Playback for 10 s Durations and 10 s Intervals

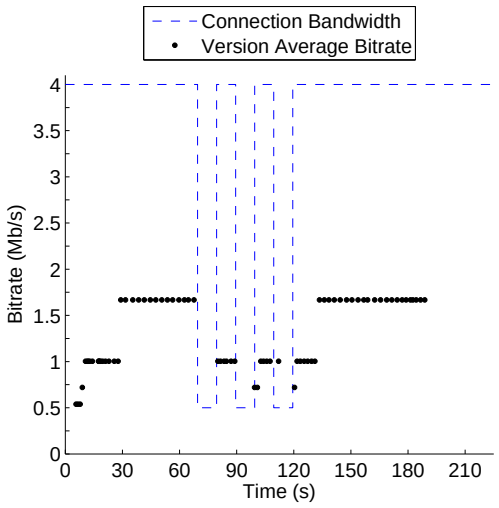

Figure 4.30: PC Browser: Drops from 4 Mbps to 0.5 Mbps at 60 s of Playback for 10 s Durations and 10 s Intervals

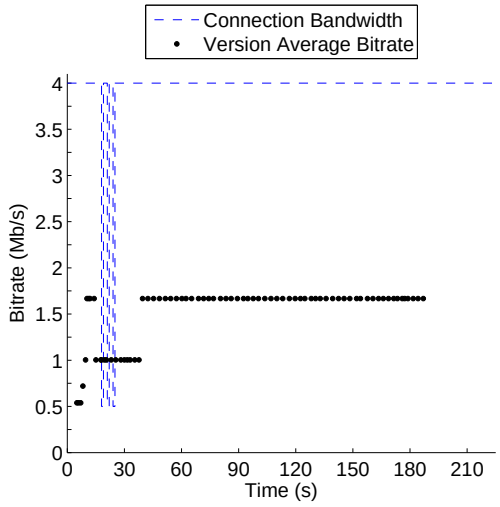

Figure 4.31: PC Browser: Drops from 4 Mbps to 0.5 Mbps at 10 s of Playback for 1 s Durations and 2 s Intervals

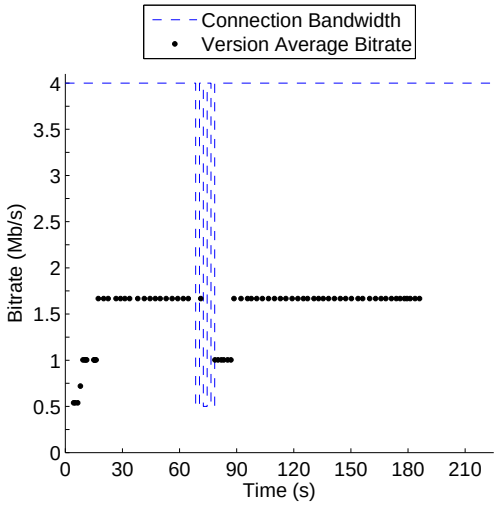

Figure 4.32: PC Browser: Drops from 4 Mbps to 0.5 Mbps at 10 s of Playback for 2 s Durations and 2 s Intervals

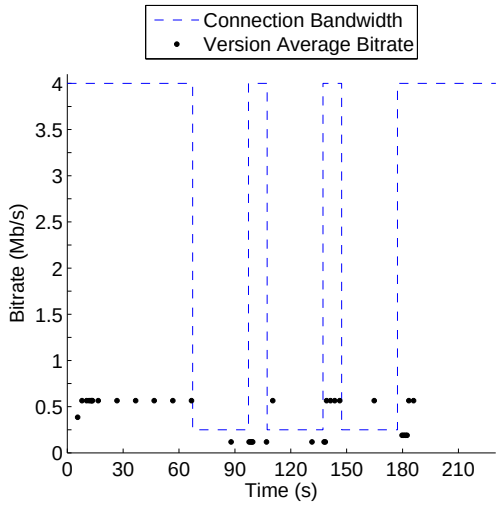

Figure 4.33: iPad: Drops from 4 Mbps to 0.25 Mbps at 60 s of Playback for 30 s Durations and 10 s Intervals

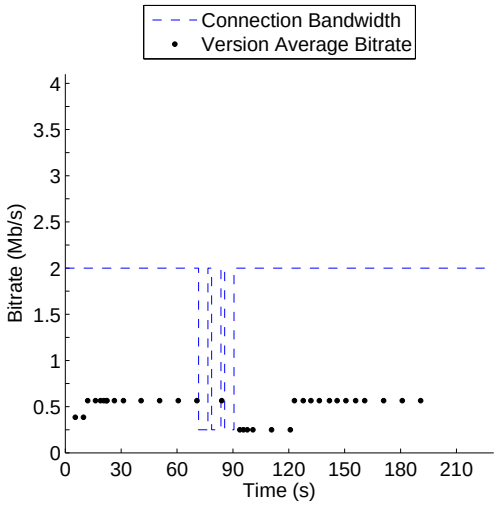

Figure 4.34: iPad: Drops from 4 Mbps to 0.25 Mbps at 60 s of Playback for 5 s Durations and 2 s Intervals

iPad: The iPad player is relatively good at absorbing short-term bandwidth variations so the player can maintain a high quality level. However, when the magnitude of the variation is large or when the variation has short intervals between the changes the player will likely reduce the requested video quality.

The magnitude of the variation has an effect on the player response. For example, Figure 4.33 shows the player behaviour during bandwidth variations between 4 Mbps and 0.25 Mbps for 30 second durations and 10 second intervals and the player is capable of requesting high quality chunks during the high available bandwidth intervals. Figure 4.34 shows the player behaviour during variations between 2 Mbps and 0.25 Mbps for 5 second durations and 2 second intervals. In this case the player takes a long time to determine that it must react and the player reacts for long after the variations had finished and the available bandwidth had increased back to 2 Mbps. For low magnitude drops the player is generally quite good at maintaining the highest video quality once the player has established steady state.

Very early in the video playback the player responds quickly to many types of variation including both large variations from very high available bandwidth to very low available bandwidth and low magnitude variations from high bandwidth to moderate bandwidth as shown in Figure 4.35. During such variations the player generally reacts harshly, and quickly requests a very low quality level. It appears that the initial reaction is an unnecessarily large reduction to the video quality as the player improves the quality level it requests during the successive variations.

Once the player has established its steady state, for variations as early as 60 seconds into playback longer variations or shorter drop intervals are needed to force the player to react. This phenomenon was not noticed with the long-term drops because short-term drops are observed by the player at a much finer time granularity. For example, dropping for 10 seconds with 10 second intervals from 4 Mbps to 0.25 Mbps does not cause the

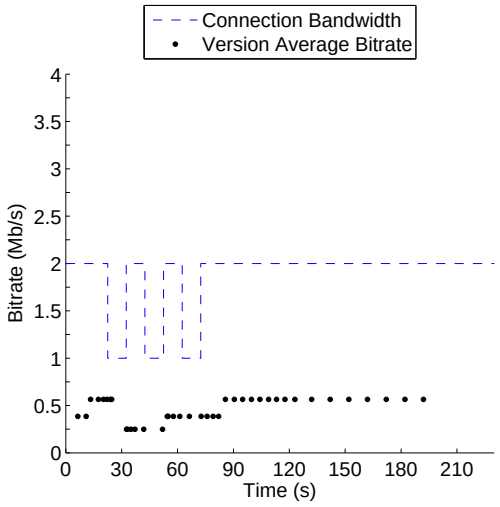

Figure 4.35: iPad: Drops from 2 Mbps to 1 Mbps at 10 s of Playback for 10 s Durations and 10 s Intervals

player to reduce quality. However if the gap is reduced to two seconds the available bandwidth seen by the player is low enough to drop quality.

4.4.3 Bandwidth Oscillations

PC Browser: For the PC Browser player oscillations in available bandwidth can cause significant quality level instability; this is particularly the case when the oscillations are 5 seconds or longer or when the oscillations have a very large magnitude. Figure 4.36 shows the player behaviour when available bandwidth oscillates between 4 Mbps and 0.5 Mbps with 10 second oscillations.

iPad: The iPad player is generally quite stable during available bandwidth oscillation. Longer oscillations and oscillations of greater magnitude result in a higher quality video than those with shorter durations and lower magnitudes. This can be attributed to the player having adequate time or bandwidth to download an entire chunk quickly either during a single long spike or multiple large spikes. See Figure 4.37 to see the behaviour of the iPad player when oscillating between 2 Mbps and 0.25 Mbps every 10 seconds.

4.5 Chapter Summary

In this chapter the properties of the video chosen for this research have been presented and experiments were conducted without prefetching. These experiments were run in order to characterize the behaviour of the PC Browser player and iPad player in several bandwidth regions under various types of bandwidth variation. The bandwidth variations considered include increasing and decreasing the available bandwidth as a single long-term change, as consecutive short-term changes, or with oscillating available bandwidth.

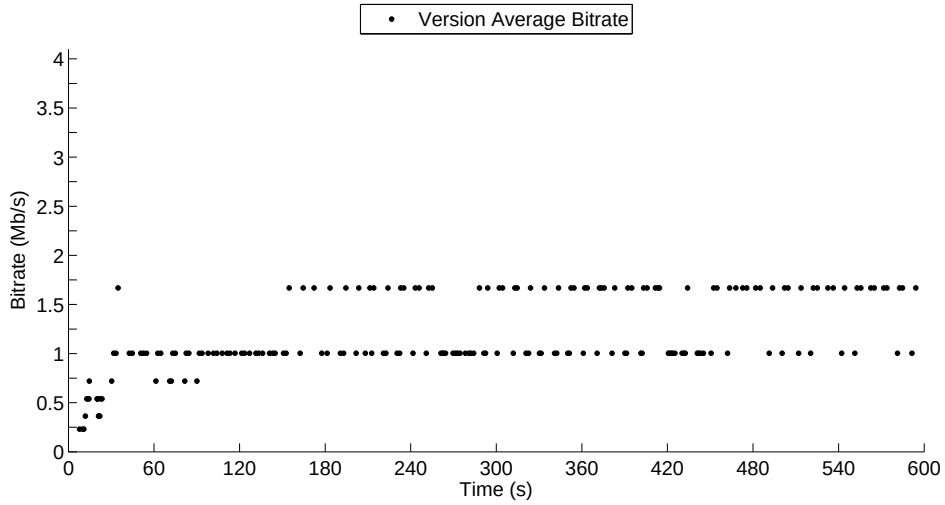

Figure 4.36: PC Browser: Oscillations between 4 Mbps and 0.5 Mbps for 5 s

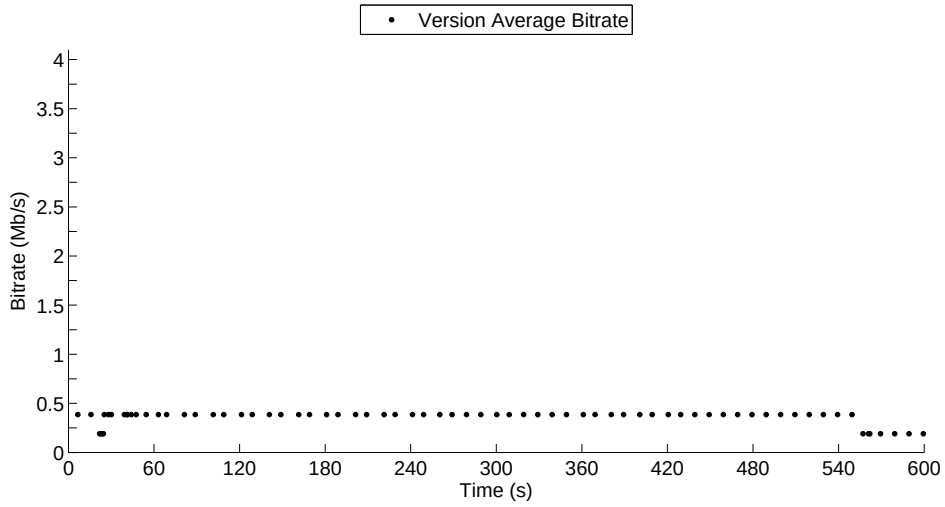

Figure 4.37: iPad: Oscillations between 2 Mbps and 0.25 Mbps for 10 s

Chapter 5

Prefetching Results

In this chapter the impact of the prefetching proxy on the PC Browser and iPad player's performance is presented. When there is a bottleneck at the server or in the network, the available bandwidth for the TCP traffic is divided up among the connections at the bottleneck. In this situation multiple connections can become very beneficial. By having a proxy open extra connections to the server the prefetching proxy can acquire more shares of the bottleneck's available bandwidth. As the proxy obtains more shares of the available bandwidth it can prefetch and cache chunks in advance of the player requesting them. Once cached the chunks can be served quickly to the player without any concerns with respect to the server's pacing or the network between the proxy and the server. As a result the player observes a higher available bandwidth by downloading the cached chunks more quickly than retrieving them from the server and the video playback is a higher quality. Prefetching also effectively increases the player's buffer capacity as chunks can be quickly served by the proxy, which is very beneficial for mobile clients with limited hardware resources. This extra buffer capacity also gives some leeway in the player's rate-adaptation algorithm, as a drop in the quantity of cached data at the proxy goes unnoticed by the player until the cached data is depleted. This allows the available bandwidth time to return to a higher level, hiding short-term reductions. Hiding short-term available bandwidth reductions between the proxy and the server prevents the player from overreacting to available bandwidth decreases so the viewer does not encounter reductions in video quality.

Experiments: The prefetching experiments in this research are conducted in a semi-controlled environment using dummynet to control the levels of available bandwidth between the proxy and the server. Since Netflix is being used there are variables beyond the control of the experiment, such as background traffic in the Internet and load on the Netflix servers. The impact of these variables is believed to be low as this research considers fairly low levels of available bandwidth and bitrates of video. The prefetching policies are evaluated through experiments and tested under various conditions. First, the prefetching proxy is evaluated with a stable level of available bandwidth between the proxy and the server. These experiments are beneficial in determining if prefetching can improve the video quality level while also paying attention to the stability in the requested versions. It is important that the player can stabilize on a particular version of video rather than having the player frequently jump between versions. Oscillations to the available bandwidth are then investigated. In these experiments the available bandwidth alternates between two levels with a constant duration between

each level change. Ideally, the player should stabilize its requested version and not be affected by the frequent changes from a high available bandwidth to a lower available bandwidth. Then increases to the available bandwidth are investigated to see how prefetching impacts the player's performance if there are short spikes in the available bandwidth. Finally, decreases to the available bandwidth are evaluated for both long-term and short-term changes. In the case of long-term change, the available bandwidth is set at a high level then decreased at a point during playback, remaining in the new region for the duration of the experiment. In these experiments it is interesting to find how the player obtains its new steady state or if the player can maintain a higher quality level. Short-term variation experiments are also run where the available bandwidth is dropped three consecutive times. These experiments are of interest to determine whether it is possible to maintain the video quality long enough for the available bandwidth to recover.

Proxy Placement: There are different options for where a proxy could be placed to provide a positive impact for the player. To be beneficial the proxy should be placed with the bottleneck occurring between it and the server rather than the bottleneck occurring between the client and the proxy. This is because the proxy uses prefetching to reduce the impact the bottleneck has on the player. If the bottleneck occurred between the player and the proxy, then the player would still feel its impact upon requesting a video chunk. Such bottlenecks could occur at the server, a congested router, or as the result of the service provider throttling connection bandwidth. A CDN could distribute them within its network to allow for clients to see improved performance without the need to set up entire servers with replicated content. Going to a more local level the proxy hardware could be placed inside a local Internet Service Provider's network to improve the service to their customers. One example where an a proxy may allow for improved performance is in cellular networks. Cellular networks have centralized components, meaning all the traffic from many cell towers goes to central locations before being routed to the wired Internet. It is expected that as cellular traffic continues to increase these central locations may become bottlenecks and Woo et al. [79] suggest that placing caches between the towers and the central locations would reduce redundant traffic to the central locations. Prefetching proxies could also be placed at these locations so the impact of the centralized bottleneck could be alleviated by prefetching. Proxies could also be placed inside residences. There would be benefit to customers using devices with minimal memory resources to use for buffering content such as mobile devices inside their homes and it also alleviates concerns with the bottleneck being the access link in the home.

Caching and Prefetching: It is interesting to compare caching and prefetching in this context but they solve inherently different problems. Namely, prefetching is designed to improve the performance for the current client whereas caching is meant to improve the performance for future clients. Considering that the majority of Netflix content is older content and no longer a first run, it is fair to say that there is little hot content so caching beyond that implemented by Netflix itself in their distributed system may not be particularly useful. However, prefetching is beneficial for all levels of popularity. With potentially hot content if the prefetching proxy is paired with a web cache then the chunks may be served or retrieved in

advance and cached. For cold content, which would not otherwise be in the cache the prefetching proxy can still retrieve the chunks in advance to provide improved performance.

