April 2012

Thin to Win? Network Performance Analysis of the OnLive Thin Client Game System

Alexander William Grant
Worcester Polytechnic Institute

Michael E. Solano
Worcester Polytechnic Institute

Thin to Win? Network Performance Analysis of the OnLive Thin Client Game System

A Major Qualifying Project Report:

submitted to the Faculty

of the

Worcester Polytechnic Institute

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by

Alexander Grant

Michael Solano

Date: April 26th, 2012

Approved:

Professor Mark Claypool

Professor David Finkel

Abstract

OnLive was one of the first companies to make use of cloud computing technology to allow users to stream games. The goal of our project was to analyze OnLive's network performance and compare these results to two popular video streaming services, YouTube and Skype. Through careful measurements, we found that OnLive handles variations in a network differently than the other two services. These results indicate that OnLive has tailored their service to adapt to many different network conditions.

Table of Contents

List of Figures

List of Tables

1 Introduction

As personal desktop computers become more popular and easier to obtain, the cost of owning and maintaining them can be unmanageable at times. Thin clients hope to solve this problem by depending on other computers or servers to help with computation and resource management. "The goal of the thin-client model is to centralize computing resources, with all the attendant benefits of easier maintenance and cheaper upgrades, while maintaining the same quality of service that could be provided by a dedicated workstation[1]." Although thin clients are more common in corporate and academic settings, recently there has been interest in using them for entertainment, such as for video games. There are a few different companies approaching video games with the thin client model in mind. The major developers thus far are:

- OnLive[1]
- Gaikai[2]
- GameString[3]
- StreamMyGame[4]

Each one of the aforementioned companies is relatively new and their technology is still being developed; GameString, for example, is still just a beta. There is almost no public information on these technologies or how these companies are trying to achieve their goals.

This project focused on a cloud gaming service called OnLive. OnLive provides a thin client that connects to the OnLive service. The OnLive service uses the cloud computing model by housing servers which contain the game and user data. Thin clients depend on other computers or servers, so OnLive combines the thin client model and the concepts of cloud computing to bring their service to users. The project explored the network and graphical information OnLive produces when run in various environments. It compared and contrasted OnLive to video streaming services such as YouTube and Skype.[5,6] A multitude of tests were be conducted on OnLive, YouTube, and Skype to measure their performance throughout this project.

---

[1] http://www.onlive.com/
[2] http://www.gaikai.com/
[3] http://www.gamestring.com/
[4] http://www.streammygame.com/smg/index.php
[5] http://www.youtube.com/
[6] http://www.skype.com/intl/en-us/home

OnLive allows users to have access to a multitude of games and play them. The difference between traditional gaming and what OnLive is achieving is that OnLive uses the ideas and concepts of cloud computing to make popular video games more accessible. Players use the OnLive service to play games on OnLive's computers while OnLive streams a live video feed of the game screen back to the user. This makes using OnLive's service easy because the game data is stored on OnLive's servers, and not on the user's personal machine. All that is needed is [2]:

- An OnLive account
- 2 Mbps (wired or Wi-Fi) connection
- Windows 7 or Vista (32 or 64-bit) or XP SP3 (32-bit) or Mac OS X 10.5.8 or later
- Most PCs and netbooks, all Intel-based Macs
- Screen Resolution: 1024x576
- Sound (but not necessary)
- Keyboard and Mouse OR OnLive Controller

These minimum requirements, coupled with the fact that user's saved game and profile data are stored on OnLive's server, make OnLive accessible to users not only in their home, but also wherever the user has access to a computer.

We decided to compare OnLive to YouTube and Skype to compare streaming technologies designed for games to YouTube and Skype, technologies designed for video. The results of this project provide insight about the quality of OnLive's service to potential and current customers of OnLive.

Our hypotheses for this project were:

- The downstream packet captures, in terms of packet size and the numbers of packets for all OnLive games are similar. This is because the player is viewing a video, so as long as the video quality is similar, the downstream packet captures are similar.
- The upstream packet captures, in terms of packet size and the number of packets are noticeably different for each game. The information being sent to OnLive is different for a slow paced game as opposed to an action packed, fast game. For example, there are more actions being used by the player in a First Person Shooter than there are in a Real Time Strategy game, so the upstream packet captures should vary depending on the game and game genre.

- Altering the bandwidth, packet loss, and delay of the network changes the quality of the experience of OnLive.
    - Adding delay affects OnLive the most and creates an environment where OnLive is nearly unplayable. With added delay, the user's actions are reflected on OnLive's service at a later time than the player originally anticipated. For example, the player may shoot at a target, but because of the delay, OnLive will recognize the shot as later, and thus the player will miss the target because the target has already moved.
    - Restricting the bandwidth and adding packet loss degrades the visual quality of of OnLive as a higher bandwidth and fewer packets lost should allow for OnLive to deliver more frames per second and a higher quality picture.

Initially, we did the background research needed to form our hypotheses. After our research had been completed, we proceeded to establish hypotheses and formulate an experiment plan for the information we intended to gather and to test the hypotheses. After obtaining resources and setting up the equipment needed for our experiments, we collected data about OnLive. When all the data was collected, we analyzed and organized the data and compared the results to our initial hypotheses.

The results in Chapter 5 highlight the following conclusions about the OnLive service. Downstream packet captures changed greatly depending on what game was played, while the upstream packet captures looked more similar than originally anticipated. OnLive also was affected differently than YouTube and Skype when the network was altered. OnLive was playable throughout every scenario (albeit with lower quality than an unrestricted network), while YouTube and Skype would sometimes skip or stop video playback altogether.

2 Background

This section provides the details on concepts that we built upon to help design our experiments and form hypotheses.

2.1 Cloud Computing

Since computers became a mainstream appliance, they have been the go-to tool for data storage and modification. Since the utilization of the Internet, being able to access one's data from anywhere with an Internet connection has been a desire for many. Cloud computing makes it possible to "access all of your personal data at any given moment[3]." The central idea behind cloud computing is that users can store data or programs in data centers that can then be accessed via an Internet connection[4]. This process makes it easier to synchronize and streamline information so that from any location, data can be shared, modified, deleted, or even created.

Cloud computing is only just now starting to become more commonplace, and the computer game industry is a sector that looks to use it to its full potential. Games and their associated information can be stored in the cloud, which can provide gamers with the ability to continue where they left off even if they are nowhere near their personal PC. Cloud computing can give low-end computers the ability to play games that they normally would not be able to play.

2.2 OnLive

OnLive is a company that saw the potential of cloud computing and developed a cloud-based gaming service. Using data centers that house powerful high-end computers, OnLive is able to give users across the United States (and soon to be other parts of the world) the ability to play a variety of games as long as they have an Internet connection.

OnLive is different from many cloud computing companies in today's market because of how it utilized cloud computing technologies. Unlike many companies, OnLive does not offer virtual machines to host websites or run data processing applications. Instead, OnLive uses the cloud systems to allow users to play video games.

After much anticipation, OnLive launched its video game service in the United States on June 17th, 2010[6]. According to Steve Perlman, OnLive's CEO, one of the largest problems that they faced since launch day was the unexpected number of users that signed up for the service. Within the first few weeks of service, the number of subscribers had already matched the

projected Fall numbers. This forced OnLive to ramp up server deployment and develop some regions quicker than others[6].

In an interview with CNET, Perlman was asked about the scalability of OnLive and how a user's experience would be affected by a large number of users on the service at the same time[6]. According to Perlman, one major issue with many online services is contention, where many users share the same connection. This is a key issue when dealing with service overloads and service interruptions. Perlman stated that OnLive has been designed from the start to eliminate contention routing and essentially provide each individual connection with its own private route. Another aspect of OnLive's scalability is that to increase capacity, OnLive only has to deploy more servers. The servers are configured in such a way to reduce the sharing of resources. This allows the servers to run independently of each other[6].

OnLive subscription service initially started out as a paid yearly subscription but in the fall of 2010, this subscription fee was removed. Currently, it is free for someone to join OnLive. With this free account, users can play free trials of games that they may want to buy and they can also add and chat with friends who also use OnLive. The OnLive service allows users to purchase individual games at retail prices, purchase monthly subscriptions to a game, and purchase monthly play pack bundles that contain anywhere from 10 – 50 games.

Currently OnLive offers two options that allow users to play across a variety of devices. OnLive gives users the option to use their personal computers, with either a Windows or Mac operating system. Perhaps the biggest selling point for OnLive's desktop app is its minimal hardware requirements. Many users are able to use an entry-level laptop or desktop to a netbook or another extremely portable computer like a MacBook Air. Another advantage is the desktop application's operating system independence. Because the application is available for both Windows and Mac, users are able to play many games that may not even be available to run locally on a Mac computer.

Users also have the option to purchase an OnLive Micro Console for $99. This Micro Console is about the size of a hard drive and allows users to play OnLive on a TV or anything with an HDMI or component input. This can be a substitute for expensive gaming computers or systems like Microsoft's Xbox or Sony's PlayStation. The user gets a wireless controller, the Micro Console and all of the cables required to hook the system up to the TV and Internet.

OnLive is currently developing apps for Apple's iPad and Android tablets. There are apps that allow users to watch live streams of their friend's games in progress, but these new apps will allow users to actually play the games from these tablet devices. This will allow tablet users to play games that typically require a powerful CPU and GPU.

2.3 GAME GENRES

For this project, we investigate three different genres:

- First Person Shooters
- Real Time Strategy
- Third Person

A First Person Shooter game is a video game in which the game world is viewed through the perspective of the main character, or shooter[7]. It is as if the user was actually in the game looking at the events of the game through their own eyes.

A Third Person game is a video game in which the perspective of the game world is viewed above the main character[7]. The user is able to see the whole main character and controls him/her while being able to see all round him/her as if the user was an observer to the world.

A Real Time Strategy video game is a subset of the strategy video game[8]. The strategy video game employs skillful thinking and tactics to achieve victory, as oppose to precision aiming and quick reactions. A Real Time Strategy video game incorporates both of these attributes into one genre of games where speed and intellect is required to obtain success.

2.4 Skype

Another popular internet program that makes use of powerful video streaming is Skype. Skype is a voice over IP (VoIP) system that allows users to video conference and voice call each other. Overall, Skype is a very popular application with 663 million registered users as of the end of 2010[9]. 65 million people sign into Skype daily and 700,000,000 minutes daily are spent talking for free with Skype to Skype calls[9].

Skype uses a proprietary Internet telephony network with limited public information available on the network protocol used. Skype is a Peer-to-Peer application as opposed to most VoIP applications that are client-server[10]. Skype uses an overlay peer-to-peer network with two different types of nodes in the network. Ordinary hosts, one of the types of nodes, allows for

voice calls and text messages to be made. A super node, the other type of node in the overlay peer-to-peer network, is an ordinary host's end-point on the Skype network. This means that any node that has sufficient computing power and a public IP address has the ability to become a super node. Ordinary hosts use the super nodes to connect to other super nodes which then connect to other ordinary hosts, thus allowing for the voice calls and text messaging to happen. This is very intriguing because as technology gets better and becomes more accessible, there will be more super nodes, thus increasing the quality of the voice calls made through Skype.

Skype's most updated version uses VP8 for all their video encoding[11]. "VP8 is a highly efficient video compression technology that was developed by On2 Technologies[12]." This includes the group video sessions as well as the one-on-one video chats. Skype allows users to make 720p HD quality video chats, but unfortunately a 1080p video chat is not available[11].

2.5 YouTube

Another extremely popular Web streaming service is YouTube. YouTube is known for its streaming of pre-recorded videos created by anyone from a corporation to an individual in their bedroom. YouTube is a video-sharing Website that allows users from all over the planet watch videos from the comfort of their computer or mobile devices. YouTube users can post their own movies or video clips and share them with the world or select individuals.

YouTube is easily the most popular video streaming service on the Internet with over 3 billion videos being viewed daily[13]. Aside from its popularity, YouTube has the technical capabilities to allow videos to be uploaded and viewed at an astounding 1080p resolution.

There have been many studies on the technical aspect of YouTube. By reading papers such as "Vivisecting YouTube: An Active Measurement Study" where members of the Computer Science and Engineering Department at the University of Minnesota studied YouTube in depth, it is possible to obtain information about YouTube's technological characteristics[14]. YouTube uses the Adobe Flash video player to stream all the videos. They use two different servers to deliver HTML webpages and video to users. One server is for the webpage that the video is located, while the other server is dedicated to holding the actual Flash video. YouTube uses both DNS resolution and HTTP redirection, for the delivery of the Flash video, to choose appropriate video servers that are best suited for the users. There are many factors that go into choosing the server. YouTube determines which servers are closest to the user, how busy a server is, and the availability of videos at various servers.

YouTube also allows users to upload their videos at many different resolutions by using different encoding techniques for a wide range of resolutions. YouTube uses Sorenson H.263 encoding for videos with 240p resolution. For videos with 360p, 480p, 720p, and 1080p, the MPEG-4 AVC (H.264) encoding is used. YouTube also supports the VP8 encoding for WebM videos[15]. The wide range of resolutions that users have access to has led to an enormous number of videos posted on YouTube.

3 Related Work

This section provides information on related research conducted on OnLive, and other relevant aspects of this project.

3.1 Cloud Gaming Services

In the paper "OnLive Cloud Gaming Service", the researchers focused on how cloud gaming services, particularly OnLive, have been severely limited by available Internet bandwidth and the time it takes to compress and decompress digital images[16]. They point out that the main problem seems to stem from the video streaming and compression it requires. After explaining how video compression works and the different encoding techniques that can be used, the authors proposed two possible substitutes to OnLive's current compression technique.

The authors begin by examining three compression techniques: H.264, VC-1(WMV-9), and MJPEG. Through their background research they determined that H.264 is unnecessarily CPU intensive whereas MJPEG is not. In order to test this hypothesis they encoded various 30 second video clips into each format. Once encoded, they did a visual analysis of each video. Four trials were ran, each trial had the same video encoded using the H.264, WMV 9, and MJPEG compression techniques. The four trials ran consisted if videos that were: a still image, moving object in stationary background, stationary object in moving background, and moving object and moving background. The researchers then watched each video and determined the quality of the video (Poor, Acceptable, Good)[16].

Overall the authors demonstrated both the advantages and disadvantages for each encoding format, but they reach the conclusion that OnLive should use the MJPEG encoding format. However, they offer no support or possible implementations for their conclusion.

Researchers from National Taiwan University studied the performance of OnLive by comparing it to another cloud gaming platform called StreamMyGame[17]. In their article, "Measuring The Latency of Cloud Gaming Systems", the researchers explain the motivation and design of their experiment. After doing a thorough investigation of the cloud-computing services available at the time, the researchers decided to compare the performance of OnLive and StreamMyGame. Unlike OnLive which has a service provided by OnLive Inc, StreamMyGame is a software solution that is managed and operated by the researchers themselves[17].

The researchers also posted information about this experiment in another article titled "Cloud Gaming Latency Analysis: OnLive and StreamMyGame Delay Measurement"[18]. In this article the authors go much more in-depth about the actual design of their testing network. The image in Figure 1 below shows how the researchers utilized a router running FreeBSD 7 with DummyNet in conjunction with two windows computers, one acting as the client for both OnLive and StreamMyGame and one acting as the server for StreamMyGame.

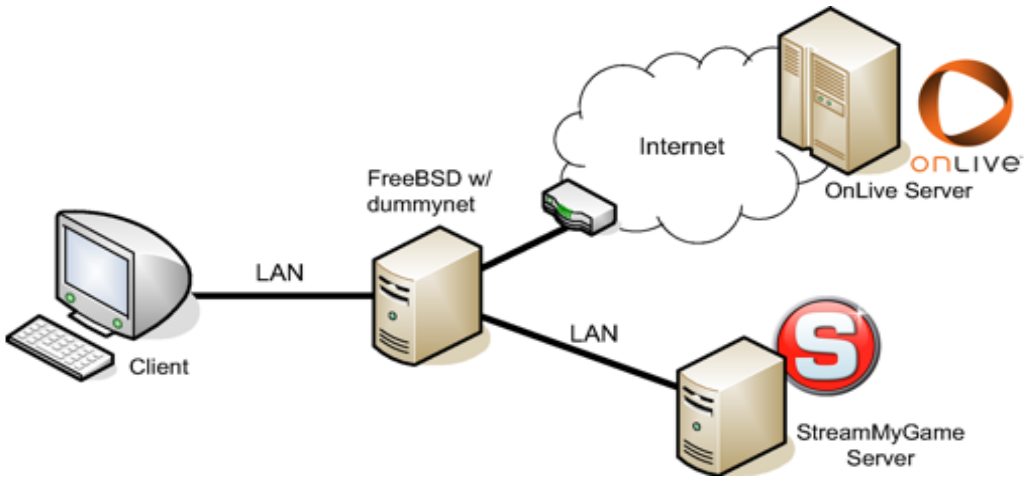


Figure 1: Network Topology of Experiment[18]

The researchers focused on the latency of commands being sent and received between the client and server. They wrote software to help measure these delays. Overall, the researchers concluded that "OnLive's overall streaming delay (i.e., the processing delay at the server plus the playout delay at the client) for the three games is between 135 and 240 ms, which is acceptable if the network delay is not significant. StreamMyGame however had streaming delays as long as 400-500 ms." The researchers pointed out that they were unsure whether or not this was a software limitation or a hardware limitation.

Overall, this research is important to us because it provides some useful techniques on how to design our own experimental network and what to expect from OnLive's gaming service in terms of latency and delay.

3.2 Latency and Gaming Research

The article, "The Effects of Loss and Latency on User Performance in Unreal Tournament 2003", is part of a larger study conducted by students at WPI for their Major Qualifying Project in May 2004[19]. The authors' main goal was to research how online multiplayer games are affected by varying network conditions, particularly network latency and

packet loss. In order to study the effects of these varying conditions, the authors set up an experiment that would measure game performance through two different layers of the game system, the application layer and the network layer.

The first thing the authors did was categorize user interactions in First Person Shooter games and design Unreal Tournament 2003 game maps for each type of interaction. Overall they determined that FPS games have two types of user interactions: movement and shooting. They further divided these categories based on complexity. From this they had the following categories: simple movement, complex movement, and precision shooting (High, Medium, and Low). Once they defined these user interactions and the sub-categories, the authors created custom game maps that focused on each particular interaction.

They then constructed a test environment to induce latency and loss while simultaneously measuring the effects of it. They did this by using a number of network tools including NIST, Ethereal, and All Seeing Eye.[7,8,9]

After conducting a few pilot studies, the authors began user testing and eventually collected data for over 200 experiments. Each user in the experiment had some previous experience with Unreal Tournament, but users were still allowed to familiarize themselves with the game before the actual experiment.

From their experiments, the authors determined that the following statements are supported by their analysis:

1. Packet loss does not have any measureable effect on user performance.
2. Latency affected precision shooting the most.
3. Latency has no measurable effect on simple or complex movements.
4. Based on user comments, packet loss was barely noticeable whereas even small amounts of latency (100ms) quickly became annoying.

Overall, the authors thoroughly tested the two major interactions of First Person Shooters and their relation to network variation. This study highlights the importance of setting up a clearly defined and concrete study. It also provides information about which software to use for network monitoring and variation.

[7] http://snad.ncsl.nist.gov/nistnet/
[8] http://www.ethereal.com/
[9] http://www.udpsoft.com/eye/

4 Methodology

This project went through four different phases: initial investigation, designing experiments, conducting the experiments, and analyzing the results. The following chapter discusses how each phase proceeded.

4.1 Initial OnLive Investigation

The initial OnLive investigation consisted of playing different demos that OnLive had available and coming up with three games that would be used to test our hypotheses.

4.1.1 Preliminary Testing and Data

After playing the demos of many different games it was apparent that OnLive's service was stable enough to handle 14 weeks of testing without giving us trouble. Using Wireshark[10], packet captures as long as 15 minutes were taken during these demos so that preliminary information on OnLive and its network behavior could be analyzed. Although OnLive is only now approaching its two year anniversary, we encountered very few bugs in the system.

OnLive's recent arrival to the gaming industry meant that it is mostly unexplored. Finding technical data about OnLive was difficult at best. Wireshark gave information regarding protocol (UDP, TCP) as well as the servers and ports used. Wireshark allowed us to look at the number of packets, packet sizes, and bytes per second among other network data. The initial graphs of the downstream and upstream data being recorded helped form our initial hypotheses and experiments about OnLive.

