Netflix Architechture Analysis

Saad Ahmed, Ralston Goes, Sanket Jain
ELEN E6776 Content Distribution Networks, Electrical Engineering
Columbia University
sa3205, sj2659, rrg2148 @columbia.edu

Abstract—**Netflix, a subscription based video-streaming service provider was founded in 1999. By 2007 it already had outperformed all its competitors and it further started to optimize and expand its services. Today Netflix alone accounts for around one-third of the peak download traffic in North America. Because of propriety issues, there are not many analyses performed for Netflix's traffic and CDN optimization. In this paper, we perform an analysis of the deployed architecture of Netflix as well as observe the traffic flow of packets. Additionally, we observe how throttling the Bandwidth changes the video quality and check if the CDN redirection takes place. We then discuss the scalability issues and further possibility for optimization of CDN selection and video delivery strategy that Netflix might be looking forward to deploying to deal with its ever-growing video traffic problem.**

I. INTRODUCTION

Netflix is one of the smartest companies of this decade. As the author of book titled *Netflixed* says, "It has more drama than most of the movies Netflix rents", the same stands for the data distribution from content origin to the end user of Netflix. One of the most commonly heard public case being bottleneck of Netflix by various ISP like Comcast and Verizon. In past decade Netflix demand increased almost exponentially resulting in bigger issues like limited bandwidth and fast lane competition. Reed Hastings the co-founder and CEO of Netflix is investing heavily to improve not only the contents but also on how they are distributed through CDN. One subscriber of Netflix uses about 3 GB per second for online streaming High Definition video and about 1GB per hour for Standard Definition and much more for Ultra High Definition. The leading subscription-video streaming of Netflix accounted for 36% of the total downstream Internet bandwidth during the peak traffic hours last March in US. Just in past six months, Netflix boosted its traffic share from 34% to 36% which is 2% increase in just 6 months. As of March 2015, Netflix's video streaming service has gathered over 50 million subscribers in 40 different countries especially the developed ones like: America, Australia, Japan and many more in Europe. Such a popular service has to solve newer engineering problems like efficient streaming, on-demand content and a huge number of requests causing heavy traffic. It becomes crucial that Netflix takes these factors into account to maintain its successful business model.

Two major factors contribute to Distribution of Content: The diameter of fiber optic cable, which has huge capacity and the infrastructure, provided by ISP. Obviously the infrastructure of distributing content needs to be improved and CDN is the main thing that needs that entire infrastructure. As Reed Hastings says, "A few more shelves of equipment might be needed in the buildings that house interconnection points...", I think it is time to rethink about the current CDN which is not only used by Netflix but also by various ISP. Hence we are presenting an analysis of current CDN of Netflix so that further improvements in infrastructure can be made to provide better Quality of Experience to user.

Netflix has taken many steps like providing service up to but not including last mile through better implementation of CDN and cloud. CDN are content distribution networks which use variety of techniques to send data from one center to end or from end to center to another end. Netflix initially outsourced streaming video delivery to three main vendors namely Level 3, Akamai and Limelight. Now Netflix has its own CDN, which accounts for more than 90% of data flow in USA and 100% outside America. It has also resorted to the use of cloud service, specifically Amazon Web Service (AWS) adding another layer to its CDN providing an IT infrastructure. Amazon Simple-DB, S3 and Cassandra are used for file storage, video streaming is served out of multiple CDNs, and a public DNS service called Ultra-DNS is used as its authoritative DNS servers. Microsoft Silverlight is the video playback platform for Netflix desktop users. As of June 2012 Netflix invested in Open Connect which is being used to cache Netflix content within its ISP's data center, bringing it close to subscriber except at the last mile hence providing better Quality of Service. Additionally, the company invested in NGINX, which is the streaming media server that runs on every Open Connect delivery appliance. Recently it made a collaborative project with Google named *Spinnaker*, which is used for cloud management (elastic cloud) and as a delivery engine. Netflix is a very smart and growing company that uses advanced

techniques and basics like downloading video based on recommendation algorithms in mid-night when users are asleep.