Bottleneck Type: If there are many clients sharing the bottleneck at a router or loaded server and the proxy opens up N connections, then the proxy obtains N shares of the available bandwidth. For example, if there are 1000 other connections sharing the bottleneck and the player does no prefetching then the player gets 1/1001 of the available bandwidth. If the proxy opens up N connections to download data in parallel then the proxy can achieve $(N)/(1000+N)$ shares of the available bandwidth, greatly improving throughput. This type of situation is the main focus of this research. However it is possible that the bottleneck link will not have many clients on it, an example of a situation like this is if the proxy is in a home and the bottleneck is the access link, when this occurs there may be a small number of other connections. If there is one other connection sharing the bottleneck link and no prefetching is used then the player gets 1/2 of the available bandwidth. If the proxy then opens up N connections then the proxy gets $(N)/(1+N)$ so rather than the proxy taking a small amount of the bandwidth share from many other connections it is taking a large chunk of the bandwidth it already had access too. This means the prefetching agents are competing for the available bandwidth with each other. This situation is also investigated and evaluated in detail in Section 5.6 where the prefetching agents are competing against each other for bandwidth rather than each having their own dedicated piece.

Overhead: When considering the case where the available bandwidth is on a per connection basis prefetching more ahead does not run the risk of reducing video performance. However, by prefetching further ahead there is a possibility of more overhead in terms of chunks that are prefetched but never requested by the player. Overhead occurs when versions change as the prefetched chunks will never be requested or when the user terminates the video before playback completes. Viewers often terminate a video early for many different reasons such as to lack of interest, lack of time to finish the video, or realizing they have viewed the video previously [19, 23, 36, 53, 69, 80]. These conditions are motivation to avoid extreme prefetching behaviour in which video data is prefetched as fast as possible. Another potential advantage of prefetching is in cases when videos are frequently terminated early. By prefetching chunks and buffering at the proxy rather than buffering large quantities of video data at the player it may be possible to reduce the necessary size of the buffer at the player and maintain the playback robustness to network variation. In doing so the number of wasted chunks that have been downloaded over costly wireless links and buffered at the player when the video is terminated early can be reduced.

This chapter starts by discussing the N-Ahead prefetching scheme. This simple scheme offers a dramatic improvement to the observed video quality levels when compared to the experiments run in Chapter 4. When downloading with an independent pipe for each connection one extreme is considered; it is also interesting to consider the other extreme where the prefetching proxy only has a single pipe of available bandwidth which

must be shared among all connections, as case which is considered later in Section 5.6. Section 5.7 presents one potential algorithmic improvement to the prefetching proxy by optimistically prefetching a higher quality and Section 5.8 concludes the chapter.

5.1 N-Ahead Prefetching

1-ahead prefetching involves the proxy prefetching chunks to stay one chunk ahead of the player, for the currently requested version. Essentially, when a request is received for chunk number n the proxy should request chunk number $n+1$ for the same version in anticipation of the player requesting it. This means if there is currently adequate bandwidth to download chunks in their entirety prior to them being requested there is only parallelism at the beginning of the video when both the initial request and the prefetched chunk are requested. For example, assume chunk n has been prefetched completely to the proxy and is cached in its entirety. At this point there is no network activity between the proxy and server. Once the player sends a request for chunk n the proxy sends a request to the server for chunk $n+1$ and chunk n is sent to the client. Again there is no parallelism in the requests being sent to the server. Now consider the case where the proxy is not able to completely download a chunk prior to it being requested. In this case the proxy has already started prefetching chunk n when the request for chunk n is received but the download has not finished. Here, the proxy still requests chunk $n+1$ from the server while streaming chunk n to the client, allowing the utilization of parallel connections between the proxy and the server. What this means is that there are a maximum of two connections downloading video from the server at any given time, however when only one connection is active the proxy does not obtain twice the available bandwidth.

N-ahead is the same concept as 1-ahead prefetching, except, rather than just prefetching one chunk in advance the proxy prefetches N chunks in advance. When a request from the player for chunk n is received at the proxy, prefetching for the next N chunks takes place as needed. This does not mean that there are always at least N concurrent connections downloading data from the server. Rather, when the quality level remains the same for consecutive chunks, some of those chunks have already been requested by the proxy and thus to remain N ahead the proxy may only need to prefetch one extra chunk. For example, say at chunk number n the quality level is switched from version y to version z and version z has not been previously requested. At this time the request for chunk number n is forwarded to the server as it has not been previously prefetched and each of the successive chunks all the way to chunk $n+N$ are also prefetched. Now when a request comes in from the player for chunk $n+1$ it is not necessary to request chunks $n+1$ through $n+N$ as they have already been requested and potentially finished download. Therefore, it will only be necessary to request chunk $n+N+1$ in order to stay N ahead. Another situation where the player will not need to send all N prefetch requests at once is if the player changes versions to a version that already has some of the relevant chunks cached or currently being prefetched. For example, if after the current quality level drops from version z to version y and the player increases back to version z again soon after then it is possible that some of

those chunks have been prefetched from the previous time version z had been requested.

The backfilling conducted by the iPad adds slight complexity as chunks are removed from the proxy once they are served to the player to reduce space overhead on the proxy. When the iPad does backfilling it is possible that the same chunk can be requested by the player multiple times. This means it could be prefetched by the proxy and served to the player but then the player could bounce between versions, which potentially results in backfilling and the request for a chunk prior to chunks that were previously served. If the proxy is to be N ahead it must be sure to check the cache and request the chunks to fill any gaps that may occur in the next N chunks at that version.

5.2 Stable Available Bandwidth

In this section the behaviour of the player with proxy prefetching and a stable available bandwidth level is presented. When there is no fluctuation to the available bandwidth using the prefetching proxy to add extra buffer capacity for safety from buffer underrun is not needed. However, the parallelism achieved during prefetching allows for the proxy to observe higher throughput so it may cache chunks in advance to reduce the time required to serve them once requested by the player. The reduced latency that occurs when prefetched chunks can be served from the proxy's cache rather than going to the server provides a significant advantage as the player is observing low chunk latency and as a result can improve the requested video quality level.

5.2.1 PC Browser

When considering the PC Browser player the stable available bandwidth experiments are conducted at 0.5 Mbps, 1 Mbps and 2.375 Mbps to represent low, moderate, and high available bandwidth levels. Using higher levels of available bandwidth is unnecessary as the bitrate of the video is low enough for the video to be played back at the highest quality level, even with no proxy prefetching. Experiments are run for 10 minutes of playback using 1-ahead prefetching though to 4-ahead prefetching inclusive. The performance results are shown in Figure 5.1, together with the results for no prefetching (labelled "0" in the plot legends).

0.5 Mbps: When no prefetching is used with the available bandwidth set at 0.5 Mbps the player is capable of streaming the lowest quality level without interruption. When 1-ahead prefetching is used the player is capable of achieving a stable playback level at version 3 at approximately 7 minutes of the experiment with some minor instability prior to that. As N increases the level of instability early in the video playback increases. At 2-ahead and 3-ahead the player stabilizes on version 5 at approximately 8 minutes of the experiment. When running experiments with 4-ahead prefetching there is significant instability prior to the player smoothing out at the highest quality level, approximately 9 minutes into the experiment. The increasing instability as N increases is related to the player observing a high throughput as prefetching buffers data, then realizing the low throughput that occurs when the player changes to a higher quality level and

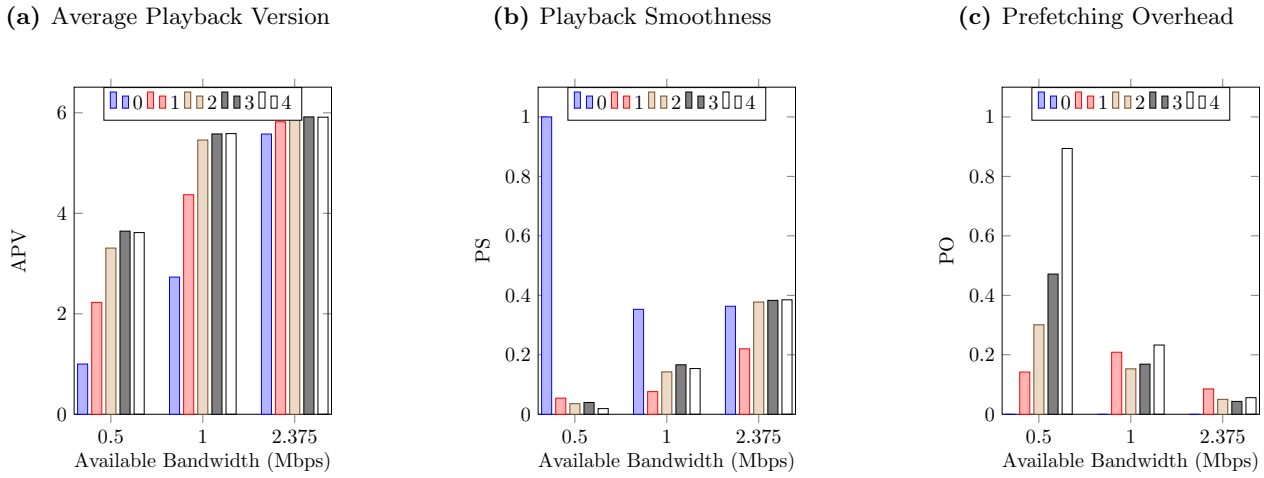

Figure 5.1: PC Browser: Stable Available Bandwidth

there is no prefetched data at the proxy to serve for that version. The stability occurs once the buffer occupancy has been built and the player remains at the higher version for a long enough period of time that the proxy can buffer the chunks sufficiently fast to maintain the higher quality. The extreme player instability of 4-ahead prefetching can be seen in Figure 5.2. It is important to note that because the player was capable of stabilizing at the highest quality with 4-ahead prefetching near the end of the experiment it is likely that the APV and the PS would both increase as the play duration increased. There is more uncertainty for the lower values of N because if they did eventually stabilize within the experiment time it was on an intermediate quality level. In these cases it is plausible that the player could remain stable but it is also possible that prefetching could eventually give the player more confidence to increase the quality level and not be able to sustain it.

1.0 Mbps: When no prefetching is used with the available bandwidth set at 1 Mbps the player is capable of streaming the third version in steady state after 2 minutes of ramp up time. When 1-ahead prefetching is used the player is capable of stabilizing on the fifth version shortly after 5 minutes of the experiment, after several previous attempts at that version. When 2-ahead prefetching is considered the player has similar behaviour, however it manages to stabilize on the highest quality level just before 4 minutes of the experiment after attempting a handful of times earlier. 3-ahead and 4-ahead prefetching display very similar behaviour to each other, with the player stabilizing at the highest quality level approximately 2 minutes into the experiment after previously failing to maintain it for a handful of earlier attempts. The behaviour where the player makes multiple attempts at a higher version but reduces quality several times before finally stabilizing can be seen in Figure 5.3 where the proxy is prefetching 2-ahead. In this case the player is much more stable than seen earlier at 0.5 Mbps in Figure 5.2 because the observed bandwidth after the player increases quality level but before there is any prefetched data is higher. This prevents the player from drastically reducing quality after increasing it when there is no buffered data.

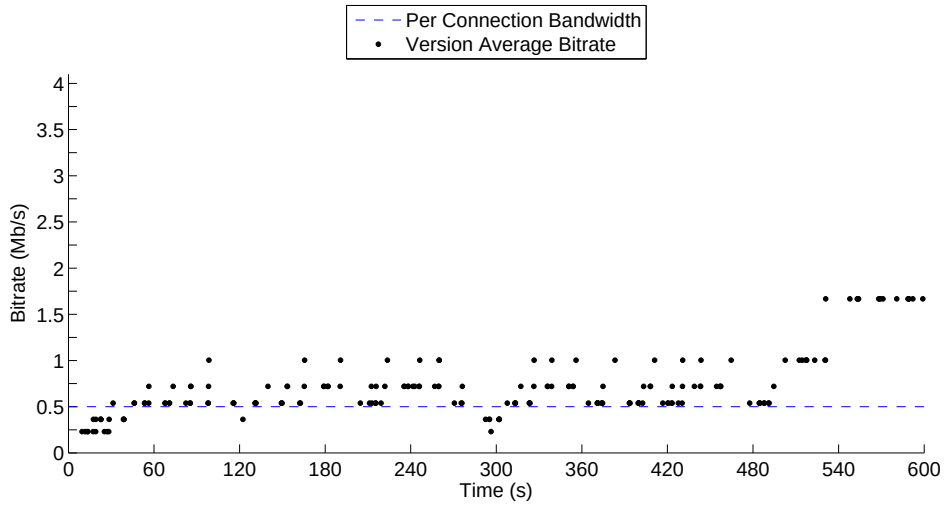

Figure 5.2: PC Browser: 0.5 Mbps Stable Per Connection Available Bandwidth with 4-Ahead Prefetching

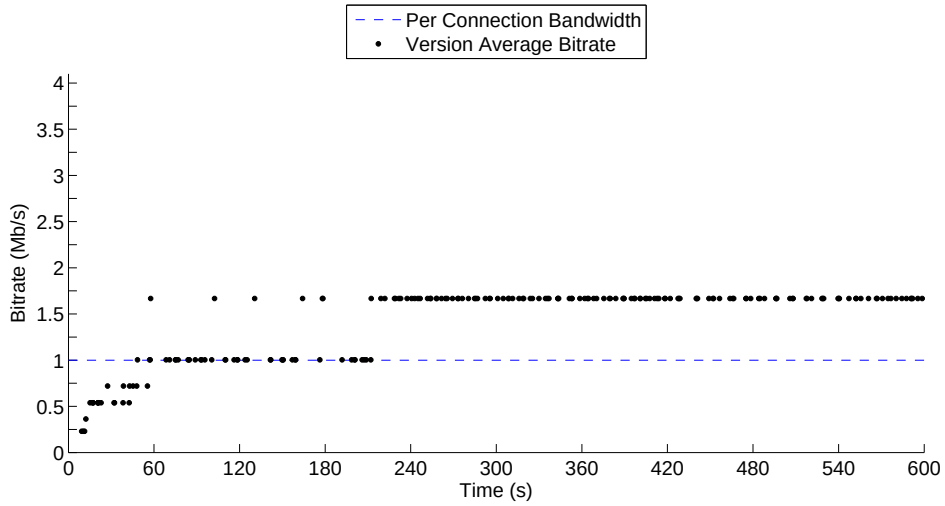

Figure 5.3: PC Browser: 1 Mbps Stable Per Connection Available Bandwidth with 2-Ahead Prefetching

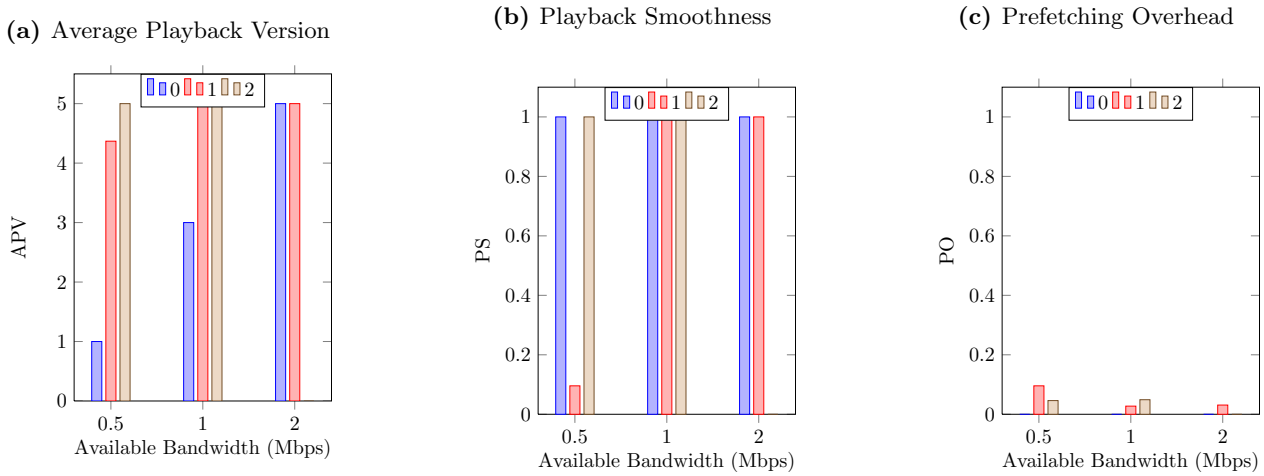

Figure 5.4: iPad: Stable Available Bandwidth

2.375 Mbps: When no prefetching is used at 2.375 Mbps the player is capable of achieving steady state for the highest quality level approximately 2.5 minutes into the experiment. With 1-ahead prefetching the player gets to the highest quality just before 2 minutes from the beginning of the experiment after trying several times earlier. 2-ahead prefetching improves on this by achieving steady state at the highest quality level at approximately 30 seconds. 3-ahead and 4-ahead offer further improvement by allowing the player to achieve steady state at the highest quality level after only a few chunks.

5.2.2 iPad

Experiments are done with the iPad player at 0.5 Mbps, 1.0 Mbps, and 2.0 Mbps to represent low, moderate and high available bandwidth levels. Experiments are run for 6 minutes using 1-ahead and 2-ahead prefetching. The results can be seen in Figure 5.4, together with the results for no prefetching (labelled "0" in the plot legends).

