Scalable Video-on-Demand with Edge Resources

Ryan Kashi
Kannan Ramchandran, Ed.

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2016-84
http://www.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-84.html

May 13, 2016

Capstone Report

Scalable Video-on-Demand with Edge Resources

Ryan Kashi

Table of Contents

Chapter 1

Technical Contributions

1 Introduction

Our capstone project, Scalable Video on Demand, aims to create a video streaming platform by using a hybrid peer to peer software architecture. A platform of this structure would be the first of its kind, and can theoretically provide video services similar to that of Netflix and Hulu at an extremely reduced cost. Users who would like to use our platform would simply have to connect to run our application and choose a video to watch. The user will be seamlessly connected to a multitude of caches, which can be thought of as nodes that choose to participate in the distribution of video content, and begin watching the video that he or she selected.

The video distribution problem is not a simple problem to solve. When subject to limited storage space, bandwidths, and varying demands, watching a simple video gets complicated extremely fast. We are building our platform off of a prototype that implements a heuristic designed to solve for an approximate solution to this problem.

2 System Architecture

By using the distributed algorithm described in "An Optimized Distributed Video-on-Demand Steaming System: Theory and Design," portions of videos are distributed to the optimal caches (participants who choose to act as a storage device for our system) that meet the video demand requirements (Lee 2012). The system that we are building is built from a prototype designed earlier that exhibits most of the core functionality of the hybrid system on local machines. The project goal is to expand upon this prototype and allow for domestic use, such that our system can be

marketed and used as a commercial product. Thus, a lot of polishing and increasing code robustness must be done to improve the prototype.

Our platform consists of four main components. These four components are the tracker, server, caches, and users. This paper will describe a basic approach to how these four components interact. A more in-depth description of the overview of our project can be seen in Jiayuan Chen's paper.

2.1 Tracker

The tracker keeps track of the IP address of each cache and the server. Its core functionality is to tell users which addresses they must connect to in order to download videos. Caches, users, and the server also relay additional information to the tracker about their states. The tracker behaves like a website, and can be connected to via any internet browser to view its contents. Because of this, the tracker is also used as a homepage for the system, where one is able to view the status of the system as whole.

2.2 Server

The server contains all of the video contents in an erasure coded format. By erasure coding each video, we are essentially turning one large file into a number of smaller, chunked files. The server distributes these video chunks to the caches based on the demand of each video and the limitations of the cache. Lastly, the server sends a user any additional video chunks that they may require for video reconstruction.

2.3 Users

Users are the ones who are currently downloading a specific video from the system. Users will begin by requesting to watch a video, at which point they will be connected to the server and a number of caches. Users will begin to accumulate chunks from the connected caches to reconstruct a small portion of the video. If any additional chunks are required for this reconstruction, they will download the remaining chunks from the server. This process completes until a user disconnects or the requested video is fully reconstructed.

2.4 Caches

Finally, caches behave as additional nodes that users are able to download their video content from. The purpose of having caches is to reduce the bandwidth load on the server. Depending on the state of the system, the caches can reduce the bandwidth usage of the server to near 0%. A person connected to the system as a cache will receive chunks from the server, which it will then distribute out to a number of users. If a person enters the system as a cache, they will also accumulate points which can then be spent to download videos from the system as a user. This point system is implemented in order to give people an incentive to contribute to our system. The more they contribute, the more videos they are able to watch on our system.

3 List of Associated Tasks

Due to the large scope of our project and small size of our team, portions of our tasks and accomplishments are intertwined. This, however, has been used as a strength, allowing for agile design flows and processes. Each of us have contributed a number of accomplishments categorized under one of these five subjects. The list of subjects and their approximate order of operations for our project is seen in Figure 1.