In this paper we:

- Study and analyze the classic architecture model used by Netflix before 2012 that is from 2008-2012.
- Discuss current architecture involving Open Connect that was implemented in 2014.
- Perform graphic analysis of packet flow from client to host. When and whom does the client contact - CDN, Amazon cloud or other options?
- Packet analysis over TCP and how encryption key is handled. Which cloud is responsible for which user and/or content?
- DASH protocol, how does low bandwidth high congestion is handled by DASH for Netflix.
- Variation in packet loss during download and of the ratio of packets lost to packets received.

II. NETFLIX ARCHITECHTURE

A. *Overview*

As the Netflix traffic increased at such a rapid pace, it had to combine multiple third party services in addition to its own platform and come up with a cloud based video streaming architecture, the one which allowed Netflix to acknowledge itself as a scalable, infrastructure-less content provider. Introduction to cloud based service was necessary since the local servers were eventually running out of space and also with the increase in traffic, bandwidth available per user was becoming the bottleneck in the video streaming performance.

Netflix moved to Software as Service (SaaS), a distribution model in which applications are hosted on a remote server to be accessed over from anywhere. It also established and built its own (PaaS) to develop Big Data Tools, HA Patterns and deploys its application and content over virtualized servers and operating systems. As far as the storage issues are concerned, it moved incremental capacity to (IaaS). Therefore, since 2008 they did not require to build any data center space and all the content and streaming app was moved to cloud. For the storage, AWS cloud services were adopted.

B. *Amazon Cloud services*

The following goals were achieved with Netflix going 100% on cloud: There was lower latency than the equivalent datacenter web pages and API calls. This was true for both, the first hit to home page as well as in-session hits. Scalability was revived as there were no more datacenters needed to accommodate the incoming new subscribers. It not only enabled substantially higher robustness and availability of content than datacenter services but also optimized agility of a large development team with automation and tools leaving behind the complex tangled datacenter code base. In order to modify the website, there is no scheduled down time since there is no central schema to change. Clean-layered interfaces and re-usable components were enforced. Figure 1 shows the constituents and distribution of deployed Netflix services on AWS Cloud.

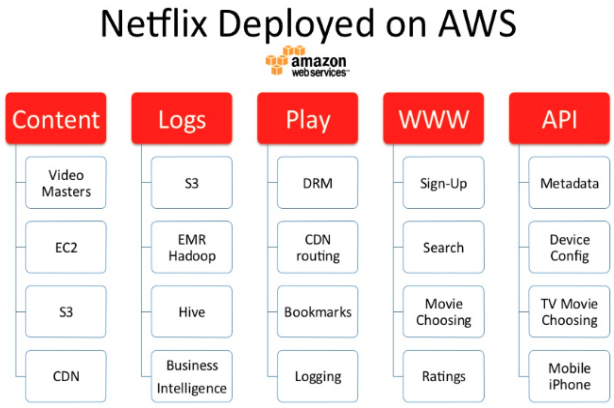

Figure 1: The constituents on distribution of deployed Netflix Services as on Amazon Web Services Cloud.

C. Key Components and Protocols

This lead to 4 key components in the architecture: 1) Netflix center, 2) Amazon AWS cloud, 3) Third party CDNs and 4) players. Figure 2 displays the architecture representation.

To observe the basic service behavior, we create an account and login into the Netflix main website. We monitor the traffic while performing all this activity. Once we get the IP addresses of the destination servers we then perform DNS resolutions to collect the canonical names of the servers that browser contacted. Figure 2 shows basic architecture of video streaming platform for Netflix.

As observed from traffic analysis, it is seen that Netflix uses its own IP address space for the host name www.netflix.com [3]. The registration of new user accounts and all billing related matters are either redirected to its own host name or Amazon. When we start streaming the movie the Netflix server does not interact again with the client.