0.5 Mbps: When no prefetching is used with the available bandwidth set at 0.5 Mbps the player is only capable of streaming the lowest video quality level. When using 1-ahead prefetching the player struggles somewhat as a result of overconfidence due to more quickly receiving prefetched content. The prefetching gives it substantial early confidence and the playback freezes for rebuffering very briefly. The player does manage to play relatively well after that, dropping the quality from the highest level only a few times during the 6 minutes of playback. However, these drops are quite large in magnitude. Figure 5.5 displays this behaviour. Using 2-ahead allows the player to successfully stream the highest quality level without drops in video quality, however the video still encountered a very brief freeze for rebuffering at the start of playback. Prefetching further ahead does not seem to have added benefit. The video freezing can be attributed to the backfilling that occurs at the player. Once the player sees the higher chunk throughput it attempts to backfill

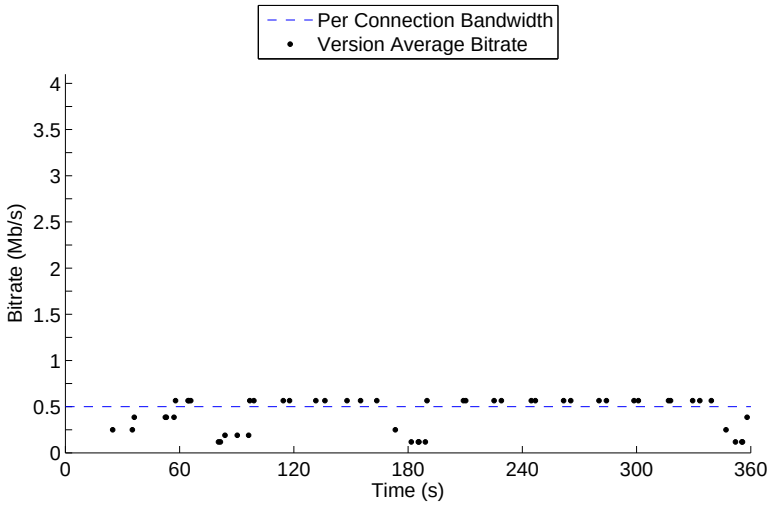

Figure 5.5: iPad: 0.5 Mbps Stable Per Connection Available Bandwidth with 1-Ahead Prefetching

the lower quality chunks with higher quality chunks and there is no data prefetched at the higher quality so the throughput decreases and the player's buffer depletes momentarily.

1.0 Mbps and 2.0 Mbps: When no prefetching is used with the available bandwidth set at 1.0 Mbps and 2.0 Mbps the player can achieve steady state in the third and fifth version, respectively. When prefetching is utilized the player is capable of streaming the highest video quality from the beginning of playback with only 1-ahead prefetching.

5.3 Oscillations

Oscillations are important to observe as competing flows can come and go from the bottleneck link; further, the available bandwidth in wireless scenarios is often very unstable.

5.3.1 PC Browser

When considering oscillations to the available bandwidth, experiments are conducted with the PC Browser player for 10 minutes of video playback using oscillations between high levels of 4 Mbps and 2.375 Mbps, and low levels of 1 Mbps and 0.5 Mbps. These oscillations occur for one of five durations: 0.25, 1, 5, or 10 seconds. Results are shown for no prefetching, 1-ahead prefetching and 2-ahead prefetching.

2.375 - 1 Mbps Oscillations: When relatively low magnitude oscillations occur such as oscillating between 2.375 Mbps and 1 Mbps no prefetching results in the player getting to the fifth version shortly after 2 minutes of the experiment, dropping to the fourth version on occasion with longer oscillations resulting in more instability. When prefetching the low level of the oscillation at 1 Mbps only has a minor effect as the player

(a) Average Playback Version **(b)** Playback Smoothness **(c)** Prefetching Overhead

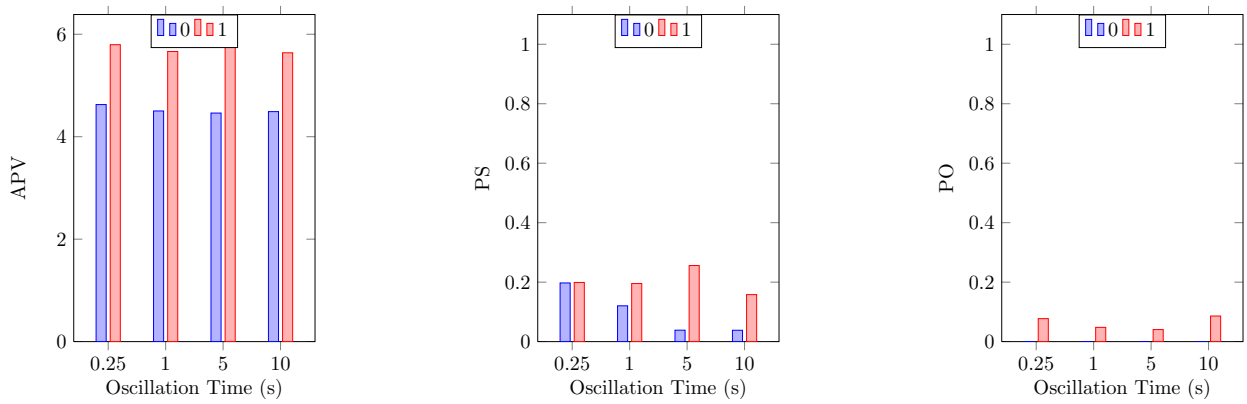

Figure 5.6: PC Browser: Prefetching Results for Oscillations 2.375 Mbps - 1 Mbps

always gets to the highest quality of video. When using 1-ahead, ramp up takes slightly longer than 1-ahead at 2.375 stable available bandwidth, as the player fails to maintain the highest quality for several attempts. The results are shown in Figure 5.6.

2.375 - 0.5 Mbps Oscillations: Increasing the magnitude of the oscillation by changing the low level from 1 Mbps to 0.5 Mbps has only a slight impact on the ramp up time when compared to 2.375 Mbps to 1 Mbps oscillations for 0.25, 1, and 5 second oscillations, when using 1-ahead prefetching. When considering longer oscillations of 10 seconds there are several short duration video quality drops throughout the playback of the video. When using 2-ahead prefetching the player is able to quickly ramp up and maintain the highest quality at approximately 30-60 seconds of the experiment, with longer ramp up times as the oscillation time increases from 0.25 seconds to 10 seconds. The performance metrics are shown in Figure 5.7 and the prefetching results for 10 second oscillations can be seen in Figures 5.8, 5.9, and 5.10.

4 Mbps - 0.5 Mbps Oscillations: For oscillations with a very large variation between the bandwidth levels, the player performed poorly with no prefetching, as indicated by the frequent changes to the quality level. Prefetching improved the performance significantly; the player ramps up quickly within a minute for the oscillations under 10 seconds, and with 10 second oscillations takes just over 2 minutes to get to the highest quality level for 1-ahead and under 1 minute for 2-ahead prefetching. The results are shown in Figure 5.11. The advantage of prefetching can be seen clearly in Figure 5.12 with 1-ahead prefetching when comparing back to Figure 4.36.

5.3.2 iPad

Experiments with oscillations to the available bandwidth are conducted with the iPad player for 6 minutes of playback and oscillations between high levels of 4 Mbps, 2 Mbps, and 1 Mbps and a low level of 0.25 Mbps.

(a) Average Playback Version **(b)** Playback Smoothness **(c)** Prefetching Overhead

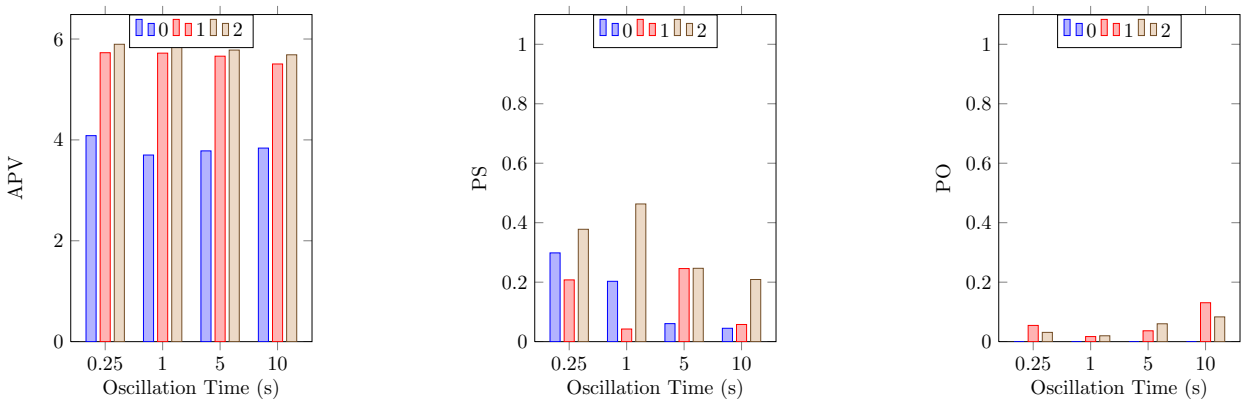

Figure 5.7: PC Browser: Prefetching Results for Oscillations 2.375 Mbps - 0.5 Mbps

.

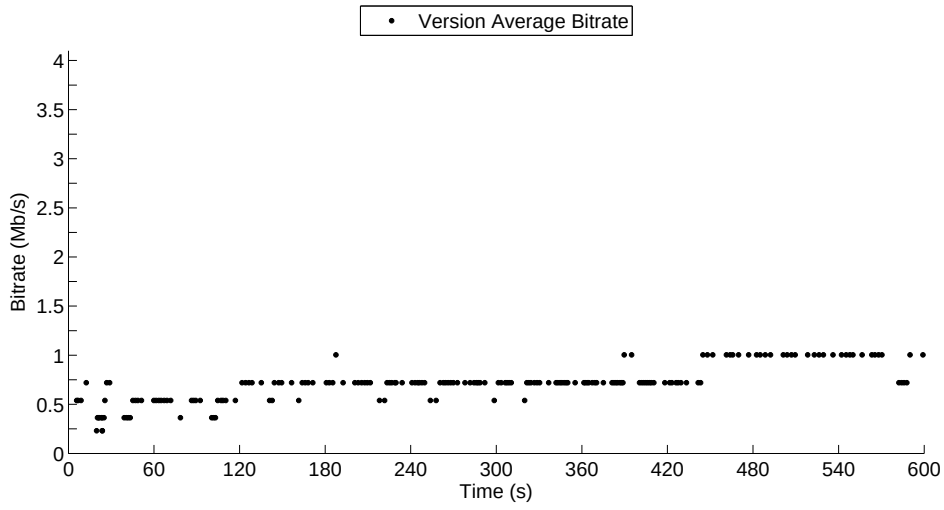

Figure 5.8: PC Browser: Oscillations from 2.375 Mbps to 0.5 Mbps for 10 s with No Prefetching

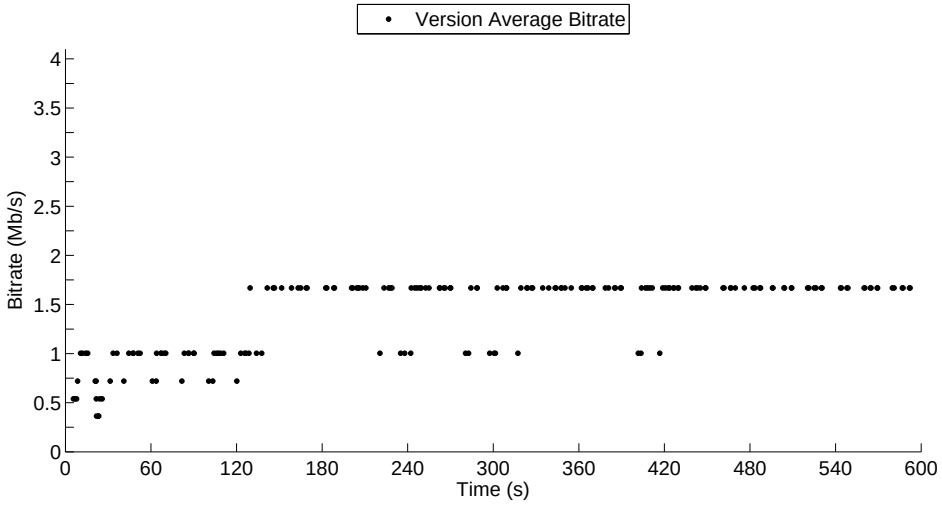

Figure 5.9: PC Browser: Oscillations from 2.375 Mbps to 0.5 Mbps for 10 s with 1-Ahead Prefetching

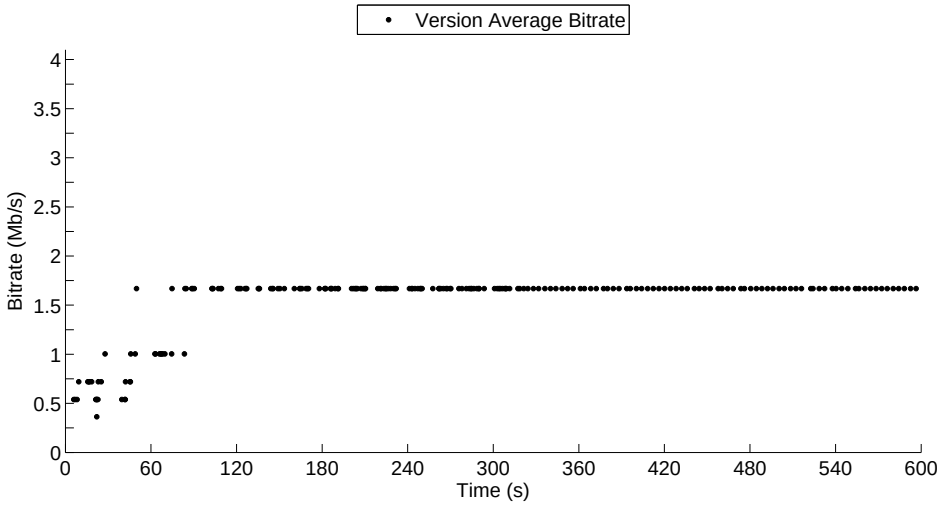

Figure 5.10: PC Browser: Oscillations from 2.375 Mbps to 0.5 Mbps for 10 s with 2-Ahead Prefetching

(a) Average Playback Version **(b)** Playback Smoothness **(c)** Prefetching Overhead

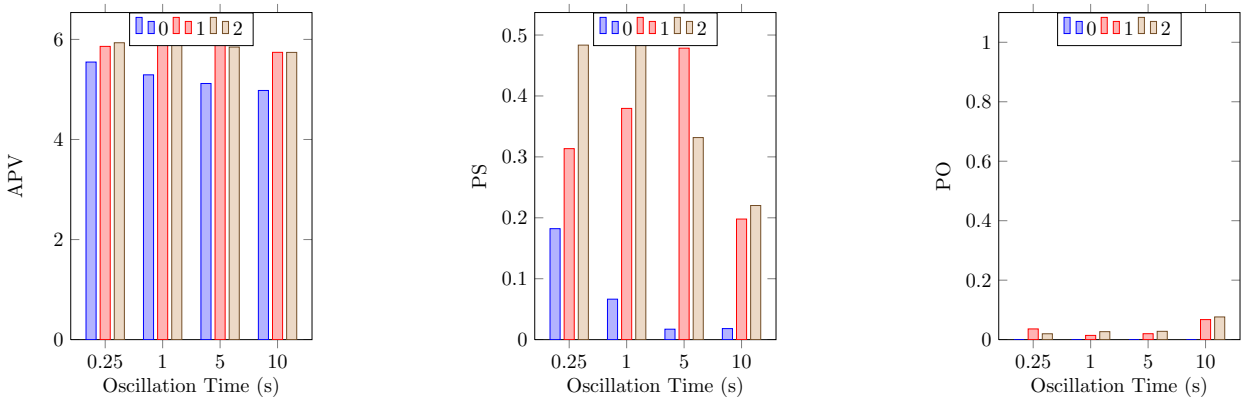

Figure 5.11: PC Browser: Oscillations 4 Mbps - 0.5 Mbps

.