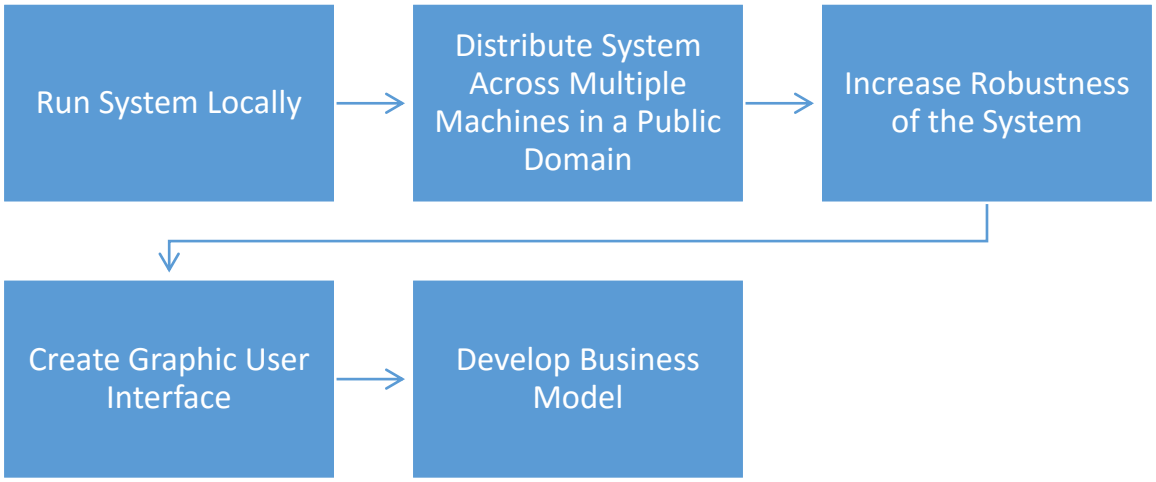

Figure 1: Project Checkpoint Diagram

4 List of Accomplishments

Over the course of the year, I have accumulated a list of accomplishments that have greatly directed us toward a commercially viable product. My list of major accomplishments and contributions is seen below.

4.1 Allow Users to Dynamically Change and Access Tracker, Server, and Caches on Public Domain Names

The initial prototype of our code was using hard-coded values for all server, tracker, and cache addresses. The components themselves were fixed onto localhost (can only be accessed by the machine that is running the code. Essentially, the server is not being broadcasted to the public). This poses as an extremely large issue when trying to run even just a portion of the service (i.e. tracker) publically. Every single user would have to manually edit the code specifically for themselves if they would like to run the code, recompile it, and then begin execution. Instead, a user is able to provide their address as input to simply run or connect to the system. The address of the tracker now acts as the "public domain name" for the system. This means that regardless of server location and connected caches, all a user needs to connect to the service is the tracker address. If a user wants to test the system locally, all changes in addresses are handled with no hassle to the user. These changes are essential for being able to run the system publically.

In addition, the original framework could only support IPv4 addresses. This means that no IPv6 addresses could connect to our service or be used for testing purposes, regardless of an internet

service providing dual stack support. This poses as an extremely large issue for scaling the system up. There are no more new IPv4 addresses being created, and in the future, all addresses will be IPv6. With my modifications to the platform, we have been able to upgrade to a much newer version of the framework, Web.py, to support IPv6 addresses. It was not until this point that we could run the system publically and have users across the world connect to it. Now, users on both IPv4 and IPv6 can use our system. Due to the differences in protocol, however, an IPv4 user cannot connect to IPv6 trackers, servers, or caches unless their internet service provider gives them dual stack support.

4.2 Ensure that Caches and Users are Successfully Disconnected from the System Under all Circumstances

Prior to this added functionality, a cache would not be able to notify the tracker that it was no longer a part of the system. This would cause all of the users connected to this cache to suddenly abort and crash due to a broken pipe error. In addition, a disconnected cache would cause the tracker to direct new users to create a connection with an imaginary cache, resulting in the crashing of the user. Thus, it was extremely important to fix this problem.

This was fixed by creating a new ftp handler inside of the server that was used to send over port data about connected caches and users. This information is then stored inside of a Python dictionary where the key is the server port that corresponds to this ftp connection. A built in handler part of the ftpdlib version 1.4.0 called on_disconnect is ran each time a port is closed unexpectedly, with the disconnected server port given as an argument. Because this argument is also the key to

the Python dictionary, we are able to successfully map this information to the port number and IP address that are listed in the tracker, allowing for a clean and safe removal from the tracker. A slightly different series of instructions are taken for a disconnected user and cache.

In addition to the changes on the server, the user code was fixed such that it will catch the error caused from a broken pipe, and safely disconnect itself from the tracker. Once this is done, the user will retain the frame number that it was currently watching and reconnect to the system once more.

