Understanding and Improving Large-scale Content Distribution

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

Vijay Kumar Adhikari

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Prof. Zhi-Li Zhang

September, 2012

Acknowledgements

I would like to thank my adviser, Prof. Zhi-Li Zhang, for his continuous support throughout my time at school.

Dedication

To Sharada, Sanchita and Sambriddhi

Abstract

The Internet architecture was not designed for delivering large-scale content. Therefore, with the increase in popularity and demand for videos on the Internet, video content providers(CPs) have to come up with a number of solutions to achieve high degree of scalability, resilience and performance requirements of video content distribution. In this thesis we aim to answer the following research questions: (a) how do large scale content distribution systems currently work and what problems do they encounter, and (b) how can we solve those problems both in the short term as well as in the long term. Towards this end, this thesis makes the following contributions:

First, we study original YouTube architecture to understand how a video delivery system with small number of large data centers handle scalability and performance challenges. Specifically, we uncover the use of location-agnostic proportional load-balancing strategy and how that affects its ISPs (Internet service providers).

Second, we investigate how a more distributed approach employed by current YouTube improves the resilience of its delivery system. Using active measurement study, we uncover the use of multiple namespaces, tiered cache hierarchy, dynamic and location aware DNS (Domain Name System). Although this approach improves the resilience and performance compared to location-agnostic approach, since YouTube uses its own content delivery infrastructure, it is likely encounter scalability challenges as its content size and popularity increases.

Third, to complement the two in-house content distribution architectures, we study Netflix and Hulu. These services make use of multiple third party content delivery networks (CDNs) to deliver their content. We find that their CDN selection and adaptation strategies lead to suboptimal user experience. We then propose inexpensive measurement-based CDN selection strategies that significantly improve the quality of the video streaming. Additionally, we find that although CDN networks themselves might be well designed the CDN selection mechanism and "intelligence" of the client software can be improved upon to provide users with better quality of service.

Finally, building upon the results of these and other recent works on understanding large scale content distribution systems, we propose a first step in the direction of an

open CDN architecture that allows for better scalability and performance. The two key ingredient of this proposal are to let any willing ISP to participate as CDNs and instrument client software to make decisions based upon measurements. This proposal is incrementally deployable. It is also economically more sustainable as it opens up new sources of revenue for the ISPs. We also provide a proof of concept implementation for this architecture using PlanetLab infrastructure.

Contents

Acknowledgements **i**

Dedication **ii**

Abstract **iii**

List of Tables **ix**

List of Figures **x**

1 Introduction **1**

1.1 Bibliographic Notes 2

2 Content distribution system design challenges **3**

2.1 Scalability 3

2.2 Resiliency 4

2.3 Performance 5

2.4 Visibility 5

2.5 Goals specific to types of content 6

2.6 Tools/protocols 6

2.7 System components 6

2.7.1 Locating content/user video interaction 7

2.7.2 Delivery 7

3 Original YouTube **9**

3.1 Introduction 10

3.2 Overview & Related Work 14
3.2.1 A Quick Overview of YouTube 14
3.2.2 Datasets and General Statistics 15
3.2.3 Related Work 16
3.3 Classification & Localization of YouTube Servers 17
3.3.1 Classifying YouTube Server IP Addresses 17
3.3.2 Locating YouTube Servers 19
3.4 Load Balancing among YouTube Data Centers 20
3.4.1 Proportional Load Balancing of Client-to-YouTube Traffic 20
3.4.2 Load Balancing among Three Web Servers 22
3.4.3 Proportional Load Balancing of YouTube-to-Client Traffic 23
3.4.4 Proportional Load Balancing and "Size" of Data Centers 24
3.4.5 Traffic Volume Asymmetry 26
3.5 PoP-Level YouTube Traffic Matrices 27
3.5.1 Single Preferred Exit in Client to YouTube Traffic 28
3.5.2 High Locality-bias in YouTube to Client Direction 29
3.6 Estimating unseen traffic 30
3.7 "What if" analysis 33
3.8 Summary 37

4 Current YouTube **38**
4.1 Introduction 39
4.2 Measurement and Data 40
4.3 Video Id Space & Namespace Mapping 41
4.4 Cache Namespaces & Hierarchy 43
4.5 Video Delivery Dynamics 45
4.5.1 Locality-aware DNS Resolution 46
4.5.2 Handling Cache Misses via Backend Fetching 48
4.5.3 HTTP Redirections Dynamics 48
4.5.4 Delay due to Redirections 50
4.6 Summary 50

5 Netflix and Hulu **52**
5.1 Netflix . . . 53
5.2 Netflix video streaming platform . . . 55
5.2.1 Overview of Netflix architecture . . . 55
5.2.2 Servicing a Netflix client . . . 57
5.2.3 Manifest file analysis . . . 60
5.2.4 CDN selection strategy . . . 62
5.3 CDN Performance Measurement . . . 63
5.3.1 Overall CDN performance . . . 66
5.3.2 Daily bandwidth variation . . . 67
5.3.3 Variation in instantaneous bandwidth . . . 68
5.4 Alternate Video Delivery Strategies . . . 68
5.4.1 Room for improvement . . . 71
5.4.2 Measurement based CDN selection . . . 71
5.4.3 Using multiple CDNs simultaneously . . . 73
5.5 Hulu . . . 73
5.5.1 Rate and CDN adaptation . . . 73
5.5.2 Status Reporting . . . 74
5.5.3 CDN Selection Strategy . . . 75
5.5.4 CDN Server Distribution . . . 79
5.6 Related Work . . . 82
5.7 Summary . . . 83

6 Open CDN Architecture **84**
6.1 Introduction . . . 84
6.2 An open CDN architecture . . . 86
6.2.1 Motivation for open CDN architecture . . . 86
6.2.2 Requirements . . . 88
6.2.3 Description of the proposed architecture . . . 88
6.2.4 Other components . . . 91
6.3 A proof of concept implementation . . . 92
6.3.1 Overview . . . 92

6.3.2 Content provider 93
6.3.3 ISPs 94
6.3.4 Clients 94
6.4 Preliminary evaluation 95
6.5 Summary 96

7 Conclusion and Discussion **99**
7.1 Future directions 101

References **102**

List of Tables

3.1 Summary Statistics for Dataset I 16
3.2 Number of video servers 20
3.3 Summary of traffic estimation 33
4.1 Youtube *Anycast* (first five) and *Unicast* (last two) Namespaces. 43
5.1 Key Netflix Hostnames 55
5.2 CDN servers for Hulu 80
6.1 Fields in table maintained by CP control server 93

List of Figures

3.1 YouTube video delivery framework. 14
3.2 Packet size distribution. 18
3.3 Traffic to YouTube data centers. 21
3.4 Client traffic distribution 22
3.5 Load balancing among front-end servers 22
3.6 Traffic from data centers to clients 24
3.7 Traffic from video servers to PoPs 24
3.8 Traffic between clients and data centers 25
3.9 Traffic proportions extracted using SVD 25
3.10 Distribution of ratios 27
3.11 Entry PoPs to exit PoPs traffic distribution 28
3.12 Servers to exit-PoP traffic distribution 30
3.13 Distribution of IP prefixes 32
3.14 Estimated unseen traffic for IP prefixes 33
3.15 Traffic at YouTube data centers 34
4.1 Number of videos mapped to each *lscache* hostname. 42
4.2 YouTube namespace hierarchy and redirection order. 42
4.3 YouTube Architectural Design. 43
4.4 Geographical distribution of YouTube Video Cache Locations. 44
4.5 Locality aware DNS mappings for *anycast* hostnames. 46
4.6 DNS resolution over time 47
4.7 Fetch time distribution at a YouTube cache server. 47
4.8 Comparison of redirection probabilities. 49
4.9 An example distribution of video initialization time. 51

5.1 Netflix architecture . . . 56
5.2 Timeline in serving a Netflix client . . . 56
5.3 CDN list in manifest file . . . 59
5.4 Video downloadable for one quality level . . . 60
5.5 CDN switching . . . 62
5.6 Best CDN at each vantage point . . . 63
5.7 CDF of average bandwidth at PlanetLab nodes . . . 64
5.8 Average bandwidth at PlanetLab nodes over the entire period . . . 64
5.9 Average bandwidth at residential networks over the entire period . . . 64
5.10 Coefficient of variance for the one-day average at PlanetLab nodes . . . 67
5.11 One-day average bandwidth at a PlanetLab node over time . . . 67
5.12 One-day average bandwidth over time at residential site 7 . . . 67
5.13 One-day average bandwidth over time at residential site 9 . . . 69
5.14 Instantaneous bandwidth at a PlanetLab node . . . 69
5.15 Instantaneous bandwidth at residential site 7 . . . 69
5.16 Instantaneous bandwidth at residential site 9 . . . 70
5.17 Average bandwidth and the upper bound at residential sites . . . 70
5.18 Average bandwidth and the upper bound at PlanetLab nodes . . . 70
5.19 Effect of number of measurements . . . 72
5.20 Best CDN vs three combined CDNs for residential hosts . . . 72
5.21 Best CDN vs three combined CDNs for PlanetLab nodes . . . 72
5.22 A section of Hulu manifest file . . . 76
5.23 CDN preference change in a short interval . . . 77
5.24 Overall CDN preference distribution . . . 77
5.25 CDN preference from geographic regions . . . 78
5.26 CDN preference for different videos . . . 78
5.27 CDN preference over time . . . 79
6.1 An open CDN architecture . . . 87
6.2 A proof of concept implementation . . . 92
6.3 Number of changes in best ISP . . . 97

Chapter 1

Introduction

Since the current Internet substrate was not primarily designed for large scale content distribution, content providers had to bring forth a number of different solutions such as geographically distributed large data centers, multi tiered caches, use of third party content distribution systems (CDNs), and use protocols such as DNS in ways not originally intended for. However, not much effort has been put forward to understand what the fundamental challenges that large scale content distributors face today in terms of storing, locating and disseminating the content. Similarly, equally important questions such as how those challenges are currently being addressed and how we can design networks that are more tailored to the needs of such large scale content distribution has not been studied sufficiently.

The single largest form of large scale content is Internet video. Unlike other content services, video exerts huge loads on network. Video delivery systems are also very complex distributed systems. In addition to networks, video requires massive amount of storage, streaming servers and other resources. Therefore, we primarily direct our focus to understanding the most popular and large Internet video content delivery systems.

Specifically, we want to examine challenges that needs to be addressed by the architects of large-scale content delivery systems and study how they address those challenges. For instance, because of the enormity of the size of the content, they can not be replicated everywhere. Since different content might be cached at different locations, how does a front end server (which has to produce such URLs for the clients) know which server will be able to serve the requested content. What happens if the selected server

is overloaded or simply failed? How should a content distribution system react to such scenarios? The system also needs to make a choice of how to store the content in different locations such that they can be quickly located and delivered so that the user perceived performance metrics such as the startup time do not degrade.

With these goals in mind, we study the following systems: (a) original YouTube, (b) current YouTube, (c) Netflix and (d) Hulu. We carefully make this selection to cover the largest of such systems and also cover most representative infrastructures and streaming technologies. In this thesis, we present a comparison of those systems, identify the common challenges they face, and propose an open CDN architecture that aims to address those challenges.