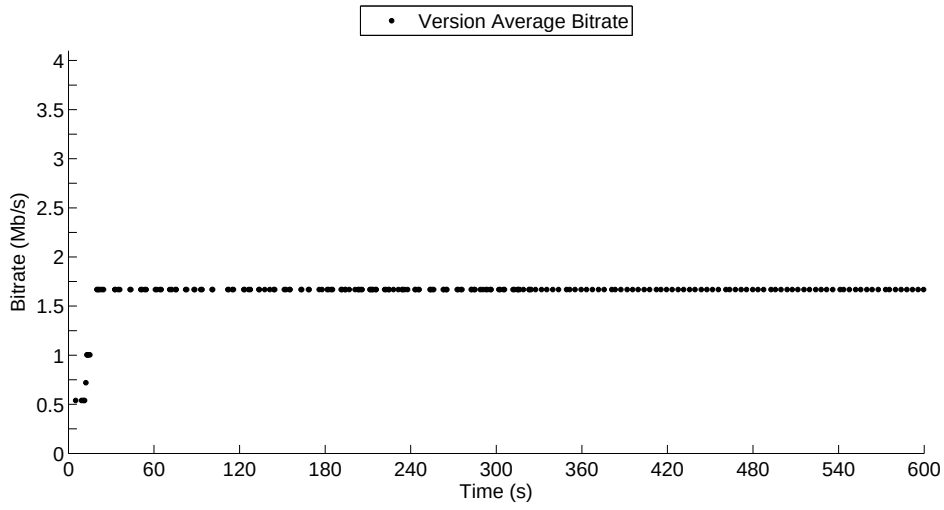

Figure 5.12: PC Browser: Oscillations from 0.5 Mbps to 4 Mbps for 5 s with 1-Ahead Prefetching

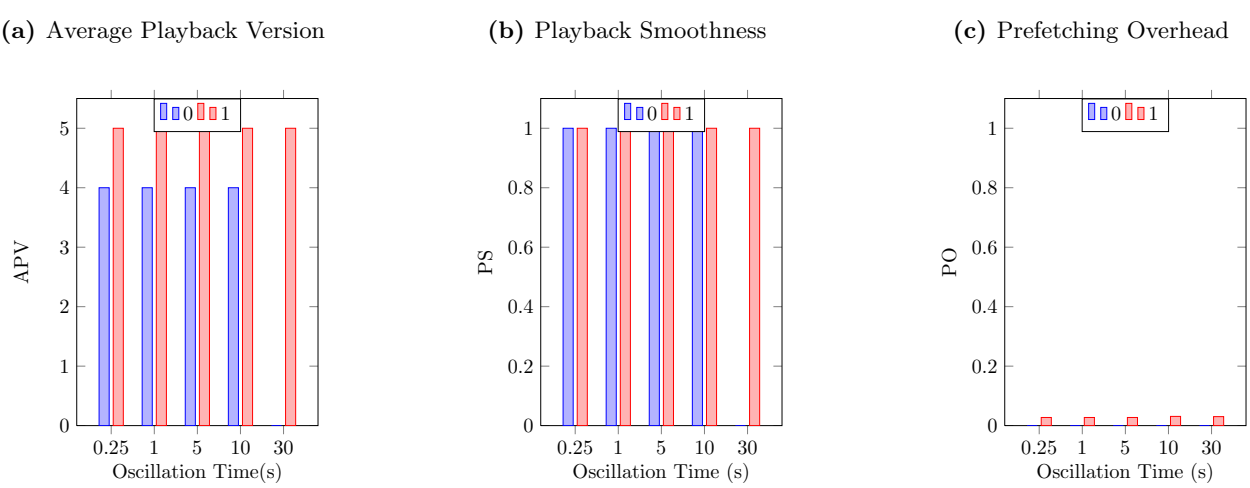

Figure 5.13: iPad: Oscillations 2 Mbps - 1 Mbps

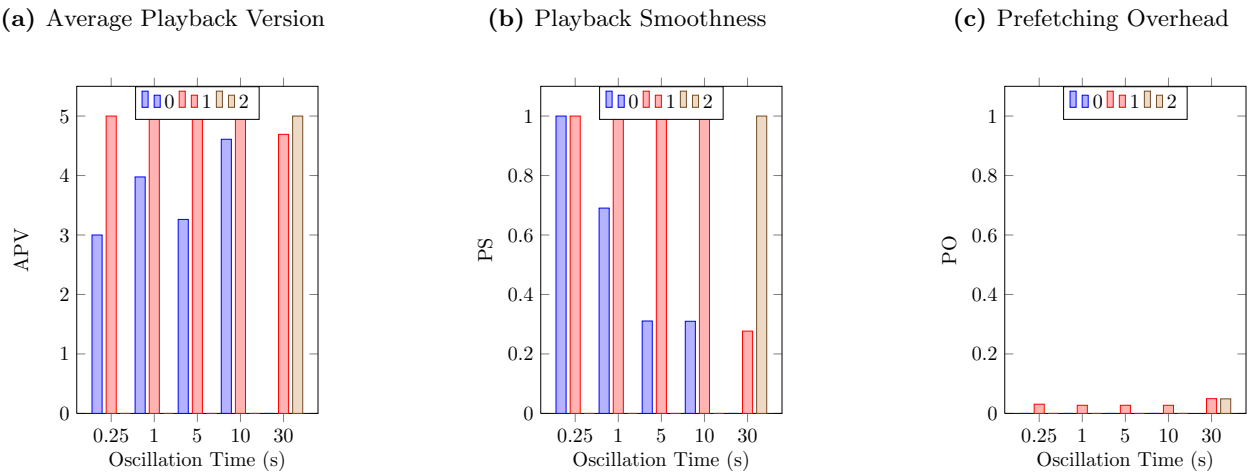

Figure 5.14: iPad: Oscillations 2 Mbps - 0.25 Mbps

These oscillations occur for the following durations of 0.25, 1, 5, 10, or 30 seconds. Results are shown for no prefetching, 1-ahead prefetching and 2-ahead prefetching.

For the case of oscillations to the available bandwidth when no prefetching is used, the iPad player is very stable. However, the average playback version is not as high as would be possible with prefetching. When prefetching is used the player is capable of backfilling to play the highest quality version from the beginning of playback using only 1-ahead prefetching in most cases. The only exception to ideal playback quality is available bandwidth oscillations between 2 Mbps and 0.25 Mbps with 30 second oscillations where the requested video quality reduced near the end of the experiment before recovering. In this case using 2-ahead prefetching was adequate in achieving steady state at the highest quality level. Figures 5.13, 5.14, and 5.15 show the performance metrics for these experiments.

(a) Average Playback Version **(b)** Playback Smoothness **(c)** Prefetching Overhead

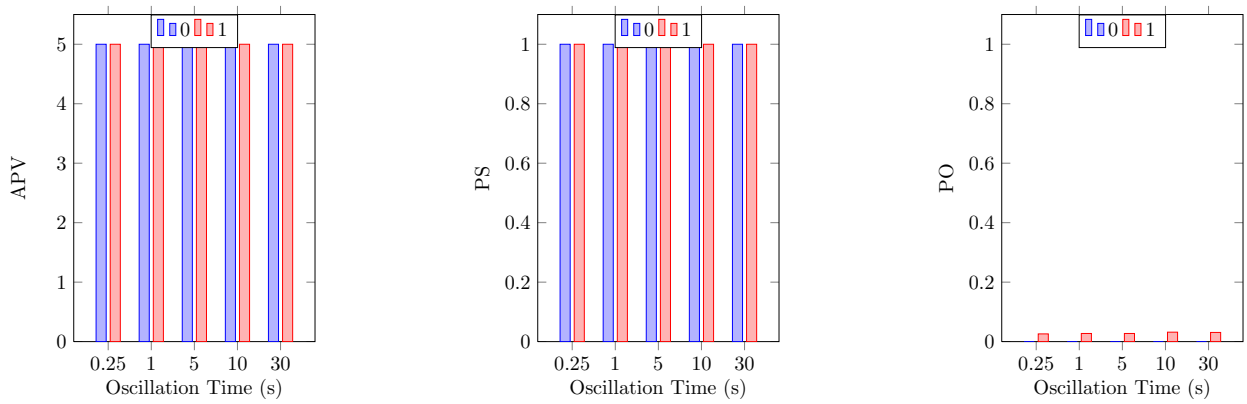

Figure 5.15: iPad: Oscillations 4 Mbps - 0.25 Mbps

5.4 Increasing Bandwidth

Long-term increase experiments are not presented due to the short-term spike experiments being adequate to get to the highest quality level during a spike's duration. Further, they are not shown as the players converge relatively quickly to their new steady state without the need for prefetching. Thus long-term spike experiments are omitted and the focus is on areas where the prefetching proxy has a greater impact.

5.4.1 Short-Term Spikes

Short-term spikes involve spiking the available bandwidth three times for short durations with short intervals between the spikes to determine how the player reacts to these spikes.

PC Browser

When considering short-term spikes for the PC Browser player the spikes were introduced at 10 seconds and 60 seconds of video playback. The spikes are either 0.5 Mbps to 4 Mbps to show large magnitude spikes or 1 Mbps to 2.375 Mbps to show moderate magnitude spikes. They lasted for 1 second or 10 second durations with 2 second or 10 second intervals between the spikes.

Prefetching inherently provides a higher APV and impacts the PS in ways unrelated to short-term spiking as seen shown in Section 5.2.1 when a stable, but low level of available bandwidth was experimented with, so comparing the performance metrics is less useful in this case. However, there is still benefit to investigating these situations from a higher level perspective.

When considering the very short spikes of 1 second in duration the behaviour is very similar to no prefetching, when there is an impact the spike generally only affects the player very briefly, increasing the requested quality level for a short duration. Once the spikes become longer (10 second durations, for example) there is more of an impact when considering prefetching. The spikes result in a larger increase to

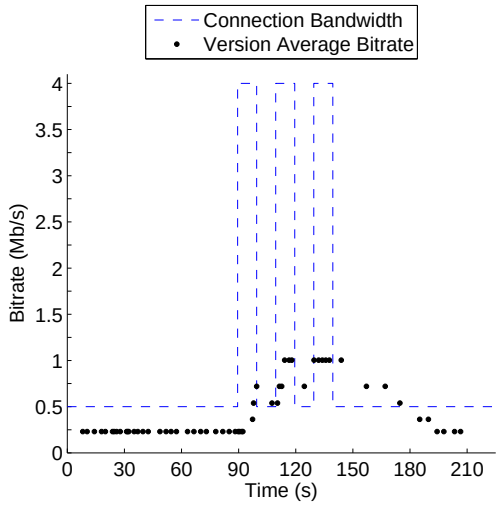

Figure 5.16: PC Browser: Spikes from 0.5 Mbps to 4 Mbps at 60 s of Playback with No Prefetching

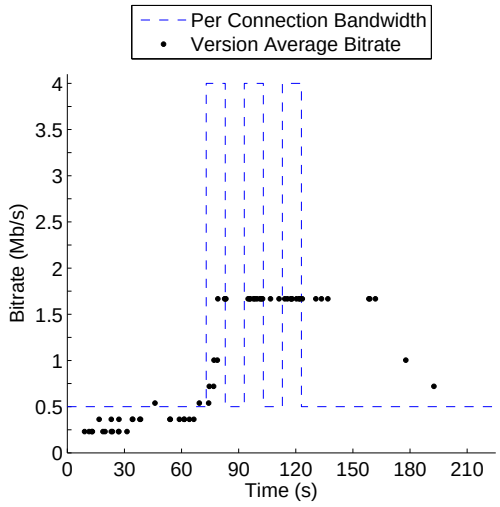

Figure 5.17: PC Browser: Spikes from 0.5 Mbps to 4 Mbps at 60 s of Playback with 2-Ahead Prefetching

the requested video quality that often lasts for a longer duration before decreasing. This means the viewer will experience longer durations at higher video qualities before it reduces back down. This behaviour can be seen when comparing Figure 5.16 and Figure 5.17, which look at no prefetching and 2-ahead prefetching for large magnitude spikes.

The time the spike occurs also has an impact on the behaviour of the player; spikes occurring later allow the player to remain more confident in maintaining the higher quality after the spikes. When looking at stable available bandwidth levels, 2-ahead prefetching at 1 Mbps of available bandwidth is adequate to play the highest quality level approximately 3.5 minutes into the experiment. Figure 5.18 and Figure 5.19 show 2-ahead prefetching for 10 second durations and 10 second intervals between 1 Mbps and 2.375 Mbps, beginning at 10 and 60 seconds of playback, respectively. The spikes that begin at 10 seconds of playback gives the player the initial confidence to request the highest video quality level but the prefetching is not enough to maintain it when the buffer occupancy is low. The spike occurring at 60 seconds again gives the player the confidence to increase to the highest video quality but at this point the buffer occupancy is high enough that the player is willing to persist at the highest quality level while the proxy gets ahead, so it stabilizes and remains at the highest video quality level. Results for the short-term spike scenarios used for these two figures but with no prefetching can be seen by referring back to Figure 4.12 and Figure 4.13.

iPad

When considering short-term spikes for the iPad player, spikes are introduced at 10 seconds or 60 seconds of video playback. These spikes occur from 0.5 Mbps to 4 Mbps. With prefetching, using spikes starting with a higher initial bandwidth (such as 1 Mbps), are unnecessary because the prefetching allows the player to play

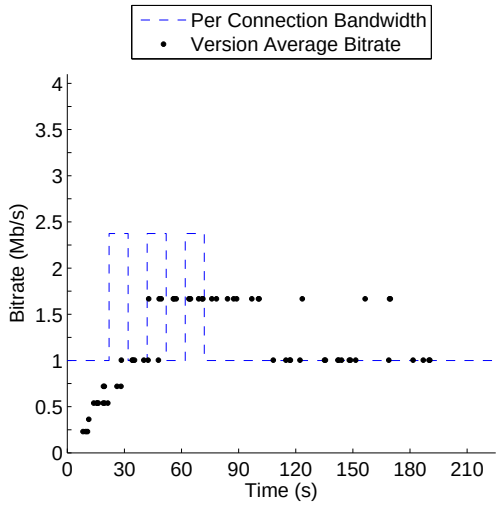

Figure 5.18: PC Browser: Spikes from 1 Mbps to 2.375 Mbps at 10 s of Playback with 2-Ahead Prefetching

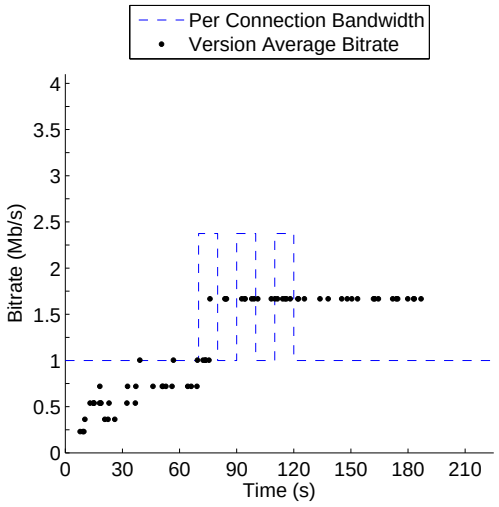

Figure 5.19: PC Browser: Spikes from 1 Mbps to 2.375 Mbps at 60 s of Playback with 2-Ahead Prefetching

the highest quality at higher levels of available bandwidth, which means spikes have no impact. Further, only 1-ahead prefetching is used as prefetching 2-ahead is enough to play the highest quality without the spikes.

One problem with prefetching at low bitrates with the iPad player is that the player receives high confidence when the proxy is serving prefetched chunks to the player so the player dramatically increases its requested bitrate. When this occurs there is no prefetched data at the proxy as the new quality level was not recently requested. The player then sees a dramatically reduced throughput and drops the requested quality back down. When a spike occurs, further benefit is possible, provided it is of large enough magnitude or for a long enough duration to allow the proxy to get ahead. Figure 5.20 and Figure 5.21 show how short-term spikes can impact the players stability when the available bandwidth is low for the majority of the experiment. Note that the two chunks of the lowest quality level during the short quality level reduction at approximately 140 seconds in Figure 5.20 were backfilled, which allowed for near optimal playback after the spikes.

5.5 Decreasing Bandwidth

Decreasing bandwidth is an important scenario to investigate since it can occur when competing flows at the bottleneck reduce the player's observed available bandwidth, which can result in a decrease in the requested video quality. Prefetching can absorb some of this reduction so it is not visible to the player and as a result the requested quality level can remain high.

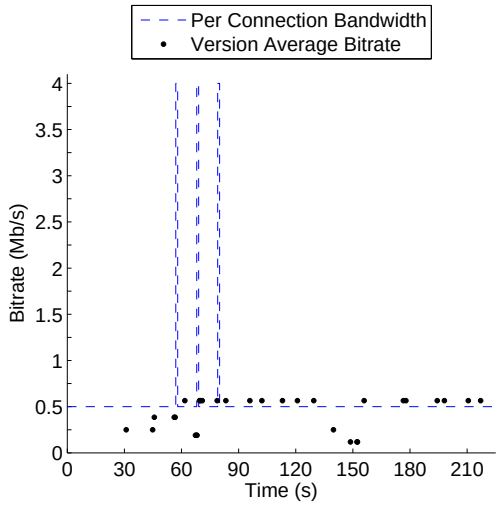

Figure 5.20: iPad: Spikes from 0.5 Mbps to 4 Mbps at 10 s of Playback for 1 s Durations and 10 s Intervals with 1-Ahead Prefetching

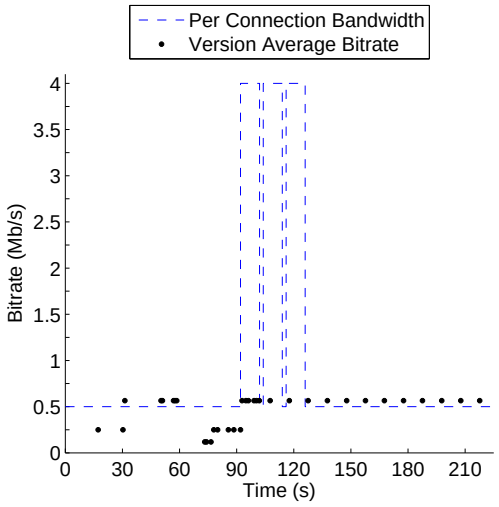

Figure 5.21: iPad: Spikes from 0.5 Mbps to 4 Mbps at 60 s of Playback for 10 s Durations and 2 s Intervals with 1-Ahead Prefetching

5.5.1 Long-Term Decreases

Long-term decrease experiments are conducted to show the impact a prefetching proxy can have during a sudden reduction to the observed available bandwidth that lasts the entire playback duration. Prefetching further ahead is likely to extend the player's time at a higher quality level, but by prefetching far enough in advance and utilizing parallel download for the chunks it may also be possible to remain at the higher quality level despite the drop in per connection available bandwidth.