4.3 Migration from Flash Video File Format to Matroska Multimedia Container Format for Stored Videos

In the prototype build of our project, all stored videos were required to be a flash video file (.flv), which are not nearly as scalable as a container format such as MPEG-4 (.mp4) or Matroska (.mkv). One cannot make a serious video streaming platform without supporting at least one of these two file formats. Matroska was chosen due to the fact that the metadata of the .mkv files is stored at the beginning of the file itself. This means that one can watch a 1080p Matroska video while the file is still being downloaded (streamed) to the user. With my changes, however, .mp4 can also be used via the implementation of the metadata reader added to the system. We, however, have chosen to stick with Matroska because it is open sourced and is overall a better video format for computers. The metadata of the videos are ripped using a python wrapper for MediaInfo called pymediainfo. MediaInfo is a command line interface that can return metadata such as “video bitrate” or “audio length,” from virtually any file format. Pymediainfo is a python wrapper for this interface that

allows any python program the power of MediaInfo. We use this to gather the required information about a video before erasure coding it such that it can be distributed in chunks to caches and users.

4.4 Auto Populate the List of Videos Stored on a Server

Originally, after erasure coding a video, information about the number of 10 second frames created, number of chunks created per frame (m), and number of chunks required to recreate the frame (k), in addition to all of the sizes of the chunks in bytes was required to be written into a comma separated file (.csv) before the server could understand its contents. This becomes an extremely large hassle, is prone to error, and can cause events where a server can tell users that it contains certain video contents when in reality it does not – crashing the system in worst case circumstances. Now, the server reads for all of the video contents that it is storing, gathers the proper metadata for each video, and returns this information to be used by the tracker without the use of a .csv file.

4.5 Send Data from User to Server about the Data Downloaded from Connected Caches

The next change is the addition of a new handler added to the server that is compatible with both the pyftpdlib package versions 0.7.0 as well as 1.4.0. After downloading video chunks from the connected caches, the user would then request additional packages from the server. By updating the prototype commands of the server, a user simultaneously tells the server exactly what cache

gave exactly what chunks. The server then stores this data as a JSON dictionary. This allows the server to know the bandwidth distribution on the caches to all of the users.

By then relaying this information to the tracker, all users are able to visit the tracker's URL in order to look at a visualization of this data in one of two ways:

1. A viewer can observe what the bandwidth distribution of each cache is. When viewing the data in this form, the viewer gets real time updates on what data (and in what amount) each cache is sending to each of the users. This allows a viewer to understand how hard the caches are working, the "active" chunks on the cache, what they are sending to the users, and which caches are sending out more data than others. An example of what a viewer will see is shown in Figure 2.