Once we are on Netflix video selection page, or if we are streaming a movie, we are served by Amazon AWS cloud. It uses many AWS zones for high availability and scalability. For our analysis, all the website traffic including the video streaming was directed to host [ec2-184-72-246-221.compute-1.amazonaws.com].

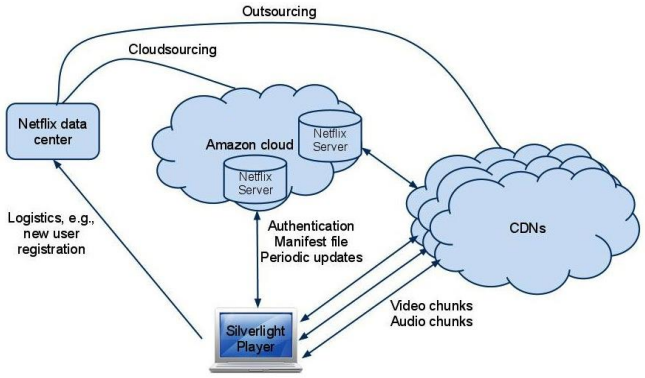

Figure 2: Architecture representation of Netflix

D. Content Distribution Networks:

Netflix employs multiple third party CDNs to deliver the video content to end-users. All the encoded and protected video content is sourced out to Amazon cloud, which then sends multiple copies to different CDNs. As shown in fig. 2, initially, just three CDNs were employed: Akamai, Limelight and Level-3. However, Netflix is further optimizing its architecture by deploying its own specialized content delivery and caching system 'Open Connect Appliances'. We will discuss this in detail in next section.

Netflix uses Silverlight to download and decode chunks on Desktop [2]. Players are also available for mobile/Ipad etc. For streaming in volatile Bandwidth environment, Netflix uses DASH (Dynamic Streaming over HTTP) protocol. DASH is a video streaming protocol that allows player to freely switch between video quality levels. It is totally different from UDP-based streaming and involves the adaption of both by the application and by the TCP. Netflix's adaption seems to default to TCP control during heavy network congestions or even during periods of volatile network conditions [3]. For every packet download, the protocol measures the received bandwidth and runs a rate-resolving algorithm to control the quality of next chunk.

E. *Architecture Improvement: Open Connect*

In June 2012, Netflix articulated its own specialized content delivery system called Open Connect. Open Connect Appliances (OCAs) are provided at no charge to Netflix's partner ISPs. Since this content delivery system is created by Netflix for its own use, it is very efficient in content delivery and performs much better than relying on proxy caches. It uses the data of its known set of users and its known set of content to result in a simplified content delivery. It uses the ISPs ability to gather routing information via BGP protocol and combines it with Netflix content routing to create a robust and extensible layer 3-7 routing engine. Layer 7 load balancing operates at the high-level application layer and deals with the actual contents of the message. It terminates the network traffic and reads the content of the message to make better load balancing decisions. The benefits of layer 7 includes the ability to apply optimizations and changes to content as well as use buffering to offload slow connections from upstream servers thus improving the overall system's performance. Therefore, the owning server and client logic allows for optimization of the streaming performance. Open Connect also uses proactive caching. Proactive caching allows for users to be able to see the most recent data while the source data and aggregations are stored in a multidimensional structure to improve performance. The old cache can be invalidated and a new one can be built with the most recent data from the source system at any set schedule. In Netflix, this is taken as an advantage by pre-populating content within ISP networks at off-peak times. This dramatically reduces upstream network utilization at peak times by 75-100%. This also reduces the load on ISPs and thus eliminates the necessity to scale, transport or set up internal links for Netflix traffic. Another advantage of using proactive caching is that it allows for central popularity calculation, which is more reliable and accurate than a cache, or proxy that determines popularity based on the requests it receives. Also, the OCAs are simple web servers that are 100% utilized for throughput and they know how to sort files and report on health.