1.1 Bibliographic Notes

Part of the content of Chapter 3 is from a conference paper, titled *YouTube traffic dynamics and its interplay with a tier-1 ISP: an ISP perspective.*, which appeared in the Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement 2010, Melbourne, Australia - November 1-3, 2010[1]. The active measurement based study of current YouTube architecture is presented in a paper titled *Vivisecting YouTube: An active measurement study.*, which appeared in the Proceedings of the IEEE INFOCOM 2012, Orlando, FL, USA, March 25-30, 2012[2]. This constitutes a part of Chapter 4 which describes the current YouTube architecture. Parts of the content in Chapter 5 are taken from two conference papers. A study of Netflix architecture is from a paper titled *Unreeling Netflix: Understanding and improving multi-CDN movie delivery* which appeared in the Proceedings of the IEEE INFOCOM 2012, Orlando, FL, USA, March 25-30, 2012[3]. The second paper that constitutes parts of Chapter 5 is titled *A tale of three CDNs: An active measurement study of Hulu and its CDNs.* This paper appeared in the Proceedings IEEE INFOCOM Workshops, Orlando, FL, USA, March 25-30, 2012[4]. The Open CDN architecture which constitutes parts of Chapter 6 is proposed in a technical report titled *Feel free to cache: enabling ISPs for content distribution* [5].

Chapter 2

Content distribution system design challenges

The challenges in developing large scale content distribution system is enormous. From the amount of content being created to the high quality of such content that require huge storage and equally huge bandwidth to be served, video delivery faces a number of scalability and performance challenges. In this section, we explore the high level design issues faced by large scale content distributors and discuss possible ways to address them.

2.1 Scalability

Video content distribution requires extremely high-level of scalability. The number of devices connected to the Internet, the number of users and their demand for Internet videos and the quality levels and length of video content have been steadily growing over time. Internet video already covers more than one third of the Internet traffic and recent projections estimate it to be more than 9/10 of all Internet traffic in a few years. In particular, YouTube now serves more than 2 billion views per day and Netflix is the single largest source of Internet traffic in the US, consuming 29.7% of peak downstream traffic. The scalability problems not only arise from the increase in videos and viewers but also from the increase in quality levels and the length of videos. For instance, Netflix offers feature-length movies in HD quality. Distributing HD-quality programming to a large audience requires tens of petabits per second of capacity[6]. Therefore, achieving

very high scalability should be one of the most important design goals for any large scale video content delivery system.

Large-scale content distribution has very specific scalability challenges. The first one relates to naming and storing of the content. As the number of available content items grow, these pieces of content have to be named (either using public external names of names used only internally) so that they can be located easily. Additionally, because of the enormity of the size of the content, they can not be replicated everywhere. This requires a careful placement of popular vs non-popular content objects. This has to be additionally complimented either by application level redirections or background fetch to deal with cache misses. One possible approach could be to divide the content space logically into multiple groups.

Similarly, most content (or individual pieces of such content) have to be addressed in some URL. URLs generally consists of a hostname and a path for the content. The most important problem related to these URLs is how to identify exactly what hostname to include in that URL. Since different content might be cached at different locations, how does a front end server (which has to produce such URLs for the clients) know which server will be able to serve the requested. For instance, should the content providers have a fixed set of servers serving a fixed set of content? Or should arbitrary servers be allowed to serve any content and the front-end just maintains a highly dynamic database of servers and the contents that they can serve? This problem gets further complicated when a content provider uses multiple third party CDNs.

2.2 Resiliency

In addition to being scalable, a large-scale video delivery system must also be resilient to server, network and other failures. Consumers of popular content expect almost 100% reliability and availability of the system. Failure to serve customers even for a short time will have serious negative impacts on brand reputation. Therefore the system must be developed in such a way that it degrades and handles failures gracefully.

Because of the scale of the system, not all content can be replicated everywhere. Although the front-end server might be able to provide the client with a URL for a piece of content, what happens if the server is overloaded or simply failed. How should a content

distribution system react to such scenarios? Should it rely on redundancy so that even if a machine behind an IP address fails, there are others to take over the responsibility? Should the system rely primarily on indirection provided by DNS and just start giving new IP addresses for the server hostname? Similarly, large scale content distribution also needs to handle cases of flash crowds. For instance, the system can be designed such that the load is evenly distributed across all the available servers irrespective of the locality of the request. Or the system can be designed rely on frequently changing DNS mappings or application-level redirections.

2.3 Performance

Another very important design goal for the system design must be high performance. A sluggish experience will be very detrimental to customer loyalty. The system must be designed in such a way that users can locate and enjoy the videos as fast as possible. For instance, the server that serves a user should be fairly close to the user in terms to network latency to avoid a high startup time. There are multiple possible approaches that can be applied to improve the startup delays. For instance, the content provider can have deeply penetrated cache locations so that users are close to at least one of the cache locations. Other approach could be to leverage the commercial CDNs which already have a very distributed cache presence.

Additionally, because of the extremely large scale of the video content, not all content can be replicated everywhere. The system needs to make a choice of how to store the content in different locations such that they can be quickly located and delivered so that the user perceived performance does not degrade.

2.4 Visibility

Related to the performance goals is the design goal of high visibility of the entire system. Content providers need to make sure the system allows them to see what the user is experiencing. The system should allow for collection of performance data at the client machine, cache server and data center locations. This allows the content providers to react to problems quickly in the short term as well as planning in the longer term.

2.5 Goals specific to types of content

In addition to aforementioned general goals for large scale content delivery systems, there are additional design challenges based upon the type of content being served. Although all content distribution systems would like to be resilient and efficient, they might differ in some of the specifics. For instance, delivery of user generated short videos is fundamentally different than delivery of feature length professional feature-length movies. For instance, the popularity distribution of these two types of content will be so different that this affects how each of these types of content can be cached. Additionally, there are security and privacy challenges in distributing professional content as compared to the distribution of user generated content and the system designers need to keep these differences in mind while developing such systems. For instance, short videos can opt for easy progressive download approach, whereas long high-quality movies have to use some form of chunked delivery system.

2.6 Tools/protocols

Ideally, the high level design of a content delivery system should be independent to the protocols or tools used to deliver and play the content. In practice, however, the protocols/tools used to deliver the content also guide the architecture of the delivery system. For instance, due to the availability of a simple redirection mechanism, HTTP based protocols are better suited for content distribution systems that want to rely on redirections to handle flash crowds and cache misses. Similarly, delivery systems using chunked delivery have more flexibility in selecting the initial server than the delivery systems that use progressive download to transfer the entire video in one transaction.

2.7 System components

We explored the high-level design goals for a content distribution system. In this section, we will discuss some of the specific services that the system needs to provide to users when developing such systems.

2.7.1 Locating content/user video interaction

The content distributor needs to design the distribution system in such a way that it is easy for users to locate and interact with the content. A content distribution containing huge amount of popular content will not find much use if users cannot easily locate the content. Therefore, the system must allow for easy ways to search and locate the content. This implies that the each piece of content must be addressed carefully. In addition to the user interface allowing users to search contents using different criteria, the system must allow for efficient naming mechanism to be used internally behind the scene. In common scenarios, users visit a front-end web server to search and locate a piece of content they are interested in. The front-end web server then serves a page that has pointers to the location of the content at a video server.

In addition to locating content, there are other ways in which users might want to interact with the content. That includes rating and commenting of the content. In this thesis, however, we will not be focusing on these two aspects of system design.

2.7.2 Delivery

The most interesting and challenging part of designing a large scale content distribution systems is concerned with "delivery" of the content. It includes decisions such as which content is to be stored where, how many data centers or cache locations to deploy and how to decide which server to select to serve any given request. Since, it is not feasible to store all the content at all locations, designers of such large scale content delivery systems need to decide how to distribute the content to different locations. Generally, a centralized service is established as a know all service that can tell which content is available at what location. Careful mapping of content namespace and server namespace can simplify the task of identifying servers best suited to serve any given content.

Additionally, the content delivery system should also be able to handle adverse phenomena such as flash crowds. Ideally, the system should be able to direct users to the server best suited for the given content at any given time. In cases where its not practical or possible to do so, the system should allow the users to be referred to a server that is more likely to be able to serve it. These mechanisms are typically deployed using

clever DNS mappings and application level redirections. DNS mappings that can be updated faster need fewer such redirections, and similarly DNS mappings that live longer generally necessitates more frequent application level redirections.

Chapter 3

Original YouTube

In this chapter we conduct an extensive and in-depth study of the original YouTube. We study traffic exchanged between YouTube data centers and its users, as seen *from the perspective of a tier-1 ISP* in Spring 2008 after YouTube was acquired by Google but before Google did any major restructuring of YouTube. Using flow-level data collected at multiple PoPs of the ISP, we first infer where the YouTube data centers are located and where they are connected to the ISP. We then deduce the load balancing strategy used by YouTube to service user requests, and investigate how load balancing strategies and routing policies affect the traffic dynamics across YouTube and the tier-1 ISP.

The major contributions of the chapter are four-fold: (1) we discover the surprising fact that YouTube does not consider the geographic locations of its users at all while serving video content. Instead, it employs a location-agnostic, *proportional* load balancing strategy among its data centers to service user requests from all geographies; (2) we perform in-depth analysis of the PoP-level YouTube traffic matrix as seen by the ISP, and investigate how it is shaped by the YouTube load balancing strategy and routing policies utilized by both YouTube and the ISP; (3) with such knowledge, we develop a novel method to estimate *unseen* traffic so as to "complete" the traffic matrix between YouTube data centers and users from the customer ASes of the ISP; and 4) we explore "what if" scenarios by assessing the pros and cons of alternative load balancing and routing policies. Our study sheds light on the "small number of big data centers" architecture employed by the original YouTube. It also highlights the interesting and important interplay between large content providers and ISPs in today's Internet.

3.1 Introduction

A significant portion of today's digital multimedia content is serviced by large content providers such as YouTube (now a subsidiary of Google), Yahoo, Google, and the like. These large content providers often employ several huge data centers – each of which is comprised of thousands of servers – to meet and serve growing user demands. For a variety of reasons, these data centers are typically located in different geographical sites, and connected to one or multiple ISPs at the nearby major Internet "interconnection regions" or Points-of-Presence (PoPs)[1] . In other words, the content serviced by a large content provider typically flows from one of its data centers through one of these PoPs to enter these ISPs, and is then carried to eventually reach various users. The dynamic *inter-dependence* and *interplay* between content providers and ISPs raise many interesting and important questions that have not been adequately studied.

Take YouTube as an example. As the most popular video sharing site on the Internet, YouTube attracts millions of users every day. Given the significant amount of traffic generated by YouTube videos, how YouTube manages its traffic dynamics and performs load-balancing among its data centers will have a considerable impact on the traffic flows across and within ISPs. Further, the (BGP) routing policies employed by both YouTube and ISPs also shape and drive the traffic dynamics between YouTube and those ISPs. Such dynamic *inter-dependence* and *interplay* therefore have significant implications in traffic management to both YouTube and the ISPs. For instance, can an ISP effectively estimate YouTube-specific traffic matrix so as to better manage its traffic dynamics?

In this chapter we take a *measurement-oriented* approach to study the YouTube traffic dynamics *from the perspective of a tier-1 ISP*, with emphasis on the *interplay* between the two players, its effect on the traffic dynamics across them, and implications in traffic management for both players. Our study is based on sampled flow-level data collected at various PoPs of a tier-1 ISP.