We purchased an OnLive micro-console to make sure that it was OnLive's system that was the limiting factor to the experiments. The console is optimized for OnLive gaming, created for the sole purpose using OnLive's system. Using the micro console eliminates any issues that may have been brought up by a computer's specifications or any background, non-OnLive-essential process.

4.1.2 Game Selection

While designing the experiments for this project, careful consideration went into which of OnLive's many games would be used. It was decided that games of different styles would best test our hypotheses and allow for unique perspectives into how OnLive delivers the gaming

[10] http://www.wireshark.org/

experience of multiple genres. Each genre chosen was picked because each selection has a distinctive graphical perspective, and therefore might have a different network footprint. The three genres chosen were First Person Shooter, Real Time Strategy, and Third Person. More information on the genres chosen can be found in Section 2.3 Game Genres.

We needed to select a game from each genre that was available on the OnLive system. It was determined that there were adequate games to choose from in the Playpack Bundle.[11] With unlimited play of over 140 games, the Playpack bundle was an excellent decision for the experiments planned. For $9.99 a month, access to every game in the Playpack bundle is given. Due to the fact that we would only need to use OnLive from the months of December to March, the most inexpensive choice was to purchase the Playpack bundle for four months.

After searching through the games in the Playpack bundle we selected the games shown in Table 1: Games Chosen for Experiments

Game	Genre
Unreal Tournament III	First Person Shooter
Grand Ages: Rome	Real Time Strategy
Batman: Arkham Asylum	Third Person

Table 1: Games Chosen for Experiments

[11] http://www.onlive.com/games/playpack#&tab=top_games

Figure 2: A Screenshot of Unreal Tournament III's Start Screen

Figure 3: A Screenshot of Unreal Tournament III Gameplay

Figure 4: A Screenshot of Grand Ages: Rome's Start Screen

Figure 5: A Screenshot of Grand Ages: Rome Gameplay

Figure 6: A Screenshot of Batman: Arkham Asylum's Start Screen

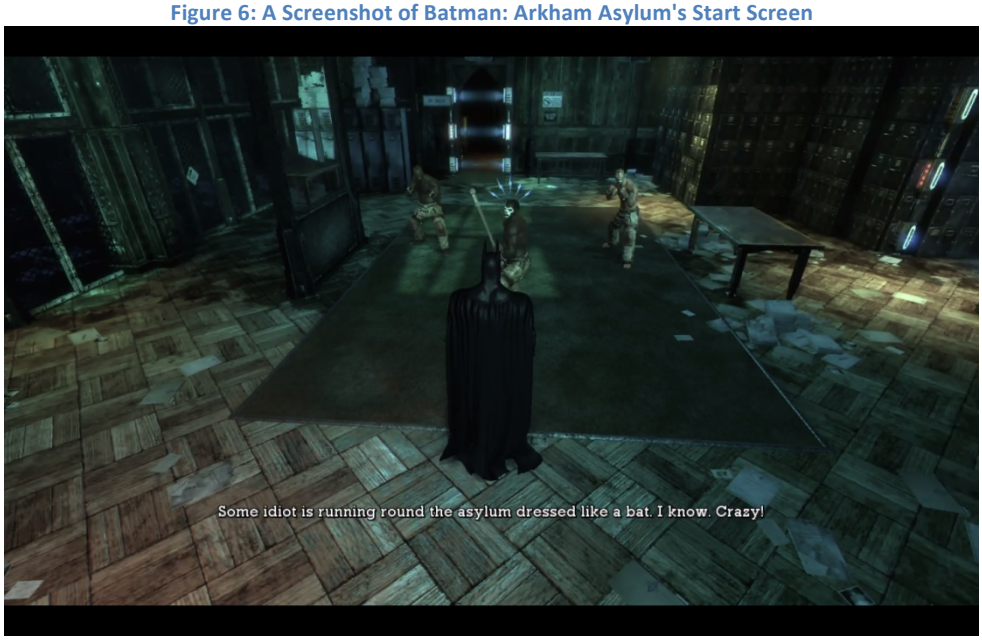


Figure 7: A Screenshot of Batman: Arkham Asylum Gameplay

As shown in Table 2: Recommended System Requirements, the system requirements for the game to be played on the PC were all very similar. Although we were playing on the OnLive system, and not on our PC, the game requirements provided a gauge of how much computer power each game required from OnLive's systems.

Recommended System Requirements			
Game	UT 3[3]	Batman[4]	Rome[5]
Intel CPU	Pentium D 2.66GHz	Pentium D 3.0GHz	Core 2 Duo E4500 2.2GHz
AMD CPU	Athlon 64 4000+	Athlon 64 X2 Dual Core 3800+	Athlon 64 X2 Dual Core 3600+
Nvidia GPU	GeForce 8800 GS	GeForce 7900 GT	GeForce 7800 GS
AMD GPU	Radeon X800 XT Platinum	Radeon X800 XL	Radeon X850 Series
RAM	1 GB	2 GB	1 GB
Direct X	DX 9	DX 9	DX 9
HDD Space	8 GB	9 GB	4 GB

Table 2: Recommended System Requirements

Another reason for choosing these games is that they were all released within the same relative time period, from 2007 to 2009. This is important because games designed and released during the same time period will require similar technology (e.g. in terms of computer power, as seen in Table 2: Recommended System Requirements) which will help keep the experiments consistent for testing our hypotheses.

4.2 Experiment Configuration

The following section explains the details of how our experiments were designed.

4.2.1 Experiment Requirements

The first step in our experiment setup was to determine what hardware and software was needed to test our hypothesis laid out in the introduction. In order to conduct experiments on OnLive, YouTube, and Skype, we determined that at the very minimum we would need the following:

- A traffic shaper.
    - The traffic shaper needed to perform the following functions:

[12] http://www.game-debate.com/games/index.php?g_id=697&game=Unreal%20Tournament%20III
[13] http://www.game-debate.com/games/index.php?g_id=461&game=Batman:%20Arkham%20Asylum
[14] http://www.game-debate.com/games/index.php?g_id=493&game=Grand%20Ages:%20Rome

        - Modify network traffic by inducing latency, creating packet loss, and limiting bandwidth.
        - Capture network traffic.
        - Hand out DHCP leases and perform NAT for machines located behind the internal network interface.
- Two Switches.
    - We decided that we needed at least one switch between the traffic shaper and the rest of the WPI network. We used this switch to connect other devices directly into WPI's network.
    - Due to our requirements, we needed to add another switch between all of the devices behind our traffic shaper. This allowed us to connect multiple devices to the one internal network port on the traffic shaper.
- A TV.
    - A television for the OnLive MicroConsole. The television needed to support 1080p resolutions and have an HDMI port available.
- A Desktop Computer.
    - A computer to run Skype and YouTube tests. The computer would have to perform the following functions:
        - Run Skype in full screen mode.
        - Run FRAPS[15] game capture software.
        - Run YouTube Videos at 1080p resolution.
        - Run OnLive's desktop application to conduct the FRAPS portion of the experiment.
- A Computer with the Ability to run Skype.
    - A MacBook laptop was used during the Skype experiments so that we knew the path of the network traffic when the aforementioned desktop computer and MacBook were connected during the Skype calls.

Based on the equipment we used and how we set it up, we created a picture of our lab network. The picture in Figure 8, shows each device and its location in the network. The devices behind the traffic shaper include the OnLive MicroConsole and the desktop computer. The

[15] http://www.fraps.com/

devices on the main WPI network include the MacBook laptop and the traffic shaper running on the box labeled router. The switch between these two devices is necessary because there was only one WPI network port available in our lab.

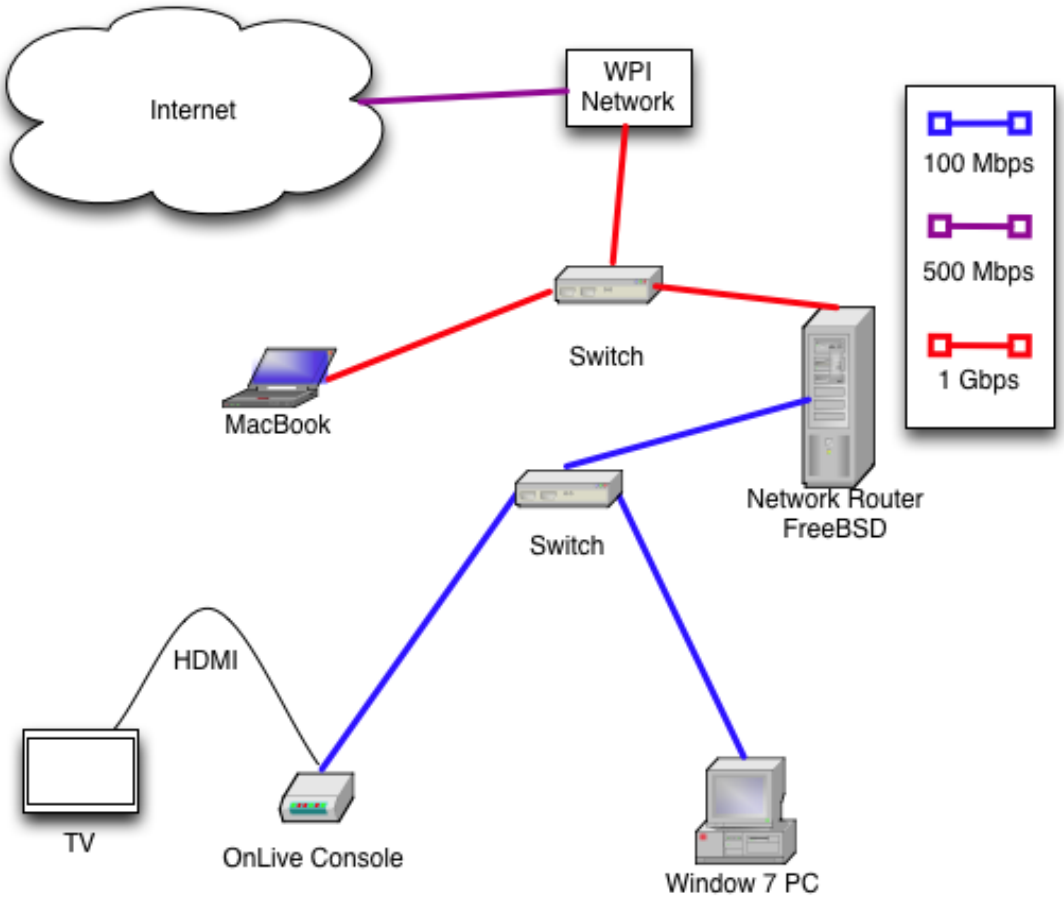


Figure 8: Network Map of Experiment Setup

4.2.2 Traffic Shaper

In order to accomplish each of the necessary traffic shaper functions, we used a custom configured traffic shaper running FreeBSD, an open source UNIX operating system. [16] FreeBSD was chosen because of its built in network functionality. We also found FreeBSD software that allowed us to modify network traffic, capture network traffic, and perform NAT, described below.

[16] http://www.freebsd.org/about.html

The first problem that we had to address was the DHCP leasing problem. To solve this problem we used a native BSD program called DHCPD[20]. This software was already built and compiled into our initial FreeBSD installation and only required some basic configuration for interface and IP specification.

We had to figure out how to perform NAT between the two network interfaces, the internal and external network links. After some online research, we initially chose to use firewall software that would allow us to implement NAT quickly and easily. This software, PF(personal firewall) [17] was easy to setup and ran well, but during the next step of our traffic shaper setup we realized that PF would not suit our needs. For our network traffic modification we wanted to use a very powerful and popular tool, DummyNet[18], but after reading about the mechanics of DummyNet, we realized that making it run in conjunction with PF could prove to be problematic. Essentially, DummyNet is built off of another FreeBSD firewall program (IPFW)[21]. After some more research we decided to remove PF and configure IPFW as our NAT and Firewall program.

In order to address the network modification functionality we chose DummyNet. As previously mentioned, DummyNet is a powerful and popular tool that is used to modify network traffic. For example, someone can use DummyNet to limit the bandwidth of a particular device on his or her network. This would be useful if someone was trying to run a home server but did not want to use their entire residential connection for that server. DummyNet can also be used to induce a wide variety of network conditions like packet loss and latency, two network metrics that we wanted to address.

4.2.3 Desktop Computer

The first part of our desktop configuration was to determine what operating system to use. For the desktop system we chose to run Windows 7. Windows has a wide variety of software available and the video services it supports particularly Skype and YouTube.

We needed a program that could gather statistics on the games we were testing that could not be obtained through packet captures. FRAPS[19] is a program that not only displays frame rate information about games, but also allows for the recording of Frames Per Second and inter-frame times directly into excel files for graphing and analyzing.

[17] http://www.openbsd.org/faq/pf/
[18] http://info.iet.unipi.it/~luigi/dummynet/
[19] http://www.fraps.com/

4.3 Testing

The following section discusses the methods in which we tested OnLive, YouTube, and Skype.

4.3.1 TCPDump Commands

To capture the network data for OnLive and other streaming services, the TCPDump[20] command on the Unix system was used. TCPDump is a command-line packet analyzer able to create pcap files of our data. A pcap file is a packet capture data file that is used in Wireshark and contains network packet data created during the live network capture[22].

4.3.2 Timing

During preliminary tests we used packet captures of fifteen minutes and five minutes. After analyzing both lengths of packet captures, it was determined that a two minute and thirty second packet capture was sufficient to observe the network characteristics of OnLive, YouTube, and Skype.

For each test, the application was run up to the point that we determined was good for testing, and then the packet capture was started. For the games needed to be at a point that represented the core gameplay. We also ensured, for each game, that the loading of the level or mode had been completed and there was a short time period for the game to stabilize before data was collected.

A similar setup was done for the Skype testing. After making sure that all non-essential background processes were turned off, the Skype video call was made. With both computers having the Skype video call on full screen, the call was given a short time period to stabilize and then on a synchronized countdown, both the FRAPs and the packet capture were started.

Capturing the network data for YouTube was a bit more complicated due to the infrastructure of YouTube's Website. Unfortunately, it is impossible to have YouTube automatically start the videos in 1080p. For this reason, the process used was:

1. The browsing history of the browser, Google Chrome, was cleared.
2. The YouTube link was pasted into the URL bar.
3. The packet capture and URL link were started simultaneously.
4. As the YouTube video came up the video was changed to full screen and then to 1080p.

[20] http://www.tcpdump.org/

4.3.3 Gameplay to Test

Each game has a wide variety of phases, from cut scenes to boss fights to the menu to mini-games. For our purpose, we needed to test gameplay that was easy to replicate and was an accurate representation of what a majority of the core gameplay was like.

Unreal Tournament III

To make each trial of Unreal Tournament III as consistent as possible, a game was set up on the same map, using the same settings for every experiment.

- A free-for-all match containing only Non-Playable-Characters (NPCs or Bots) was started.
- The map was set to Rising Sun.
- The number of Bots was set to 10.
- The Time and Score Limit were set to Infinity.
- No mutators (additional options such as one hit kills, low gravity, etc.) were selected.
- Forced Respawns were also chosen.

It was impossible for the player to perform the same actions every time due to the nature of the opposing AI and a free-for-all match, but the gameplay and actions taken during each trial were done to achieve the same goal each time. Each trial consisted of gathering the weapons, armor, and health laid out throughout the level and using them to the advantage of the player to defeat the Bots in game.

Batman: Arkham Asylum

Keeping the trials of Batman: Arkham Asylum similar each time proved to be more difficult than originally anticipated. There are a lot of cut scenes and differing gameplay within this game, so playing throughout the levels would lead to being interrupted by gameplay that was inconsistent with what we wanted to test. It was also complicated to replicate trials because the saving system was progress based and could not be controlled. This meant that once the player were successful in an area, it auto saved and could not go back. This made it impossible to test the same part of gameplay over and over again.

After progressing through the game, several modes were unlocked. One of the game modes was a challenge to stay alive as long as possibly while fighting off an infinite number of

enemies. This challenge mode provided non-stop action without cut scenes, but it ensured that each trial was relatively similar to the others in terms of actions taken.

Each time the challenge mode was started, the player would start in a small square room with 3 enemies. As the player started to fight the Bots, more would appear. It became increasingly difficult to dispose of the incoming Bots because the player could never focus their attention on one enemy for too long. If an enemy was focused on for too long, the other enemies would interrupt any action being taken, and damage would be taken by the main character. For this reason, it was simple enough to merely wound or injure the enemies, but never fully get rid of them due to the overwhelming amount of Bots that would continually show up. Although the actions taken each trial were different, like Unreal Tournament III, each trial consisted of using attack combos to incapacitate as many enemies as possible until the number of Bots was overwhelming.

Grand Ages: Rome

Grand Ages: Rome was the easiest game to keep consistent. The same level was picked each time and the same actions were taken for every trial following a consistent pattern of what to build next and where to build it. During the gameplay, no enemies were encountered so it was easier to keep everything consistent because the player was the only one able to change the outcome of the game. We had full control of what actions the player could take. The buildings were all placed in the same place. Because of the small number of resources given to us in the beginning of the game, it is easy to build the same objects over and over again in the same pattern and around the same time due to the low income that our society obtains during the early phase of the game. The actions taken were:

1. Build 3 insulas.
2. Build a pig farm.
3. Build a wheat farm.
4. Build an aqueduct.
5. Build a large water fountain.
6. Build a logging shed.
7. Build 2 more insulas.
8. Build a butcher shop.
9. Build a farmer's market.

10. Build a grape farm.
11. Build 2 logging sheds.

It was after the 11th step that the two and a half minute mark was surpassed and the packet capture ended.

4.3.4 YouTube Tests

A video of a Real Time Strategy game, StarCraft 2, was chosen for the YouTube tests. The video can be found at the following link: http://www.youtube.com/watch?v=0NTeyF6wQUs.

The video consists of a game between two opponents, only showing the gameplay of StarCraft 2. For the YouTube tests, the Google Chrome[21] browser was used and the actions taken were:

1. Clear the browsing history of Google Chrome.
2. YouTube video link was pasted into the URL bar and enter was hit.
3. The packet capture was started simultaneously with step 2 as the enter button was hit.
4. As quick as possible the video quality settings were set to 1080p.
5. As quick as possible the video was set to full screen.
6. The packet capture was stopped after two and a half minutes.

4.3.5 Skype Tests

A Skype video call was set up between the desktop computer and the MacBook. Each camera was pointed at the individual operating the specific computer (Alexander with the MacBook, Michael with the desktop computer). After the video call was set up and both computers were on full screen, the packet capture began. After two and a half minutes the packet capture was stopped.

4.3.6 DummyNet Tests

Once the baseline data was taken with simple packet captures, DummyNet was utilized to further analyze the applications being tested. DummyNet was used to mimic different network situations. With DummyNet we were able to restrict the bandwidth, add random packet loss, and add latency when capturing data with TCPDump. This allowed us to test the boundaries of each

[21] https://www.google.com/chrome

application and how they react to network instability. We tested the following conditions independently of each other:

- Downstream bandwidth restricted to 10 Mbps, upstream bandwidth restricted to 2 Mbps.
- Downstream bandwidth restricted to 5 Mbps, upstream bandwidth restricted to 1 Mbps.
- Downstream random packet loss of 1%, upstream random packet loss of 1%.
- Downstream random packet loss of 1.5%, upstream random packet loss of 1.5%.
- Downstream added delay of 20ms, upstream added delay of 20ms.
- Downstream added delay of 35ms, upstream added delay of 35ms.

4.3.7 FRAPS Tests

FRAPS[22] allowed us to measure the frames per second, the frame times, and the minimum, maximum, and average frames per second of the games being tested. FRAPS was used when the network was not affected, but also when the aforementioned conditions mentioned in Section 4.3.6 DummyNet Testswere altered through the use of DummyNet. To capture the data with FRAPS only a few settings had to be changed. The length of the capture was set by inputting, the number of seconds (150), the folder in which the files should be created, which statistics to capture, and what button starts the capture.

[22] http://www.fraps.com/

5 Results

The results section contains much of the relevant data that we collected and helps explain in detail the information collected through packet captures of OnLive, YouTube, and Skype.

5.1 Initial OnLive Packet Captures

The following data was collected through the use of TCPDump commands and analyzed in Excel. Packet captures that lasted two and a half minutes were organized into tables and graphs so as to analyze the different applications. Only the OnLive games, Grand Ages: Rome, Unreal Tournament III, and Batman: Arkham Asylum were analyzed.