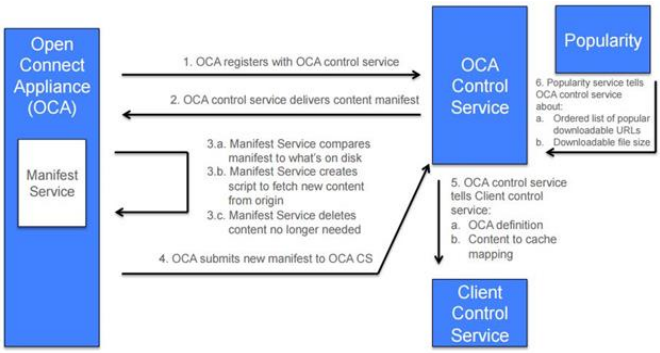

Figure 3: Open Connect Appliance

The content flow in OCA can be described with reference to the above diagram. The main components we consider is the OCA, with manifest service, the OCA Control Service, the Client Control Service and a Popularity service. The OCA registers itself with the OCA control service. The OCA control service delivers the content manifest to the OCA. The manifest service on OCA then compares the received manifest from the OCA control service with the manifest that's on the disk. If it finds new content that it requires, the manifest service creates a script to fetch the new content from the origin. The manifest service also deletes content that is no longer in use.

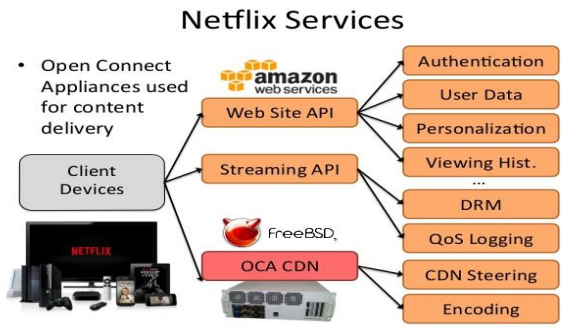

Figure 4: Netflix OCA-CDN Implementation

The OCA then submits a new manifest to the OCA control service (OCA CS). The OCA CS in turn communicates with a Client Control Service, providing the definition of the OCA as well as the content to cache mapping. The OCA CS is also fed by a Popularity service, which tells it about the ordered list of popular downloadable URLs and the downloadable file sizes. As any control service, the OCA CS plays a critical role here as well in maintaining and updating the OCA.

To obtain the best service, the client needs to know its closest OCA for video stream delivery. It first contacts the Netflix control servers to obtain the address of the local Netflix cache.

III. Traffic Analysis

A. *DNS and Packet Routing*

Once we go to the Netflix.com and select any movie title to play, the DNS resolves which CDN to contact by first contacting the Netflix server based on ec2. As we can see below in figure 5, standard query is first made to host Netflix.com. Once we receive the response back with the information about what CDN to connect to, standard query for connection is sent to that particular CDN base. Once we get the response, the connection is established.

```
DNS      75 Standard query 0x61f7 A www.netflix.com
DNS     148 Standard query response 0x61f7 A www.netflix.com CNAME www.geo.netflix.com CNAME
DNS      77 Standard query 0x2aa2 A cdn-0.nflximg.com
DNS      75 Standard query 0xd6a7 A cdn.nflximg.com
DNS     184 Standard query response 0x2aa2 A cdn-0.nflximg.com CNAME dscg.netflix.com.edgesu
DNS     181 Standard query response 0xd6a7 A cdn.nflximg.com CNAME images.netflix.com.edgesu
DNS     100 Standard query 0xf0f8 PTR lb._dns-sd._udp.0.10.20.172.in-addr.arpa
DNS     100 Standard query response 0xf0f8 PTR lb._dns-sd._udp.0.10.20.172.in-addr.arpa
DNS      78 Standard query 0x3c0e A assets.nflxext.com
```

Figure 5: DNS Resolution

Also, Netflix assigns a fixed CDN to a user the moment he subscribes for the service. This CDN remains the same for that particular user regardless of its location, IP address, computer, traffic condition etc. For our account, our traffic was all redirected to Akamai CDN from Amazon ec2 cloud, no matter from where and when we accessed the site.