From BGP routing tables, YouTube (AS36561) advertises 3 prefixes, namely, 64.15.112.0/20, 208.65.152.0/22, and 208.117.224.0/19. Using this information, we first

[1] For example, according to [7], there are eight major "interconnection regions" within the United States – namely, New York, Washington D.C. and Atlanta on the east coast, Dallas and Chicago in the central US, and Los Angles, the Bay Area and Seattle on the west coast – at which content providers typically buy transit from or peer with various ISPs.

extract all flows related to YouTube from the ISP flow-level measurement data. Through reverse DNS look-ups and other analysis, we infer and deduce that YouTube employs *seven* data centers[2] to service user demands, and that it is connected to the tier-1 ISP at six out of the eight Internet "interconnection regions" (or PoPs) mentioned in [7].

Using the extracted YouTube traffic, we perform an extensive and in-depth analysis of the traffic dynamics between YouTube and the ISP, and explore how load balancing strategies and routing policies employed by both players drive and affect the traffic dynamics between them. In the following we provide a brief overview of the major observations and contributions of our study along four inter-related lines of investigations.

Location-Agnostic Load Balancing: We analyze and infer the load balancing strategies used by YouTube to service user requests. We find that YouTube employs a *proportional* load balancing strategy among the seven data centers (as opposed to, say, a locality- or proximity-aware strategy), where the proportionality seems to be determined by the "size" of the data centers, e.g., as measured by the number of public IP addresses seen in the data that are associated with (front-end) video servers at each data center. This proportionality stays fairly constant over time and across different geographical locations (PoPs) of the tier-1 ISP, and holds for both the client-to-YouTube and YouTube-to-client traffic. *This finding is of particular interest, as it provides an important contrast to the findings of several earlier studies on CDNs [8, 9], which show the prevalence of* proximity-based *content distribution among several CDN services.*

Prevalence of Early-Exit Routing in the ISP Network: With knowledge of the YouTube load balancing strategy, we examine how the YouTube traffic flows between various PoPs of the ISP and the six PoP locations where YouTube is connected to the ISP. In other words, we estimate and analyze the *PoP-level YouTube traffic matrix* as seen by the ISP. Due to the routing asymmetry, we consider the client-to-YouTube and YouTube-to-client traffic separately. We find that the PoP-level *client-to-YouTube* traffic matrix is highly *geographically biased* in the sense that the client-to-YouTube traffic originated from a source PoP always leaves the ISP and enters YouTube at the "closest" of the six destination PoPs, *regardless of* which YouTube data center the traffic

[2] As inferred based on DNS names as well as traceroutes, the seven YouTube data centers are located respectively in New York City, Ashburn (near Washington D.C.), Miami, Chicago, Dallas, Los Angles, and the San Jose Bay Area. YouTube is connected to the tier-1 ISP in question at six PoP locations: New York City, Washington D.C., Chicago, Dallas, Los Angles, and the San Jose Bay Area.

is destined to. For instance, the client-to-YouTube traffic originated at a source PoP near New York City, despite being proportionally load balanced across seven YouTube data centers, always exits the ISP and enters YouTube at the New York PoP itself. This observation suggests that YouTube somehow has to carry the client traffic from New York across its "internal" network to the destination data center. On the other hand, the *YouTube-to-client* traffic originated from six YouTube data centers always enters the ISP at the corresponding PoP that they are closest to. For instance, the YouTube-to-client traffic originated from the YouTube New York data center always enters the ISP at the New York PoP. In particular, the YouTube-to-client traffic from the Miami data center is *not* carried by the ISP, i.e., "unseen" by the ISP. This suggests that the YouTube-to-client traffic from the Miami data center is carried by another ISP to the clients. The asymmetry in the client-to-YouTube and YouTube-to-client traffic matrices can be attributed to the BGP routing policies such as "early exit" and route preferences set by the ISP and YouTube.

Estimating Unseen Traffic: Despite the highly asymmetric nature of the client-to-YouTube and YouTube-to-client traffic, the proportionality of traffic split across YouTube data centers (when seen by the ISP) still holds, even when we zero in on a specific customer AS (or prefix) of the ISP. This observation leads us to develop a novel method to estimate the *unseen* YouTube traffic (traffic that is being carried outside of *ISP-X* network), and "complete" the YouTube traffic matrices between YouTube data centers and client ASes of the ISP. This ability to estimate unseen traffic is useful and important both in theory and in practice: It allows us to infer and estimate the total traffic demand matrices between YouTube and various customer ASes of the ISP; it also enables an ISP to estimate the "new" traffic demand and prepare for the potential impact on its network, when routing changes lead YouTube to re-route this unseen traffic through the ISP instead.

Better Understanding of Interplay through "What If" Analysis: Using the YouTube traffic matrices estimated above, we investigate several "what if" scenarios to examine the pros and cons of different load balancing strategies and routing policies, and the impact of the resulting traffic dynamics on both YouTube and the ISP. For example, we find that due to uneven traffic demand distribution across the geographical locations, a locality-aware data center selection strategy [8], while may reduce the overall

video download latency for users, can also lead to un-balanced loads across different data centers, especially when the capacities of data centers do not match the geographical load distribution. This plausibly explains why YouTube prefers a location-agnostic, proportional load balancing strategy, as server-load and network bandwidth are likely more critical than latency to the performance of video download/streaming services. We also explore how selective route announcements can shift traffic between YouTube and the tier-1 ISP, and how the ISP can prepare for such changes in YouTube traffic dynamics using the unseen traffic estimation.

To our best knowledge, this chapter provides the first extensive study of traffic exchanged between YouTube data centers and users, as seen *from the perspective of a large (tier-1) ISP*. More importantly, while YouTube is one of many (albeit one of great importance) large content providers on the Internet, our work provides a valuable case study that demonstrates the importance of understanding the interplay between large content providers and ISPs. In particular, our study illustrates that the load balancing strategies employed by a content provider (be it proportional load-balancing, locality-aware, latency-aware, or other more sophisticated strategies) *and* (BGP) routing policies can have a significant impact on the traffic dynamics across content providers and ISPs, and have important implications in traffic management and performance engineering for both content providers and ISPs. With the increasing prominence of data centers in content delivery and emergence of cloud computing, we believe that the interplay between large content (or application service) providers and ISPs will have broad ramifications in a number of key areas such as traffic engineering, network management, business agreements and economic models. Our work is only a first attempt in addressing this broad set of questions.

The remainder of the chapter is structured as follows. In Sec. 3.2 we overview the YouTube video delivery framework and datasets used in our study. In Sec. 3.3 we present the method used for inferring YouTube data centers, and in Sec. 3.4, we analyze and deduce the load balancing strategies used by YouTube among its data centers. In Sec. 3.5 we study the YouTube traffic matrices as seen by the ISP, and investigate the effects of load balancing and routing policies. In Sec. 3.6 we present a novel method for estimating the *unseen* traffic, and in Sec. 3.7 we explore "what-if" scenarios. The chapter is concluded in Sec. 3.8.

3.2 Overview & Related Work

3.2.1 A Quick Overview of YouTube

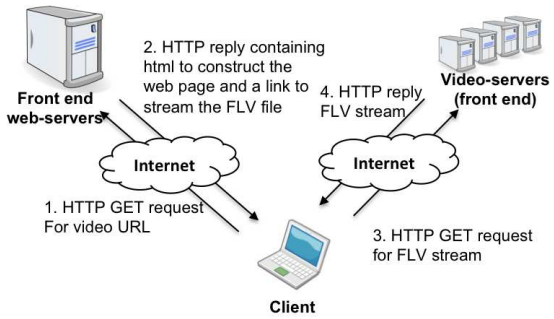


Figure 3.1: YouTube video delivery framework.

YouTube is the most popular video serving website that serves user-generated and other videos. It lets any user upload videos in a number of formats. YouTube then converts them in flash-video format and makes them available for viewing. Users with a browser that have Adobe Flash player (or some other compatible player) can watch those videos on YouTube.com or some other sites that embed those videos.

Fig. 3.1 schematically depicts a typical sequence of steps involved when a user watches a YouTube video. When a user goes to `www.youtube.com` to watch a video, a HTTP request is sent to one of the servers corresponding to `www.youtube.com`, which we refer to as the `(front-end) web-servers`, and a web page is returned that also contains a place-holder for the video. When the user clicks on the video or when the video starts playing automatically, the front-end web server instructs the browser (or the flash plug-in inside the browser) to stream video from another server within one of the YouTube data centers, which actually serves the Flash video content. We refer to these servers as the `(front-end) video-servers` – they are publicly visible in the sense each is assigned a public IP address (and a DNS name). Hence, to watch any YouTube video, a user's browser typically has to communicate with both one of the three front-end web servers and one of many (front-end) Flash video-servers. All parts of this communication - including the video streaming - happens over HTTP.

3.2.2 Datasets and General Statistics

In our study we use sampled Netflow records collected by a tier-1 ISP, which we will refer to as *ISP-X*, at various PoPs in the US and Europe. The collected flow data is augmented with BGP routing updates and other relevant information such as *AS Paths*, source and destination *IP prefixes* advertised by the neighbor ASes, *egress router* from which the destination prefix is learned (hence this is the router at which the flow exits *ISP-X* network), and so forth. The augmented routing information allows us to attribute the source and destination IP addresses contained in the IP headers to the source/destination ASes and the (specific) prefixes they own and announce via BGP. For our study, we use two flow datasets, each collected at 44 PoPs in US and Europe of the ISP for three consecutive days in two separate weeks in Spring, 2008. Using these datasets, we then extract all YouTube-related traffic, i.e., all flows containing IP addresses belonging to the 3 prefixes announced by YouTube.

Table 3.1 summarizes the basic statistics for the first dataset, where the statistics for the *client-to-YouTube* traffic is listed in the second column, and those for *YouTube-to-client* traffic in the last column. (The statistics for the second dataset is similar; we omit them here for lack of space.) In total, there are approximately 172 million sampled flows containing YouTube IP addresses, out of which 54 million flows are from clients (users) to YouTube, and 118 million flows from YouTube to clients. There are no flows from YouTube to YouTube traffic. The difference in the numbers of flows from clients to YouTube vs. YouTube to clients is partly due the *traffic asymmetry* in the two directions: the amount of traffic sent from YouTube to clients are in general far larger than the amount of traffic sent from clients to YouTube. (This traffic asymmetry also manifests in the disparate byte counts of flows in each direction, not shown here, see Sec. 3.5.) Another contributing factor is the effect of *routing asymmetry*: we see that the number of BGP prefixes for client IPs and the number of client AS numbers in the YouTube-to-client traffic are more than double of those in the client-to-YouTube traffic. Despite the routing and traffic asymmetries, there are ∼3,000 ASes which are seen in both client-to-YouTube and YouTube-to-client traffic, and there are ∼10,000 BGP advertised prefixes that are seen in both directions. Hence these client ASes (or prefixes) use *ISP-X* for connecting to YouTube in both directions. In addition to the aforementioned augmented flow datasets, we also conduct reverse DNS look-ups and

traceroutes to classify YouTube traffic, infer and discover the locations of YouTube data centers, etc. We also collect several gigabytes of *tcpdump* [10] data by playing hundreds of randomly selected YouTube videos on a couple of our lab machines. Collected data includes IP, TCP, HTTP headers and as well as first 200 bytes of the payload. The *tcpdump* data is used to confirm and validate the YouTube traffic characteristics inferred from the sampled Netflow records.