5.1.1 Downstream Packet Captures

The data from the packet captures from each game was organized into Table 3, depicting different statistics. These statistics included the standard deviation, minimum, maximum, and average number of packets as well as the size of each packet. From this table we were able to quickly see the differences across each game.

Unreal Tournament III								
Trial	Packet Min	Packet Max	Packet Average	Packet STDEV	Kilobytes Min (kb)	Kilobytes Max (kb)	Kilobytes Average (kb)	Kilobytes STDEV (kb)
1	687	775	736.3	18.1	730.257	850.061	786.8	26.6
2	674	784	733.9	18.8	711.337	844.795	780.8	23.6
3	671	777	731.0	19.2	701.891	848.983	779.5	26.0
Batman: Arkham Asylum								
Trial	Packet Min	Packet Max	Packet Average	Packet STDEV	Kilobytes Min (kb)	Kilobytes Max (kb)	Kilobytes Average (kb)	Kilobytes STDEV (kb)
1	727	806	761.5	14.2	755.3	819.6	795.6	11.5
2	729	794	759.2	15.0	766.2	817.8	791.6	10.6
3	717	797	757.4	14.8	759.5	818.5	794.6	11.6
Grand Ages: Rome								
Trial	Packet Min	Packet Max	Packet Average	Packet STDEV	Kilobytes Min (kb)	Kilobytes Max (kb)	Kilobytes Average (kb)	Kilobytes STDEV (kb)
1	311	661	506.5	62.6	140.5	707.1	482.5	94.8
2	389	737	500.5	51.1	316.3	816.7	477.1	74.9
3	379	680	509.3	50.9	308.0	710.8	486.8	67.4

Table 3: Downstream Packet Captures of Each Game

Our initial hypothesis stated that because OnLive is comparable to a video streaming service, the packet captures of each game should be very similar. Both Batman: Arkham Asylum and Unreal Tournament III have similar downstream packet capture statistics, but even these two games differ more than we originally suspected. It was not until looking at the Grand Ages: Rome results that we realized the packet captures of different games do vary.

The results from Grand Ages: Rome were surprising and completely contradicted our initial hypothesis. Every single one of Grand Ages: Rome's data was significantly smaller than both Unreal Tournament III and Batman: Arkham Asylum except for the standard deviation. It

was these results that prompted us to reconsider our hypothesis and how OnLive was working to deliver its products to users.

To grasp the differences of each game more easily, the data collected from all three games were plotted on the same graph. For each game the second trial was chosen and plotted on the same graph to clearly compare and contrast the packet captures of each game.

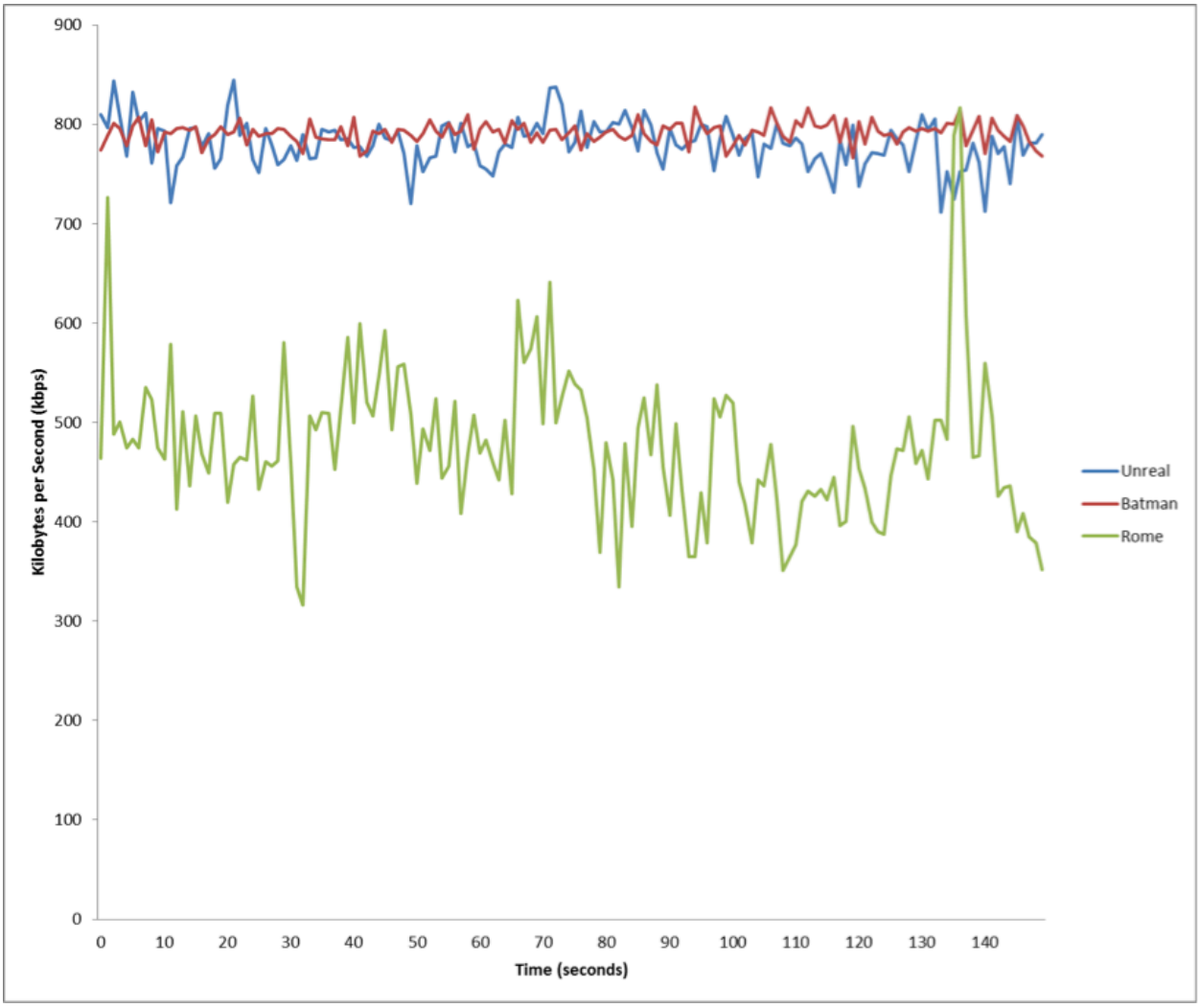


Figure 9: Batman, Unreal, and Rome Downstream: Kilobytes vs. Time

Figure 9 shows a clear visual difference between Grand Ages: Rome and the other two games. At the same time, it is possible to see the subtle differences between Unreal Tournament III and Batman: Arkham Asylum. This graph shows the consistency of Unreal Tournament III and Batman: Arkham Asylum while also showing the large standard deviation of Grand Ages: Rome. A graph similar to this was also plotted for packet size. It is visually similar to Figure 9 and can be found in the appendix.

5.1.2 Upstream Packet Captures

Interestingly the upstream packet capture data contradicted our hypothesis as well. We assumed that because each game was from a different genre, and the speed at which you play each game is different, the upstream would vary greatly across all three games.

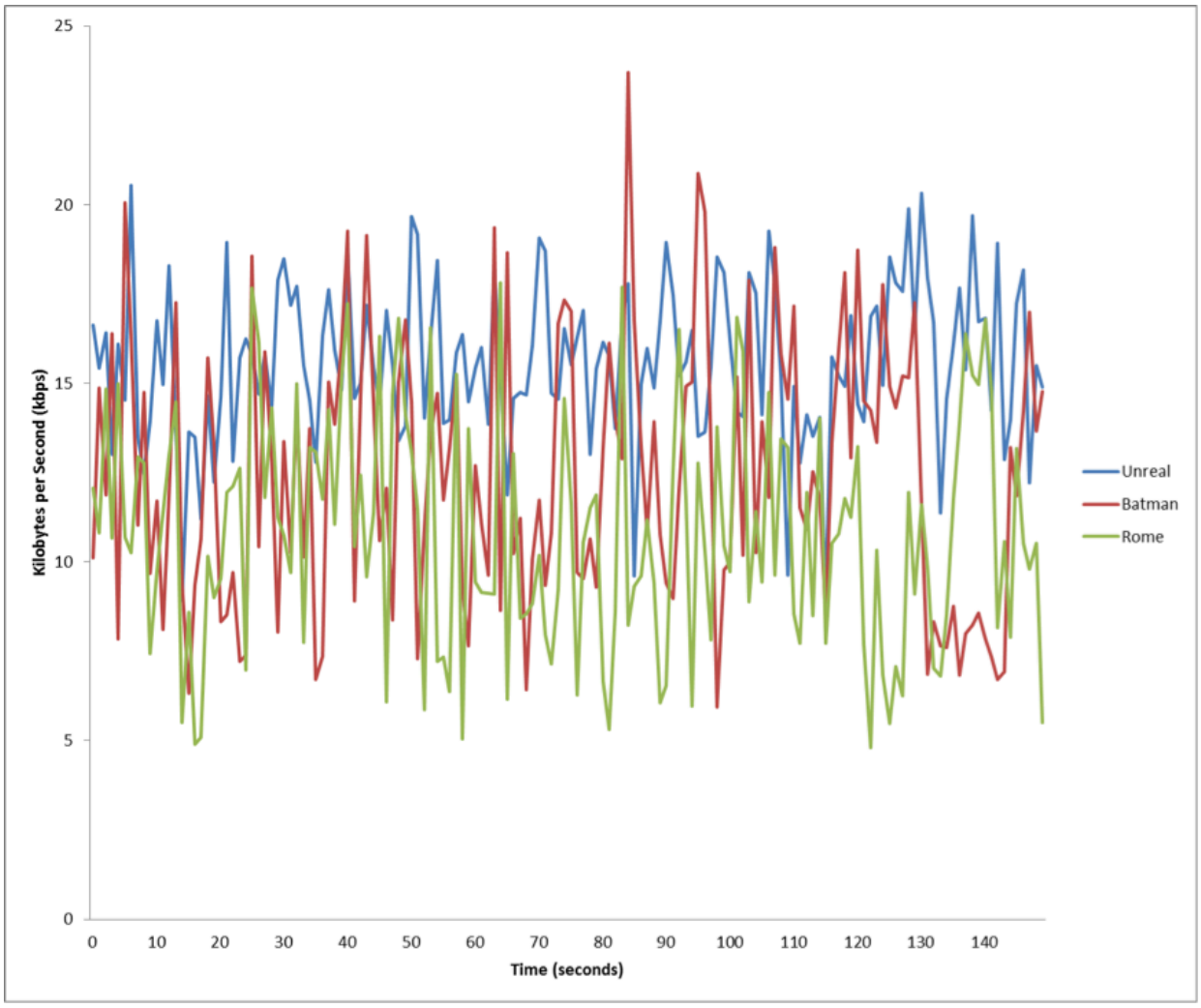


Figure 10: Batman, Unreal, and Rome Upstream: Kilobytes vs. Time

As evident from Figure 10, the data collected for the upstream information of each game is similar. All three games have a similar range of Kilobytes per second. Similar to the downstream packet captures, the Number of Packets vs. Time graphs for the upstream are visually similar and can be found in the appendix.

5.2 OnLive Versus Video Streaming Services

The following section contains information from experiments in the previous section and the information collected from the trials for Skype and YouTube.

5.2.1 Downstream Packet Captures

Overall when compared with the results of the OnLive trials were much different than the results of the Skype and YouTube trials. Table 4 shows statistics for the two video applications and the three OnLive games.

Trial	Packet Min	Packet Max	Packet Average	Packet STDEV	Kilobytes Min (kb)	Kilobytes Max (kb)	Kilobytes Average (kb)	Kilobytes STDEV (kb)
Unreal	674.0	784.0	733.9	18.8	711.3	844.8	780.8	23.6
Batman	729.0	794.0	759.2	15.0	766.2	817.8	791.6	10.6
Rome	389.0	737.0	500.5	51.1	316.3	816.7	477.1	74.9
Skype	78.0	212.0	135.7	30.3	35.8	201.7	102.3	42.7
YouTube	7.0	3512.0	2981.5	651.3	6.7	5317.2	4511.1	993.2

Table 4: All 5 Applications Downstream Packet Capture

Looking at Table 4, YouTube had the greatest lower and upper bounds and was the most erratic in terms of the number of packets per second and packet size.

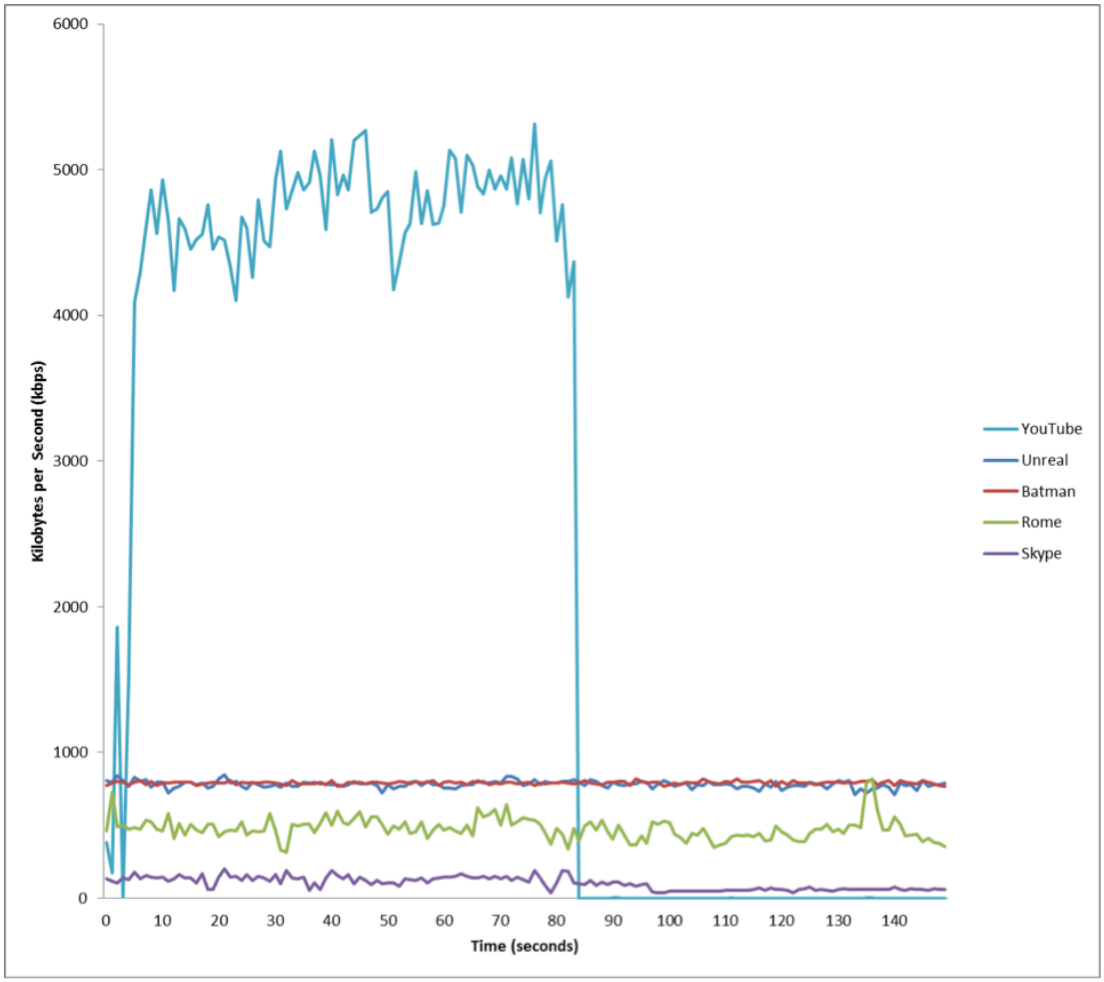


Figure 11: Unreal, Batman, Rome, Skype, and YouTube Downstream: Kilobytes vs. Time

From Figure 11 it is possible to see the differences in network traffic for each system. YouTube's packets are much larger than any of the other applications tested. Similar to our other tests, the Number of Packets vs. Time graph looks very similar to Figure 11 and can be found in the appendix.

5.2.2 Upstream Packet Captures

When looking at the upstream packet captures of each application, Skype produced results that were unexpected. All four other applications data looks similar to what we expected and what we saw before, but the Kilobytes vs. Time graph shows Skype with the largest average upstream.

Trial	Packet Min	Packet Max	Packet Average	Packet STDEV	Kilobytes Min (kb)	Kilobytes Max (kb)	Kilobytes Average (kb)	Kilobytes STDEV (kb)
Unreal	54.0	142.0	106.9	15.9	8.8	20.5	15.6	2.2
Batman	33.0	160.0	83.9	26.5	6.0	23.7	12.5	3.7
Rome	29.0	127.0	75.2	24.5	4.8	17.8	10.8	3.2
Skype	133.0	187.0	168.5	6.9	107.2	163.5	140.9	6.7
YouTube	15.0	2989.0	2161.0	634.9	0.9	179.3	130.3	37.4

Table 5: All 5 Applications Upstream Packet Capture

Unlike the OnLive games, Skype's Kilobytes per second manages to stay on par with what is being displayed by YouTube. As seen in Figure 12, Skype's upstream bandwidth shows that Skype uploads much more data per second than the OnLive games. This difference is most likely because of the two-way video functionality of Skype.

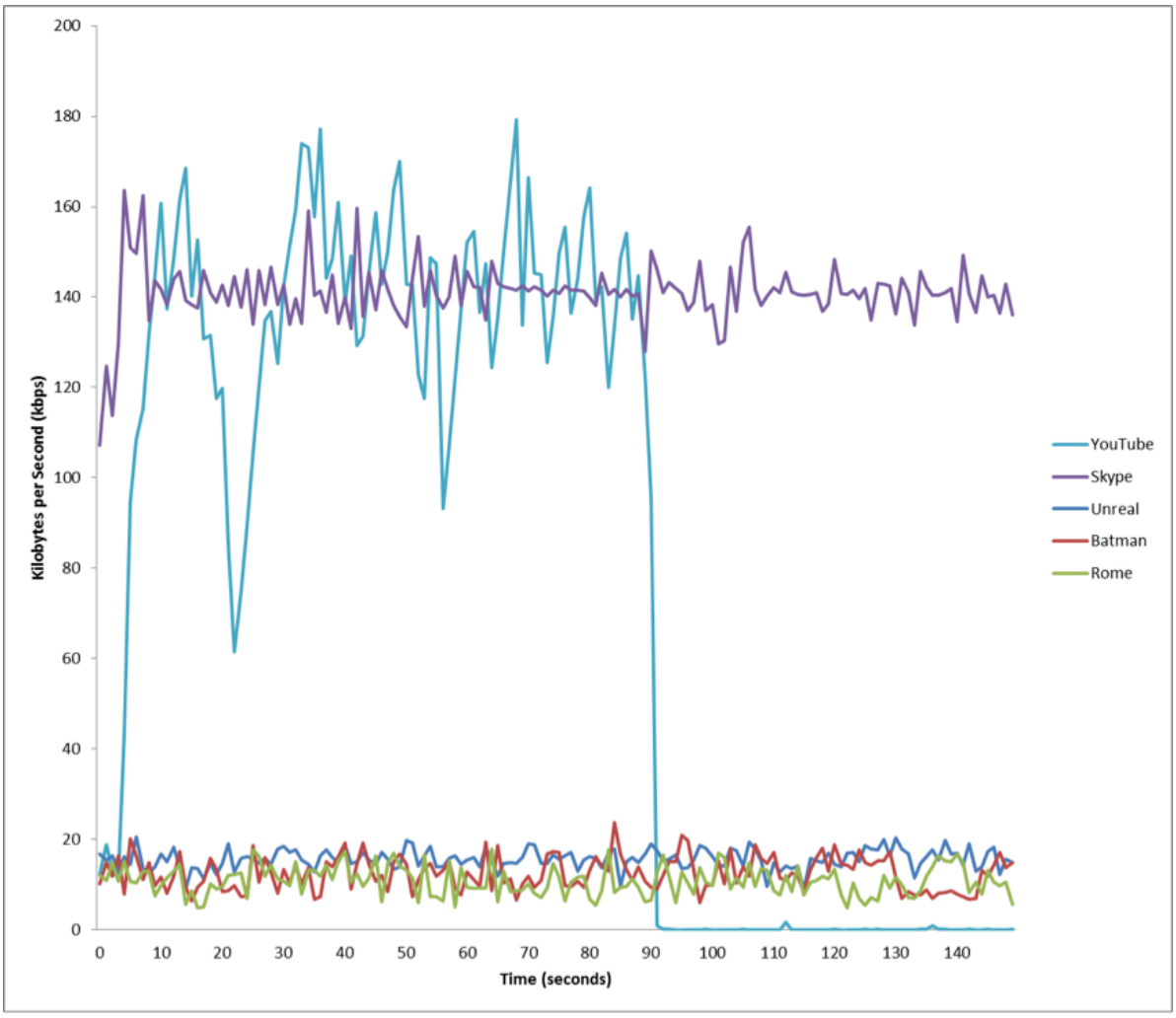


Figure 12: Unreal, Batman, Rome, Skype, and YouTube Upstream: Kilobytes vs. Time

5.3 OnLive and Network Variation

All of the packet captures collected thus were with an unrestricted network and no changes to the traffic shaper had been made. The following results will display how a restricted network can affect the data captured for the OnLive games, YouTube, and Skype. For this part of our investigation DummyNet was used on our FreeBSD router to modify the network and cause disruptions of service. For the OnLive portion of our investigation we used Unreal Tournament III.