11:26:49 PM	Google Chrome	ec2-54-204-41-202.compute-1.amazonaws.com
11:26:49 PM	Google Chrome	ec2-54-204-41-202.compute-1.amazonaws.com
11:26:49 PM	Google Chrome	ec2-54-204-41-202.compute-1.amazonaws.com
11:26:50 PM	Google Chrome	128.59.47.208
11:26:53 PM	Google Chrome	ec2-54-204-41-202.compute-1.amazonaws.com
11:26:54 PM	Google Chrome	ec2-50-19-235-7.compute-1.amazonaws.com
11:26:54 PM	Google Chrome	a23-203-29-222.deploy.static.akamaitechnologies.com

Figure 6: IP Addresses showing traversed traffic

B. *Traffic Analysis on Wireshark*

As we go on Netflix.com with our account logged in, Hello message is sent to host (54.243.242.133 – Amazon ec2) for handshake over TLSv1.2. Server sends back the HTTP text indicating that it accepted the connection and this confirmation is received via TLSv1.2. Then server and client exchange the keys and change cypher specifications of the file as seen in figure 7. Once the encryption handshake messages are exchanged, the host is ready to send the requested application data over TCP. All the packets are requested and sent over a TCP protocol. All frames sent to and received from host are of length 66 bytes and 1454 bytes respectively

```
172.20.10.5     54.225.199.51   TL…  Client Hello
54.243.242.1…   172.20.10.5     HT…  HTTP/1.1 202 Accepted  (text/plain)[Malformed Packet
172.20.10.5     54.243.242.133  TCP  58097 → 80 [ACK] Seq=17520 Ack=19317 Win=129696 Len=
23.196.59.18    172.20.10.5     TL…  Application Data
172.20.10.5     23.196.59.18    TCP  58101 → 443 [ACK] Seq=931 Ack=4426 Win=130752 Len=0
54.225.199.51   172.20.10.5     TCP  443 → 58102 [ACK] Seq=1 Ack=243 Win=15616 Len=0 TSva
54.225.199.51   172.20.10.5     TL…  Server Hello
54.225.199.51   172.20.10.5     TCP  [TCP segment of a reassembled PDU]
54.225.199.51   172.20.10.5     TL…  Certificate
172.20.10.5     54.225.199.51   TCP  58102 → 443 [ACK] Seq=243 Ack=2777 Win=129664 Len=0
172.20.10.5     54.225.199.51   TCP  58102 → 443 [ACK] Seq=243 Ack=4097 Win=128352 Len=0
54.225.199.51   172.20.10.5     TL…  Server Key Exchange
172.20.10.5     54.225.199.51   TCP  58102 → 443 [ACK] Seq=243 Ack=4345 Win=130816 Len=0
172.20.10.5     54.225.199.51   TL…  Client Key Exchange
172.20.10.5     54.225.199.51   TL…  Change Cipher Spec
172.20.10.5     54.225.199.51   TL…  Encrypted Handshake Message
54.225.199.51   172.20.10.5     TCP  443 → 58102 [ACK] Seq=4345 Ack=324 Win=15616 Len=0 T
54.225.199.51   172.20.10.5     TL…  Change Cipher Spec, Encrypted Handshake Message
172.20.10.5     54.225.199.51   TCP  58102 → 443 [ACK] Seq=369 Ack=4396 Win=131008 Len=0
172.20.10.5     54.225.199.51   TCP  [TCP segment of a reassembled PDU]
172.20.10.5     54.225.199.51   TL…  Application Data
```