Table 3.1: Summary Statistics for Dataset I

	to YouTube	from YouTube
Total flows	$\sim 54 \times 10^6$	$\sim 118 \times 10^6$
Client IP addresses seen	$\sim 10 \times 10^6$	$\sim 20 \times 10^6$
BGP advertised client prefixes	$\sim 28,000$	$\sim 61,000$
Number of client ASes	$\sim 4,000$	$\sim 9,000$
YouTube IP addresses seen	$\sim 2,000$	$\sim 2,000$

3.2.3 Related Work

Existing studies of YouTube traffic characteristics have either focused on user behaviors or relied on data collected at end hosts or edge networks. For example, the authors in [11] examine video access patterns, user behavior, popularity life cycle of videos, etc., and compare these characteristics with the traditional web, whereas in [12] the authors explore the "social networks" of YouTube videos. The studies in [13, 14] utilize packet traces collected at campus networks to study the YouTube traffic characteristics from the perspective of an edge network. In a more recent work [15], Fahmy *et al.* analyze and compare the underlying distribution frameworks of three video sharing services, YouTube. The focus of the work is to investigate the variation in service delay experienced by users in different geographical locations, and when accessing videos of different popularity and ages. In contrast, using the flow-level data collected at multiple PoPs of a tier-1 ISP, we study the interplay between YouTube and the ISP by investigating the effects of load balancing strategies and routing policies (of both YouTube and the ISP) on the YouTube traffic dynamics. Our work also differs from existing studies [16, 17, 18] on characterizing large-scale *Content Distribution Networks* such as Akamai. With increasing reliance on large data centers for content delivery, our work sheds light on the interesting and important interplay between large content providers and ISPs.

3.3 Classification & Localization of YouTube Servers

As explained in Sec. 3.2.1, YouTube has a two-tier video delivery framework, where servers can be divided into mainly two categories, *front-end* and *video-servers*. Although it might not be very difficult to get a list of data centers YouTube uses, it is a non-trivial task to identify the geographic locations and the roles played by individual YouTube servers in the video delivery framework. In this section we use the Netflow data along with DNS records to classify YouTube server IP addresses into these categories. We also use another dataset collected using the *tcpdump*, by playing a large number of videos on couple of client machines, to validate the classification. We explain how we map YouTube server IP addresses to different locations using reverse DNS mapping and traceroutes from multiple vantage points.

3.3.1 Classifying YouTube Server IP Addresses

We see a total of 2093 YouTube IP addresses in the datasets, and perform reverse DNS look-ups on each of them. Using the host names thus resolved, we classify YouTube IP addresses and corresponding flows into three categories: i) 3 YouTube IP addresses (i.e., *front-end web servers*) whose host names are all mapped to `www.youtube.com`; the corresponding flows are referred as the *web front-end* flows. ii) more than 2000 YouTube IP addresses (83% of the total YouTube IP addresses) whose host names are of the format `loc-`v`xx.loc.youtube.com` (e.g., `dal-`v`26.dal.youtube.com`), where `loc` is one of the 8 city codes, `ash, chi, dal, lax, mia, nyc, sjc, sjl`, and `xx` is an integer. We deduce and later validate that these host names correspond to *front-end (Flash) video servers* within various YouTube data centers. iii) The remaining IP addresses are either mapped to host names with other formats (roughly 5%), containing keywords such as *smtp*, *dns, db, webcam*, or are not resolvable, i.e., with no public host names (roughly 11%). For flows associated with these IP addresses, many of them are either UDP, or are associated with TCP port numbers other than 80. Hence we deduce that these flows are unrelated to YouTube video delivery, and hence ignore them in the remainder of our study. For flows belonging to the first two categories, we further classify them based on the directions of the flows: *client-to-YouTube* for traffic sent from the clients to YouTube servers, or *YouTube-to-client* for the traffic in the other direction.

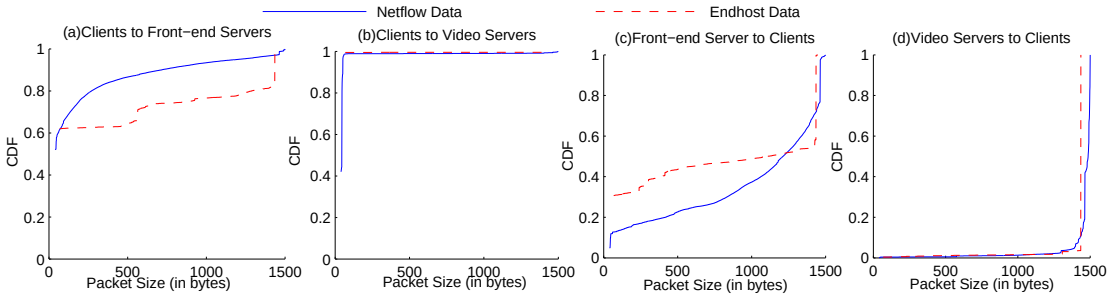


Figure 3.2: Packet size distribution.

Validating YouTube Server IP Address Classification using *tcpdump* Data. To validate the YouTube IPs and flow classification presented above, we also collect *tcpdump* data (packet traces) at end hosts by playing hundreds of randomly selected YouTube videos on these hosts. By examining the payload, we obtain the *ground-truth* regarding the role of the YouTube IP addresses seen in the *tcpdump* data. As above, we classify the flows into four (sub-)categories and characterize the overall traffic characteristics of each category: *i) Clients to (web) front-ends:* These packets carry the HTTP requests to front-ends for downloading various objects in the web-page. These packets are likely to be smaller but of variable sizes. *ii) Clients to Video servers:* These packets mainly carry the acknowledgments for the data sent by the video servers. These packets generally contain a smaller number of bytes. *iii) Front-ends to clients:* These flows carry packets mostly with HTML and javascripts etc which describes the downloaded web-page. These packets are likely to be larger packets, with varying number of bytes because of the varying sizes of the web objects. *iv) Video servers to clients:* These flows carry the video content, therefore are likely to be containing a constant but large number of bytes.

For flows of each category, we obtain the overall traffic characteristics of the flows obtained using the *tcpdump* data collected at the end hosts, and compare them with those obtained flows within the same category in the ISP (sampled) flow datasets. As an example, in Fig. 3.2, we plot the cumulative distribution of the packet sizes of flows in each category computed using the *tcpdump* data as well as the same distribution computed using the ISP datasets. The distribution of the former is shown as the dashed line, and the latter the solid line. We see that the two distributions in each category match quite well, and are visibly distinct from those in the other categories. These

results corroborate and confirm that the YouTube IP address and flow classification presented is indeed valid.

3.3.2 Locating YouTube Servers

The host names of video server IP addresses discovered through reverse DNS look-ups seem to suggest that there are eight data centers. We perform traceroutes from multiple vantage points to geo-locate and verify the locations for the IP addresses. We find that traceroute queries corresponding to the IP addresses located in the same city produce the same last-hop IP addresses. This traceroute-based analysis also reveals that the two city codes `sjc` and `sjl` point to the same (data center) location, namely within the San Jose Bay Area. We therefore group these two together as a single data center location. This yields a total of seven data centers, as shown in Table 3.2. The last column of the table also provides the percentage of video server IP addresses belonging to each data center, as seen in the ISP datasets.

We further associate the data center locations (city codes) with the PoP locations of the ISP. More specifically, we compare these origin PoP locations for each YouTube IP address with our 3-letter city code based classification. Except for those YouTube video server IP addresses containing the city code `mia`, in all other cases flows containing (as the source IP address) YouTube video server IP addresses originate from either a source PoP with the same city code or from a source PoP that is nearby. For instance, `nyc-v24.nyc.youtube.com` appears as a source IP address only in flows from New York PoP of the ISP, whereas `ash-v22.ash.youtube.com` appears as a source IP address only in flows from Washington D.C. PoP of the ISP. Furthermore, flows with the source IP addresses containing the city code `sjc` and `sjl` are only seen at the San Jose Bay Area PoP of the ISP. This analysis also reveals that YouTube is connected with the ISP at six PoP locations, Washington DC, New York City, Chicago, Dallas, Los Angles and San Jose.

Lastly, in terms of the three front-end web server IP addresses, they only show up in flows (as source IP addresses) originating from the San Jose PoP. Moreover, other YouTube IP addresses within the same /24 block are also mapped to the San Jose PoP. Traceroute analysis also reveals that these three front-end web server IP addresses are located within the San Jose Bay Area. Hence we deduce that all three front-end web

servers are located within the San Jose Bay Area.

Table 3.2: Number of video servers

Code	City	% of IP addresses
ash	Ashburn, VA	24.77
chi	Chicago, IL	17.27
dal	Dallas, TX	07.89
lax	Los Angeles, CA	17.33
mia	Miami, FL	7.37
nyc	New York City	7.26
sjc, sjl	San Jose Bay Area	18.07

3.4 Load Balancing among YouTube Data Centers

In this section we analyze and infer the load balancing strategy used by YouTube among its data centers to service user requests. The key discovery we make is that YouTube employs a *proportional* load balancing strategy among the seven data centers and not based on the geographical proximity to clients, where the proportionality seems to be determined by the "size" of the data centers; this is in contrast to several studies [8, 9] on CDNs which typically exploit *proximity-based* server selection strategy to reduce end-to-end latency[3] .

3.4.1 Proportional Load Balancing of Client-to-YouTube Traffic

We now examine how client-to-YouTube traffic is distributed among the seven YouTube data centers. Our initial hypothesis is that YouTube does proximity-aware content distribution meaning clients are served videos primarily from the data-center that results in lowest latency. To our surprise, we found that this was not the case. For instance, when we examined a "/24" prefix belonging to New York Public Library, it was exchanging only about 7% of the total flows with YouTube NYC data-center. Similarly, another prefix owned by Illinois Century Network was only exchanging 19% of its traffic with YouTube Chicago data-center. It was also very clear that the flows were not

[3] In Sec. 3.7 we will discuss the pros and cons of using *proportional* load-balancing vs. *proximity-aware* load-balancing strategies. We also note that the authors in [8] find that simply relying on *proximity* for serving client requests lead to poor CDN performance for some users; the quality of paths should be also taken into account.

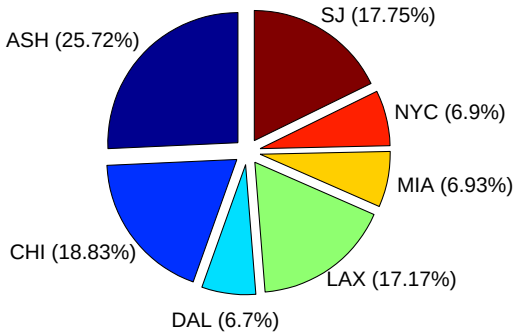


Figure 3.3: Traffic to YouTube data centers.

being equally divided among data-centers because YouTube Ashburn data-center was receiving the highest share of traffic from both the networks and was significantly higher from traffic shares that other data-centers were getting. When we examined the traffic distribution more closely, we noticed that the traffic was being distributed at a fixed proportion to YouTube data-centers from all the *ISP-X* PoPs irrespective of which PoP was geographically closer to which data-center.