5.3.1 Bandwidth Restriction and OnLive

The first network characteristic we changed was the bandwidth of the network connection. The OnLive game was played under the following bandwidth restrictions: Unrestricted Upstream and Downstream, 10Mbit/s Downstream and 2Mbit/s Upstream, and 5Mbit/s Downstream and 2Mbit/s Upstream. These bandwidth restrictions were chosen because they are similar to a many consumer Internet connections available today.

The downstream bandwidth measurements of OnLive are shown in Table 6. These measurements indicate that OnLive is consistent over different Internet bandwidths. The standard deviations of both the packets per second and the kilobytes per second both decrease as the bandwidth decreases. Overall this table indicates that as the bandwidth decreases, the packets per second and kilobytes per second proportionally decrease.

	Packet Min	Packet Max	Packet Average	Packet STDEV	Min (kB)	Max (kB)	Average (kB)	STDEV (kB)
Unrestricted	687	797	751.9	19.1	691.4	839.1	773.1	25.7
10 Mbps	482	564	537.4	13.9	480.3	579.2	546.2	17.4
5 Mbps	297	353	328.1	9.9	222.1	275.2	257.0	10.1

Table 6: Vary Bandwidth- Unreal Tournament Downstream

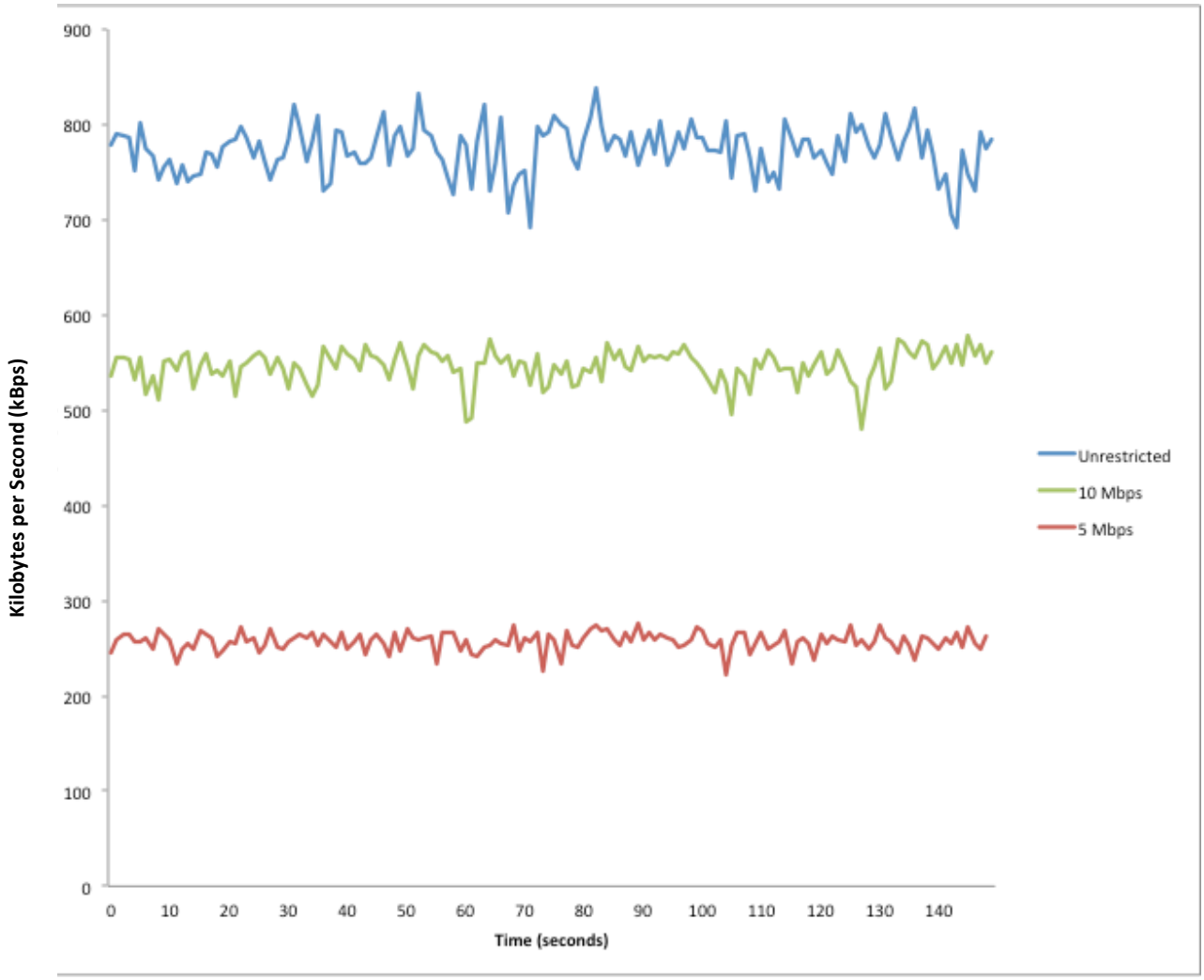


Figure 13: Vary Bandwidth-Unreal Tournament Downstream KiloBytes Versus Time

The graph in Figure 13 shows the measured downstream bandwidth in Kilobytes versus Seconds. The lowest line (5 Mbps) averages around 250 kBps, which is equivalent to 2 megabits per second, which also matches OnLive's network requirements of at least 2 megabits per second of Internet bandwidth. This graph supports the conclusion drawn from Table 6 that OnLive makes use of whatever bandwidth is available.

The CDF in Figure 14 shows the packet sizes for OnLive across varying bandwidths. This CDF shows that nearly 40% of the packets in the 5Mbps test were either 200 bytes or 1400 bytes in size while 60% of the packets from the 10Mbps and unrestricted tests were 1400 bytes in size. This indicates that as the available bandwidth changes, so do the sizes of the packets sent.

The CDF in Figure 15 shows the interpacket times for OnLive across varying bandwidths. This CDF shows that almost 60% of the packets are less than 2ms apart for all of the trials. Overall, this CDF indicates that the interpacket times for the 5Mbps and 10Mbps bandwidths are slightly larger than the interpacket times for the unrestricted bandwidth.

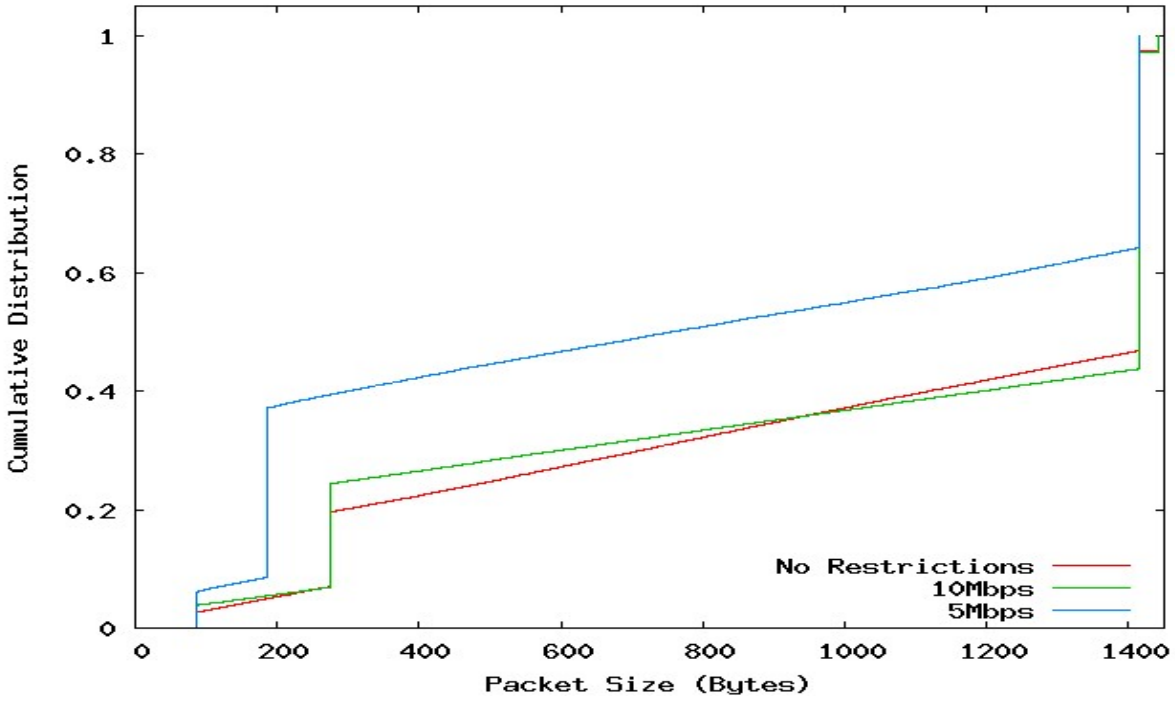


Figure 14: Vary Bandwidth- CDF Packet Size: Unreal Tournament

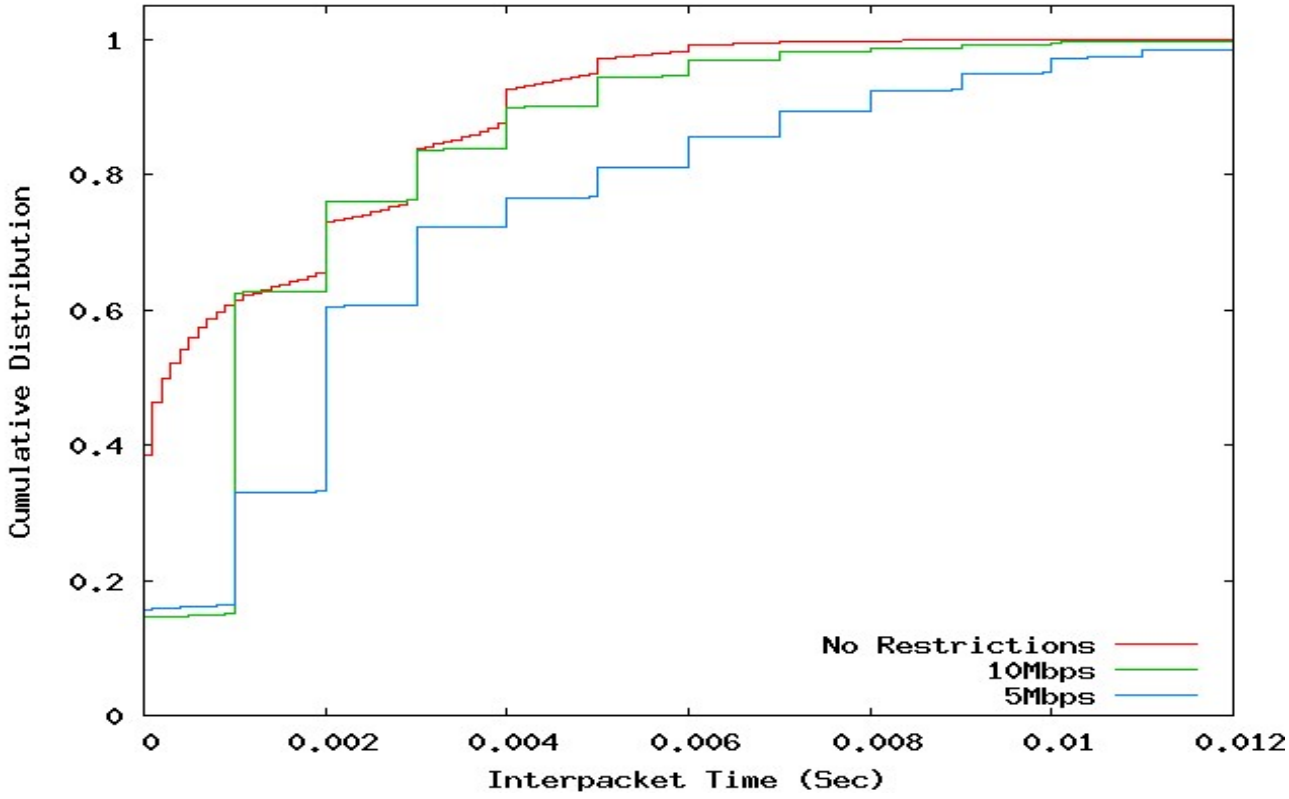


Figure 15: Vary Bandwidth- CDF Interpacket Time: Unreal Tournament

5.3.2 Packet Loss and OnLive

The next network characteristic modified was the packet loss rate of the Internet connection. For these tests, 0%, 1%, and 1.5% packet loss were used.

	Packet Min	Packet Max	Packet Average	Packet STDEV	Min (kB)	Max (kB)	Average (kB)	STDEV (kB)
0% PL	687.0	797.0	751.9	19.1	691.5	839.1	773.1	25.7
1% PL	521.0	788.0	692.3	37.5	536.5	846.1	762.3	44.1
1.5% PL	547.0	727.0	674.1	33.4	572.6	812.2	746.9	47.8

Table 7: Vary Bandwidth- Unreal Tournament Upstream

The downstream bandwidth measurements with varying packet loss for OnLive are shown in Table 7 below. The measurements indicate that OnLive downstream bandwidth is consistent even with varying packet loss rates. The standard deviations of both the packets per second and the kilobytes per second increased as the packet loss increased, but overall the averages were similar.

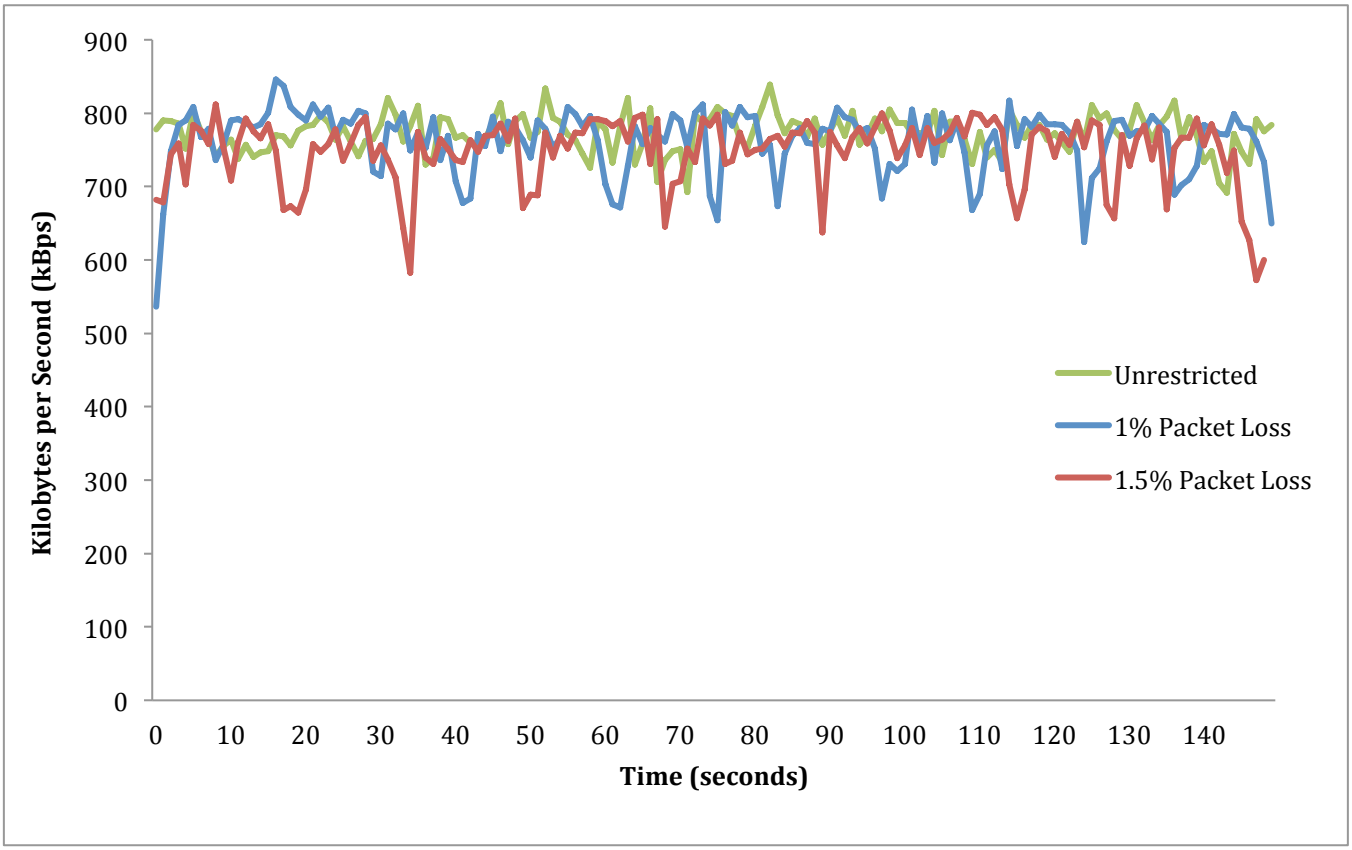


Figure 16: Vary Packet Loss- Unreal Tournament Downstream Kilobytes versus Time

The graph in Figure 16 shows the kilobytes per second downstream for Unreal Tournament across varying packet loss rates. The graph supports the conclusion drawn from Table 7 that OnLive is consistent even with packet loss.

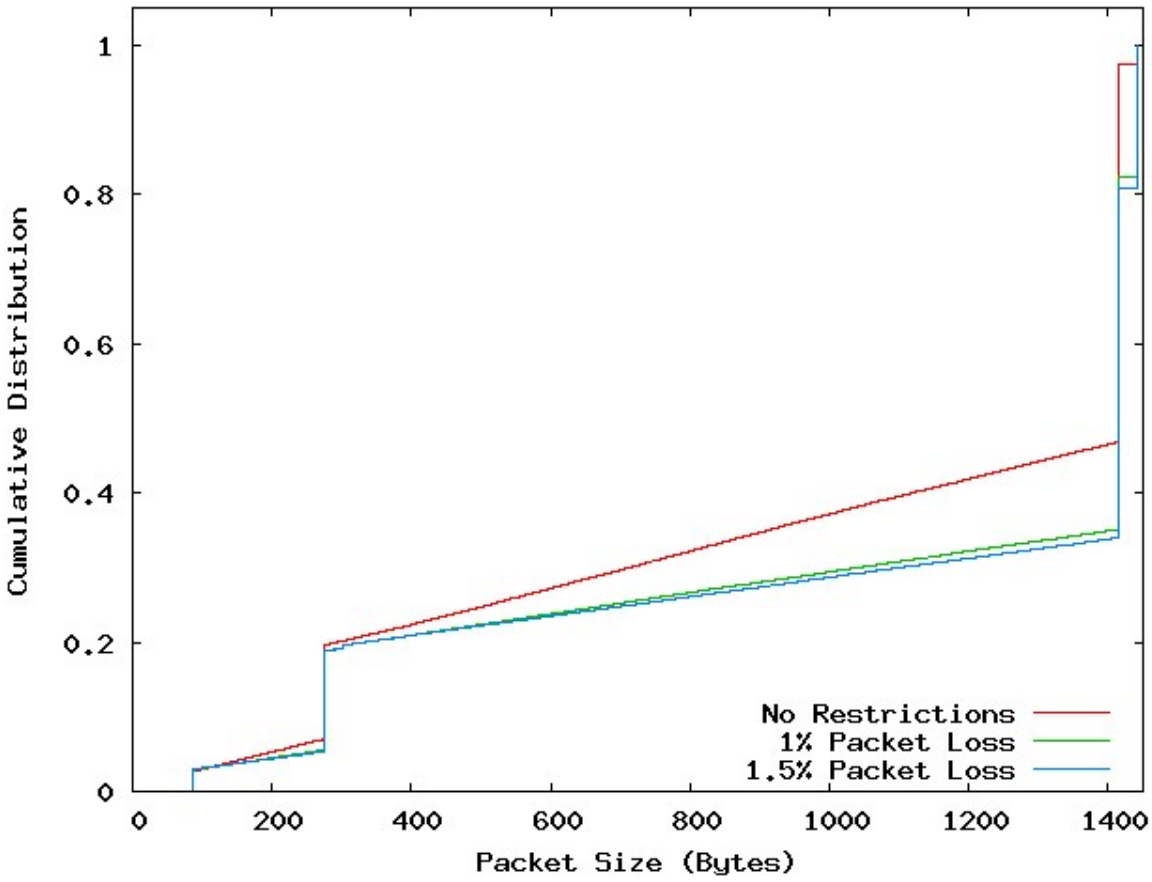


Figure 17: Vary Packet Loss- CDF Packet Size: Unreal Tournament

The CDF in Figure 17 shows the packet sizes for OnLive across varying packet loss rates. This CDF shows that the packet sizes for 1% and 1.5% packet loss are nearly identical. Overall, this CDF indicates that the higher the packet loss rate, the larger the percentage of the largest packets sent, in this case packets with a size of around 1400 bytes.