Figure 7: Wireshark Analysis

As displayed, the encryption keys are exchanged over TLSv1.2.

```
TLSv1.2 Record Layer: Handshake Protocol: Encrypted Handshake Message
   Content Type: Handshake (22)
   Version: TLS 1.2 (0x0303)
   Length: 40
   Handshake Protocol: Encrypted Handshake Message
```

Figure 8: Encryption handshake message header

C. *TCP Packet Flow*

Once we choose to stream a video content, it is observed that the destination IP changes from 54.243.242.133 to

208.54.37.30. We can say that the requests for the content have now been directed to Akamai CDN closest to our ISP.

172.20.10.5	17.143.164.93	TCP	57614 → 5223 [PSH, ACK] Seq=1 Ack=1 Win=4096 Len=7
172.20.10.5	208.54.37.30	TCP	58104 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS
172.20.10.5	208.54.37.30	TCP	58103 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS
172.20.10.5	208.54.37.30	TCP	58105 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS
172.20.10.5	208.54.37.30	TCP	58106 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS
172.20.10.5	208.54.37.30	TCP	58107 → 80 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS
17.143.164.93	172.20.10.5	TCP	5223 → 57614 [ACK] Seq=1 Ack=710 Win=429 Len=0 TSv
208.54.37.30	172.20.10.5	TCP	80 → 58104 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
172.20.10.5	208.54.37.30	TCP	58104 → 80 [ACK] Seq=1 Ack=1 Win=131840 Len=0 TSva
172.20.10.5	208.54.37.30	HT…	GET /range/0-36995?o=AQG749G7vEKcyoWD-5cm2u1wdJp47
208.54.37.30	172.20.10.5	TCP	80 → 58103 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
208.54.37.30	172.20.10.5	TCP	80 → 58106 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
208.54.37.30	172.20.10.5	TCP	80 → 58105 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0
172.20.10.5	208.54.37.30	TCP	58103 → 80 [ACK] Seq=1 Ack=1 Win=131840 Len=0 TSva
172.20.10.5	208.54.37.30	TCP	58106 → 80 [ACK] Seq=1 Ack=1 Win=131840 Len=0 TSva
172.20.10.5	208.54.37.30	TCP	58105 → 80 [ACK] Seq=1 Ack=1 Win=131840 Len=0 TSva
208.54.37.30	172.20.10.5	TCP	80 → 58107 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0

Figure 9: TCP Packets Flow

CDN host then sends the video content in packets with varying frame window size, depending on the bandwidth limitation. Frame length of 1454 bytes is consistent for all the received video chunks. Client requests the packets through sequence numbers and acknowledgements for the received packets are also piggybacked along with the next request. The figure below shows the transmission flow of few segments of TCP packets:

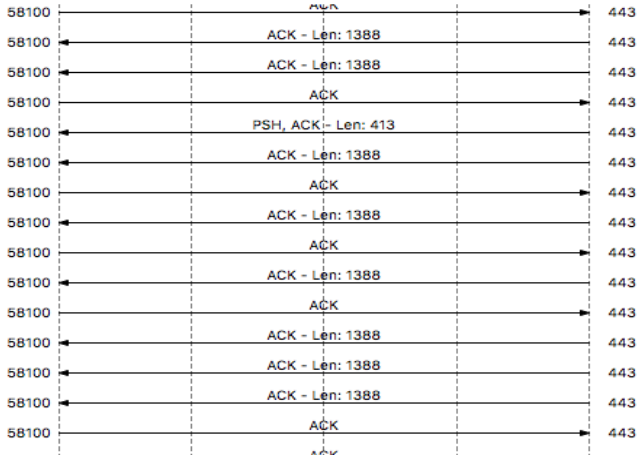

Figure 10: Transmission of Packets

We also analyzed the packet flow per second for one of the streamed movies for several minutes. There were several packet losses and retransmission observed. Figure 11 is the graph obtained. Bars in blue represent the packet losses. As we can see the packets received per second is the highest as soon as we started streaming the video. It is mainly because the buffer is empty and extra packets are received to fill it.