The distribution of the overall client-to-YouTube traffic is shown in Fig. 3.3. We can see that the client-to-YouTube traffic is clearly not equally divided among the data centers. In particular, we see that the ASH data center receives slightly more than a quarter of the traffic, while the LAX, CHI, and DAL data centers receive about 17% of the total traffic. On the other hand, the DAL, MIA and NYC data centers receive only $\sim$7% of the total traffic each. This is very similar to what we observed in case of the two prefixes we discussed above.

Fig.3.3 shows how the total traffic is divided. Next, we look at how traffic coming from individual source PoPs of *ISP-X* is distributed. In Fig. 3.4, we show how client-to-video server traffic is distributed among 7 YouTube data centers for traffic originating at each *ISP-X* PoP. In this figure x-axis shows different PoP for *ISP-X*, and y-axis shows the fraction of total traffic going to different YouTube data centers. As seen in this figure, the fraction of traffic received by each YouTube data center from each PoP is the same for all PoPs. It implies that although PoPs of *ISP-X* and YouTube data centers are located at geographically diverse locations, they do not have any preferences to each

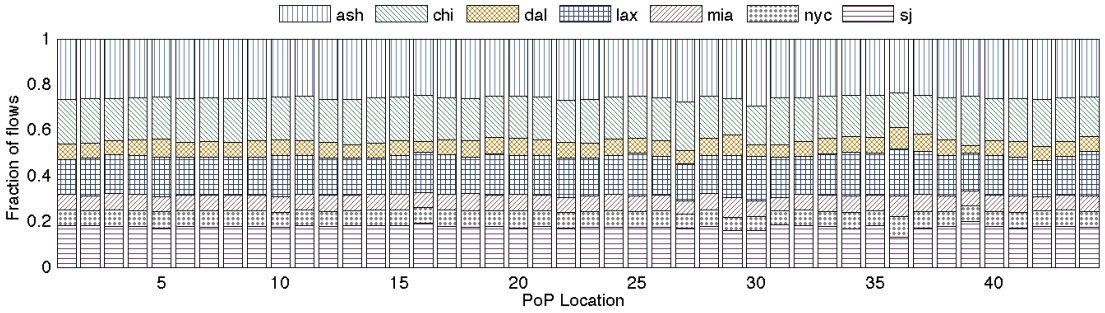


Figure 3.4: Client traffic distribution

other in terms of network/geographic "proximity". More specifically, traffic coming from clients is distributed to 7 YouTube data centers according to *fixed proportions*. We have also verified that these proportions hold at various other levels such as over time, at AS level, and individual IP prefix level (plots not shown due to space limitations).

3.4.2 Load Balancing among Three Web Servers

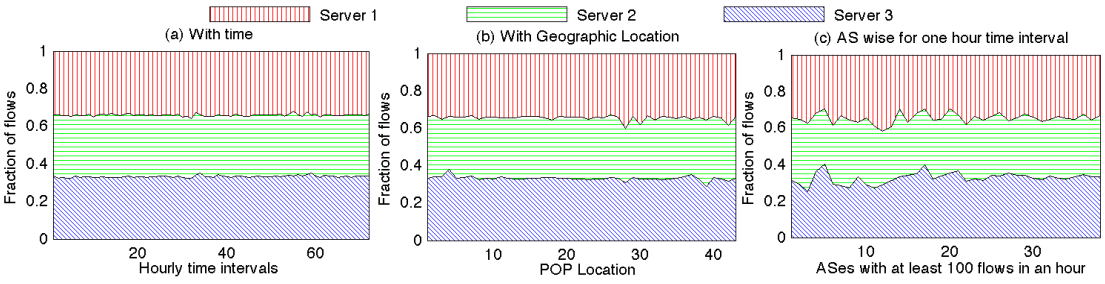


Figure 3.5: Load balancing among front-end servers

We now analyze how YouTube performs load balancing among the three front-end web servers. Since all the web servers are located inside San Jose data-center, all the traffic to front-end web servers have to come to San Jose. An interesting but not so surprising discovery is that the traffic is equally distributed among those 3 front-end server IP addresses. This observation holds true at different time granularity, e.g., over different hours or over different days, as well as across different spatial granularity, e.g., at the level of individual PoPs of the ISP or with respect to individual client ASes. These observations are shown in Fig. 3.5, where Fig. 3.5(a) plots the traffic distribution

among the three front-end web servers over different hours, Fig. 3.5(b) plots the traffic distributions among the three front-end web servers at various PoPs of the ISP, and Fig. 3.5(c) plots the traffic distributions among the three front-end web servers at various (large) client ASes of the ISP. That the traffic is equally distributed among the front-end web servers is expected, as YouTube front-ends employs the round-robin DNS resolution [19] for mapping `www.youtube.com` to the IP addresses of front-end web servers.

3.4.3 Proportional Load Balancing of YouTube-to-Client Traffic

In the discussion so far we used only one direction of the traffic. i.e, traffic going from clients to YouTube data centers. Our findings show that this traffic is proportionally distributed among various YouTube data centers. In the following, we analyze the traffic from the reverse direction, i.e., from YouTube data centers to clients. Comparing distributions in these two directions is inherently challenging because of asymmetric routing in the Internet. Routing in the Internet is not symmetric, that is the network path taken by the traffic going from a client to one of the YouTube data centers may not be the same as the path taken by the reply traffic coming from the YouTube data center to the client. Furthermore, since the network flow data available to us is only collected at various PoPs of the *ISP-X*, it may not have the YouTube to client traffic for some of the client-to-YouTube traffic, as the reply traffic from YouTube can go through different ISPs. Therefore, in order to analyze how the traffic from YouTube data centers to various *ISP-X* PoP is distributed, we only consider the IP prefixes for which we see the traffic coming from all the YouTube data center locations. In addition, some of the YouTube data centers may not use *ISP-X* to send the reply traffic to customers at all. In our data, we do not find any reply traffic from Miami YouTube data center. Hence we ignore traffic from Miami YouTube data center in the following analysis.

Fig. 3.6 shows how much traffic is sent by each of the YouTube data center to clients. As seen in this figure, the largest fraction of video traffic is sent by the YouTube ASH data center.

Next, we investigate if the proportional traffic distribution holds for the YouTube data center-to-client traffic at the PoP level. As mentioned before due to asymmetric routing in Internet, for this analysis, we consider traffic for only those IP prefixes which receive traffic from all the YouTube data centers (except Miami data center) through

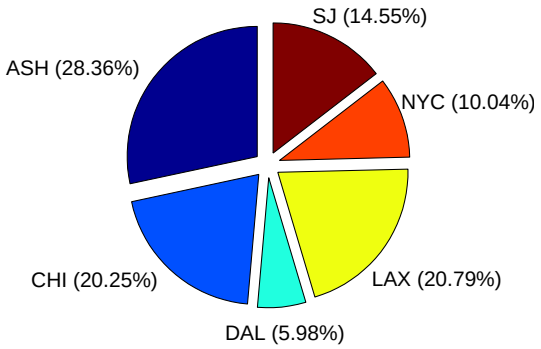


Figure 3.6: Traffic from data centers to clients

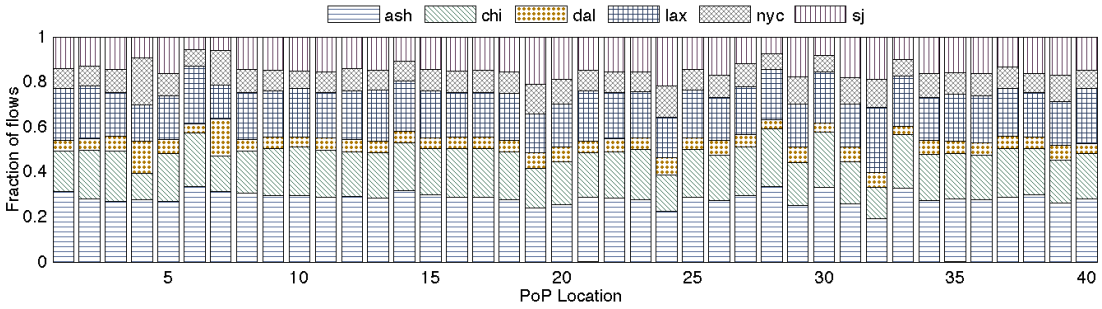


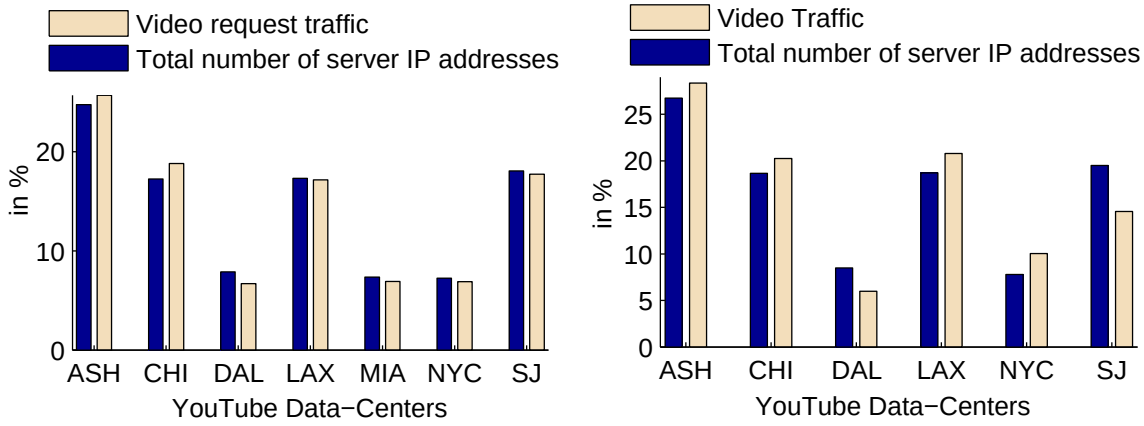
Figure 3.7: Traffic from video servers to PoPs

ISP-X. Furthermore, we group all the IP prefixes using *ISP-X* PoPs. For this we assign a PoP to each IP prefixes, which is the PoP from where traffic with the corresponding IP prefix enters in the ISP network. We plot the fraction of traffic sent by each data center to various *ISP-X* PoPs in Fig. 3.7. In this figure different destination *ISP-X* PoPs are shown on the x-axis, and y-axis shows the fraction of the traffic as number of sampled flows received by the PoP, from a given YouTube data center. As seen in this figure, traffic from different YouTube data center to *ISP-X* PoP is distributed proportionally. These proportions are the same as the total proportions shown in Fig. 3.6.