The CDF in Figure 18 shows the interpacket times for OnLive across varying packet loss rates. This CDF shows that the interpacket times for both packet loss rates are again nearly identical, but overall they vary from the trial with no packet loss.

Overall, this data indicates that OnLive maintains a consistent bandwidth even though some of the underlying network characteristics like packet size and interpacket time are slightly different.

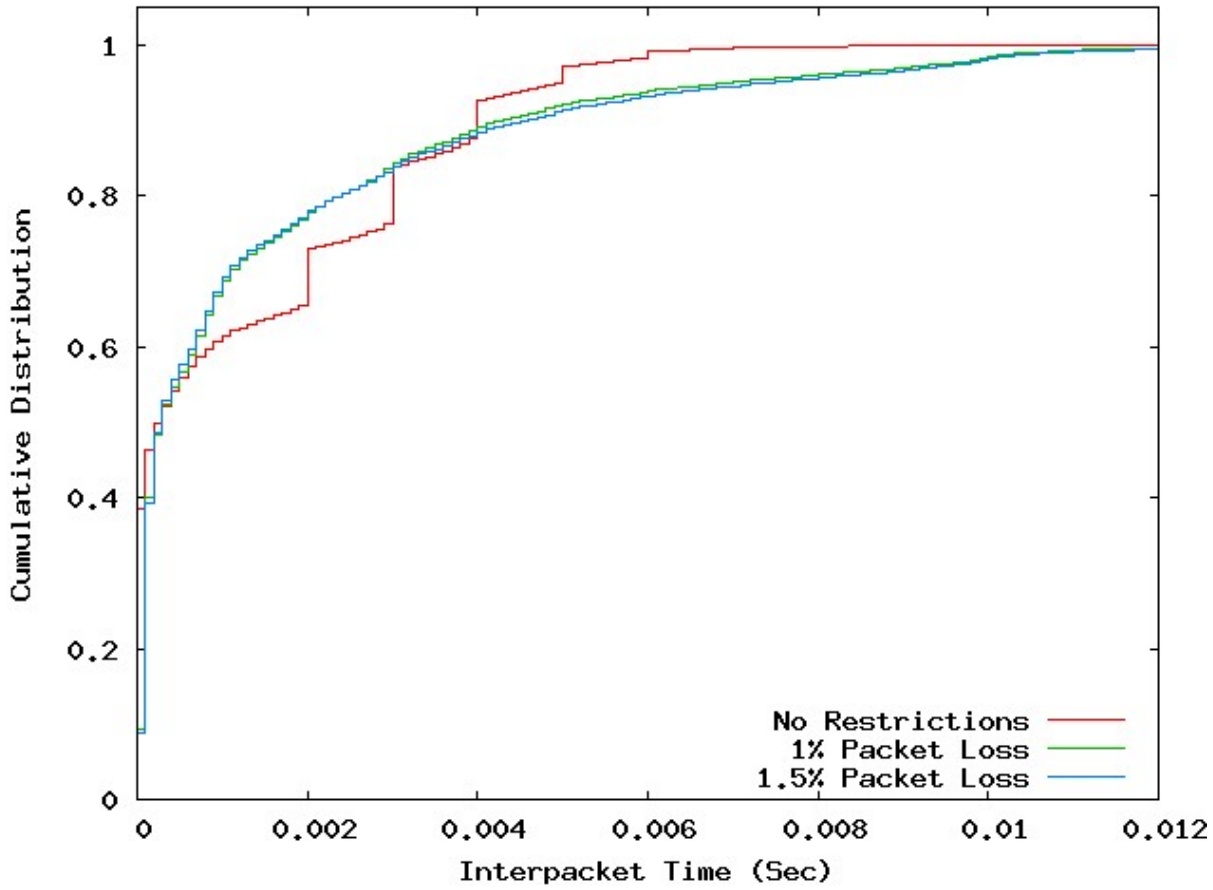


Figure 18: Vary Packet Loss- CDF Interpacket Time: Unreal Tournament

5.3.3 Latency and OnLive

The last network characteristic that was modified was the latency that was induced in the system. For these tests, 0ms, 20ms, and 35ms were used.

The downstream bandwidth measurements with varying latencies for OnLive are shown in Table 8. The measurements again indicate that OnLive downstream bandwidth is consistent even across varying latencies.

	Packet Min	Packet Max	Packet Average	Packet STDEV	Min (kB)	Max (kB)	Average (kB)	STDEV (kB)
0 ms	687	797	751.9	19.1	691.4	839.1	773.1	25.7
20 ms	456	790	747.7	33.5	343.4	826.2	766.2	46.7
35 ms	699	795	758.4	15.0	710.2	836.4	780.1	20.8

Table 8: Vary Latency- Unreal Tournament Downstream

The graph in Figure 19 shows the kilobytes per second downstream for Unreal Tournament across varying latencies. The graph supports the conclusion drawn from Table 8 that OnLive is consistent even with additional latency.

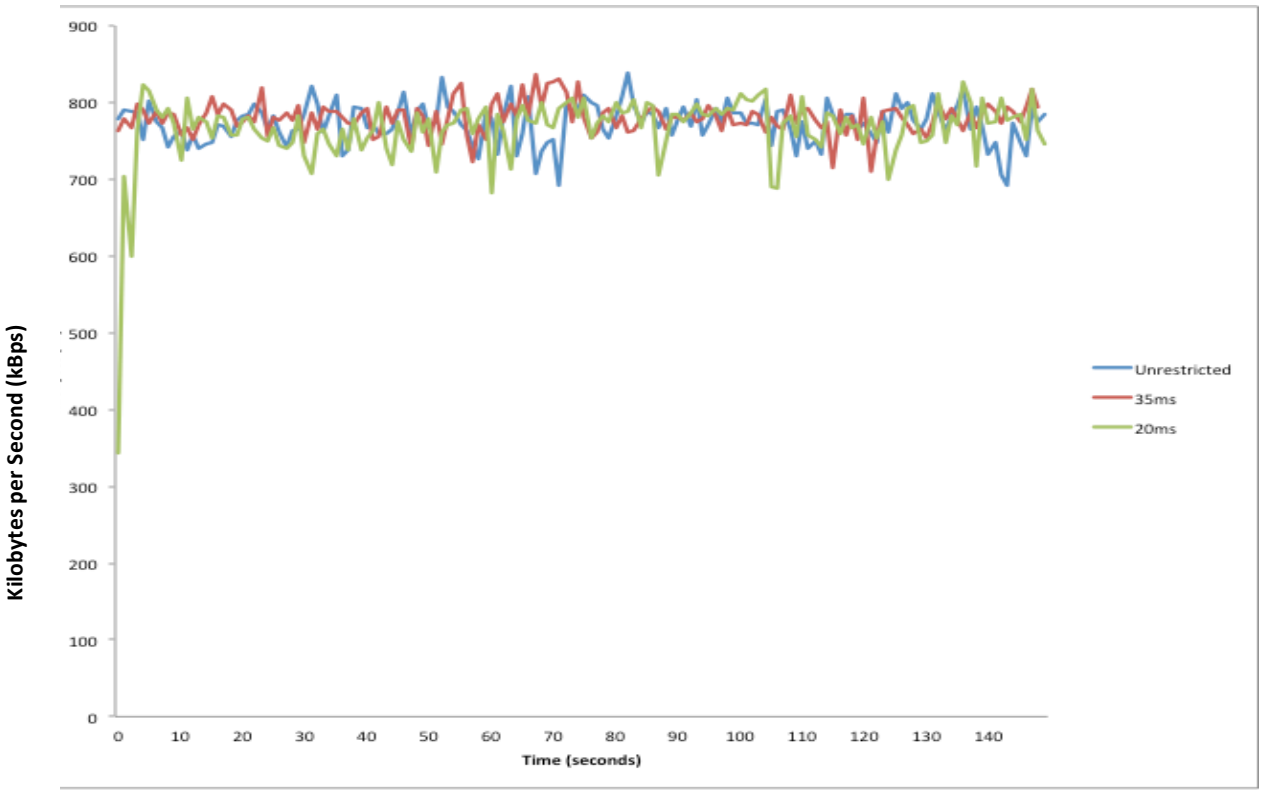


Figure 19: Vary Latency- Unreal Tournament Downstream Kilobytes Versus Time

The CDF in Figure 20 shows packet sizes for OnLive across varying latencies. This CDF shows that latency has little to no effect on the packet sizes of the OnLive service. The CDF in Figure 21 shows the interpacket times for OnLive across varying latencies. The distinct steps

seen in the two latency lines in the CDF reveal that the interpacket times are much more rigidly defined. The difference between these two lines and the unrestricted line indicates that OnLive specifically modifies its service to handle particular latencies.

Overall these results indicate that OnLive tailors its service to specifically handle varying latencies.

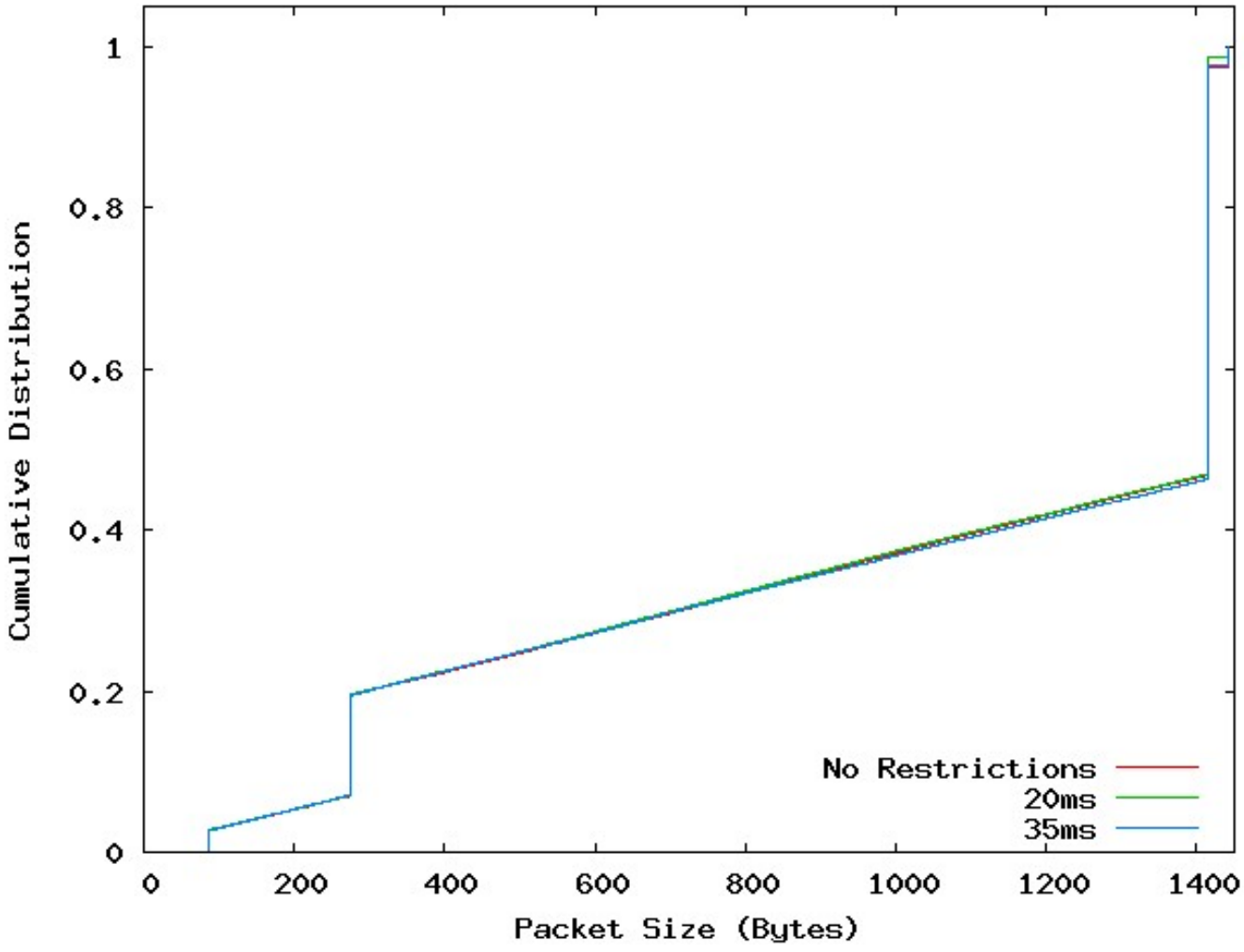


Figure 20: Vary Latency- CDF Packet Size: Unreal Tournament

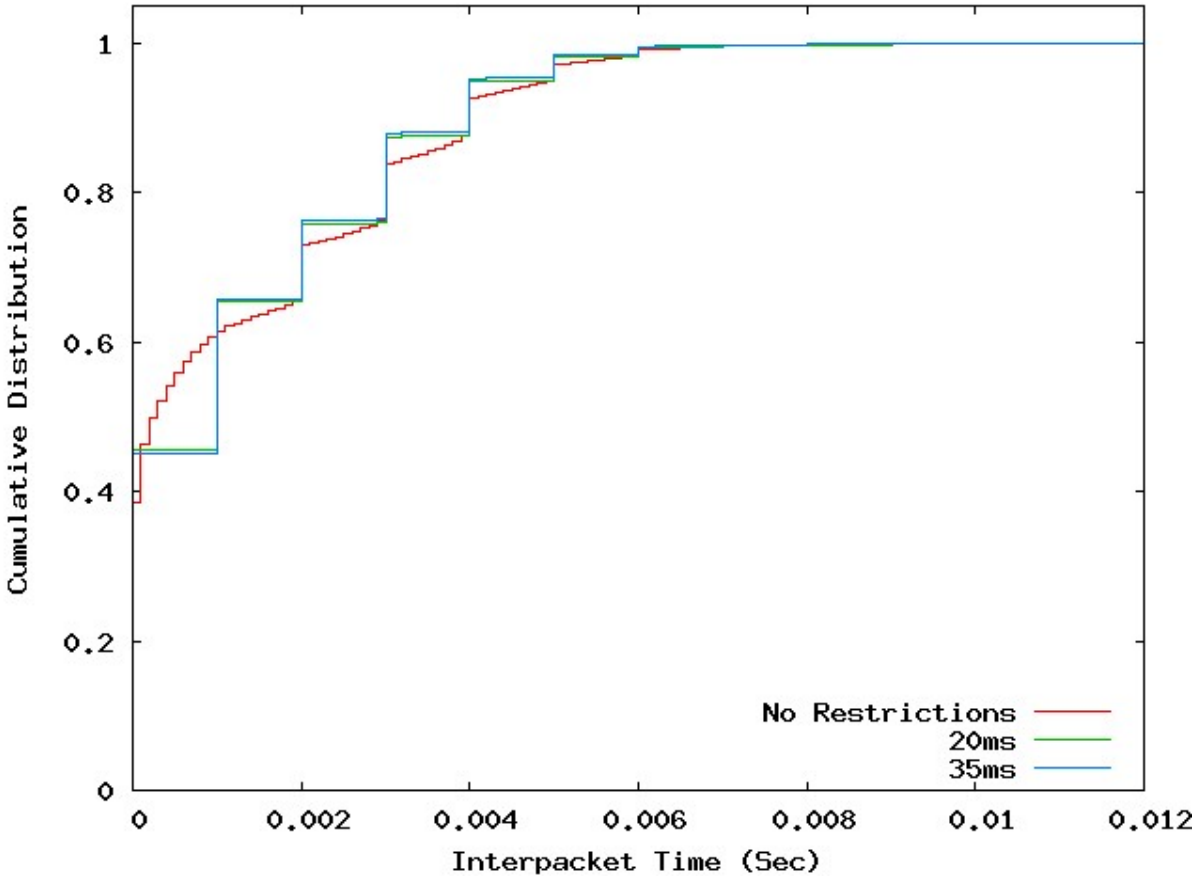


Figure 21: Vary Latency- CDF Interpacket Time: Unreal Tournament

5.4 Video Streaming Services and Network Variation

The next step in our experiment was to use the DummyNet settings for YouTube and Skype.

5.4.1 Bandwidth Restriction and YouTube

The downstream bandwidth measurements of YouTube are shown in Table 9 below. The Table demonstrates that YouTube makes use of whatever bandwidth it can. The max kilobyte per second measurement for the unrestricted bandwidth is much higher than any other bandwidth measurements for OnLive and Skype.

	Packet Min	Packet Max	Packet Average	Packet STDEV	Min (kB)	Max (kB)	Average (kB)	STDEV (kB)
Unrestricted	0	3512	1670.1	1557.8	0.0	5317.2	2526.2	2359.3
10 Mbps	183	436	413.1	28.3	216.0	642.0	623.0	49.6
5 Mbps	38	228	205.6	21.4	44.6	332.0	308.8	35.6

Table 9: Vary Bandwidth- YouTube Downstream

The graph located in Figure 22 shows the downstream bandwidths for YouTube at varying bandwidth restrictions. The graph also supports the conclusion drawn from Table 9 that YouTube will use whatever bandwidth is available. The line representing the unrestricted bandwidth is much higher than the other two bandwidth restrictions.