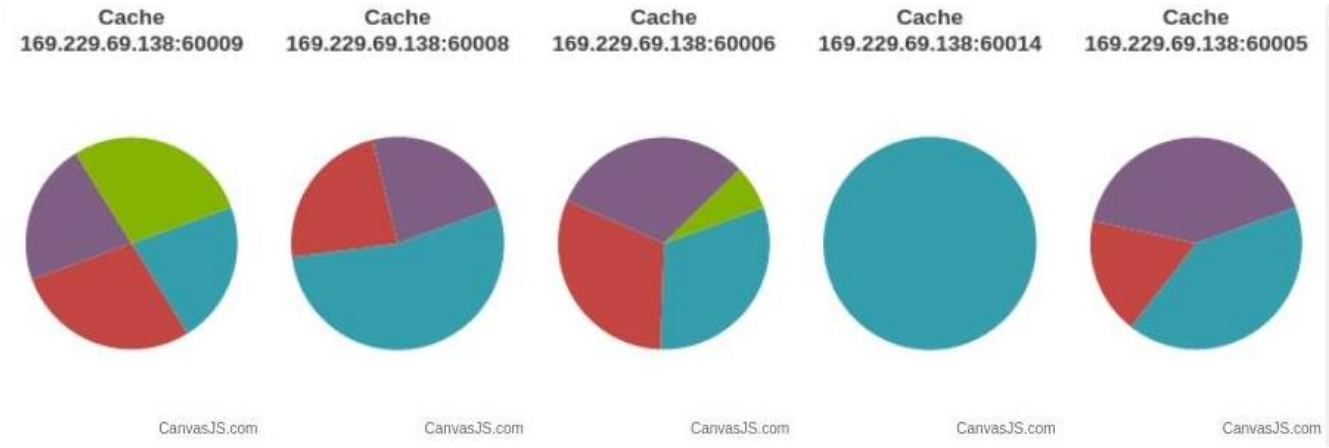

Figure 2 – Cache Chunk Download Distribution. Each pie chart represents a different cache. What is displayed here is the amount of bytes sent to each of the connected users to a cache. By scrolling over the cache with your mouse, it will display the user name and number of bytes sent.

2. A viewer can observe where a given user is receiving its chunks from. For each user, a visual is displayed designating the number of chunks received from each of the connected caches. This updates as the user requests each additional frame. By displaying this, we are given a detailed viewing on where the distribution of a cache's bandwidth is directed. It is often the case that certain users receive more chunks than others from the connected caches. This is due in part to the popularity of a given video. An example of what a viewer will see is shown in Figure 3.

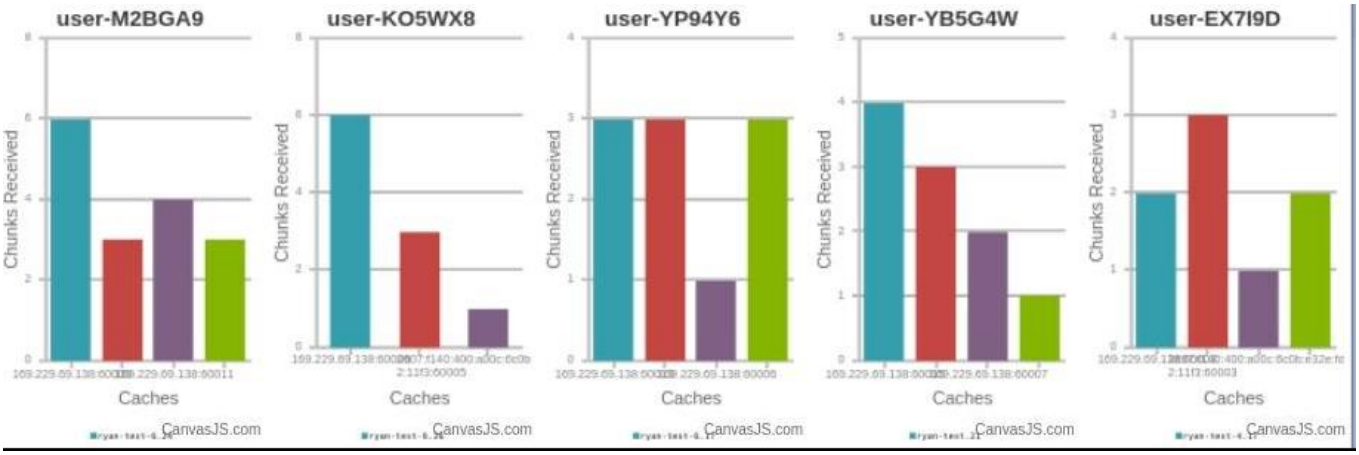

Figure 3 – User Download Distribution. Each bar graph represents a different user watching a different video. The five users each receive a number of chunks from their connected caches. In this instance, a total of 20 chunks are required to fully reconstruct the current video frame. All remaining chunks are downloaded from the server. In this current state, 54% of the required bandwidth is being sent by connected caches.

4.6 Allow Users and Caches to Log into Tracker via a Session

Once accounts were supported on the tracker, it was possible to log into the tracker via browser and view its contents. The python code running caches and users, however, were no longer able to view the contents that the tracker provided, and thus could no longer connect to the server or any other caches. This was due to the limitations provided with the most widely used python library for HTTP requests, urllib2. This library provides no idea of a session object, which keeps track of internet history from a given source, and is essential in order to log in to a service. Thus, in order to continue progress, the urllib2 library was replaced with a more extensive library.

The urllib2 library was replaced with the requests library which contained a session object. This object was built off of the lesser known urllib3 library, and allows for maintaining private keys given to a browser that are required for logging in to a website. The cache and user python scripts now create a session object that allow for logging in to the tracker, and is passed as an input to all HTTP requests to connect to the tracker.