PC Browser

When considering long-term decreases to the available bandwidth with the PC Browser player decreases were conducted from 4 Mbps to 0.5 Mbps to show large magnitude decreases and 2.375 Mbps to 1 Mbps to show moderate magnitude decreases. These decreases occurred at 10 seconds (early), 60 seconds (mid), and 240 seconds (late) of playback to determine the impact that extra time to increase buffer occupancy at the player and prefetch chunks at the proxy would have on the decrease.

2.375 Mbps - 1 Mbps: When the decreases to the available bandwidth occurs early in the video playback at 10 seconds using 1-ahead prefetching, the proxy does not have time to build its cache and there is instability resulting in decreased PS and increased PO for a moderate improvement to the APV. As N increases to two and three, the APV gets closer to optimal while PS improves with little impact to the PO (Figure 5.22). As the time that the decrease occurs becomes later in the video the prefetching shows increases in APV and PS, these increases are the result of the proxy having time to establish its cache at the high quality, then when the drop occurs the cache can simply maintain it. When the drop happens earlier, the cache has not been

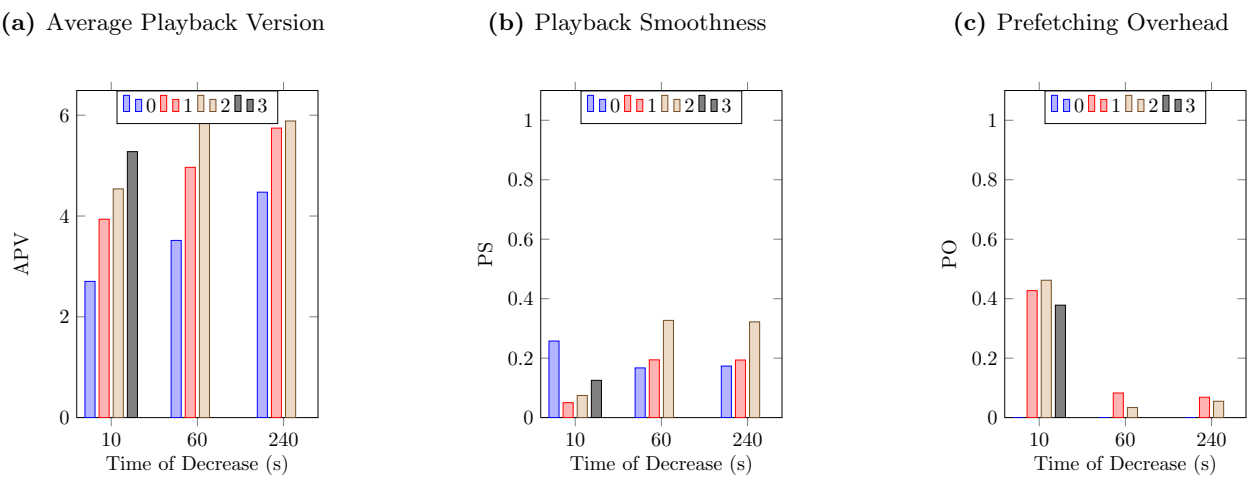

Figure 5.22: PC Browser: Long-Term Decreases 2.375 Mbps - 1 Mbps

built on the highest quality so there is instability as the player works up to the highest quality. This can be seen when comparing the 3-ahead prefetching experiment in Figure 5.23 for a decrease at 10 seconds and 2-ahead prefetching experiment in Figure 5.24 for a decrease at 60 seconds.

4 Mbps - 0.5 Mbps: Larger magnitude decreases behave similarly to lower magnitude decreases. When the decreases to the available bandwidth occur early in the video playback at 10 seconds the prefetching proxy does not have time to build its cache and as a result there is significant instability resulting in decreased PS and increased PO for a small improvement to the APV. However, the player can maintain a higher APV and increase the PS while progressively decreasing the PO when this decrease occurs later in the video. This scenario allows the proxy to build its cache and as a result serve chunks to the player quickly enough for the player to maintain confidence in its currently requested quality level while the proxy downloads future chunks in parallel. Figure 5.25 shows the results for early, mid and late decreases. Note that when a change occurs at a mid or late time the player is capable of maintaining the highest quality level for high levels of prefetching. This is because when the available bandwidth drops the chunks are downloaded in parallel, and that allows for adequate time to finish a chunk download in time for delivery to the player.

iPad

When considering long-term decreases to the available bandwidth with the iPad player decreases were conducted from high levels of 4 Mbps and 2 Mbps to low levels of 0.5 Mbps and 0.25 Mbps at 10 seconds and 60 seconds of playback. For determining if the players could maintain the current version after a bandwidth decrease experiments were run for a minimum of 6 minutes of playback.

When dropping from 4 Mbps to 1 Mbps at 10 seconds of playback the APV was 3.58 with a PS of 0.44 without the benefits of prefetching. When prefetching is utilized the APV remains at its maximum of 5 and PS is also maximized at 1. To increase the stress on the prefetching proxy experiments were then conducted

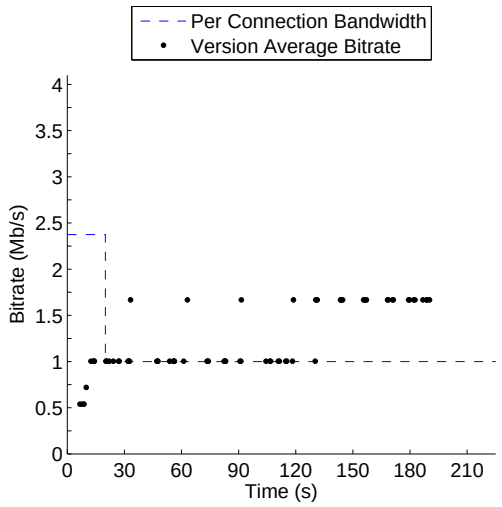

Figure 5.23: PC Browser: Long-Term Decrease from 2.375 Mbps to 1 Mbps at 10 s of Playback with 3-Ahead Prefetching

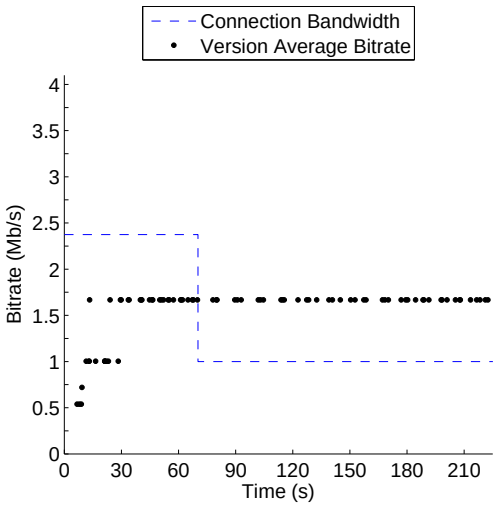

Figure 5.24: PC Browser: Long-Term Decrease from 2.375 Mbps to 1 Mbps at 60 s of Playback with 2-Ahead Prefetching

(a) Average Playback Version

(b) Playback Smoothness

(c) Prefetching Overhead

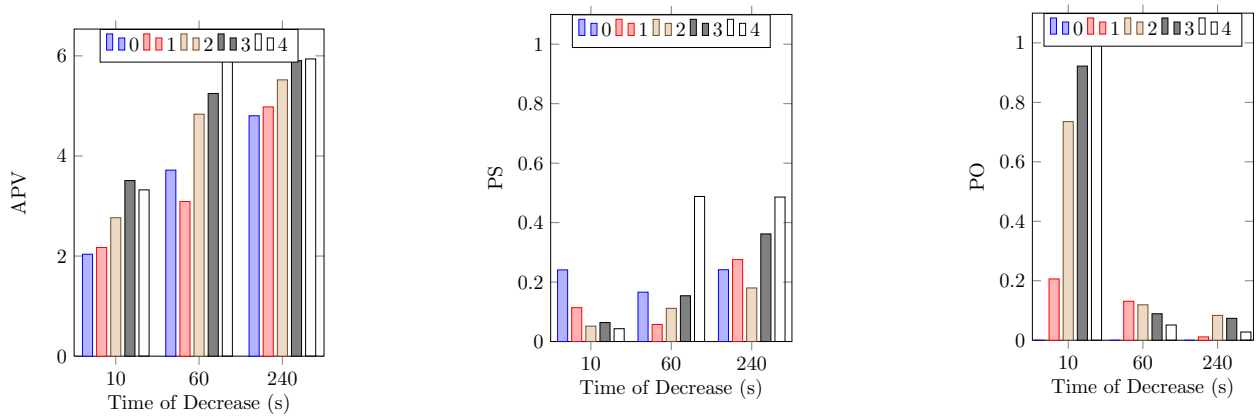

Figure 5.25: PC Browser: Long-Term Decreases 4 Mbps - 0.5 Mbps

with decreases from 4 Mbps to 0.5 Mbps and the player was still able to maintain the highest quality level with only 1-ahead prefetching as observed in Figure 5.28. To test an extreme case, decreases to 0.25 Mbps were made. When decreasing to 0.25 Mbps the player was capable of maintaining the highest quality level when using 3-ahead prefetching. When using less than 3-ahead prefetching the player would eventually drop video quality and encounter freezing for rebuffering and significant quality level instability.

Figure 5.26 and Figure 5.27 show how 3-ahead prefetching with available bandwidth drop at 10 seconds of playback outperforms 2-ahead prefetching with a drop at 60 seconds when decreases to 0.25 Mbps are done; not even the added time at a high level of available bandwidth to establish the proxy's buffer and increase the player buffer occupancy is sufficient for 2-ahead to remain effective in this case.

5.5.2 Short-Term Drops

Short-term drops are an important set of experiments as they show the prefetching proxy's ability to absorb brief decreases to the available bandwidth to maintain a high requested video quality.

PC Browser

Short-term drops for the PC Browser player are considered for durations of 1 second or 10 seconds, with 2 second or 10 second intervals. These changes occur at 10 seconds, 60 seconds or 240 seconds of playback to determine the impact of the player's buffer occupancy. Two available bandwidth regions are used, with large magnitude drops occurring between 4 Mbps and 0.5 Mbps and moderate magnitude changes occurring between 2.375 Mbps and 1 Mbps. Experiments are conducted using 1-ahead and 2-ahead prefetching, when beneficial.

Section 4.4.2 described how the buffer occupancy played a role in the player's ability to absorb bandwidth variation. Prefetching effectively increases the buffer occupancy by downloading chunks in advance to be served to the player quickly. In doing so the APV is increased and the PS is increased, with minimal PO. The result is smoother playback at the highest quality level.

2.375 Mbps - 1 Mbps For moderate magnitude drops 1-ahead prefetching is capable of reducing the impact of short-term drops and 2-ahead further reduces it by absorbing all variations at and beyond 60 seconds of playback. The player is still impacted by drops at 10 seconds of playback as the prefetching proxy cannot get ahead at the highest quality prior to the drops occurring. This behaviour can be seen by comparing Figure 5.30 and Figure 5.31 which show the player behaviour during drops of 10 second durations with 2 second intervals between 2.375 Mbps and 1 Mbps at 10 seconds of playback. Figure 5.29 details the performance metrics for moderate magnitude drops between 2.375 Mbps and 1 Mbps occurring at 60 seconds of playback.

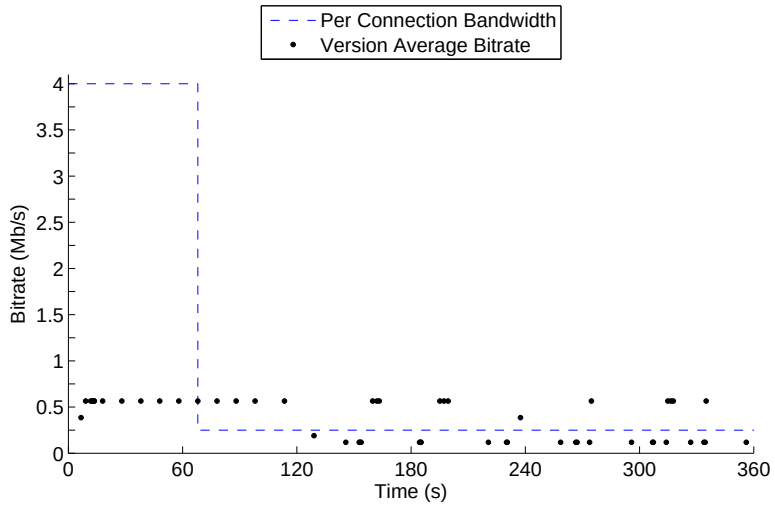

Figure 5.26: iPad: Long-Term Decrease from 4 Mbps to 0.25 Mbps at 60 s of Playback with 2-Ahead Prefetching

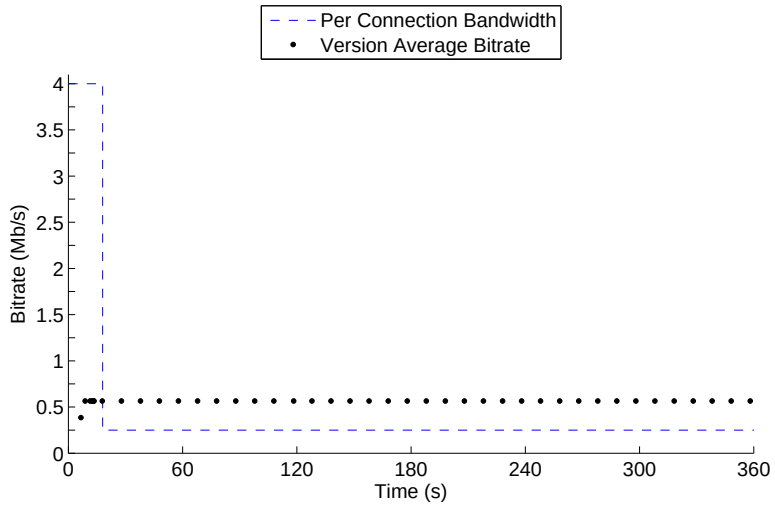

Figure 5.27: iPad: Long-Term Decrease from 4 Mbps to 0.25 Mbps at 10 s of Playback with 3-Ahead Prefetching