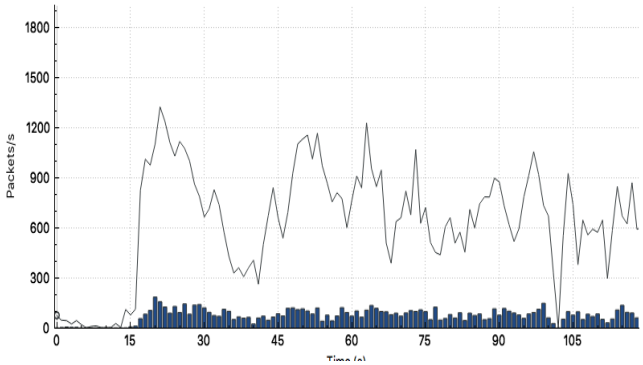

Figure 11: Packets received at user end per second.

Next, we throttled the incoming Bandwidth to observe the following:

1) Does the Netflix allow switching of CDN to keep up with the streaming video quality
2) Observe the DASH protocol by noticing the video quality adaption with variation in Bandwidth
3) Notice the change in window size and variation in packet losses

The analysis performed by [2] shows that once the available bandwidth for the linked CDN decreases, the Netflix architecture does not switch the client to some other CDN. Instead, it uses DASH protocol to lower the available quality to adjust it to the available bit rate. Only when it is no longer able to support even the lowest quality (usually at 100kb/s), it directs client to some other CDN that is providing better performance and lower latency.

Since we cannot throttle the bandwidth at the CDN end, in order to notice the switching of clients connection to different CDN under traffic congestion, we will virtually create such an environment by varying limiting the bandwidth at our end to have a basic idea. We have used '*Network Link Conditioner'* to throttle the inbound bandwidth to the client.

We have investigated and analyzed the instantaneous bandwidth variation during several minutes of video playing. This is important as DASH player constantly monitors the available bandwidth to determine the level of video quality to stream. The small time scale bandwidth can significantly affect the viewer's experience. Figure 12, 13 and 14 shows the packets received per second at different Bandwidths: 0.1MB/s, 0.2 MB/s, 1.5 MB/s and 10 MB/s. When the Bandwidth was limited to 100kbps, Netflix video failed to load and our result is consistent with the analysis performed in [2]. Once we increase the bandwidth to 200kbps, the video content takes a while to load/buffer and streams with a poor quality. As we can see from graph of figure 12, the number of packets received per second is at

maximum, 150, in the start. Also, the number of packets lost is much higher.

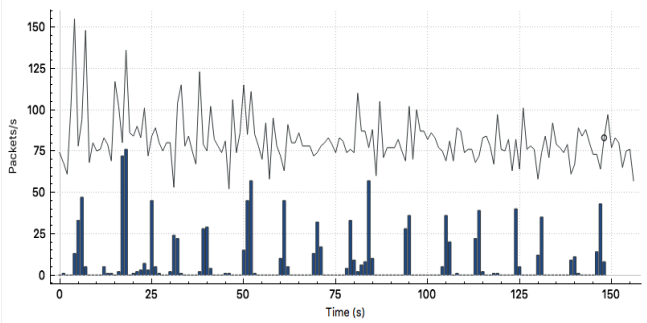

Figure 12: Packets received and lost per second at bandwidth of 200 kbps

As we increase the Bandwidth to 1.5 MB/s, we observe a little better video quality (medium quality) and as seen from the graph the number of video chunks received per second increases to ~280 packets per second. The packet loss is also decreased, hence resulting in fewer pauses due to buffering.