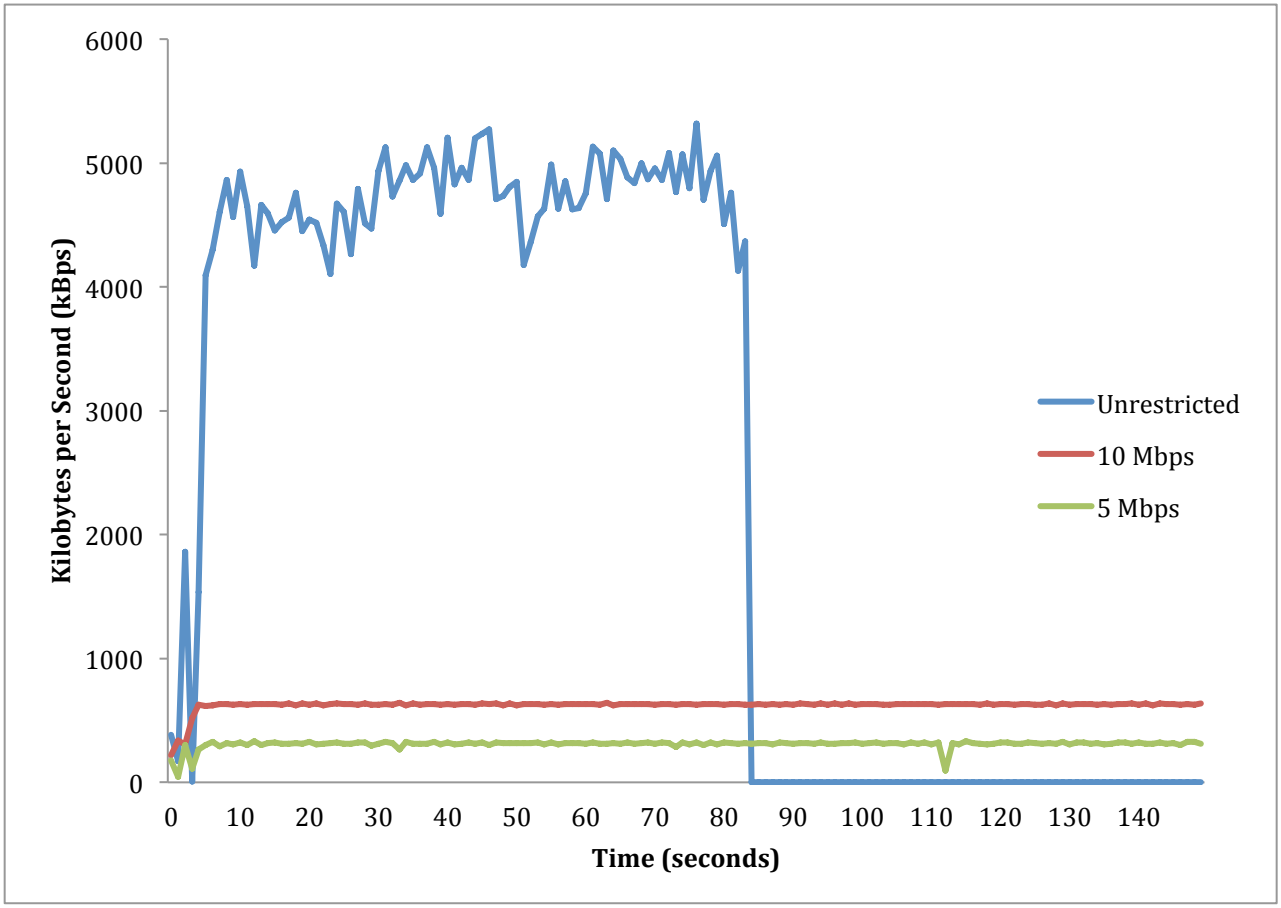


Figure 22: Vary Bandwidth- YouTube Downstream Kilobytes versus Time

5.4.2 Packet Loss and YouTube

The downstream bandwidth measurements for varying packet loss with YouTube are shown in Table 10. This table demonstrates that YouTube's use of bandwidth is extremely dependent on packet loss. In fact during the YouTube tests where there was packet loss, the video actually stopped playing about 90 seconds in.

	Packet Min	Packet Max	Packet Average	Packet STDEV	Min (kB)	Max (kB)	Average (kB)	STDEV (kB)
0% PL	0	3512	1670.1	1557.8	0.0	5317.2	2526.2	2359.3
1% PL	95	798	494.9	125.1	110.4	1208.2	747.3	191.8
1.5% PL	18	589	315.1	138.2	26.8	891.7	474.9	210.2

Table 10: Vary Packet Loss- YouTube Downstream

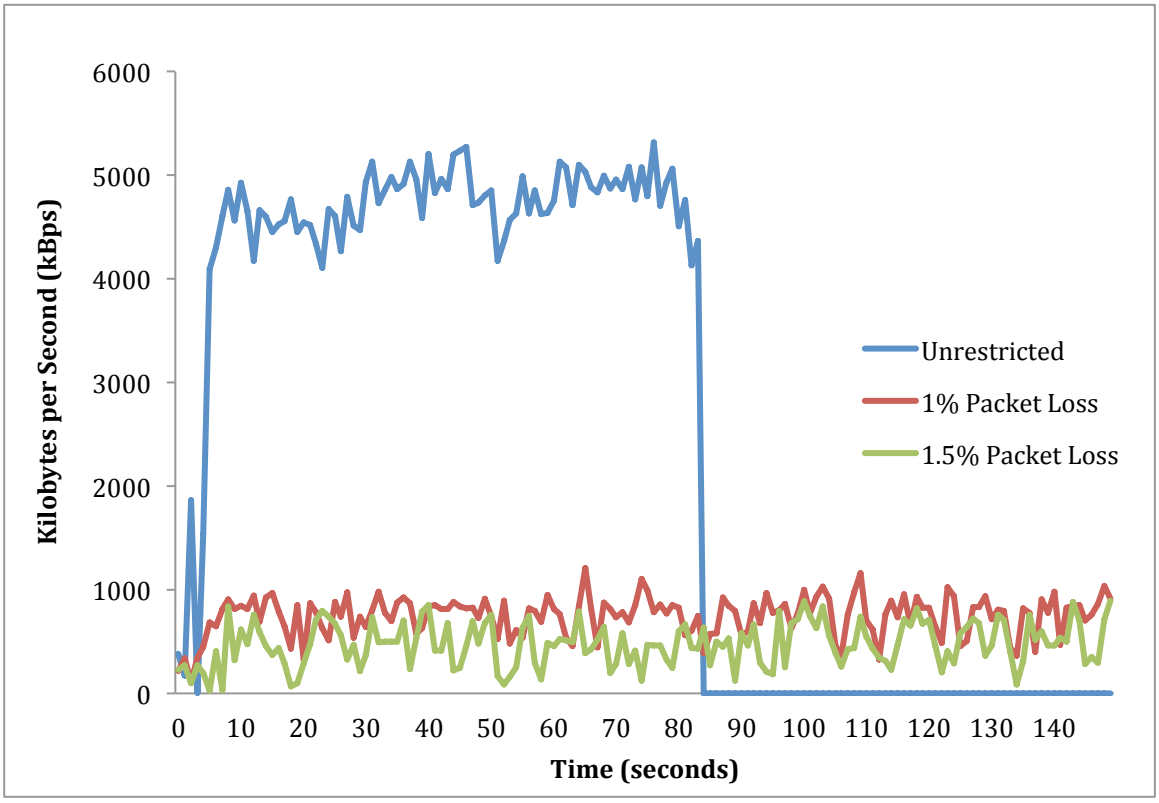


Figure 23: Vary Packet Loss- YouTube Downstream Kilobytes versus Time

The graph in Figure 23 shows the bandwidth of YouTube in kilobytes per second. The graph also supports the conclusion drawn from Table 10 that YouTube is extremely sensitive to packet loss.

5.4.3 Latency and YouTube

The downstream bandwidth measurements for varying latencies with YouTube are shown in Table 11. This table demonstrates that YouTube's use of bandwidth changes extensively depending on the latency in the system. As the latency increases the max kilobytes per second also increases but the average still remains similar.

	Packet Min	Packet Max	Packet Average	Packet STDEV	Min (kB)	Max (kB)	Average (kB)	STDEV (kB)
0 ms	0	3512	1670.1	1557.8	0.0	5317.2	2526.2	2359.3
20 ms	0	7619	1694.7	2400.3	0.0	11534.1	2563.3	3634.6
35 ms	0	8136	1672.0	2017.9	0.0	12280.9	2528.0	3052.9

Table 11: Vary Latency- YouTube Downstream

The graph in Figure 24 shows the YouTube downstream for varying latencies in Kilobytes per second. The graph supports the conclusion drawn from table 11 that YouTube's bandwidth changes extensively as the latency increases. The two lines representing the two tested latencies vary between 0 and up to 12000 kilobytes per second. The graph also indicates that YouTube has some sort of latency compensation in place to deal with varying network conditions.

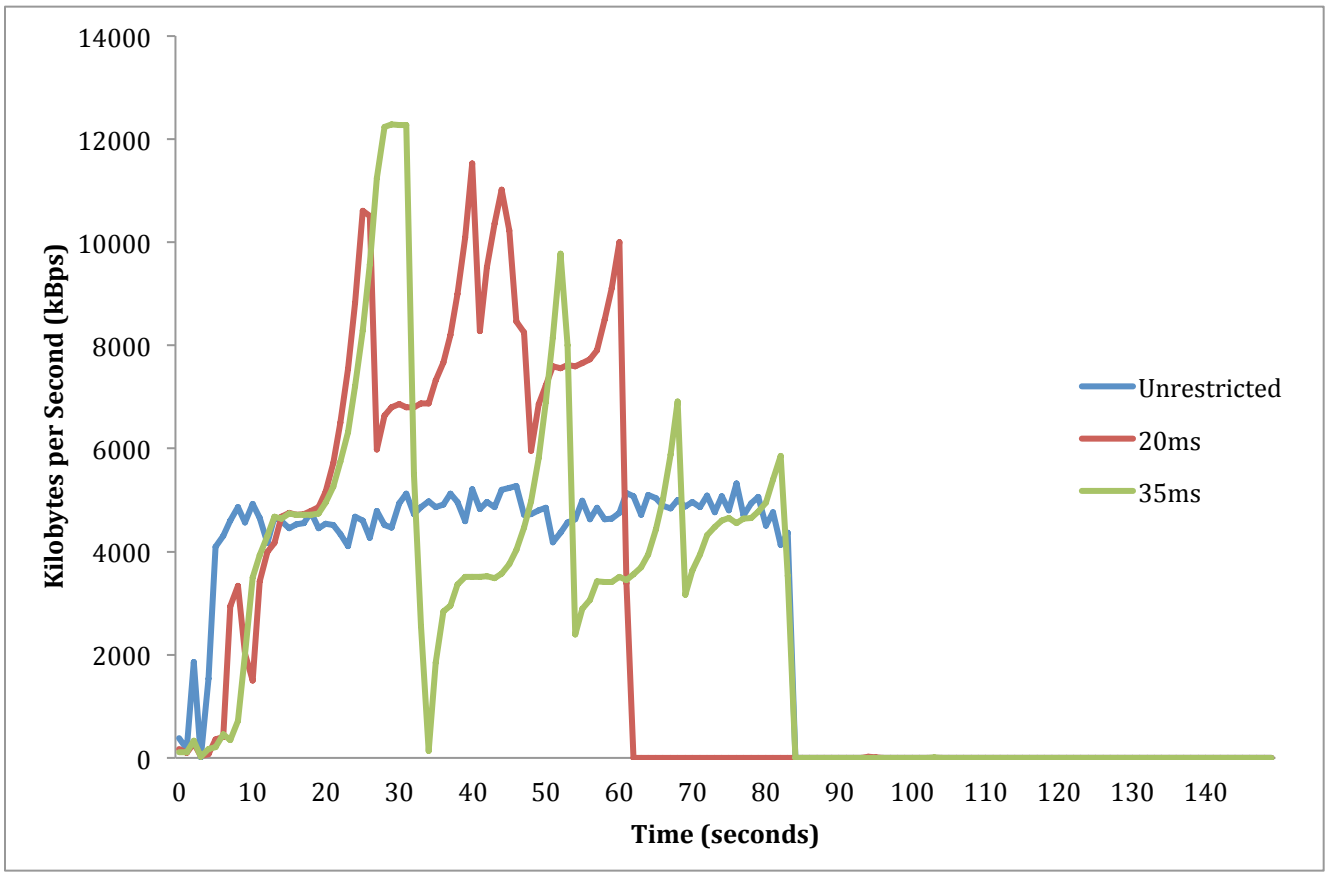


Figure 24: Vary Latency- YouTube Downstream Kilobytes versus Time

5.4.4 Bandwidth Restriction and Skype

The downstream bandwidth measurements with Skype are shown in Table 12. This table demonstrates that Skype is able to operate at many different bandwidths.

	Packet Min	Packet Max	Packet Average	Packet STDEV	Min (kB)	Max (kB)	Average (kB)	STDEV (kB)
Unrestricted	105	199	156.1	15.2	91.2	196.5	135.0	19.7
10 Mbps	126	202	156.5	12.2	97.8	195.9	135.0	16.5
5 Mbps	107	193	156.7	14.1	69.7	186.0	138.7	19.3

Table 12: Vary Bandwidth Skype Downstream

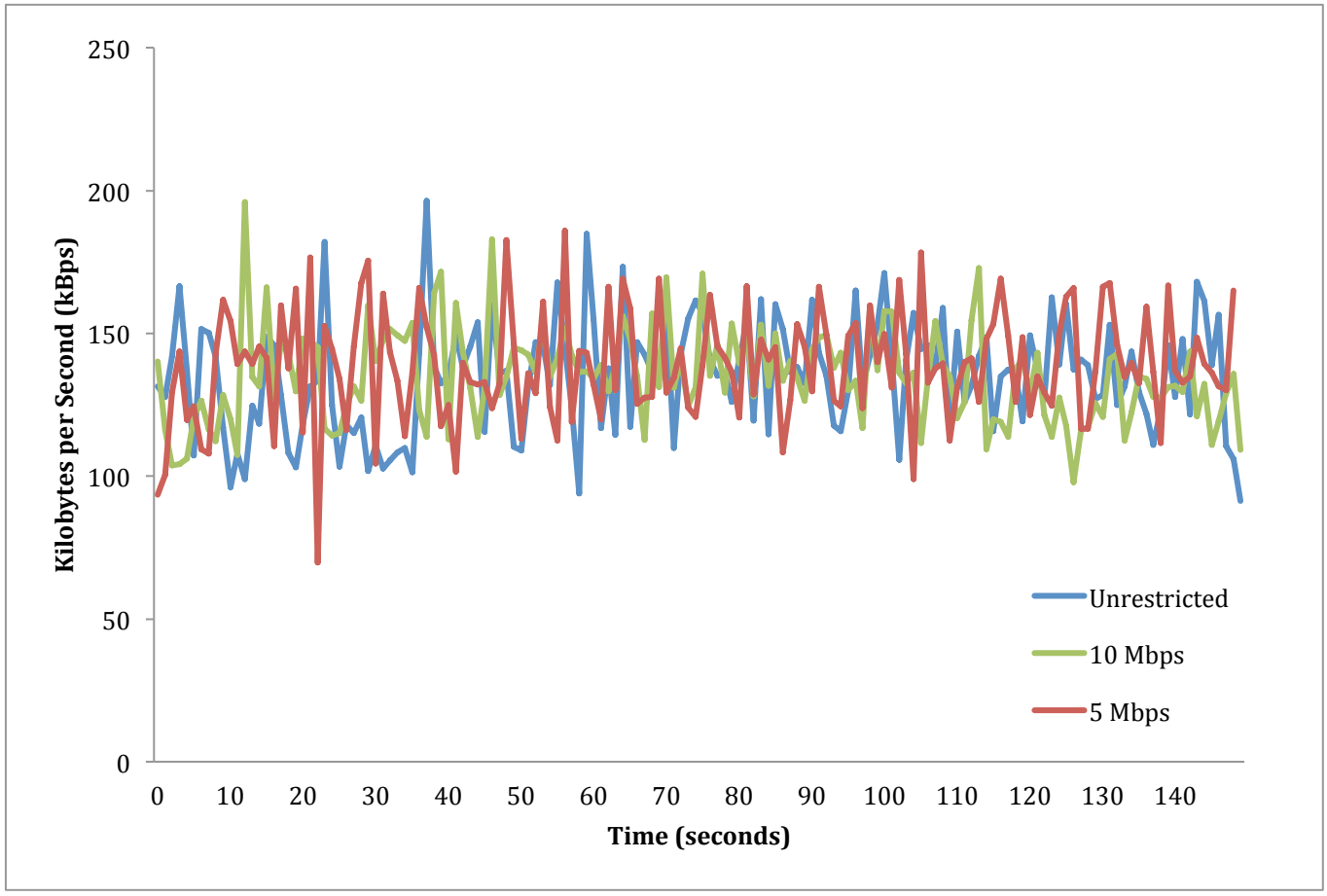


Figure 25: Vary Bandwidth- Skype Downstream Kilobytes versus Time

The graph in Figure 25 shows the Skype downstream in Kilobytes per second. It also supports the conclusion drawn from Table 12 that Skype is able to operate at many different bandwidths and that it is consistent across all of the bandwidths.

5.4.5 Packet Loss and Skype

The downstream bandwidth measurements for Skype with varying packet loss rates are shown in Table 13. The table shows that as the packet loss rate increases so does the bandwidth used by Skype.

	Packet Min	Packet Max	Packet Average	Packet STDEV	Min (kB)	Max (kB)	Average (kB)	STDEV (kB)
0% PL	114	198	159.3	14.0	97.8	189.8	133.1	17.3
1% PL	149	236	187.3	17.1	131.8	249.1	182.8	23.3
1.5% PL	143	273	192.5	26.5	121.7	299.6	190.2	36.5

Table 13: Vary Packet Loss Rates Skype Downstream

The graph in Figure 26 shows the Skype downstream measurements across varying packet loss rates in kilobytes per second. The graph supports the conclusion drawn from Table 13 that, as the packet loss rate increases the bandwidth also increases.

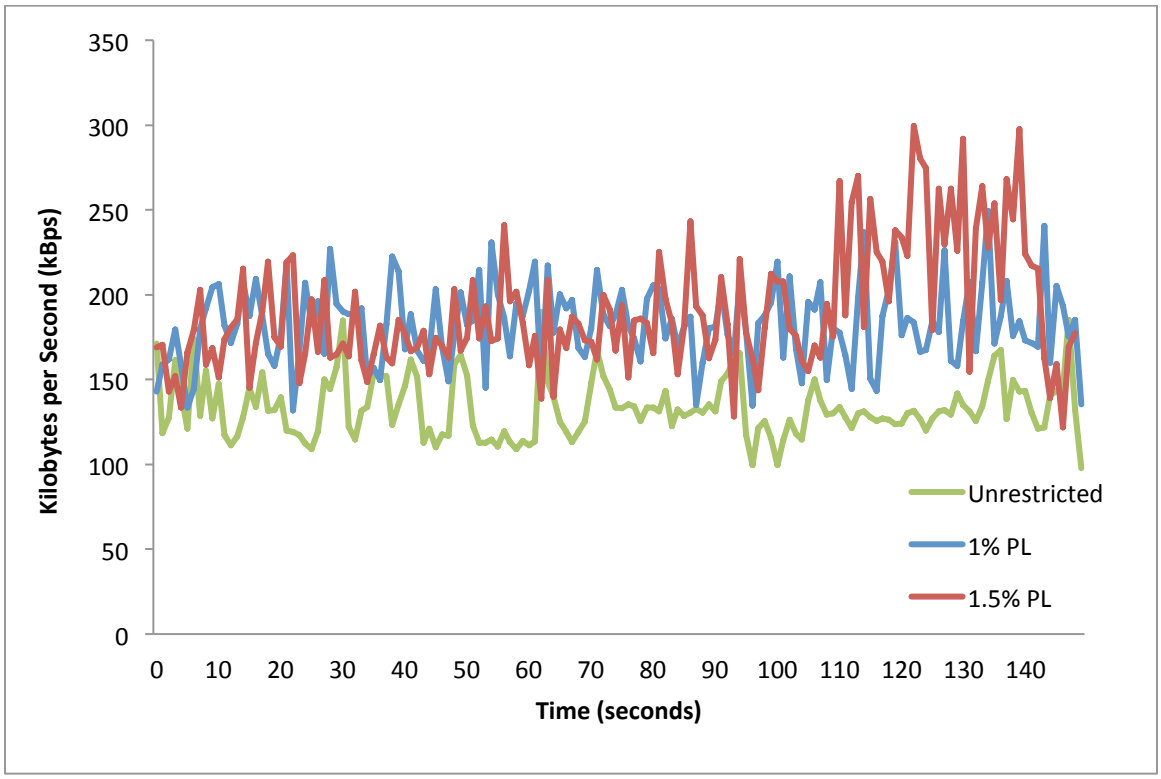


Figure 26: Vary Packet Loss Rates- Skype Downstream Kilobytes versus Time

5.4.6 Latency and Skype

The downstream bandwidth measurements for Skype across varying latencies are represented in Table 14. The average packets per second and kilobytes per second indicate that Skype is extremely resistant to latency.