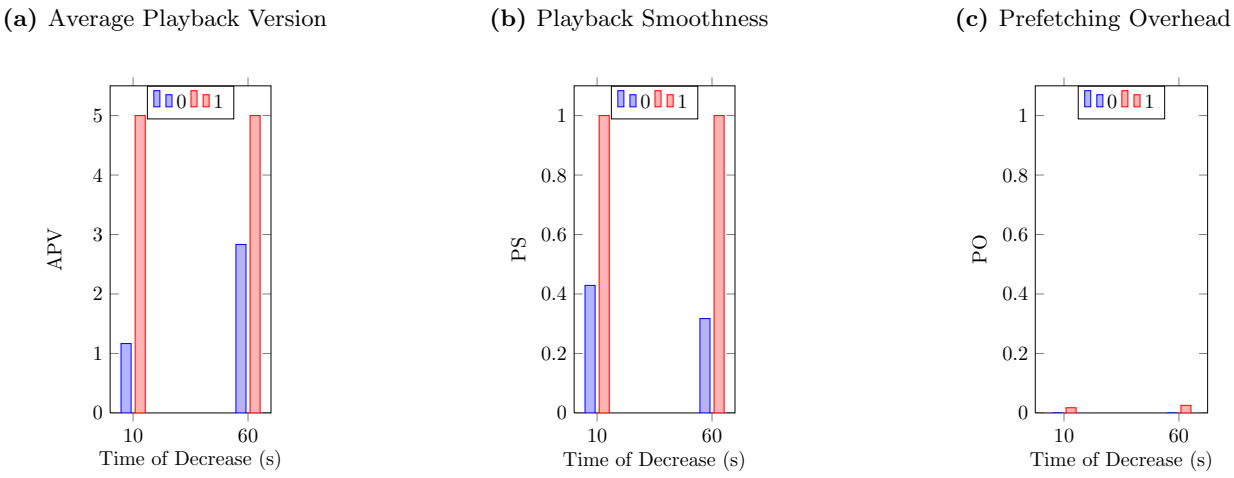

Figure 5.28: iPad: Long-Term Decreases 4 Mbps - 0.5 Mbps

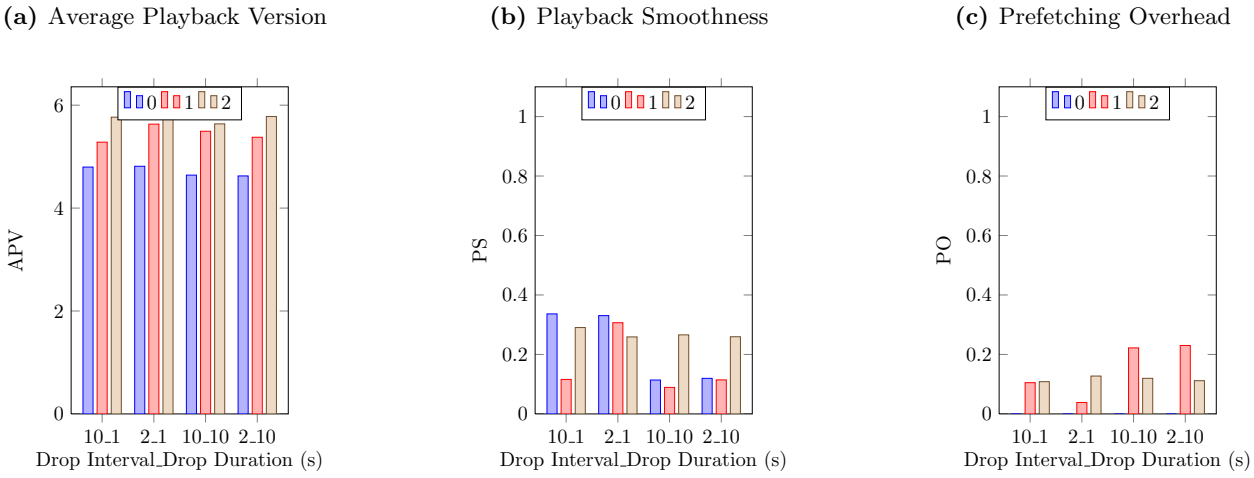

Figure 5.29: PC Browser: Drops 2.375 Mbps - 1 Mbps at 60 s

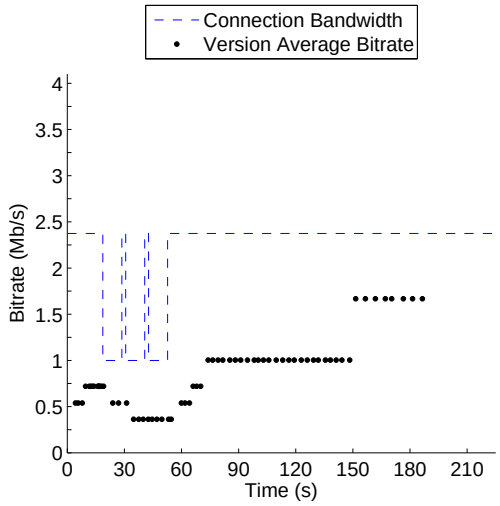

Figure 5.30: PC Browser: Drops from 2.375 Mbps to 1 Mbps at 10 s of Playback for 10 s Durations and 2 s Intervals with No Prefetching

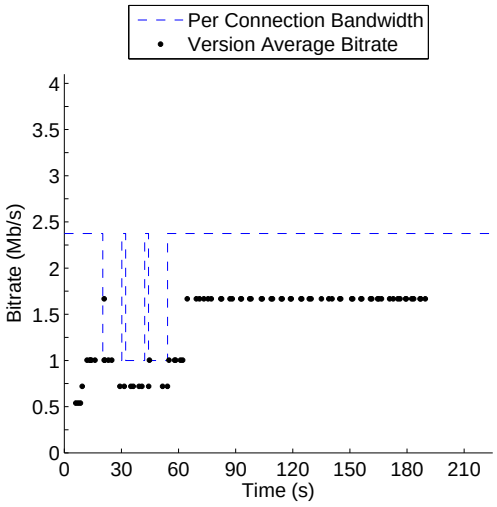

Figure 5.31: PC Browser: Drops from 2.375 Mbps to 0.5 Mbps at 10 s of Playback for 10 s Durations and 2 s Intervals with 1-Ahead Prefetching

4 Mbps - 0.5 Mbps When considering large magnitude drops the prefetching proxy has a significant impact. It is capable of absorbing short-term drops in all of the experiments with drops beginning at 60 seconds onward with only 1-ahead prefetching. Early in the video the effectiveness of prefetching is improved by prefetching 2-ahead as can be seen when comparing 1-ahead prefetching in Figure 5.32 and 2-ahead in Figure 5.33 with the no prefetching case in Figure 5.34. The player only makes minor decreases to the video quality as it quickly requests the highest quality. When considering the very short drops prefetching is able to completely absorb the variation as seen in Figure 5.35 and Figure 5.36 when comparing drops for 1 second with 2 second intervals at 10 seconds of playback with no prefetching and 2-ahead prefetching, respectively.

iPad

When considering the short-term drops for the iPad player experiments were run for 3 minutes of playback with consecutive drops starting at either 10 seconds or 60 seconds of playback. Only drops of either 10 seconds or 30 seconds in duration are presented as longer drops are needed to challenge the prefetching proxy. The time intervals between drops are 2 seconds or 10 seconds to provide different burst intervals. These drops occur between high levels of available bandwidth at 4 Mbps or 2 Mbps and drop to a low level of available bandwidth at one of 0.5 Mbps or 0.25 Mbps.

With a large magnitude of the change such as having drops from 4 Mbps to 0.25 Mbps the player was able to handle all drop durations up to 10 seconds, whether the drops occurred early or late, with optimal playback from the viewer's perspective. When drop duration increased to 30 seconds the player struggled more in the absence of prefetching, with 10 second intervals producing an APV of 4.21 and a PS of 0.15 and

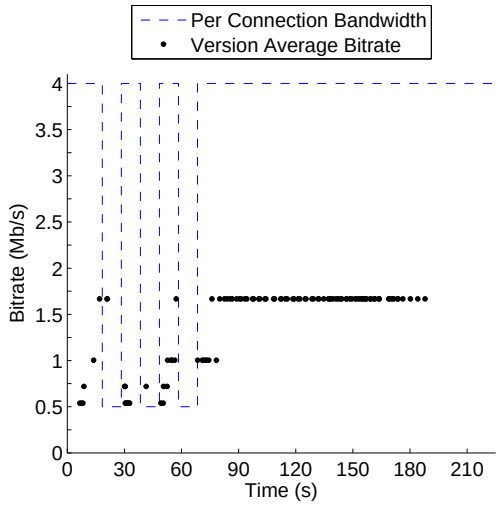

Figure 5.32: PC Browser: Drops from 4 Mbps to 1 Mbps at 10 s of Playback for 10 s Durations and 10 s Intervals with 1-Ahead Prefetching

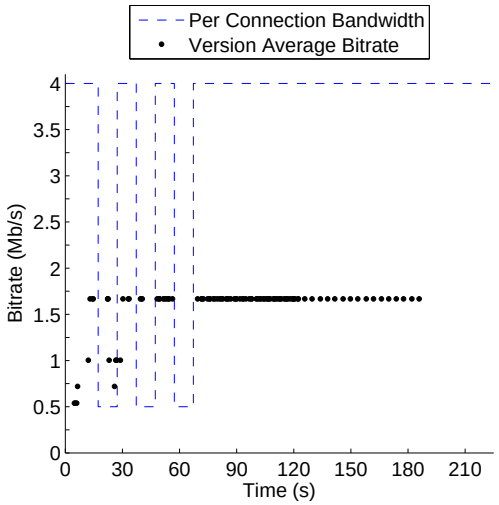

Figure 5.33: PC Browser: Drops from 4 Mbps to 1 Mbps at 10 s of Playback for 10 s Durations and 10 s Intervals with 2-Ahead Prefetching

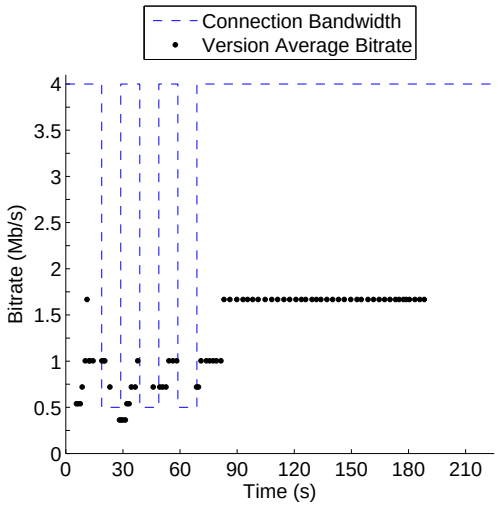

Figure 5.34: PC Browser: Drops from 4 Mbps to 1 Mbps at 10 s of Playback for 10 s Durations and 10 s Intervals with No Prefetching

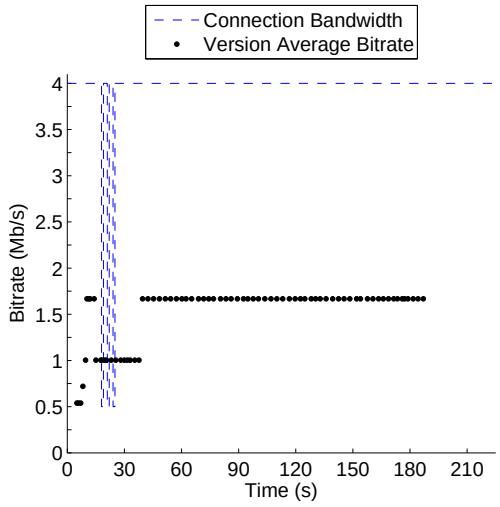

Figure 5.35: PC Browser: Drops from 4 Mbps to 1 Mbps at 10 s of Playback for 1 s Durations and 2 s Intervals with No Prefetching

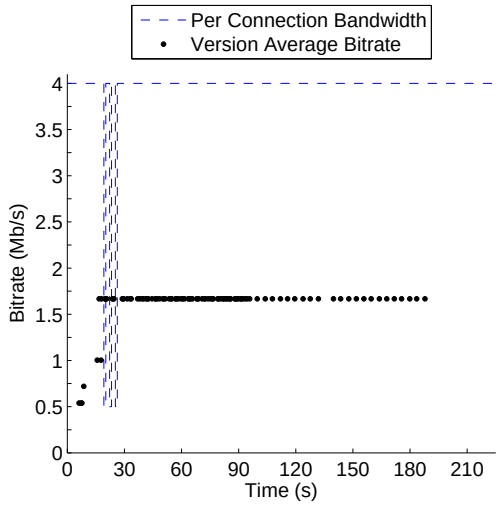

Figure 5.36: PC Browser: Drops from 4 Mbps to 1 Mbps at 10 s of Playback for 1 s Durations and 2 s Intervals with 2-Ahead Prefetching

2 second intervals producing an APV of 4.36 and a PS of 0.28. The player is capable of keeping the APV high due to backfilling; however, backfilling introduces another form of overhead as chunks are requested and never played. This overhead is referred to as player overhead. Both the previously presented experiments incurred a high player overhead with overheads of 0.58 and 0.55, respectively. Prefetching 1-ahead resulted in optimal playback in both cases and a reduction in player overheads to 0.22 and 0.15, but at a cost of PO of 0.03 and 0.07, respectively. By increasing to 2-ahead prefetching the player overheads can be further reduced to approximately 0.03 in both cases with a PO of 0.12 and 0.16, respectively. Figure 5.38 and Figure 5.39 show the difference between 1-ahead prefetching and 2-ahead prefetching with 30 second drops and 10 second intervals at 60 seconds of playback. These can be compared to the corresponding results with no prefetching shown in Figure 5.37. Even in cases where no prefetching was necessary to obtain optimal viewer playback such as 10 second drops with 2 second intervals at 10 seconds of playback, the player overhead is still reduced from 0.23 to 0.02 when 1-ahead prefetching is used, displaying benefits to the system beyond those of the viewer experience.

5.6 Shared Bandwidth N-Ahead Prefetching

This research has focused on scenarios with a bottleneck that is shared by many clients or traffic flows so the prefetching proxy can open N connections to achieve N shares of the available bandwidth. However, it is possible that the bottleneck could have only the prefetching proxy on it or a very limited number of other flows, such as a scenario in which the bottleneck is a home access link with the proxy in the home. In this situation the available bandwidth is being divided up among the prefetching agents rather than the proxy

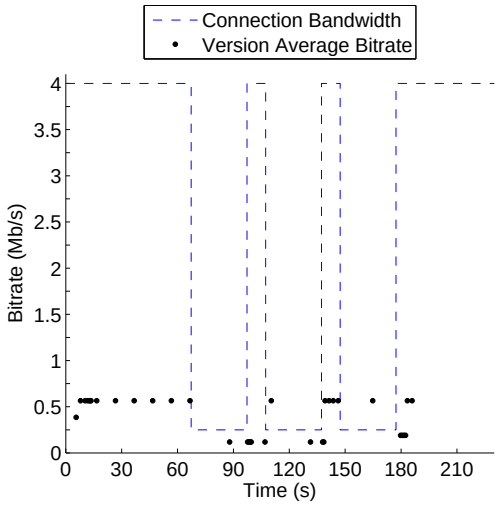

Figure 5.37: iPad: Drops from 4 Mbps to 0.25 Mbps at 60 s of Playback for 30 s durations and 10 s intervals with No Prefetching

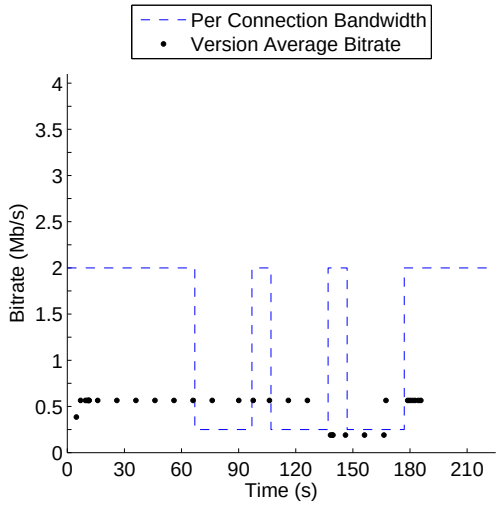

Figure 5.38: iPad: Drops from 4 Mbps to 0.25 Mbps at 60 s of Playback for 30 s durations and 10 s intervals with 1-Ahead Prefetching

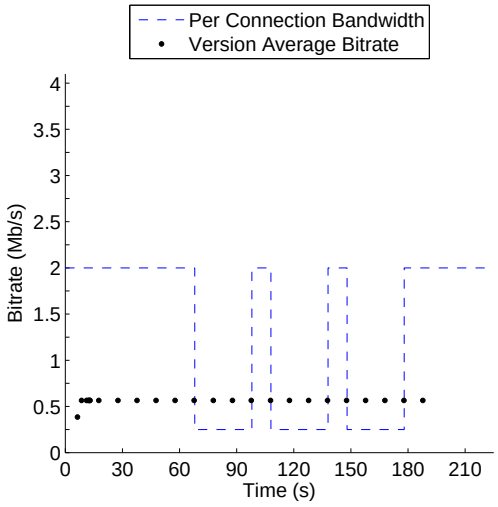

Figure 5.39: iPad:Drops from 4 Mbps to 0.25 Mbps at 60 s of Playback for 30 s durations and 10 s intervals with 2-Ahead Prefetching

obtaining extra shares by taking bandwidth from other competing flows. In this case prefetching can still be beneficial when there is initially an excess of available bandwidth, which allows the prefetching proxy to cache chunks in advance of their requests and stay far enough ahead so that the player does not reduce the requested quality level. With these cached chunks the available bandwidth can drop and the client's requests can be served by the proxy as the prefetching agents continue to download chunks. When the bandwidth recovers the proxy has allowed the player to maintain its quality level despite the reduction in available bandwidth and it can begin refilling the cache.

Drawbacks to prefetching when the bandwidth is shared among the prefetching agents occur in cases when the available bandwidth does not recover after a decrease, or is initially very low and the prefetching agents battle for bandwidth with each other. In these cases the proxy should stop prefetching, or reduce the number of prefetching agents and realize that prefetching is not as beneficial because the currently needed chunks are being downloaded more slowly than they would be without prefetching.

An example of where prefetching in a single pipe scenario is often advantageous would be the case of consecutive drops. With drops from 2 Mbps to 0.5 Mbps, for 30 second drops with 10 second intervals at 60 seconds of playback, 2-ahead prefetching is capable of achieving optimal playback of APV 5 and a PS of 1, with PO of 0.05 and player overhead of 0.05. The same experiment with no prefetching has an APV of 3.9 with PS of 0.19 and a player overhead of 0.39. This advantage can be seen by comparing the player behaviours in Figure 5.40 and Figure 5.41.