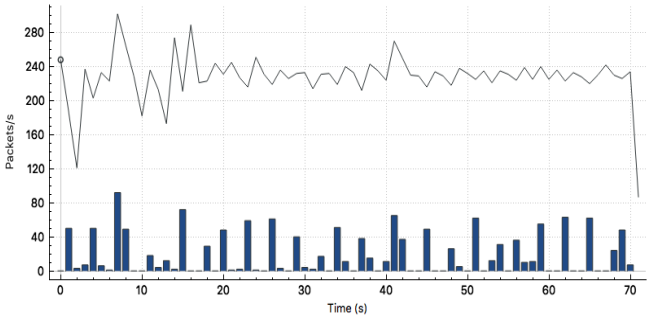

Figure 13: Packets received and lost per second at bandwidth of 1.5 Mbps

Finally we increase the Bandwidth to 10 MB/s and observe a tremendous improvement in the performance of video streaming. Not only does the video quality become HD, but also the ratio of packets received to packet lost increases tremendously. The maximum number of packets received in the start of playback is 1500 per seconds. The latency time is also negligible.

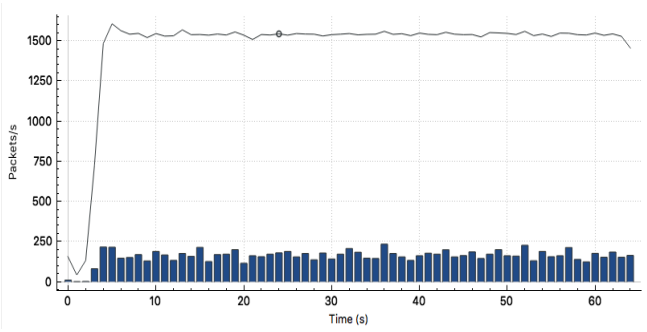

Figure 14: Packets received and lost per second at bandwidth of 10 Mbps

IV. Conclusion

For this project, we analyzed basic as well as improvised Netflix's video-streaming services and CDN modeling to uncover the overall architecture of Netflix, which is currently by far, the leading video-streaming subscription based platform. We have observed the routing of TCP video chunks and overall traffic flow with in its architecture. We have also evaluated the performance of Netflix's current architecture by performing traffic measurements under bandwidth variation. We have also observed that CDN for each user is fixed for each subscriber and does not change regardless of location, time or traffic congestions in CDN's networks.

Netflix is already planning to introduce Ultra High definition movies. Due to the size of UHD files, the traffic at the CDNs and the network load would increase significantly resulting in throttling the available Bandwidth and video bit rate for each user. In order for Netflix's to keep up with this new feature while further improving on scalability and better user appearance, it needs to make its CDN assignment strategies more elastic and dynamic.

Netflix also has the capacity to improve recommendation algorithm to ensure that the recommendation titles for a particular user are being cached pro-actively at user's closest CDN or caching platform.

Acknowledgment

We would like to express our gratitude to and thank Professor Anwar Walid for sharing his vital knowledge to help us grasp and understand the essentials of CDNs and related topics

References

[1] http://people.cs.clemson.edu/~jmarty/projects/DASH/CCNC2013SUBMISSION.pdf

[2] Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery. By Adhikari, Vijay Kumar

[3] http://people.cs.clemson.edu/~jmarty/projects/DASH/CCNC2013SUBMISSION.pdf

[4] Slides on Netflix CDN: http://www.slideshare.net/adrianco/netflix-global-cloud

[5] Peers of Netflix in New York: https://openconnect.netflix.com/peeringLocations/

[6] Spinnaker http://googlecloudplatform.blogspot.com/2015/11/Netflixs-Spinnaker-available-now-on-Google-Cloud-Platform.html

[7] http://techblog.netflix.com/2015/11/global-continuous-delivery-with.html

[8] New optimization Netflix has come up with to reduce bit rate (they started testing this only last week or so):