The importance of this is such that the user and cache scripts must be associated with an account stored inside of the tracker's database in order to accumulate points (via running a cache) and subtract points (via watching a movie). The points are used as a currency for the system, and are used to determine if a person is able to watch a video or not (points are explained further in the next section). If the python code running caches and users was unable to log into the system, then

no progress on creating and developing an incentive mechanism for people to use our system could be created.

4.7 Implementation of a Freemium Points Service for Viewing Videos

Once the account system was created, a freemium currency was then developed that would reward persons for running caches, and place a monetary value on viewing a video. Current video streaming platforms require a monthly fee in order to fully view their content. This is due to the lack of business models available for their system architecture. Because the system described in this paper improves with many peers running caches, an incentive mechanism was developed in order to persuade people to allow the system to use their resources as a cache. This developed into a freemium points system that allows users to watch videos for free so long as they have contributed to our system as a cache prior.

When a user downloads a video, the user uploads the information regarding the amount of bytes that they received from each connected cache to the tracker. The tracker then takes this information, and accumulates the amount of points for each of the accounts running caches. For every gigabyte of data that a cache sends, the associated account gains one point. In order to watch a video, a user plus its account must purchase the video for the price of five points. These numbers were chosen because a 20 minute, 1080p video ranges from 700 megabytes to 1,000 megabytes, meaning they must upload 5-10 videos worth of content before watching one. After purchasing a

video, the tracker stores this information such that a user is not double charged for watching the same video twice.

5 Challenges and Solutions

Before creating a truly robust product, we must address the issue of caches dynamically disconnecting and reconnecting. This problem currently addresses two issues with our system. If a cache randomly crashes, the state of this cache is not properly reflected in the tracker. This means that future users, when connecting to our system through the tracker, will be redirected to connect to a non-existing cache. This poses an issue because a user will not be able to make a successful peer-to-peer connection with a host device, causing various issues. In addition, this issue means that we cannot dynamically update the caches that a user is connected to. This becomes especially important when we are running our server with full length, 1080p videos, where the duration of a connection is requires exponentially more bandwidth and time.

In order to address this feature, we must upgrade to pyftpdlib version 1.4.0, which allows us to run certain exceptions based on when a connection is disconnected. For example, by using pyftpdlib 1.4.0, if a cache crashes when it is connected to the server, the server can run its “on disconnect” exception to notify the tracker that a cache was disconnected. Simultaneously, a user can have an “on disconnect” exception for when a cache crashes, and can then close the connection and connect to the most qualified cache for the data that the user requests.

6 Future Improvements

Despite the amount of work that has been put into this project, there is still much room for improvement. A list and description of notable improvements are as follows:

6.1 Switch the Tracker Framework from Web.py to Something Else

Currently, development of Web.py has stopped. The framework itself is simple to use, however is not robust enough for large scale. The head developer has created a new framework called Flask that has currently been gaining a lot of steam in the software industry. It is recommended to switch to Flask (or any other framework) sooner rather than later before the tracker itself becomes too complicated.

6.2 Change the Tracker Database

In certain circumstances, the tracker website will not load properly (however is fixed by simply refreshing the browser). The MySQL database is not strong compared to others. It will be wise to implement a stronger database for the storage of user and cache data, especially as the amount of data grows.

6.3 Upgrade pyftpdlib to version 1.5.0

As of late December 2015, pyftpdlib upgraded to version 1.5.0. This library no longer supports the ftpserver object, which the system's server is built on. The upgrade from version 1.4.0 to 1.5.0 will not be trivial, however it is advised to keep the system's libraries up with the most up to date software.

6.4 Stop Server from Wasting Bandwidth on a Stable Environment

In the current version of the project, when the number of users is fixed, continually watching the same set of movies with fixed caches (all of which are full), the server will remove a chunk for a given video in place of a different chunk for the same video. For 1080p videos, this ends up wasting a lot of bandwidth for essentially no improvement to the system. As the number of caches and users scales up, this affect can amount for a larger amount of bandwidth and a great deal of money wasted.

6.5 Allow for People to Purchase Points with their Credit Cards

Currently, the only way to accumulate points with our system is for a person to run a cache. In order to be truly freemium, there must be a way for you to purchase the freemium currency with

real money. This feature is one of the most important parts of a freemium service, and should be implemented if a company were to use this application.