	Packet Min	Packet Max	Packet Average	Packet STDEV	Min (kB)	Max (kB)	Average (kB)	STDEV (kB)
0 ms	105	199	156.1	15.2	91.2	196.5	135.0	19.7
20 ms	116	208	154.4	15.3	84.1	209.6	133.1	20.7
35 ms	116	196	157.2	15.1	78.2	188.5	134.4	20.8

Table 14: Vary Latency- Skype Downstream

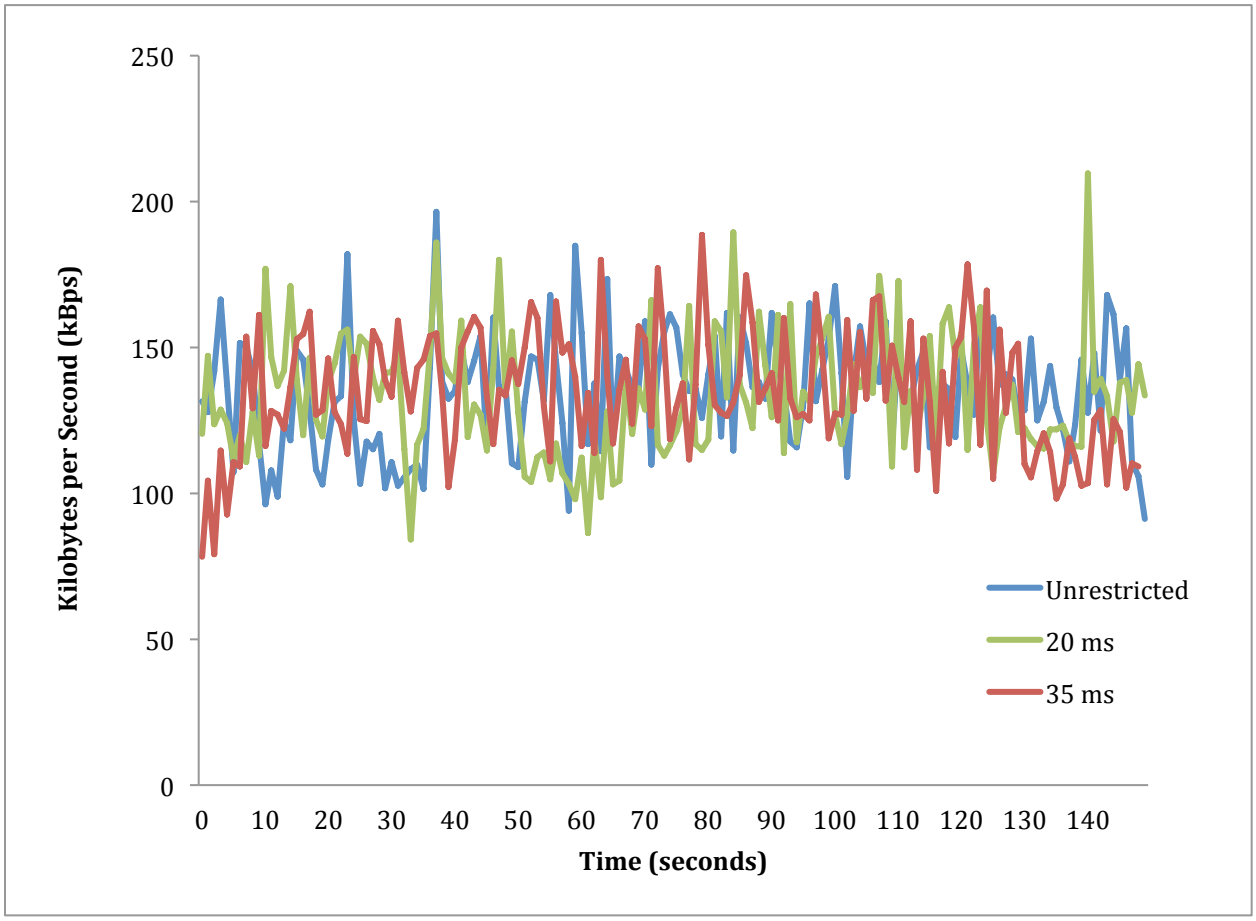


Figure 27: Vary Latency- Skype Downstream Kilobytes versus Time

The graph in Figure 27 shows the downstream bandwidth for Skype across varying latencies in kilobytes per second. This graph also demonstrates that Skype is extremely resistant to latency variation.

5.5 Overall Results

5.5.1 OnLive Individual Game Bandwidth

One of the major results from our experiments is that the downstream bandwidth of OnLive varies depending on the game being played while the upstream bandwidth remains constant. The graph in Figure 28 shows the overall downstream and upstream bandwidths used by the 3 OnLive games tested. This graph reveals that the Grand Ages: Rome used considerably less bandwidth than the other two games, Unreal Tournament and Batman.

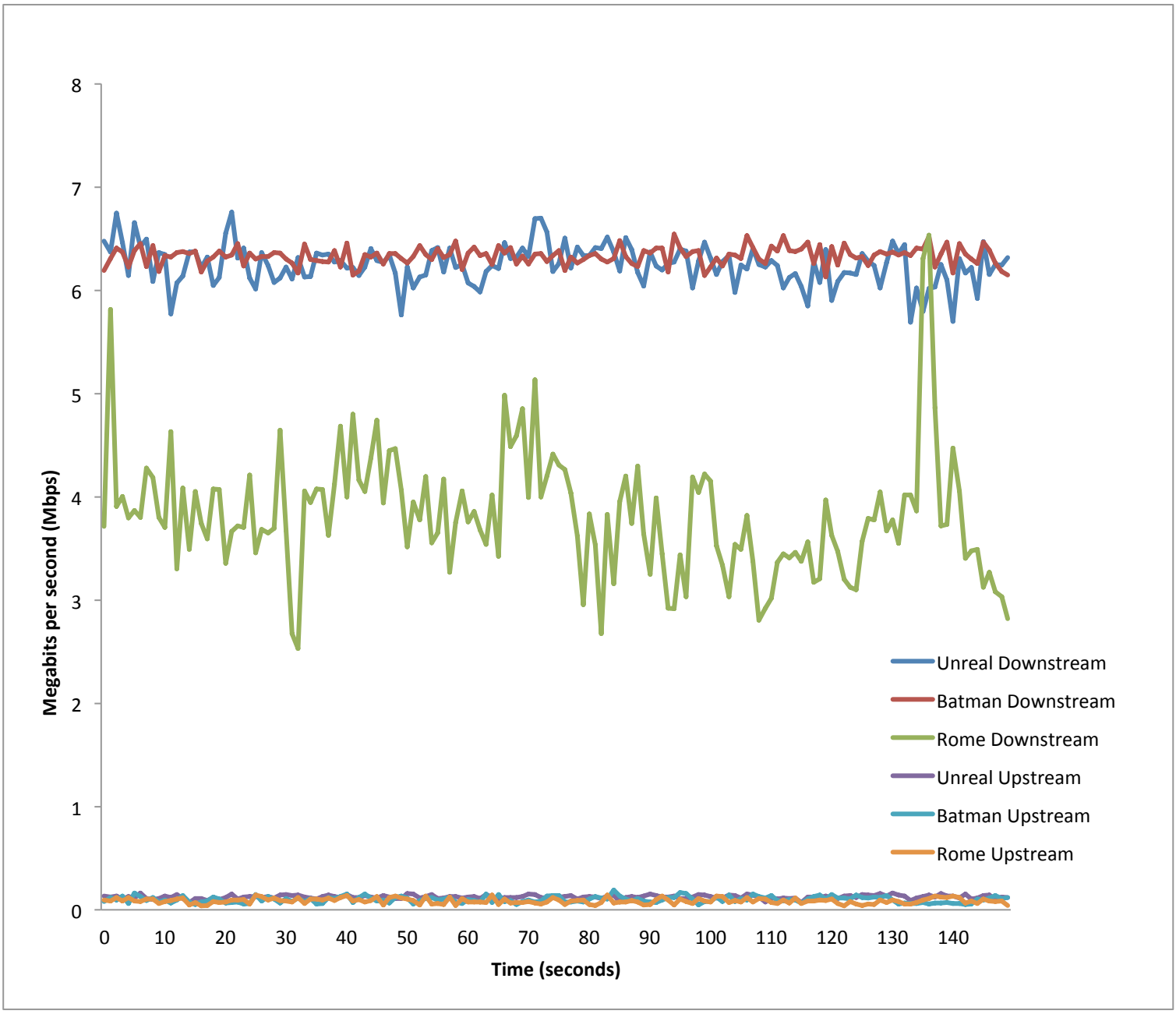


Figure 28: 3 OnLive Games Upstream and Downstream: Megabits vs Time

5.5.2 Frame Rate Measurements During Network Variations

During the DummyNet tests for OnLive FRAPS data was collected that allowed us to measure the frame rates of the OnLive session being played. The graph in Figure 29 show the variations in frame rate across the varying network conditions. This graph indicates that the frame rates of the OnLive game will change depending on the condition of the network. This is important because it reveals that OnLive must do some network analysis of its own in order to modify the frame rates of the game.

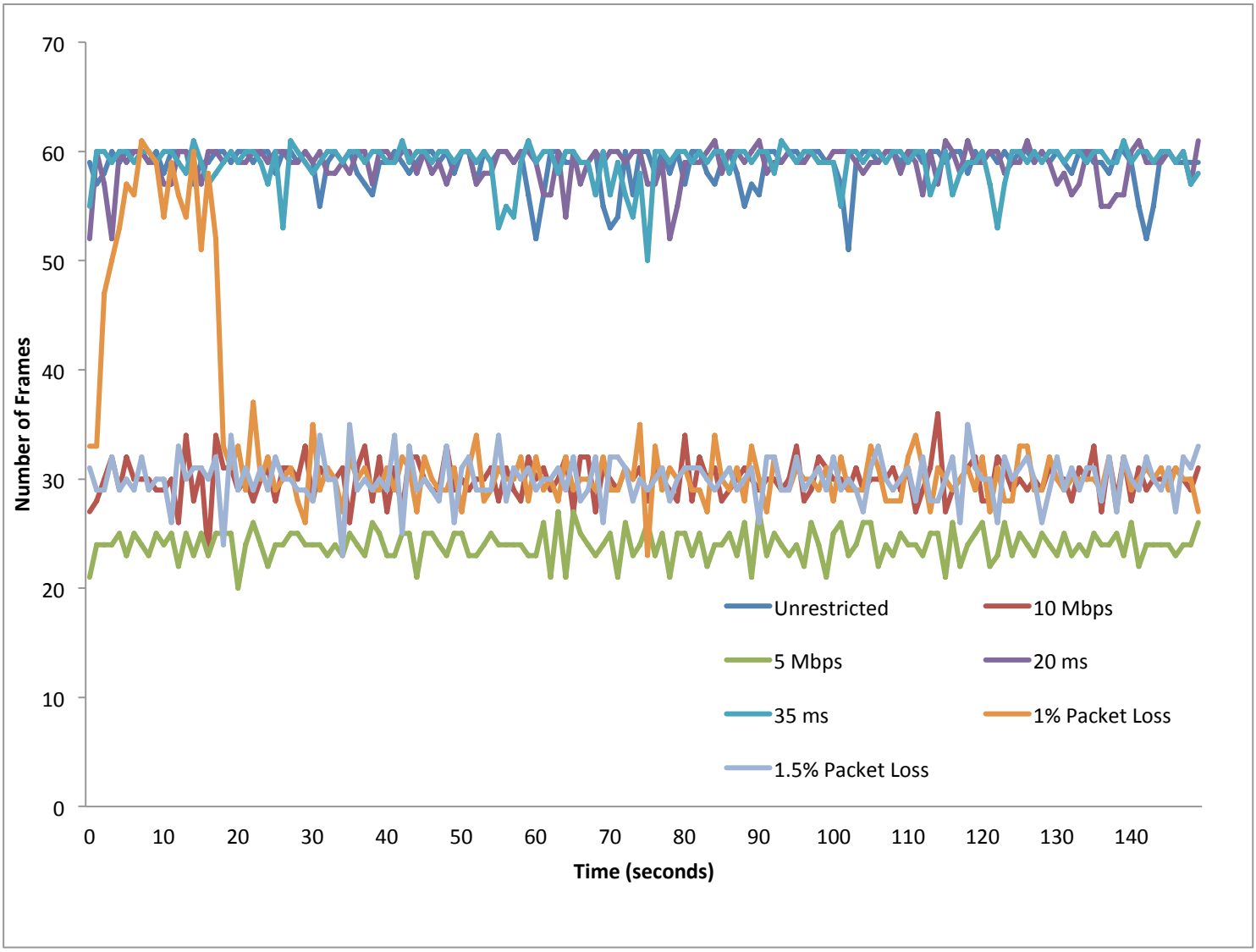


Figure 29: Network Restrictions in Unreal Tournament 3: Frames per Second

6 Conclusion

Throughout the world today, cloud computing is changing the way people use their home Internet connection. With new technologies like OnLive utilizing and depending on higher quality Internet connections, it is imperative that both companies and their customers understand the basics of how these technologies work and what resources they require. As stated in Chapter 1 our goal for this project was to analyze the underlying network characteristics and performance of the OnLive gaming service. By reaching this goal, the project also hopes to provide potential and current customers of OnLive with a basic understanding of how the system utilizes the Internet Connection that they are paying for.

To meet this goal we decided to initially study the basic network performance metrics of OnLive. After this initial investigation, we realized that a majority of the bandwidth used by OnLive is for audio and video data. This revealed to us that OnLive is essentially a video streaming service that shares characteristics with other video streaming services like Skype and YouTube. Upon coming to this realization we furthered our analysis by comparing our initial results with results from a basic network study of Skype and YouTube.

Based on our results and comparisons we have reached the following conclusions about OnLive:

- The network footprint of the OnLive gaming service varies depending on the game being played.
- OnLive as a video streaming service is fundamentally different than Skype and YouTube.
- OnLive modifies properties of the game depending on the type of Internet service available.

6.1 Basic Network Characteristics of OnLive

The first part of our testing began with a basic investigation of the network behind the OnLive gaming service. One of the major conclusions that we have drawn from these initial tests is that OnLive downstream bandwidth varies depending on the type of game being played while OnLive upstream bandwidth is consistent across all three types of games.

The first part of that conclusion is clearly demonstrated by the graph in Figure 28. Overall the data seems to suggest that first person shooter games require many frame updates per second and therefore need more bandwidth to perform. Conversely real-time strategy games like

Grand Ages: Rome, do not require the same kinds of frame updates and therefore do not need as much bandwidth.

In contrast, the upstream bandwidth measurements across all three games were extremely similar. This seems suggests that OnLive is designed to send client responses independently of the game. In other words, regardless of the game being played, OnLive will always send constant client commands to the server.

6.2 OnLive and Other Video Streaming Services

The next part of our testing investigated the basic network characteristics of two other popular video services, Skype and YouTube. When compared with the results from the initial OnLive tests, these tests indicate that OnLive is drastically different from Skype and YouTube. Overall, YouTube and Skype had the highest upstream bandwidths and YouTube had the largest downstream bandwidths (Figures 11 &12).

Skype's large upstream bandwidth is most likely due to the fact that Skype was tested as a two-way video service. In other words, video was not only being received but also sent. This idea is also supported by the fact that Skype's up and down streams were nearly symmetrical at 102KBps and 140KBps (Tables 4 & 5).

Additionally, YouTube's large downstream is most likely due to the design of its infrastructure. In other words, YouTube's popularity has allowed the service to utilize some of the most cutting edge technologies available. As stated in our background section, YouTube streams over 3 Billion videos per day. In order to maintain such a service, YouTube has to ensure that its content is available in many locations for optimized speed. With WPI's large Internet connection and peering with other providers, it is no surprise that YouTube's downstream was able to reach the level that it did.

Lastly, the content provided by YouTube differs from OnLive because it is prerecorded content. Unlike Skype and OnLive, YouTube provides content that has already been optimized for their particular service. Overall this allows YouTube to have much more control over the characteristics of the video being streamed. Additionally, once the video is downloaded to the browser cache, the bandwidth drops drastically. OnLive's bandwidth doesn't drop because the content is being created and streamed almost instantaneously. This is similar to Skype's video streaming, but OnLive uses much more downstream bandwidth (Figure 11).

6.3 OnLive and Network Variation

After our initial network investigation of OnLive, we investigated how OnLive handles variations in the network. In summary, OnLive was able to provide a consistent and playable gaming experience across many network conditions.

One of the first network characteristics to indicate this conclusion was bandwidth. Across the United States today, there are many different types of Internet connections available. In order for OnLive to become such a popular service, it must be able to deal with these varying connections and bandwidth speeds. Our results in Figure 13 support this conclusion.

Packet loss is another network characteristic that varies greatly depending on the Internet connection available. Our results from Section 5.3.2 demonstrate that OnLive is also able to deal with variations in packet loss.

Lastly, OnLive must also handle variations in the latency of an Internet connection. Our results from Section 5.3.3 demonstrate that OnLive is able to again provide a consistent and playable gaming experience even across these varying latencies.

6.4 Overall

In summary, OnLive is not a typical video streaming service. Our results indicate that OnLive network conditions differently, but at least as effectively as Skype and YouTube. The results from our FRAPS data demonstrate that while OnLive's network metrics such as packet size and interpacket time may vary, the frame rates are either similar or at the minimum high enough to provide a user with a playable game.

7 Future Work

OnLive pushes the boundaries of cloud computing and gaming. At the end of 2011, OnLive released an OnLive application for iOS and Android systems but unfortunately our project only focused on the console and PC applications of OnLive[25]. As unexplored as OnLive is, the iOS and Android versions of OnLive are even more enigmatic. Comparing OnLive with a 3G or 4G connection versus a Wi-Fi connection would be an interesting concept to explore.

Comparing and contrasting the tablet, phone, MicroConsole, and PC versions of OnLive may yield very interesting results. This would be helpful for potential customers to decide which OnLive system would best suit their needs. Skype and YouTube also have applications for each of the platforms mentioned. This project could be extended easily to include the tablet and phone applications of OnLive, Skype and YouTube.

Another aspect of OnLive is the OnLive Desktop application. Released in early 2012, OnLive Desktop lets users access a powerful PC from an iOS or Android tablet. [23] Instead of using OnLive's system to access popular video games, OnLive Desktop gives users the ability to access a PC with Microsoft Office on a Windows 7 environment. Users can even browse the Internet and use the Adobe Flash player, an uncommon feature on most tablets. This application is completely unexplored and the performance testing of an application like this versus a PC or laptop with Microsoft would most definitely produce interesting results. OnLive Desktop could prove to be faster and more powerful than some laptops or netbooks. This would have a great effect on the consumer, because it would be cheaper and more portable to buy a tablet and use OnLive Desktop than to buy a laptop or netbook.

An idea originating from the results of our experiments and our original hypotheses would be the testing of other games on OnLive. There are many different genres of games and OnLive has a multitude of games to choose from, ranging from old to new. A larger study could build off of our original games and capture a vast amount of data from a much larger array of games. The limitation of the single player experience can be researched as well because some games on OnLive have a multiplayer aspect. The comparison between all the games would be an intriguing research venture. There are many different categories to explore – Onlive offers games from many different genres including[24]:

[23] http://desktop.onlive.com/

- Action
- Adventure
- Casual
- Classic
- Family
- Fighting
- Horror
- Indie
- Platform
- Puzzle
- RPG
- Racing
- Shooter
- Simulation
- Sports
- Strategy

With 261 games, a comprehensive study on all the downstream and upstream packet captures of each game would prove useful to OnLive. OnLive would be able to view which games are the most taxing on their systems or which games are affected the most by network instability.

Although many of the future work ideas have been technical, there is a user-level aspect to OnLive that has yet to be explored. Focus groups and user studies could be used to determine the quality of experience of OnLive versus other gaming consoles. Playing the same games on the OnLive console compared to the same game played on an XBOX 360, Playstation 3, Nintendo Wii, or PC would give valuable information on a typical consumer's attitude towards OnLive against more popular gaming consoles/mediums.

Works Cited

[1] Jason Nieh, Naomi Novik, S. Jae Yang, A Comparison of Thin-Client Computing Architectures: Columbia University, November 2000. Online at: http://www.nomachine.com/documentation/pdf/cucs-022-00.pdf

[2] OnLive Technical FAQ, OnLive, http://www.onlive.com/support/performance, Retrieved: January 2012.

[3] Rivka Tadjer. What is Cloud Computing?, PC Magazine, http://www.pcmag.com/article2/0,2817,2372163,00.asp, November 18, 2010.

[4] Dave Perry. Taking Gaming into the 'Cloud', BBC, http://news.bbc.co.uk/2/hi/programmes/click_online/8085937.stm, June 9, 2009.