3.4.4 Proportional Load Balancing and "Size" of Data Centers

Our analysis has shown that YouTube does not do a proximity-aware load balancing. Instead, it does load balancing among its data centers in a fixed proportions. As we show

next, the distribution of incoming video-requests to its data centers based on the "size" or the "resources" available at those data centers. Since we do not have any access to internal system design of each YouTube data center, we use the number of IP addresses seen at each location as the rough estimate of the size of a data center. We plot the fraction of traffic received by each YouTube data center along with the fraction of IP addresses seen from different data centers in Fig. 3.8(a). As we see in this figure, the amount of traffic received by each data center is directly proportional to number of IP addresses seen at that location. Furthermore, Fig. 3.8(b) shows that fraction



(a) Client to YouTube traffic

(b) YouTube to client traffic

Figure 3.8: Traffic between clients and data centers

of traffic sent by different YouTube data centers to *ISP-X* PoPs, is proportional to the fraction of total number of IP addresses seen at the corresponding data center. It shows that *YouTube performs load-sharing among its data centers based on their "size", not location.*

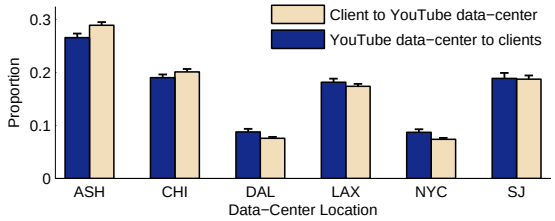


Figure 3.9: Traffic proportions extracted using SVD

Extracting proportions using SVD: Next, we use the rank-1 approximation using SVD[4] [20] to estimate the proportions used to distribute the traffic among different data centers. For this analysis, we construct the traffic matrices C and S for client-to-YouTube and YouTube to-client traffic respectively. Here C_{ij} is the traffic originating from ith /24 IP prefixes which is destined to jth data center, and S_{ij} is the traffic received by ith prefix from jth data center. It is possible that some IP prefixes may not be using *ISP-X* network to reach all the data centers and vice versa, therefore, for the unbiased estimate of proportions, we ignore the /24 IP prefixes which do not send data to all the data centers for the traffic matrix C, and prefixes which do not receive data from all the data centers for the traffic matrix S. Since we do not see traffic coming from YouTube Miami data center, we ignore it in this analysis so that we can compare the proportions seen in both directions of the traffic. We construct a large set of matrices c and s by randomly selecting 1000 rows from matrices C and S respectively to verify if the proportions hold for all the prefixes for different random selection of IP prefixes. We perform the SVD based rank-1 approximation on matrices c and s. Fig. 3.9 shows the proportions estimated for both client-to-YouTube and YouTube-to-client traffic using matrices c and s. In this figure, black horizontals line on top of each bar show the 95% confidence interval for the mean values shown in the figure. As seen in this figure, traffic is distributed in the same proportions for both matrices, with very little variances.

3.4.5 Traffic Volume Asymmetry

As we have already observed, the amount of client-to-YouTube flows is much smaller compared to YouTube-to-client flows(see Sec. 3.3). In this section we further analyze the YouTube-to-client and client-to-YouTube traffic ratios to get a better insights of the YouTube traffic. For this analysis, we define "two-way traffic pairs" as $\langle c2s(p, dc), s2c(p, dc)\rangle$ for different /24 IP prefixes p and YouTube data centers dc. We use term $c2s(p, dc)$ to represent the traffic sent by IP prefix p to dc data center of YouTube. Here we are interested in learning how the ratios $r(p, dc) = \frac{s2c(p,dc)}{c2s(p,dc)}$ are distributed.

Due to asymmetric routing in Internet some of the two-way traffic pairs may contain

[4] Albeit the proportionality can also be verified directly, the SVD automatically yields the proportionality (or a rank-1 matrix approximation) that best fits the data.

zero values. Therefore we only consider the pairs with non-zero values in our analysis. Fig. 3.10(a) shows the distribution of $r(p, dc)$ for all such two-way traffic pairs for the number of bytes exchanged between IP prefix p and data center dc, and Fig. 3.10(b) shows the distribution of $r(p, dc)$ using number of flows. As seen in this figure, the ratio of the bytes exchanged between IP prefixes and data centers stays between values 40 and 80 for more than 80% of the pairs, on the other hand the ratio of the flows exchanged is most of the times close to 2. *It demonstrates that traffic exchanged between IP prefixes and data centers is fairly consistent, and can be used for estimating the YouTube traffic that is not seen in the ISP-X network.* (See Sec. 3.6).

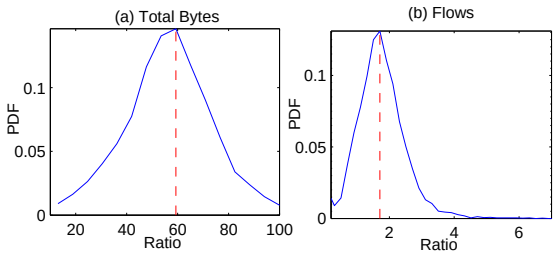


Figure 3.10: Distribution of ratios

3.5 PoP-Level YouTube Traffic Matrices

In Section 3.4, we made the observation that the YouTube traffic from clients at all locations are divided in a fixed proportion irrespective of whether a data-center is nearer to a client or very far from it. We also saw that the traffic from the data-centers to the clients also follow similar proportional distribution. This behavior of end-to-end traffic matrices naturally raises the question of whether the traffic also gets divided in a similar way inside the *ISP-X* backbone. In this section we look at the *entry PoP to exit PoP traffic-matrix* from the perspective of *ISP-X* with the objective to better understand how the traffic enters *ISP-X* network and gets divided to different exit PoPs. We present the similarities and differences between those two matrices below.

To study this interaction, we generate *entry-exit* traffic matrices based on the entry and exit PoPs for the YouTube traffic from the perspective of *ISP-X*. We have following two entry-exit matrices based on the direction: (i) from clients to YouTube servers, and

(ii) from YouTube servers to clients.

3.5.1 Single Preferred Exit in Client to YouTube Traffic

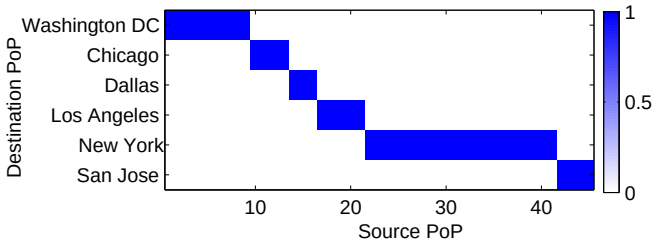


Figure 3.11: Entry PoPs to exit PoPs traffic distribution

For the clients to YouTube servers traffic we look at how much traffic enters from an *ISP-X* PoP and which PoP it exits from. We see that the traffic exits from only 7 of the *ISP-X* PoPs where YouTube is present. We also see that for each of the 44 PoPs from which the client to YouTube traffic enters *ISP-X*, all of the traffic from that PoP exits *ISP-X* from exactly one of the 7 exit PoPs. If we represent the entry PoPs as columns of the traffic matrix and exit PoPs as the rows, we see that each column has exactly one non-zero entry. A closer inspection of this matrix reveals that for any PoP P, it has a single exit-pop E_P such that E_P is the geographically closest *ISP-X* PoP where YouTube is present. Clearly, for the seven PoPs where YouTube data centers are present, $P = E_P$.

Next, we compare this entry-exit traffic matrix with the "end-to-end traffic demand and supply" matrices from YouTube's perspective seen in Sec. 3.4. We see that they are significantly different. For instance, the end-to-end traffic matrix which shows how the total traffic to YouTube video-servers are divided among different data centers, is a rank-one matrix, and follows the gravity model[21]. On the other hand the entry-exit traffic matrix seen by *ISP-X*is a full-rank sparse matrix where each column has exactly one non-zero row. From the point of view of *ISP-X*, traffic flows do not seem to be getting divided to different locations.

To understand the reason behind this difference between the two traffic matrices, we examined BGP routing information. From the BGP data, we see that YouTube aggregates all of its IP addresses in all the data centers in only 3 IP prefixes: 64.15.112.0/20,

208.65.152.0/22, and 208.117.224.0/19. All of these three prefixes are then announced to *ISP-X* at each of the 7 PoPs where YouTube is present. This allows *ISP-X* to do an early exit (or hot-potato) routing [22]. Since all the YouTube prefixes are announced from each of the locations where YouTube is present, *ISP-X* can transfer the YouTube-bound traffic at the exit PoP that is nearest to the source PoP irrespective of the final destination data center for the traffic.

Fig. 3.11 shows how the traffic entering from different *ISP-X* PoPs exits the ISP network. As we can see PoPs 1 to 9 send all of their YouTube-bound traffic towards Washington DC PoP. Similarly, there are 20 PoPs for which New York is the nearest exit PoP to reach YouTube. It shows that each *ISP-X* PoP has exactly one preferred exit PoP for all the YouTube traffic, irrespective of the final destination data center. We use this mapping of source PoPs to their nearest exit-PoP in Sec. 3.7 for some of the "what if" analysis.

Traffic carried by YouTube: The early-exit routing employed by *ISP-X* has another implication in terms of how the traffic reaches the final destination data center. Since *ISP-X* hands over all the YouTube-bound traffic to YouTube at local PoPs, YouTube has to deliver this traffic to the final destinations using its own "internal" network.

To estimate the amount of client to YouTube traffic that YouTube has to deliver to its different data centers, we look at how much traffic *ISP-X* hands over to YouTube at each location and how much of it is destined to the local YouTube data center. For instance, from all the clients in nearby PoPs, if *ISP-X* hands over 100 units of traffic to YouTube at New York PoP, then 7%, 7%, 17%, 7%, 18%, 16% and 17% of it are destined to New York, Miami, Los Angeles, Dallas, Chicago, Ashburn and San Jose data centers respectively. We see that only 7 units of traffic is actually delivered locally to New York data center, while remaining 93 units of the traffic still has to be carried somehow to other data centers. Overall, we see that only ∼10% of the total traffic remains local, while remaining ∼90% of the client to YouTube traffic is delivered to the final destination data centers by YouTube using its "internal" network.

3.5.2 High Locality-bias in YouTube to Client Direction

Although YouTube uses a proximity-agnostic mechanism to load balance the traffic between YouTube servers and client, the entry-exit traffic matrix at PoP level shows strong

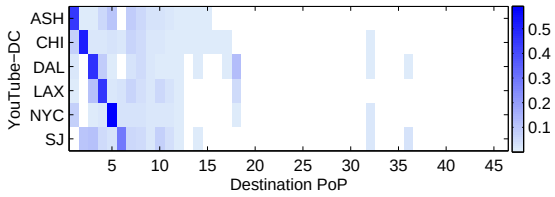


Figure 3.12: Servers to exit-PoP traffic distribution

locality bias. In Fig. 3.12, each row represents the distribution of traffic originating from a YouTube data center, which enters the *ISP-X* network through the corresponding local PoP, to all the PoPs as a fraction of the total traffic coming from that data center. Also, in this figure first 6 destination PoP are the ones where the six data centers connect to the ISP in the same order as YouTube data centers shown in y-axis. This figure shows from each PoP from which YouTube to client traffic enters *ISP-X*, about 40% to 60% of the traffic exits *ISP-X* network from the local PoP itself, and only the remaining traffic enters the *ISP-X* backbone.

Not surprisingly, early-exit routing is again the reason behind high locality bias in the YouTube to client traffic. When we look at the BGP information available in the flows, a very large number of prefixes are announced from multiple *ISP-X* PoPs. Therefore, in many cases *ISP-X* selects an exit PoP from a list of PoPs where the destination prefix is announced and in general it prefers the PoP that is geographically closest to the entry PoP.

These results show that even when the content-providers do not do geography-aware load balancing, large ISPs still can still employ early-exit routing and carry less traffic in their backbone.