7 Conclusion

Throughout the course of the year, I have expanded upon a bare-bones prototype of a hybrid architecture video streaming service. With my added implementations, I have allowed the system to behave under a distributed environment with the support of 1080p full videos contained in a multitude of file types. I have upgraded the framework to support IPv6 addresses, and allow for easy, intuitive connection to a cloud such as Amazon Web Services running a central server. I have created a robust system that allows for registered accounts to keep track of their online currency used to purchase and collect a library of videos. The idea behind this translates directly to a viable business model that could only be applied to a system architecture such as ours, and is novel to a platform like this.

In addition, I have implemented numerous file transfer protocol handlers that allows for the seamless communication of real time data that is both simple to read and understand, as well as detailed to show an experienced operator an overlook of the system.

I have ensured that Scalable Video on Demand system described here is able to support a dynamic setting of connecting and disconnecting caches and users that want to have a continual, rewarded progress of contributing to our system. By making everything bug free and easy to use to the point that a new person could easily continue work on the project without running into any hard-to-fix

errors, I have ensured the prolonged life of this system and a foundation for a possibly incredible breakthrough.

8 References

1. K. Lee, H. Zhang, Z. Shao, M. Chen, A. Parekh and K. Ramchandran, “An Optimized Distributed Video-on-Demand Streaming System: Theory and Design”*, The 50th Allerton Conference on Communication, Control and Computing, Monticello, IL, October, 2012.

2. K. Lee, L. Yan, A. Parkeh and K. Ramchandran, “A VoD System for Massively Scaled, Heterogeneous Environments: Design and Implementation”, IEEE 21st International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2013), San Francisco, CA, August, 2013.

Chapter 2

Engineering Leadership

(Team Written)

1 Introduction

Among all engineering leadership topics and tools we learnt over the last year, we chose the three most relevant modules to our system. This paper will analyze the current industry of video content delivery, market research that can validate the commercial success of our product, and the management of the software intellectual property.

2 Industrial Analysis

As of 2014, broadband internet is nearly ubiquitous around the world with the “global average connection speed exceeding the 4 Mbps broadband threshold” (Young 2014) according to a report generated from the Akamai Intelligent Platform. With the increasing availability of high speed internet and internet connected devices, the delivery of entertainment video directly over the internet has become a convenient solution for consumers. Video streaming services have increasingly been accepted by consumers as the alternative to multichannel video programming distributors (MVPD) such as satellite and cable television systems and physical media rental services.

Our platform is backed behind a novel heuristic that reduces bandwidth costs on a centralized server to potentially nothing, which gives our system an edge over current video streaming services such as Netflix, Amazon Video, and Hulu. Despite this, it is important to observe and analyze the current competitors of this industry to see what could be done for us to successfully enter this line of business.

In the realm of entertainment video delivery over the Internet, Netflix dominates and controls the market. Netflix members are able to enjoy streaming media over their televisions, mobile devices, and computers. Members can play and pause content at their leisure without commercials, commitments, or restrictions. Netflix, Inc. is a membership service, but members have unlimited access to a pool of available content during their time of membership. As of July 2011, Netflix, Inc. has increased its emphasis on the removal of DVD based entertainment, focusing more on its robust video streaming platform,

An advantage Netflix, Inc. has over the market for entertainment video delivery is its emphasis on making its service compatible with many devices while maintaining streaming functionality. The company holds partnerships with "satellite and telecommunications operators to make their service available through the television set-up boxes of these service providers," (Netflix 10-K 2014).

Netflix, Inc. "strives for consumers to choose (Netflix) in their moment of free time," (Netflix 10-K 2014). Netflix, Inc. refers to this part of their company's objective as the "winning moments of truth" (Netflix 10-K 2014) and places emphasis on their "recommendation and merchandising technology to predict and recommend titles for members to enjoy," (Netflix 10-K 2014). In addition, Netflix has begun rolling out its own Content Delivery Network (CDN) called Open Connect that will reduce cost and improve the delivery of the service's content by reducing network overhead (Netflix 2014). Because Netflix is a large percentage of the traffic ISPs deliver to their end-users, Netflix, Inc. has offered a solution to peer directly with Netflix. CDN,

however, will only provide a reduction in latency for users and has no effect on the total amount of data sent from Netflix servers to its customers. Thus, despite their claims, our platform has the capability of providing customers with the same video at a greatly reduced price.