An example of the negative impact of prefetching occurs when the bandwidth does not recover in time after the drop and the prefetching agents try to stay too far ahead. The player struggles when there are no chunks buffered at the proxy as the chunk latency becomes large due to the bandwidth being shared among multiple connections. Figure 5.42 and Figure 5.43 show the difference in behaviour between 1-ahead and 3-ahead shared bandwidth prefetching for a bandwidth decrease from 4 Mbps to 1 Mbps. In this case the 1-ahead prefetching performed reasonably well, allowing for an APV of 4.63, a PS of 0.27, a PO of 0.13 and a player overhead of 0.20. When prefetching 3-ahead the APV drops to 3.6, the PS drops to 0.07, and the PO also drops to 0.066, but the player overhead increases to 1.12. Also, if the available bandwidth drops to a very low level such as to 0.25 Mbps the video is essentially unplayable due to significant freezing. However, the video was playable at this level with both no prefetching and 3-ahead prefetching when the available bandwidth was set per connection. In the case of the shared available bandwidth the freezing occurs because multiple prefetching agents are attempting to download chunks in parallel resulting in insufficient bandwidth to download each chunk without affecting the currently needed chunk's latency.

5.7 Periodic Optimistic Prefetching

All the results that have been presented thus far involve the proxy only prefetching the current quality level. This has some drawbacks. When the player observers prefetched chunks coming fast it may increase

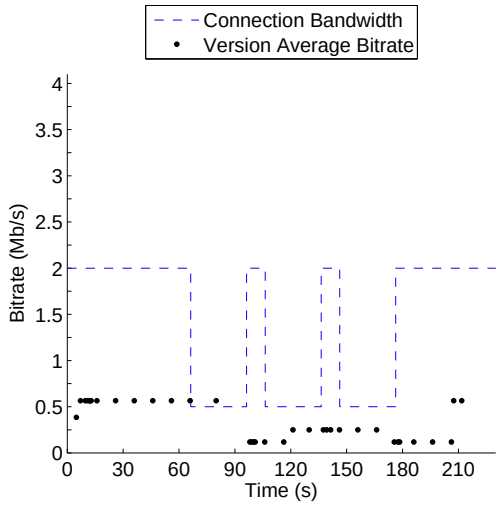

Figure 5.40: iPad: Drops from 2 Mbps to 0.5 Mbps at 60 s of Playback for 30 s Durations and 10 s Intervals with No Prefetching

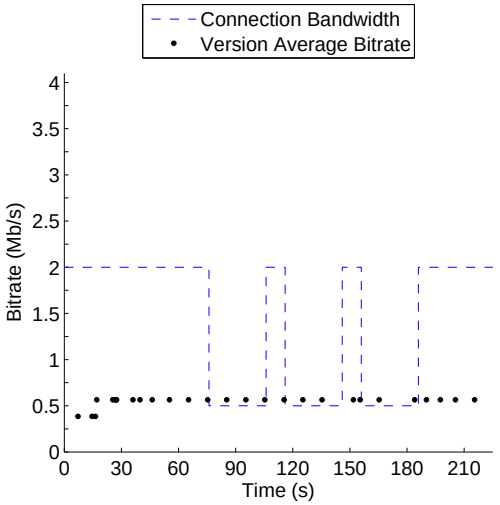

Figure 5.41: iPad: Drops from 2 Mbps to 0.5 Mbps at 60 s of Playback for 30 s Durations and 10 s Intervals with Shared Bandwidth 2-Ahead Prefetching

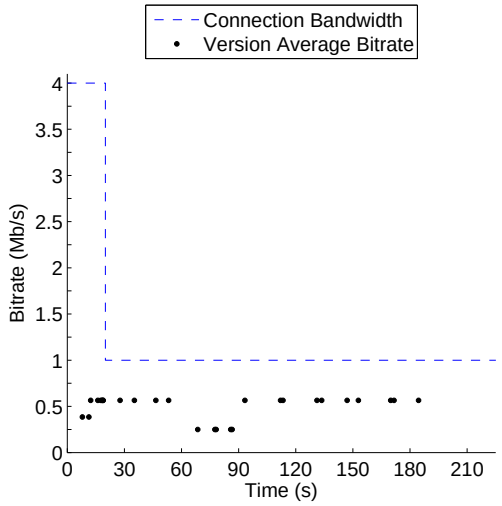

Figure 5.42: iPad: Long-Term Decrease from 4 Mbps to 1 Mbps at 10 s of Playback with Shared Bandwidth 1-Ahead Prefetching

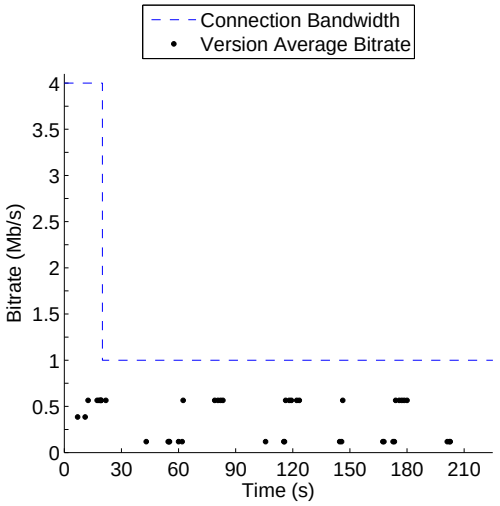

Figure 5.43: iPad: Long-Term Decrease from 4 Mbps to 1 Mbps at 10 s of Playback with Shared Bandwidth 3-Ahead Prefetching

the requested quality level, but if there are no prefetched chunks for that quality level the request must go to the server. In this case the player only observes one share of the available bandwidth on the first chunk at the higher quality level. As a result the player may determine that it cannot sustain that quality level and the quality level of the requested version drops. This results in quality level instability which is undesirable [22, 36, 46, 59, 80, 81].

One way to address this is to prefetch further ahead. That way when the player requests a high quality level it will begin prefetching far ahead. After the player determines it cannot sustain the new quality level, the player will reduce the requested quality level and on the player's next attempt at the high quality level there may be chunks cached from the previous attempt that will allow the high quality version to be served quickly while future chunks are being prefetched. The downside of this approach is that the proxy must prefetch further in advance than otherwise necessary, which results in higher prefetching overhead. Also the player must request the high quality level again soon enough, before the cached chunks are no longer useful.

Another option is to allow the prefetching proxy to guess when it is appropriate to prefetch higher quality levels so there may be chunks cached prior to the player requesting those chunks. This method results in more overhead in the sense that chunks are getting requested without the player actively requesting that version in hopes that the player will make another attempt at playing the higher quality. However, if the proxy algorithm has an accurate understanding of the player's rate-adaptation algorithm this overhead could be minimal. A small number of experiments were conducted to determine if optimistically prefetching a higher quality may be a feasible option.

The way the two players that are used in this research adapt their requested quality level differs. The PC Browser always increases its requested version one quality level at a time as opposed to the iPad which may change from the lowest to the highest quality version in one change. Further, the iPad may also backfill its buffer. The optimistic prefetching algorithm was designed to accommodate the PC Browser player as it is more predictable. Several considerations should be made when designing such an algorithm. First, the higher version must have been previously requested. If the player is streaming at version three and no version higher than three has ever been requested then the player simply prefetches N-ahead at version three as prefetching version four when there is no evidence it will be requested is wasted bandwidth. If a higher quality level had previously been requested then the version above the currently requested version is a candidate for optimistic prefetching. One final concern is that a higher version must have been requested recently. If for example version five was last requested at chunk number ten, and version four has streamed a hundred chunks smoothly since, the proxy does not want to be prefetching at version five as it is unlikely to be requested again and those bytes would be wasted prefetching overhead.

Experiments are conducted with 2-ahead prefetching using 2-ahead optimistic prefetching, and 3-ahead prefetching using 3-ahead optimistic prefetching. In these experiments less emphasis was made on prefetching overhead as the optimal value for how long the player should prefetch the higher quality was not known, so the proxy always prefetched the next higher quality version if it had previously been requested. The

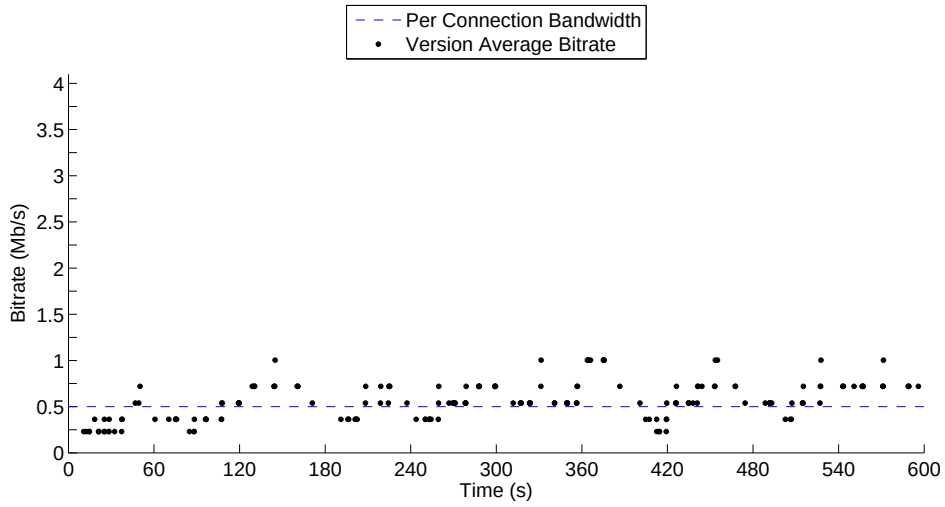

Figure 5.44: PC Browser: 0.5 Mbps Stable Available Bandwidth with 3-Ahead Prefetching and 3-Ahead Optimistic Prefetching

experiments used 0.5 Mbps and 1 Mbps stable available bandwidth. The results were not favourable for 0.5 Mbps as the APV was 3.1 and 3.0, and the PS was 0.03 and 0.02, for 2-ahead and 3-ahead respectively. It is believed that this method is ineffective because as the player changes versions it can do so very quickly, too quickly for the optimistic prefetching to get ahead when the player is in the buffering state.

If the optimistic prefetching does manage to get ahead for the player at a higher quality level, but only long enough to request a few chunks at the new quality level then when the player requests down a quality level there is nothing prefetched at that level and the player decreases its requested quality level further. Figure 5.44 displays this behaviour clearly. It is reasonable to believe that prefetching further ahead would aid in alleviating these issues. Setting the per connection available bandwidth to 1 Mbps showed more promise with APV of 5.52 and 5.61 and PS of 0.13 and 0.19 for 2-ahead and 3-ahead, respectively.

5.8 Chapter Summary

This chapter presented the performance results for the prefetching proxy. Its utility and impact was evaluated and shown for various networking conditions including under stable available bandwidth, during available bandwidth oscillations, and during long-term and short-term increases and decreases to the available bandwidth. The proxy has shown to provide a substantial performance benefit in many cases by improving the requested version quality and the playback smoothness, with only limited prefetching overhead.

When prefetching is used at stable levels of available bandwidth it can introduce instability in the playback but by prefetching further ahead it is possible for the player to eventually stabilize and achieve a high APV. Higher levels of available bandwidth result in less initial instability and the need to prefetch fewer ahead. The impact of oscillations or short-term drops to the available bandwidth proved to be greatly reduced by

prefetching. Spikes had little impact on the player, although sometimes causing the player to request even higher quality levels when prefetching was conducted. During long-term decreases to the available bandwidth the player could often maintain the highest video quality for the entire duration of the video provided. The main focus of the research has concerned scenarios with a bottleneck shared by many connections, however Section 5.6 has shown that the proxy has potential to work in situations where the bottleneck may only have a limited number of contending flows. Further, this chapter has shown some potential for other prefetching algorithms, that are more fine tuned to the player's behaviour as discussed in Section 5.7.

Chapter 6

Conclusions

To conclude, this chapter summarizes the thesis, lists the main contributions, and briefly outlines future work.

6.1 Thesis Summary

This thesis is evidence of the improvement potential to Internet video streaming performance by using a prefetching proxy with an HTTP adaptive streaming system. There have been some recent works which investigate caching or prefetching in conjunction with caching for adaptive streaming video players and others that have looked at buffering of video at the player. However, this research shows that prefetching alone, with a proxy server, can provide substantial performance benefits. It can offload the stress of buffering data from player devices that have limited resources, such as smartphones and tablets, onto the proxy hardware. It also allows the player to be isolated from short-term bandwidth reduction, preventing an unnecessary overreaction to the observed fluctuations in the available bandwidth. Further, by utilizing parallel connections it is possible for the proxy server to obtain more shares of the available bandwidth at the bottleneck. This increases throughput from the server to the proxy and allows for chunks to be delivered to the player with reduced latency, effectively improving the video quality when the available bandwidth is low.

Chapters 1 and 2 introduced the topic, described the goals of the thesis and presented the relevant background. Chapter 3 laid out the methodology for this research. In this chapter the hardware, the software and the communication protocols were discussed as well as the experiments, including the factors and metrics used for evaluation. Chapter 4 characterized both the PC Browser and the iPad Netflix player's rate-adaptation logic under varying network conditions. These conditions included behaviour in several bandwidth regions, and at various times during playback while under stable available bandwidth, short-term sequential spikes and drops to the available bandwidth, long-term increases and decreases to the available bandwidth as well as oscillations to the available bandwidth. Chapter 5 discussed prefetching in detail and presented the experimental results. This research shows strong potential for a prefetching proxy to improve the quality of video being delivered over the Internet.

6.2 Thesis Contributions

The major contributions of this thesis are as follows:

- An experimental platform in which traffic is routed between the client devices and the server through the proxy software. By manipulating the traffic and using a VPN it is possible to insert proxy hardware between the client devices and the server. This allows for evaluation of the clients and the prefetching proxy in a semi-controlled environment.
- A prefetching proxy application designed to be used with Netflix, an enormously popular closed source system.
- A characterization of the Netflix rate-adaptation algorithm for both the PC Browser and iPad Netflix players. The players were found to both quickly react to bandwidth variation, indicating a potential for improving the performance using prefetching.
- Analysis of the performance of the prefetching proxy under various network conditions for the PC Browser and iPad Netflix players. Even though prefetching far ahead will not reduce performance, prefetching only 1 or 2 ahead was adequate in greatly improving video quality in many scenarios.

6.3 Future Work

As outlined in Chapter 5 the prefetching proxy increases the buffer capacity of the player by allowing chunks to be quickly served from the proxy's cache. Using this strategy could offload some of the buffering from the player to the proxy, reducing the amount of wasted video data when the user terminates the video early. The research in this thesis does effectively increase the potential buffer capacity by caching chunks at the proxy, however it does not reduce the number of chunks the player downloads. Future work could involve reducing the size of the player buffer and measuring the impact of buffering more at the proxy rather than the client.

This research shows the potential improvements to video playback performance that are made possible by prefetching with a proxy. However, due to the Netflix rate-adaptation algorithm being closed source the optimal optimistic prefetching algorithms for the two players are unknown. If the proxy could determine with more certainty which chunk would be requested next from any particular quality level it could start prefetching that chunk immediately and provide a smoother ramp up period. The further in advance the proxy can predict the more useful it could become. This thesis presented a small set of experiments involving optimistic prefetching. Future work may involve a more precise characterization of the players or using open source systems for tuning more advanced prefetching algorithms.

References

[1] 3GP-DASH. 3gp-dash 26.247, March 2013. URL `http://www.3gpp.org/ftp/Specs/html-info/26247.htm`.

[2] S. Akhshabi, A. C. Begen, and C. Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *Proceedings of the 2nd Conference on Multimedia systems*, MMSys '11, pages 157–168, San Jose, California, USA, February 2011. ACM.

[3] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen. What happens when http adaptive streaming players compete for bandwidth? In *Proceedings of the 22nd International Workshop on Network and Operating System Support for Digital Audio and Video*, NOSSDAV '12, pages 9–14, Toronto, Ontario, Canada, June 2012. ACM.

[4] S. Akhshabi, L. Anantakrishnan, C. Dovrolis, and A. C. Begen. Server-based traffic shaping for stabilizing oscillating adaptive streaming players. In *Proceeding of the 23rd Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '13, pages 19–24, Oslo, Norway, February 2013. ACM.

[5] T. Andelin, V. Chetty, D. Harbaugh, S. Warnick, and D. Zappala. Quality selection for dynamic adaptive streaming over http with scalable video coding. In *Proceedings of the 3rd Conference on Multimedia Systems*, MMSys '12, pages 149–154, Chapel Hill, North Carolina, USA, February 2012. ACM.

[6] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang. A quest for an internet video quality-of-experience metric. In *Proceedings of the 11th Workshop on Hot Topics in Networks*, HotNets-XI, pages 97–102, Redmond, Washington, USA, October 2012. ACM.

[7] A.C. Begen, T. Akgul, and M. Baugher. Watching video over the web: Part 1: Streaming protocols. *IEEE Internet Computing*, 15(2):54–63, March-April 2011.

[8] S. Benno, J. O. Esteban, and I. Rimac. Adaptive streaming: The network has to help. *Bell Labs Technical Journal*, 16(2):101–114, September 2011.