3.6 Estimating unseen traffic

As seen in Sec. 3.4, the proportionality of traffic split across YouTube data centers (as seen by the ISP) holds at various levels such as over different time intervals or at individual AS or IP prefix level. Furthermore, we showed that ratio of the traffic exchanged between YouTube data centers and individual client prefixes is fairly stable (See Fig. 3.10). These observations led us to develop a method to estimate the *unseen*

and *partially seen* YouTube traffic – namely the (portion of) traffic carried by other ISPs – and "complete" the traffic matrix between YouTube data centers and users from the customer ASes of the ISP.

Basically we have two ways of estimating the unseen traffic: a) based upon the proportional distribution among data-centers, and b) based upon the ratio of client-to-YouTube and YouTube-to-client traffic. These approaches have different error characteristics. The proportions at which the traffic gets distributed to (and from) different data-centers have very small variations compared to the ratio of traffic in two direction (see Figures 3.9 and 3.10). Therefore, we try to use the proportionality of traffic distribution to do the estimate as much as possible. Only when its not possible to use that piece of information (such as when we see no client to YouTube data at all), we make use of the ratio of traffic in different directions.

• **Formulating a matrix completion problem:** To facilitate the discussion in this section, we represent the client-to-YouTube traffic matrix by C, where $C_{i,j}$ = Total bytes from ith IP prefix to jth data center, and the YouTube-to-client matrix by S, where $S_{i,j}$ = Total bytes to ith IP prefix from jth data center. Both C and S have unknown entries representing the traffic carried between corresponding client prefix and YouTube data center. To get a complete picture of how much traffic is actually exchanged between an IP prefix and and a YouTube data center including the traffic not carried by *ISP-X*, we need to estimate all such unknown values. The problem therefore can be viewed as a matrix completion problem where some entries are known and some unknown and we use the known entries to compute the values of the unknown entries.

• **Estimation using proportional distribution:** For our matrix completion problem, we first use the proportions at which the YouTube traffic is divided among its 7 data centers. First, we divide the rows of the matrices C and S to form three sub-matrices each: C^f and S^f which consist of all the rows for which all the column entries are known, C^p and S^p which consist of all the rows for which at least one column entry is known, and C^0 and S^0 which consist of all the rows for which none of the column entries are known.

Next we use the proportion $x_1 : x_2 : x_3 : x_4 : x_5 : x_6 : x_7$ (extracted in Sec. 3.4) to fill the missing entries of C^p as follows. For each row, we first get the known values, say $C^p_{i,j_1}, C^p_{i,j_2}, \ldots, C^p_{i,j_m}$ where $1 \leq m \leq 7$. Each of the unknown entry $C^p_{i,k}$ in that row

can be estimated as $\frac{(C^p_{i,j_1}+C^p_{i,j_2}+...+C^p_{i,j_m})\times x_k}{(x_{j_1}+x_{j_2}+...+x_{j_m})}$. We apply the same mechanism to fill the missing entries in S^p by using the proportions.

For filling the values in C^0 and S^0 we use a slightly different approach. Since we do not have any known entries in any of the rows, we exploit the mean ratio between client-to-YouTube and YouTube-to-client traffic to estimate one entry per row for C^0 using the corresponding non-zero value from S^p or S^f. Since all of the IP prefixes in C and S must have either sent or received traffic from YouTube data centers through *ISP-X*, for each completely unknown row in C, the corresponding row in S must have at least one known entry. For each row in C^0, we pick one known field from the corresponding row in S and divide it by the mean ratio to get the corresponding column in C^0 and vice versa for S^0. Finally we combine the rows of C^f, C^p and C^0 to get the completed matrix C' for C, and S' by combining S^f, S^p and S^0 for in-complete initial traffic matrix S.

• **Results:** For the estimate of "unseen" traffic, we first extracted all the IP prefixes from the Netflow data, which were seen either in client-to-YouTube traffic or YouTube to-client traffic or both. There are around 150,000 such IP prefixes in our Netflow data. Using these prefixes we prepared matrices C and S by extracting the total number of bytes exchanged by these prefixes with different YouTube data centers.

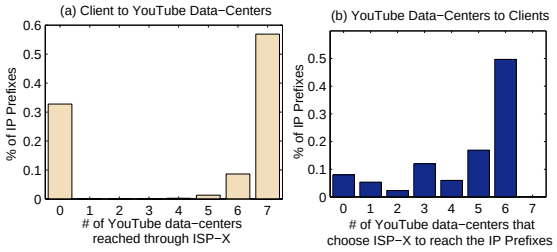


Figure 3.13: Distribution of IP prefixes

Fig. 3.13 shows the distribution of IP prefixes seen in the YouTube traffic going through the *ISP-X*'s network, on the basis of the total number of YouTube data centers reached by them using *ISP-X*. In Fig. 3.13(a) we see that more than 50% prefixes use *ISP-X* to reach all 7 YouTube data centers, on the other hand less than 50% prefixes received traffic from 6 YouTube data centers through *ISP-X*. These figures show that there is a large amount of unseen traffic for these IP prefixes which is delivered through other ISPs.

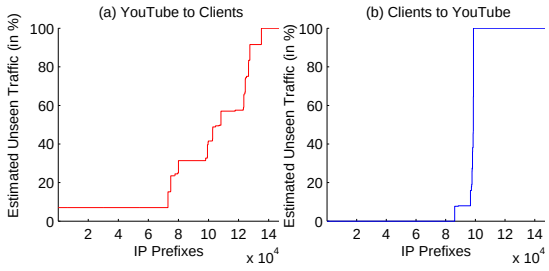


Figure 3.14: Estimated unseen traffic for IP prefixes

Table 3.3: Summary of traffic estimation

	to YouTube	from YouTube
Total seen traffic	0.59	39.7
Estimated unseen traffic	0.34	20.4

We show the distribution of estimated unseen traffic for IP prefixes using our methodology in Fig. 3.14. In this figure, x-axis shows the IP prefixes and y-axis shows the corresponding unseen traffic as a fraction of total traffic received or sent by these prefixes to all the YouTube data centers. Moreover, for the client-to-YouTube data centers traffic we see that 36.5% of the estimated total traffic is not seen by *ISP-X*. Similarly, for the YouTube-to-client traffic 34% of the estimated total traffic is unseen. Overall the percentage of unseen traffic is 34.1% of the estimated total traffic. We summarize these results in Table 3.3.

We conclude from above that *ISP-X* does not see a huge portion of the traffic exchanged by its customer ASes with YouTube. Therefore, any changes in the routing policy used by its customer ASes and YouTube can have significant impact on the *ISP-X*'s network. Furthermore, this estimation of the unseen traffic can now enable the *ISP-X* to estimate the "new" traffic demand and prepare for the potential impact on its network, when routing changes lead to YouTube to re-route unseen traffic through the ISP instead.

3.7 "What if" analysis

In Section 3.4, we observed that YouTube does a location-agnostic proportional load-balancing. We also saw in Section 3.5 that YouTube announces all-inclusive large prefixes

from all the locations where it peers with *ISP-X*. Finally, we were able to estimate some of the traffic that is exchanged between clients and YouTube that *ISP-X* does not see. These results give us insights on what is happening now. However, since the dynamics of the network change over time, it is natural to ask what happens if things change. In this section we use the insights gained about YouTube and its traffic characteristics with respect to *ISP-X* to investigate some of the "what if" scenarios to examine the pros and cons of different load-balancing strategies and routing policies, and the impact of the resulting traffic dynamics on both YouTube and the ISP.

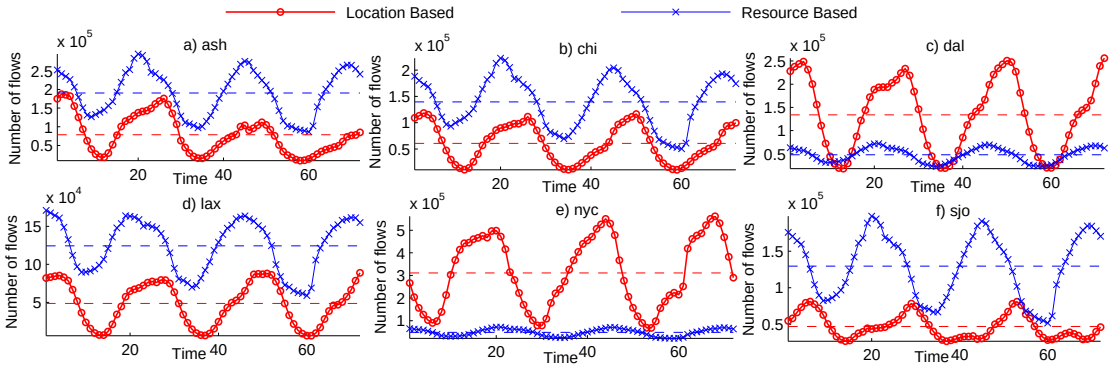


Figure 3.15: Traffic at YouTube data centers

• **Scenario 1: What if YouTube employs a location aware load balancing among its data centers?**

We investigate the scenario when instead of being location-agnostic, YouTube employs a "location based" load balancing scheme. One way YouTube can achieve this is by configuring its front-end web servers in such a way that they direct the clients' video requests to a data center that is nearest to the location of the requesting client.

We already know when the client to YouTube traffic enters *ISP-X* network from a PoP it exits from exactly one exit PoP where YouTube is attached. We will use this information to assign each PoP to its nearest YouTube data-center. For instance, YouTube data center near the New York PoP will be the only data center that will deliver video traffic to all the PoPs that use New York PoP as the exit PoP for YouTube bound traffic. We then evaluate how it affects the *ISP-X*'s traffic matrix and also the load at different YouTube data centers.

Effect on the ISP traffic matrix: Since all the client PoPs will fetch videos from

the nearby YouTube data center in this scenario, the traffic that is carried in the ISP backbone will be reduced significantly. For instance, when we computed the amount of traffic that remains local at the source PoP under this approach, we see that about 44.5% of overall YouTube to client traffic does not even enter the ISP backbone compared to the same number for location-agnostic load balancing. Moreover, it eliminates most of the "long-distance" communication for the ISP. For example, Los Angeles data center will never have YouTube to client traffic going to Chicago PoP under the location based load balancing. *Hence, location based load balancing will have a positive impact on ISP-X's PoP level traffic matrix, since it reduces the traffic on ISP-X's backbone network significantly.*

Poor resource utilization under geographical load-balancing. Although geographical load-balancing helps in reducing the amount of traffic carried in the *ISP-X*'s backbone, it results in poorer resource utilization and increased peak load at each data center.

To analyze the impact of location based load balancing on YouTube data centers, we compute the amount of video requests served by each data center during different time of the day. We estimate the amount of video requests served by each data center by rearranging the client to YouTube data center matrix on the basis of nearest PoP. In Fig. 3.15, we compare the loads at six data centers under location based and resource based load balancing schemes. In each figure here, x-axis represents the time and y-axis represents the total number of flows received by the corresponding data center at different time. We observe that the load on New York and Dallas data center increases significantly as they are the nearest data centers for a large number of *ISP-X* PoPs.

Furthermore, let us assume that the maximum number of flows each data center receives in any hour indicates the load at that data center. Let L_i^j indicate the number of flows received by data center i in hour j, then we define the load $L_{max}(i), i = 1, 2, \ldots, 6$ as, $L_{max}(i) = \max_{j=1\ to\ 72} L_i^j$. Under these assumptions, the sum of peak load at each data center ($ML = \sum_{i=1}^{6} L_{max}(i)$) will represent the amount of computing resources needed during the peak load time to serve all the client requests. If we compare MLs under the two schemes, we see that the total units of resources that are needed in case of location based load balancing is about 30% higher than that of the current resource based load balancing.