[5] Chris Roper. GDC 09: OnLive Introduces The Future of Gaming. Next-Generation "Cloud" Technology Could Change Videogames Forever, IGN, http://pc.ign.com/articles/965/965535p1.html, March 23, 2009.

[6] Rich Brown. Interview: OnLive CEO Steve Perlman gives us his post-launch perspective, CNET, http://news.cnet.com/8301-17938_105-20010687-1.html, July 15, 2010.

[7] Ernest Adams, Andrew Rollings. Andrew Rollings and Ernest Adams on Game Design, Pearson, 1st Edition, May 11, 2003.

[8] Bruce Geryk. A History of Real-Time Strategy Games, GameSpot, March 31, 2008.

[9] Skype Grows FY Revenues 20%, Reaches 663 Min Users, TelecomPaper, http://www.telecompaper.com/news/skype-grows-fy-revenues-20-reaches-663-mln-users, March 8, 2011.

[9] Jennifer Caukin. A Day In The Life of Skype #Infographic, Skype: The Big Blog, http://blogs.skype.com/en/2011/09/a_day_in_the_life_of_skype_inf.html, September 6, 2011.

[10] Salman A. Baset and Henning G. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol, Columbia University, http://www.cs.columbia.edu/~salman/publications/skype1_4.pdf, September 15, 2004.

[11] Skype Moves to VP8 for All Video Calls, The H, http://www.h-online.com/open/news/item/Skype-moves-to-VP8-for-all-video-calls-1318315.html, August 4, 2011.

[12] General Questions, The WebM Project, http://www.webmproject.org/about/faq/, Retrieved: April 4, 2012.

[13] Statistics, YouTube, http://www.youtube.com/t/press_statistics, Retrieved December 14, 2011.

[14] Vijay Kumar Adhikari, Sourabh Jain, Yinyin Chen, and Shi-li Zhang, Vivisecting YouTube: An Active Measurement Study, University of Minnesota, July 11, 2011. Online at: http://www.cs.umn.edu/tech_reports_upload/tr2011/11-012.pdf

[15] YouTube HTML5 Video Player, YouTube, http://www.youtube.com/html5, Retrieved: April 9, 2012.

[16] Talhah Asharaf, Kelvin Leung, Xiang Liu. OnLive Cloud Gaming Service, May 2011. Online at: http://www.sjsu.edu/people/rakesh.ranjan/courses/cmpe272/s1/Team%20WS%20OnLive%20Cloud%20Gaming%20Service.pdf

[17] Kuan-Ta Chen, Yu-Chun Chang, Po-Han Tseng, Chun-Ying Huang, and Chin-Laung Lei, "Measuring The Latency of Cloud Gaming Systems," In *Proceedings of ACM Multimedia 2011*, Nov 2011.

[18] Sheng-Wei (Kuan-Ta) Chen, Yu-Chun Chang, Po-Han Tseng, Chun-Ying Huang, Chin-Laung Lei. Cloud Gaming Latency Analysis: OnLive and StreamMyGame Delay Measurement, http://www.iis.sinica.edu.tw/~swc/onlive/onlive.html, April 4, 2012.

[19] Tom Beigbeder, Rory Coughlan, Corey Lusher, John Plunkett, Emmanuel Agu, and Mark Claypool. The Effects of Loss and Latency on User Performance in Unreal Tournament 2003, In *Proceedings of ACM Network and System Support for Games Workshop (NetGames)*, Portland, OR, USA, September 2004. Online at: http://www.cs.wpi.edu/~claypool/papers/ut2003/

[20] Ted Lemon. dhcpd.conf, linuxmanpages.com, http://linuxmanpages.com/man5/dhcpd.conf.5.php, Date Accessed: December 11th, 2011.

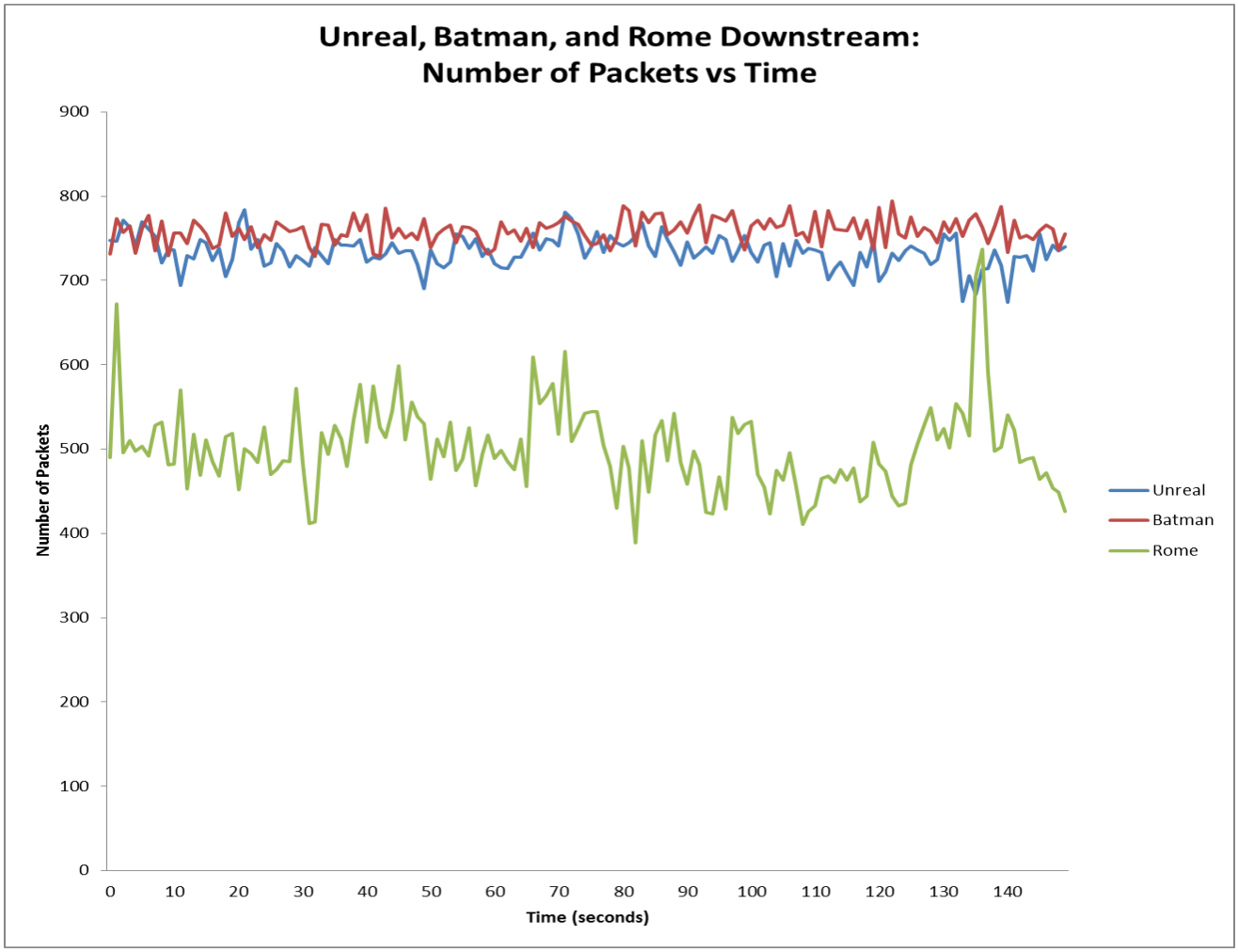
[21] FreeBSD Handbook: Chapter 31 Firewalls, freebsd.org, http://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/firewalls-ipfw.html, Date Accessed: December 11th, 2011.

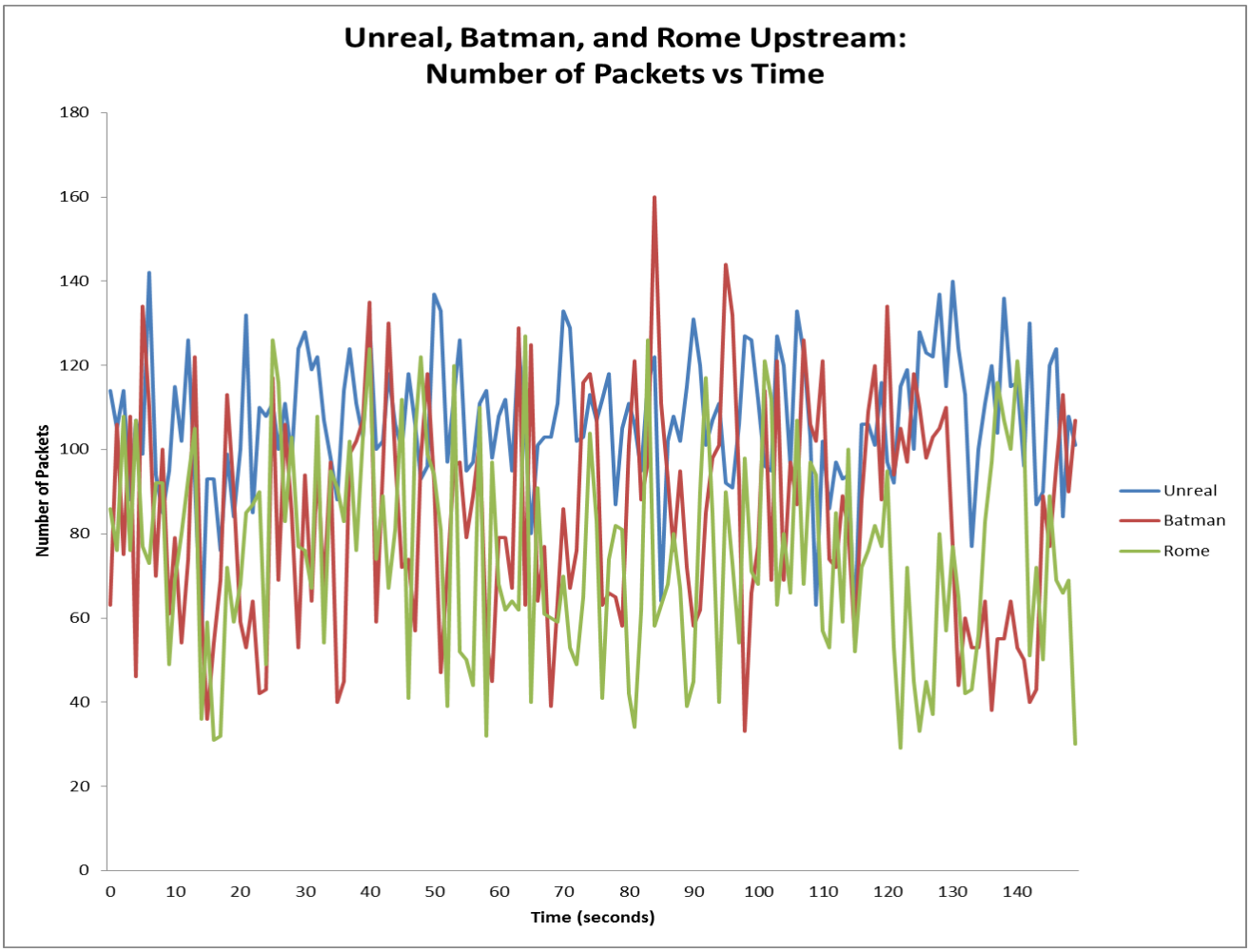
[22] .PCAP File Extension, fileinfo.com, http://www.fileinfo.com/extension/pcap, July 5th, 2011.

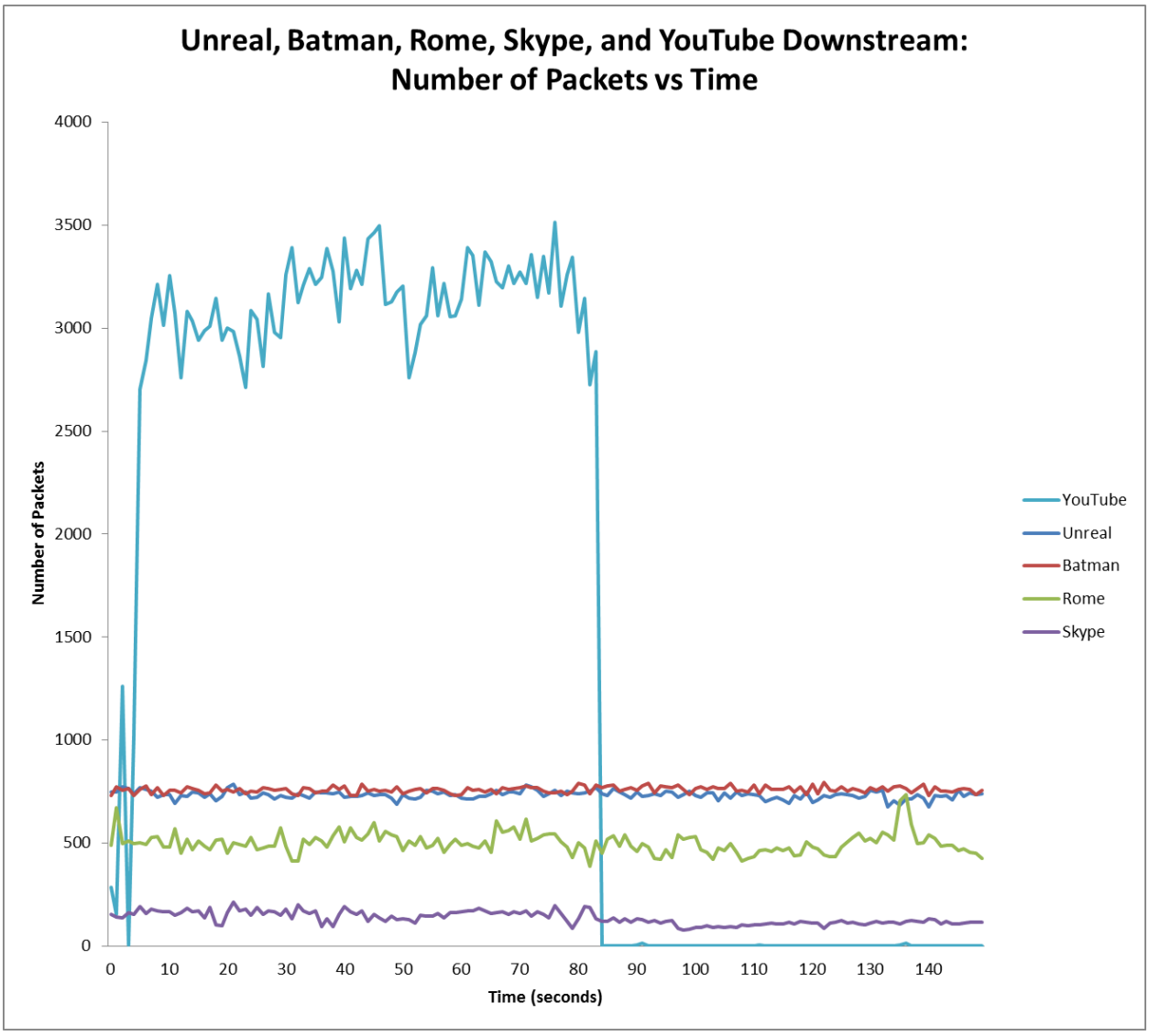
[23] Mike Schramm. OnLive releases iOS and Android apps, custom touch controls, and a wireless controller, joysitq.com, http://www.joystiq.com/2011/12/07/onlive-releases-ios-and-android-apps-custom-touch-controls-and/, December 7, 2011.

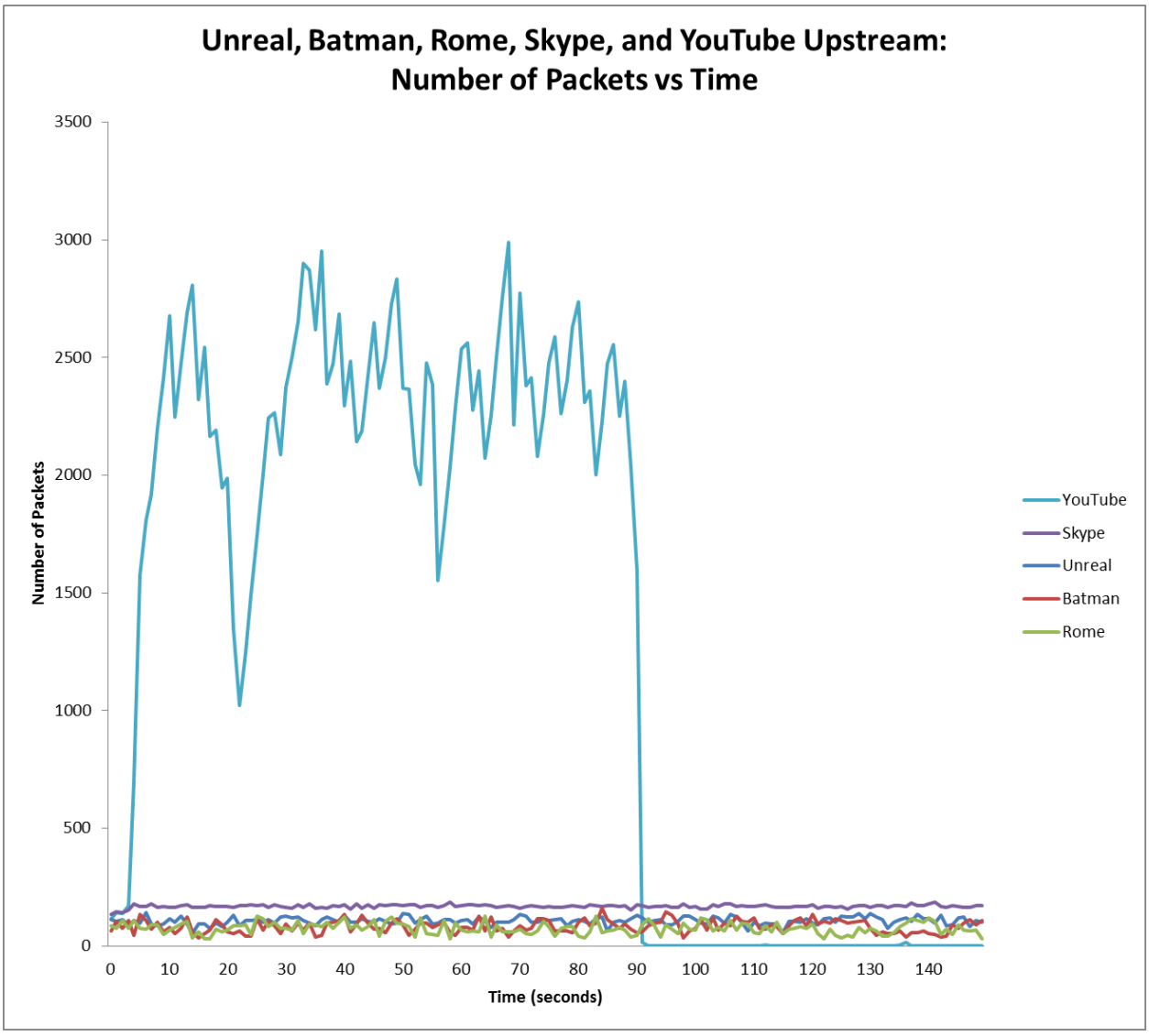
[24] OnLive. Games, OnLive, http://www.onlive.com/games/featuredgames#&tab=all_games&per_page=252, Retrieved April 10, 2012.

Appendix


Unreal, Batman, and Rome Downstream:
Number of Packets vs Time
Number of Packets
900
800
700
600
500
400
300
200
100
0
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
Time (seconds)
Unreal
Batman
Rome


Unreal, Batman, and Rome Upstream:
Number of Packets vs Time
Number of Packets
Time (seconds)
0
20
40
60
80
100
120
140
160
180
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
Unreal
Batman
Rome


Unreal, Batman, Rome, Skype, and YouTube Downstream:
Number of Packets vs Time
Number of Packets
4000
3500
3000
2500
2000
1500
1000
500
0
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
Time (seconds)
YouTube
Unreal
Batman
Rome
Skype


Unreal, Batman, Rome, Skype, and YouTube Upstream:
Number of Packets vs Time
Number of Packets
3500
3000
2500
2000
1500
1000
500
0
0
10
20
30
40
50
60
70
80
90
100
110
120
130
140
Time (seconds)
YouTube
Skype
Unreal
Batman
Rome