[9] S. Bhandarkar, L. Ramaswamy, and H. K. Devulapally. Collaborative caching for efficient dissemination of personalized video streams in resource constrained environments. In *Proceedings of the 3rd Conference on Multimedia Systems*, MMSys '12, pages 185–190, Chapel Hill, North Carolina, USA, February 2012. ACM.

[10] ITU-R Rec. BT.500. Methodology for the subjective assessment of the quality of television pictures. *ITU Radiocommunication Sector*, 2012.

[11] M. Carbone and L. Rizzo. Dummynet revisited. *ACM SIGCOMM Computer Communications Review*, 40(2):12–20, April 2010.

[12] C. Chen, S. Lee, and H. W. Stevenson. Response style and cross-cultural comparisons of rating scales among east asian and north american students. *Psychological Science*, 6(3):170–175, May 1995.

[13] Cisco. Cisco visual networking index: Global mobile data traffic forecast update 2012-2017. Technical report, Cisco, 2013.

[14] Cisco. Cisco visual networking index: Forecast and methodology, 2012-2017. Technical report, Cisco, 2013.

[15] N. Cranley, P. Perry, and L. Murphy. User perception of adapting video quality. *International Journal of Human-Computer Studies*, 64(8):637–647, August 2006.

[16] A. Dastpak, J. Liu, and M. Hefeeda. Video streaming over cognitive radio networks. In *Proceedings of the 4th Workshop on Mobile Video*, MoVid '12, pages 31–36, Chapel Hill, North Carolina, USA, February 2012. ACM.

[17] L. De Cicco, S. Mascolo, and V. Palmisano. Feedback control for adaptive live video streaming. In *Proceedings of the 2nd Conference on Multimedia Systems*, MMSys '11, pages 145–156, San Jose, California, USA, February 2011. ACM.

[18] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang. Understanding the impact of video quality on user engagement. In *Proceedings of the 2011 Conference on Special Interest Group on Data Communications*, SIGCOMM '11, pages 362–373, Toronto, Ontario, Canada, August 2011. ACM.

[19] J. Erman, A. Gerber, K. K. Ramadrishnan, S. Sen, and O. Spatscheck. Over the top video: The gorilla in cellular networks. In *Proceedings of the 2011 Conference on Internet Measurement*, IMC '11, pages 127–136, Berlin, Germany, November 2011. ACM.

[20] K. Evensen, D. Kaspar, C. Griwodz, P. Halvorsen, A. Hansen, and P. Engelstad. Improving the performance of quality-adaptive video streaming over multiple heterogeneous access networks. In *Proceedings of the 2nd Conference on Multimedia Systems*, MMSys '11, pages 57–68, San Jose, California, USA, February 2011. ACM.

[21] K. Evensen, A. Petlund, H. Riiser, P. Vigmostad, D. Kaspar, C. Griwodz, and P. Halvorsen. Demo: Quality-adaptive video streaming with dynamic bandwidth aggregation on roaming, multi-homed clients. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 355–356, Bethesda, Maryland, USA, June 2011. ACM.

[22] K. Evensen, A. Petlund, H. Riiser, P. Vigmostad, D. Kaspar, C. Griwodz, and P. Halvorsen. Mobile video streaming using address-based network prediction and transparent handover. In *Proceedings of the 21st Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '11, pages 21–26, Vancouver, British Columbia, Canada, June 2011. ACM.

[23] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao. Youtube everywhere: impact of device and infrastructure synergies on user experience. In *Proceedings of the 2011 Conference on Internet Measurement*, IMC '11, pages 345–360, Berlin, Germany, November 2011. ACM.

[24] N. Gautam, H. Petander, and J. Noel. A comparison of the cost and energy efficiency of prefetching and streaming of mobile video. In *Proceedings of the 5th Workshop on Mobile Video*, MoVid '13, pages 7–12, Oslo, Norway, February 2013. ACM.

[25] G. Ghinea, J. P. Thomas, and R. S. Fish. Multimedia, network protocols and users - bridging the gap. In *Proceedings of the 7th ACM conference on Multimedia (Part 1)*, MULTIMEDIA '99, pages 473–476, Orlando, Florida, USA, October 1999. ACM.

[26] M. Ghobadi, Y. Cheng, A. Jain, and M. Mathis. Trickle: Rate limiting youtube video streaming. In *Proceedings of the 2012 USENIX Annual Technical Conference*, USENIX ATC '12, pages 17–17, Boston, Massachusetts, USA, June 2012. USENIX Association.

[27] A. Goel, C. Krasic, and J. Walpole. Low-latency adaptive streaming over tcp. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 4(3):20:1–20:20, September 2008.

[28] S. Gouache, G. Bichot, A. Bsila, and C. Howson. Distributed amp; adaptive http streaming. In *Proceedings of the 2011 International Conference on Multimedia and Expo*, ICME '11, pages 1–6, Barcelona, Spain, July 2011. IEEE.

[29] D. Havey, R. Chertov, and K. Almeroth. Receiver driven rate adaptation for wireless multimedia applications. In *Proceedings of the 3rd Conference on Multimedia Systems*, MMSys '12, pages 155–166, Chapel Hill, North Carolina, USA, February 2012. ACM.

[30] R. Houdaille and S. Gouache. Shaping http adaptive streams for a better user experience. In *Proceedings of the 3rd Conference on Multimedia Systems*, MMSys '12, pages 1–9, Chapel Hill, North Carolina, USA, February 2012. ACM.

[31] T. Huang, N. Handigol, B. Heller, N. McKeown, and R. Johari. Confused, timid, and unstable: Picking a video streaming rate is hard. In *Proceedings of the 2012 Conference on Internet Measurement*, IMC '12, pages 225–238, Boston, Massachusetts, USA, November 2012. ACM.

[32] incoming. Incoming android mobile video application, May 2013. URL `http://watch.incoming.tv`.

[33] ITU-T Rec. J.246. Perceptual visual quality measurement techniques for multimedia services over digital cable television networks in the presence of a reduced bandwidth reference. *ITU Telecommunication Standardization Sector*, 2008.

[34] ITU-T Rec. J.247. Objective perceptual multimedia video quality measurement in the presence of a full reference. *ITU Telecommunication Standardization Sector*, 2008.

[35] ITU-T Rec. J.341. Objective perceptual multimedia video quality measurement of hdtv for digital cable television in the presence of a full reference. *ITU Telecommunication Standardization Sector*, 2011.

[36] J. Jiang, V. Sekar, and H. Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 97–108, Nice, France, December 2012. ACM.

[37] S. Khemmarat, R. Zhou, L. Gao, and M. Zink. Watching user generated videos with prefetching. In *Proceedings of the 2nd Conference on Multimedia Systems*, MMSys '11, pages 187–198, San Jose, California, USA, February 2011. ACM.

[38] J. Klaue, B. Rathke, and A. Wolisz. Evalvid - a framework for video transmission and quality evaluation. In *Proceedings of the 13th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, TOOLS '03, pages 255–272, Urbana, Illinois, USA, September 2003. Springer-Verlag.

[39] V. Krishnamoorthi, N. Carlsson, D. L. Eager, A. Mahanti, and N. Shahmehri. Helping hand or hidden hurdle: Proxy-assisted http-based adaptive streaming performance. In *Proceedings of the 21st Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, MASCOTS '13, page to appear., San Francisco, California, USA, August 2013. IEEE.

[40] R. Kuschnig, I. Kofler, and H. Hellwagner. An evaluation of tcp-based rate-control algorithms for adaptive internet streaming of h.264/svc. In *Proceedings of the 1st Conference on Multimedia systems*, MMSys '10, pages 157–168, Phoenix, Arizona, USA, February 2010. ACM.

[41] R. Kuschnig, I. Kofler, and H. Hellwagner. Improving internet video streaming performance by parallel tcp-based request-response streams. In *Proceedings of the 7th International Conference on Consumer Communications and Networking*, CCNC '10, pages 1–5, Las Vegas, Nevada, USA, January 2010. IEEE.

[42] R. Kuschnig, I. Kofler, and H. Hellwagner. Evaluation of http-based request-response streams for internet video streaming. In *Proceedings of the 2nd Conference on Multimedia Systems*, MMSys '11, pages 245–256, San Jose, California, USA, February 2011. ACM.

[43] C. Liu, I. Bouazizi, and M. Gabbouj. Parallel adaptive http media streaming. In *Proceedings of 20th Conference on Computer Communications and Networks*, ICCCN '11, pages 1–6, Maui, Hawaii, USA, July 2011. IEEE.

[44] C. Liu, I. Bouazizi, and M. Gabbouj. Rate adaptation for adaptive http streaming. In *Proceedings of the 2nd Conference on Multimedia Systems*, MMSys '11, pages 169–174, San Jose, California, USA, February 2011. ACM.

[45] K. Miller, E. Quacchio, G. Gennari, and A. Wolisz. Adaptation algorithm for adaptive streaming over http. In *Proceedings of the 19th International Workshop on Packet Video*, PV '12, pages 173–178, Munich, Germany, May 2012. IEEE.

[46] R. K. P. Mok, X. Luo, E. W. W. Chan, and R. K. C. Chang. Qdash: a qoe-aware dash system. In *Proceedings of the 3rd Conference on Multimedia Systems*, MMSys '12, pages 11–22, Chapel Hill, North Carolina, USA, February 2012. ACM.

[47] R.K.P. Mok, E.W.W. Chan, and R.K.C. Chang. Measuring the quality of experience of http video streaming. In *Proceedings of the 2011 Symposium on Integrated Network Management*, IM '11, pages 485–492, Dublin, Ireland, May 2011. IEEE.

[48] R.K.P. Mok, E.W.W. Chan, X. Luo, and R.K.C. Chang. Inferring the qoe of http video streaming from user-viewing activities. In *Proceedings of the 1st Workshop on Measurements Up the Stack*, W-MUST '11, pages 31–36, Toronto, Ontario, Canada, August 2011. ACM.

[49] ITU-T Rec. P.800. Methods for subjective determination of transmission quality. *ITU Telecommunication Standardization Sector*, 1996.

[50] ITU-T Rec. P.911. Subjective audiovisual quality assessment methods for multimedia applications. *ITU Telecommunication Standardization Sector*, 1998.

[51] A. Perkis, S. Munkeby, and Odd Inge Hillestad. A model for measuring quality of experience. In *Proceedings of the 7th Symposium on Nordic Signal Processing*, NORSIG '06, pages 198–201, Reykjavik, Iceland, June 2006. IEEE.

[52] M. Pinson and S. Wolf. Comparing subjective video quality testing methodologies. In *Proceedings of the SPIE: Video Communications and Image Processing*, volume 5150 of *VCIP '03*, pages 573–582, Lugano, Switzerland, 2003.

[53] L. Plissonneau and E. Biersack. A longitudinal view of http video streaming performance. In *Proceedings of the 3rd Conference on Multimedia Systems*, MMSys '12, pages 203–214, Chapel Hill, North Carolina, USA, February 2012. ACM.

[54] L. Popa, A. Ghodsi, and I. Stoica. Http as the narrow waist of the future internet. In *Proceedings of the 9th Workshop on Hot Topics in Networks*, Hotnets-IX, pages 1–6, Monterey, California, USA, October 2010. ACM.

[55] X. Qiu, H. Liu, D. Li, S. Zhang, D. Ghosal, and B. Mukherjee. Optimizing http-based adaptive video streaming for wireless access networks. In *Proceedings of the 3rd Conference on Broadband Network and Multimedia Technology*, IC-BNMT '10, pages 838–845, Beijing, China, October 2010. IEEE.

[56] A. Rao, A. Legout, Y. Lim, D. Towsley, C. Barakat, and W. Dabbous. Network characteristics of video streaming traffic. In *Proceedings of the 7th Conference on Emerging Networking Experiments and Technologies*, CoNEXT '11, pages 1–12, Tokyo, Japan, December 2011. ACM.

[57] A. Rath, S. Goyal, and S. Panwar. Streamloading: Low cost high quality video streaming for mobile users. In *Proceedings of the 5th Workshop on Mobile Video*, MoVid '13, pages 1–6, Oslo, Norway, February 2013. ACM.

[58] R. Rejaie and J. Kangasharju. Mocha: A quality adaptive multimedia proxy cache for internet streaming. In *Proceedings of the 11th Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '01, pages 3–10, Port Jefferson, New York, USA, June 2001. ACM.

[59] R. Rejaie, M. Handley, and D. Estrin. Layered quality adaptation for internet video streaming. *Journal on Selected Areas in Communications*, 18(12):2530–2543, December 2000.

[60] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen. Bitrate and video quality planning for mobile streaming scenarios using a gps-based bandwidth lookup service. In *Proceedings of the 2011 International Conference on Multimedia and Expo*, ICME '11, pages 1–6, Barcelona, Spain, July 2011. IEEE.

[61] H. Riiser, H. S. Bergsaker, P. Vigmostad, P. Halvorsen, and C. Griwodz. A comparison of quality scheduling in commercial adaptive http streaming solutions on a 3G network. In *Proceedings of the 4th Workshop on Mobile Video*, MoVid '12, pages 25–30, Chapel Hill, North Carolina, USA, February 2012. ACM.

[62] L. Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *ACM SIGCOMM Computer Communications Review*, 27(1):3–41, January 1997.

[63] Y. Sánchez de la Fuente, T. Schierl, C. Hellge, T. Wiegand, D. Hong, D. De Vleeschauwer, W. Van Leekwijck, and Y. Le Louédec. idash: Improved dynamic adaptive streaming over http using scalable video coding. In *Proceedings of the 2nd Conference on Multimedia Systems*, MMSys '11, pages 257–264, San Jose, California, USA, February 2011. ACM.

[64] Sandvine. Global internet phenomena report: Fall 2010. Technical report, Sandvine, 2010.

[65] Sandvine. Global internet phenomena report: Spring 2011. Technical report, Sandvine, 2011.

[66] Sandvine. Global internet phenomena report: 1h 2012. Technical report, Sandvine, 2012.

[67] Sandvine. Global internet phenomena report: 2h 2012. Technical report, Sandvine, 2012.

[68] Sandvine. Global internet phenomena report: 1h 2013. Technical report, Sandvine, 2013.

[69] N. Sharma, D. Krishnappa, D. Irwin, M. Zink, and P. Shenoy. Greencache: Augmenting off-the-grid cellular towers with multimedia caches. In *Proceedings of the 4th Conference on Multimedia Systems*, MMSys '13, pages 271–280, Oslo, Norway, February 2013. ACM.

[70] M. Siekkinen, M. A. Hoque, J. K. Nurminen, and M. Aalto. Streaming over 3G and lte: How to save smartphone energy in radio access network-friendly way. In *Proceedings of the 5th Workshop on Mobile Video*, MoVid '13, pages 13–18, Oslo, Norway, February 2013. ACM.

[71] M. Siller and J. Woods. Using an agent based platform to map quality of service to experience in conventional and active networks. *IEE Proceedings Communications*, 153(6):828–840, December 2006.

[72] I. Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *Proceedings of the International Symposium on Multimedia*, 18(4):62–67, October 2011.

[73] T. Stockhammer. Dynamic adaptive streaming over http –: Standards and design principles. In *Proceedings of the 2nd Conference on Multimedia Systems*, MMSys '11, pages 133–144, San Jose, California, USA, February 2011. ACM.

[74] J. Summers, T. Brecht, D. Eager, and B. Wong. Methodologies for generating http streaming video workloads to evaluate web server performance. In *Proceedings of the 5th Conference on International Systems and Storage*, SYSTOR '12, pages 2:1–2:12, Haifa, Israel, June 2012. ACM.

[75] J. Summers, T. Brecht, D. Eager, and B. Wong. To chunk or not to chunk: Implications for http streaming video server performance. In *Proceeding of the 22rd Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '12, pages 15–20, Toronto, Ontario, Canada, June 2012. ACM.

[76] G. Tian and Y. Liu. Towards agile and smooth video adaptation in dynamic http streaming. In *Proceedings of the 8th Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 109–120, Nice, France, December 2012. ACM.

[77] C. Timmerer and C. Griwodz. Dynamic adaptive streaming over http: From content creation to consumption. In *Proceedings of the 20th International Conference on Multimedia*, MM '12, pages 1533–1534, Nara, Japan, October 2012. ACM.

[78] S. Wolf and M. Pinson. Video quality measurement techniques. Technical report, National Telecommunications and Information Administration, June 2002.

[79] S. Woo, E. Jeong, S. Park, J. Lee, S. Ihm, and K. Park. Comparison of caching strategies in modern cellular cackhaul networks. In *Proceeding of the 11th International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, pages 319–332, Taipei, Taiwan, June 2013. ACM.

[80] S. Xiang, L. Cai, and J. Pan. Adaptive scalable video streaming in wireless networks. In *Proceedings of the 3rd Conference on Multimedia Systems*, MMSys '12, pages 167–172, Chapel Hill, North Carolina, USA, February 2012. ACM.

[81] M. Zink, O. Künzel, J. Schmitt, and R. Steinmetz. Subjective impression of variations in layer encoded videos. In *Proceedings of the 11th Conference on Quality of Service*, IWQoS '03, pages 137–154, Berkeley, California, USA, June 2003. Springer-Verlag.