A concern and risk factor for Netflix, Inc. (Netflix 10-K 2014) is the retention and attraction of members to the service. Consumers need to seek value with the service which can be primarily found in the speedy delivery of content as well as "compelling content choices, as well quality experience for selecting and viewing TV shows and movies," (Netflix 10-K 2014).

3 Market Research

Over the last two decades, there has been an exponential increase in the interest shown in watching. This indicates that the VoD system has a huge market. The current leading competitors in the VoD market are Netflix, Hulu, and Amazon Prime Video. The revenue generated by these companies indicate consumer's inclination towards the video streaming service is increasing and this technology is slowly replacing traditional satellite and cable television systems.

The core idea of success behind our VoD system is based on our conclusion from market research that by caching the few popular videos that are on high demand we can significantly reduce the bandwidth of the system, while delivering content to users at an improved speed. According to a Video statistics and Market research by Tubular, a video intelligence software, only 5% of videos uploaded to YouTube drive most of the views (Marshall 2015b). Out of 1.1

billion videos on YouTube, only 58.6 million have more than 10,000 views and these videos alone have generated 7.4 trillion views (Marshall 2015b).

Anyone who owns an internet connected device is a potential customer to our product. The Interactive Advertising Bureau, a nonprofit organization, states that there has been an increase of 35% in views collected by people watching videos on their smart devices in the past year and also one third of the people surveyed said that they watch at least one video lasting five minutes every day (Marshall 2015a). Smartphones with bigger screen, improved display, memory, and battery life indicate that the technological advancements in edge devices go hand in hand with these social trends.

Cisco predicts that by 2018, 84% of the consumer internet traffic will come from video (Zccone 2014). Consumers are streaming more video content than ever before, and Internet Service Providers are not able to maintain sufficient bandwidth required to enable constant high quality video streaming. Being well aware of this problem, Verizon and Comcast have made special agreements with the heaviest traffic producing consumers that, for an additional cost, the traffic from these customers will be handled at a higher priority (Grill 2015).

The technological innovations like ‘capture camera’ and ‘retina display TV with UHD 4k videos’ worsen the problem, as the backend is not able to keep up with the user demands. Reducing video bitrates is not a solution, as it will only lead to poor viewing experience and lose market value (Grill 2015). Our system comes up with a feasible solution to this problem. If we cache the popular videos on edge devices, the user would not have to get the data from the

central server, thus reducing internet traffic and improving the performance in terms of speed and quality.

This project can be made into an open source project contributing to technological evolution. An ideal marketing strategy for this system would be to develop a final product with a good user interface, and advertise it as a commercial product. Our system, can provide high quality video at a cheaper price, making it easier to sell our product and replace existing VoD companies. Our market research indicates potential success of our product and the ability to revolutionize the VoD market.

4 Intellectual Property

The core technology of this project is the VoD architecture, including the concepts of using caches as edge resources, algorithms ruling the data flow, and methods of chunking videos into parts. Several related papers have been published (Lee 2012) (Lee 2013), and a working prototype has been built on top of them. We believe that this system architecture is patentable, and holding such patent could bring numerous advantages to the product in business.

First, a patent enables us to provide the stable and fast VoD experience to customers in an exclusive way. It will maintains the differentiated value from the technology we developed. Second, protection of the core technology allows the ongoing product development to focus on the product itself, without worrying about similar services being released by competitors. Last

but not least, a patent can add value to our project and thus make it more likely that another company will buy our technology. Typically, filing a utility patent in U.S. will cost around $20,000 (Quinn 2016). Nevertheless, this is a small investment compared to the future return, as current market competitors are generating approximately 1 billion dollars of revenue (MarketsAndMarkets 2016).

On the other hand, there are difficulties in protecting this IP as a patent. This market has become extremely competitive in recent years. Companies such as Rovi, AT&T, and Microsoft hold large patent portfolios. Moreover, software patents are hard to define. As a result, improperly defined IPs can lead to lawsuits if filed without prior research. For example, in 2013, the author of the patent "Video-on-demand systems (Nomura 2007)" sued Youtube and Amazon's CloudFront services of infringing his entire patent (Randles 2016). Also, the product's bandwidth and resource-saving advantages might be invalidated by network technology advancement, as the network resources will not be an issue any more.

Alternative solutions to protect our IP include transforming it into a trade secret, sell the project to existing VoD vendors, or make it an open source project. Our project can be a valuable addition to the existing VoD platforms as an upgrade to their original services.

5 Conclusion

Analysis of the industry landscape, market research, and IP have helped us to understand the potential of the product, as well as a clearer idea of how to commercialize it. Tools learnt from

the engineering leadership courses have played a crucial role our analysis will be utilized in future stages of development.

6 References

Gill, Dror. "As The Internet Grows Bigger, What Do We Do About Video?" *ReelSEO*. Reelseo & Tubular Labs, Inc., 15 Jan. 2015. Web. 05 Mar. 2016. <http://www.reelseo.com/internet-bigger-video/>.

Lee, Kangwook., Hao Zhang, Ziyu Shao, Minghua Chen, Abhay Parekh, and Kannan Ramchandran. "An Optimized Distributed Video-on-demand Streaming System: Theory and Design." *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*(2012): n. pag. Web.

Lee, Kangwook., Yan Lisa, Parekh Abhay, and Kannan Ramchandran. "A VoD System for Massively Scaled, Heterogeneous Environments: Design and Implementation." *2013 IEEE 21st International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems* (2013): n. pag. Web.

MarketAndMarket "Video on Demand (VOD) Market worth $61.40 Billion by 2019." *Video on Demand (VOD) Market worth $61.40 Billion by 2019*. Marketsandmarkets, Feb. 2015. Web. 05 Mar. 2016.

Marshall, Carla. "We're All Watching More Video on Mobile Devices [Study]." *ReelSEO*. Reelseo & Tubular Labs, Inc., 10 June 2015a. Web. 05 Mar. 2016. <http://www.reelseo.com/increase-mobile-video-consumption/>.

Marshall, Carla. "5% of Videos on YouTube Drive 95% of the Views." *ReelSEO*. Reelseo & Tubular Labs, Inc., 04 Sept. 2015b. Web. 05 Mar. 2016. <http://www.reelseo.com/5-percent-youtube-videos-drive-95-percent-views/>.

Netflix Inc. 2014 10-K Report. Netflix Inc. 03 Feb. 2014. Web. 06 Mar. 2016

Netflix. *OpenConnect-Deployment-Guide.pdf*. N.p.: Netflix, 2014.*Oc.nflxvideo.net*. Netflix, 2014. Web. 06 Mar. 2016. <http://oc.nflxvideo.net/docs/OpenConnect-Deployment-Guide.pdf>

Nomura, Tetsuya. Video-on-demand System. Tetsuya Nomura, Tommy Sun, assignee. Patent 7254622. 7 Aug. 2007. Print.

Randles, Jonathan. "YouTube, Amazon Escape Video On Demand Patent Suits - Law360." *YouTube, Amazon Escape Video On Demand Patent Suits - Law360*. Law360, 13 Sept. 2013. Web. 05 Mar. 2016.<http://www.law360.com/articles/472533/youtube-amazon-escape-video-on-demand-patent-suits>

Quinn, Gene. "The Cost of Obtaining a Patent in the US - IPWatchdog.com | Patents & Patent Law." *IPWatchdogcom Patents Patent Law The Cost of Obtaining a Patent in the US Comments*. IPWatchdog, 04 Apr. 2015. Web. 05 Mar. 2016. <http://www.ipwatchdog.com/2015/04/04/the-cost-of-obtaining-a-patent-in-the-us/>

Young, Jeff. "Akamai Releases Second Quarter 2014 'State of the Internet' Report." *Akamai*. Akamai, 30 Sept. 2014. Web. 06 Mar. 2016.

Zaccone, Emanuela. "Why Video Consumption and Instant Messaging Rule Social Trends." *Medium*. N.p., 29 Jan. 2015. Web. 05 Mar. 2016. <https://medium.com/@zatomas/why-video-consumption-and-instant-messaging-rule-social-trends-5d86e51e4355#.rfvko98rx>.