The reason for this increase is the following. Since the peak load time for client requests coming from different PoP is different, the peak load at one of the PoPs of *ISP-X* is proportionally shared by all the data centers in the location agnostic load balancing. However, in case of location based load balancing it does not allow the distribution of peak load, and all the load accumulates at the nearest YouTube data center.

In summary, we see that while location based load balancing may reduce the load on *ISP-X*'s backbone network, it results in the poor resource utilization and creates more "hot-spots" for YouTube by increasing the load on the individual data centers during the peak load hours.

• **Scenario 2: What if YouTube only announces IP prefixes specific to each data center at the corresponding location where it exchanges traffic with *ISP-X*?**

Since YouTube announces larger all-inclusive prefixes at each location where it exchanges the traffic with *ISP-X*, the ISP can do early exit routing and dump all YouTube-bound traffic to YouTube at the nearest exit (see Sec. 3.5). This forces YouTube to carry that traffic internally to its different data centers. In this section, we investigate how traffic matrices will change if YouTube only announces specific prefixes from each of its location.

To see how the traffic matrix for *ISP-X* changes if YouTube starts announcing only the specific local prefixes at each location, we divided YouTube IP addresses in smaller prefixes so that each prefix has IP addresses belonging to only one YouTube data center. We combined smaller prefixes to form larger prefixes as long as the larger prefixes included IP addresses from a single data center. From the three large prefixes we obtained 2 /25s, 9 /24s and 5 /23s where each of the prefixes belonged to exactly one data center.

If YouTube announces only the specific prefixes from each location corresponding to the local data center, then *ISP-X* will have to carry the traffic corresponding to each YouTube prefix to the data center where it is present. This means most of client to YouTube server traffic will be carried in the ISP backbone.

However, as we have already seen in Sec. 3.4.5, the ratio between client-to-YouTube and YouTube to client traffic is approximately 1 : 59, the additional volume of traffic that is carried in *ISP-X* backbone will only increase by 2% for the total YouTube traffic.

• **Scenario 3: What if due to BGP routing changes the number of prefixes**

YouTube reaches using *ISP-X* increases?

As seen in the Sec. 3.6, a significant portion of the traffic the customers of *ISP-X* exchange with YouTube is not visible to *ISP-X*. They are likely being carried by some other ISPs. If YouTube changes its routing policy and decides to send all that traffic through *ISP-X*, the amount of YouTube to client traffic handled by *ISP-X* would increase significantly: by about 50% of the current YouTube to client traffic volume. This might also happen if YouTube experiences routing failures in its other providers' network. Since YouTube traffic amounts for about 5.8%, this results in about 2.9% increase in the total traffic carried by the ISP. On the other hand, being able to estimate this unseen traffic accurately, *ISP-X* can prepare and plan for such scenarios in advance.

3.8 Summary

In this chapter, we presented the first extensive study the original YouTube content distribution architecture. We presented an study of traffic exchanged between YouTube data centers and users, as seen *from the perspective of a large tier-1 ISP*. Using our trace-driven analysis of the interaction between YouTube and one of its large transit-providers, we inferred and analyzed the load balancing scheme employed by YouTube and how these choices affect the traffic dynamics across a large content provider and a tier-1 ISP. We further formulated and solved unseen traffic estimation problem that ISPs can use to get an estimate of the "new" traffic demand and prepare for the potential impact on its network, when routing changes lead their to re-route unseen traffic through the ISP instead. We also conducted several "what if" analysis to understand how the load balancing strategies chosen by content providers and routing policies used by the ISPs affect each other.

Chapter 4

Current YouTube

We deduce key design features behind the YouTube video delivery system by building a distributed active measurement infrastructure, and collecting and analyzing a large volume of video playback logs, DNS mappings and latency data. We find that the design of YouTube video delivery system consists of three major components: a "flat" video id space, multiple DNS namespaces reflecting a multi-layered *logical* organization of video servers, and a 3-tier physical cache hierarchy. We also uncover that YouTube employs a set of sophisticated mechanisms to handle video delivery dynamics such as cache misses and load sharing among its distributed cache locations and data centers.

The current YouTube architecture could be considered better than the original one from an ISPs point of view. First, although not always accurate, DNS based localization is helpful. Therefore, most of the time the distance between clients and the content cache in minimized and that reduces the load on ISP backbones as most of the content delivery remains local. However, since YouTube uses its own distribution network, it is likely to encounter scalability challenges as its content popularity increases.

Additionally, HTTP redirections are an integral part of the current YouTube architecture and each redirection takes the user farther away to a more centralized cache location. This also increases load on ISP backbone.

4.1 Introduction

Given the traffic volume, geographical span and scale of operations, the design of YouTube's content delivery infrastructure is perhaps one of the most challenging engineering tasks. Before Google took over YouTube and subsequently re-structured the YouTube video delivery infrastructure, it was known that YouTube employed several data centers in US [1] and used third-party content delivery networks to stream videos to users. While it is widely expected that Google has incorporated the YouTube delivery system into its own infrastructure in the past few years, little is known how Google has re-designed and re-structured the YouTube video delivery infrastructure to meet the rapidly growing user demands as well as performance expectations. This chapter attempts to "reverse-engineer" the YouTube video delivery system through large-scale measurement, data collection and analysis. The primary goal of our study is to understand the *design principles* underlying Google's re-structuring of the YouTube video delivery system. Understanding YouTube is important for future content providers and content delivery system designers, because YouTube video delivery system represents one of the "best practices" in Internet-scale content delivery system. Additionally, because of the significant volume of traffic that YouTube generates, this reverse-engineering work also helps Internet service providers to understand how YouTube traffic might impact their traffic patterns.

The rest of the chapter is organized as follows. We describe the measurement infrastructure and collected data in Section 4.2. In Section 4.3, Section 4.4 and Section 4.5 we present details regarding how we derive our findings, including the analysis performed, the methods used, and additional experiments conducted to verify the findings. We conclude the chapter in Section 4.6.

Related Work. Most existing studies of YouTube mainly focus on user behaviors or the system performance. The authors in [11, 12] examined the YouTube video popularity distribution, popularity evolution, and its related user behaviors. The authors in [13] investigate the YouTube video file characteristics and usage patterns such as the number of users, requests, as seen from the perspective of an edge network.

In [1], Adhikari et al. uncover the YouTube data center locations, and infer the load-balancing strategy employed by YouTube at the time. In [23], the authors examine data

collected from multiple networks to uncover the server selection strategies YouTube uses. To the best of our knowledge, our work is the first study that attempts to reverse engineer the current YouTube video delivery system to understand its overall architecture.

4.2 Measurement and Data

We developed a distributed active measurement and data collection platform consisting of: i) PlanetLab nodes that are used for both crawling the YouTube website and performing DNS resolutions, as well as functioning as proxy servers for YouTube video playback; ii) open recursive DNS servers that are used for issuing DNS requests. Our platform utilizes 471 PlanetLab nodes that are distributed at 271 geographical dispersed sites and 843 open recursive DNS servers located at various ISPs and organizations. We also developed an emulated YouTube Flash video player in Python which performs the two-stage process involved in playing back a YouTube video: In the first stage, our emulated video player first connects to the YouTube's website to download a web page, and extracts the URL referencing a Flash video object. In the second stage, after resolving the DNS name contained in the URL, our emulated video player connects to the YouTube Flash video server thus resolved, and follows the HTTP protocol to download the video, and records a detailed log of the process.

In addition, our emulated YouTube Flash video player can be configured to use an open recursive DNS server (instead of the default local DNS server of a PlanetLab node) for resolving YouTube DNS names. This capability enables us to use the 843 open recursive DNS servers as additional vantage points.

We adopt a multi-step process to collect, measure, and analyze YouTube data. First, we crawl the YouTube website from geographically dispersed vantage points using the PlanetLab nodes to collect a list of videos, record their view counts and other relevant metadata, and extract the URLs referencing the videos. Second, we feed the URLs referencing the videos to our emulated YouTube Flash video players, download and "playback" the Flash video objects from the 471 globally distributed vantage points, perform DNS resolutions from these vantage points, and record the entire playback processes including HTTP logs. This yields a collection of detailed video playback traces. Third, using the video playback traces, we extract all the DNS name and IP

address mappings from the DNS resolution processes, analyze the structures of the DNS names, and perform ping latency measurements from the PlanetLab nodes to the IP addresses, and so forth. Furthermore, we also extract the HTTP request redirection sequences, analyze and model these sequences to understand YouTube redirection logic.

YouTube Videos and View Counts. We started by first crawling the YouTube homepage (www.youtube.com) from geographically dispersed vantage points using the PlanetLab nodes. We parsed the YouTube homepage to extract an initial list of (unique) videos and the URLs referencing them. Using this initial list as the seeds, we performed a breadth-first search: we crawled the web-page for each video from each PlanetLab node, and extracted the list of related videos; we then crawled the web-page for each related video, and extracted the list of its related videos, and so forth. We repeated this process until each "seed" video yielded at least $10,000$ unique videos (from each vantage point). The above method of collecting YouTube videos tends to be biased towards popular videos (at various geographical regions). To mitigate this bias, we take multiple steps. First, we add our own short empty videos to the list. We also search YouTube for different keywords and add to our list only those videos that have very small view counts. After all these steps, we have a list of $434K$ videos (including their *video id*s, the (most recent) *view-count* and other relevant information).

Video Playback Traces and HTTP Logs. Using the list of videos we collected, we fed the URLs referencing the videos to our emulated YouTube Flash video players to download and "playback" the Flash video objects from the 471 globally distributed vantage points. We recorded the entire playback process for each video at each vantage point. This includes, among other things, the DNS resolution mappings, all the URLs, HTTP GET requests and the HTTP responses involved in the playback of each video.

4.3 Video Id Space & Namespace Mapping

YouTube references each video using a unique "flat" *video id*, consisting of 11 *literals*. We refer to the collection of all *video id*s as the *video id space*. While we find that the literals in the first 10 positions can be one of the following 64 symbols: {a-Z, 0-9, _, -}, only 16 of these 64 symbols appear in the 11th (last) position. Therefore, while the size of the YouTube *video id space* is 64^{11}, the *theoretical* upper bound on the number

of videos in YouTube is $63^{11} \times 16$, still an astronomical number. Analyzing the $434K$ *video id*s in our list, we find that they are *uniformly distributed* in the *video id space*.

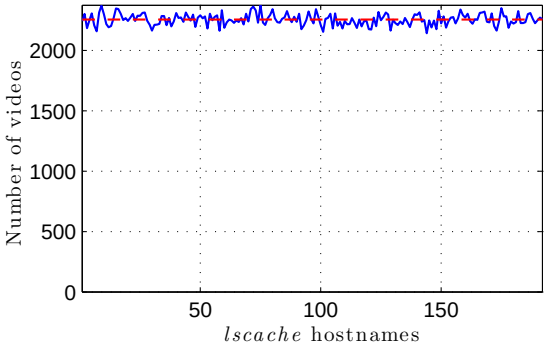


Figure 4.1: Number of videos mapped to each *lscache* hostname.

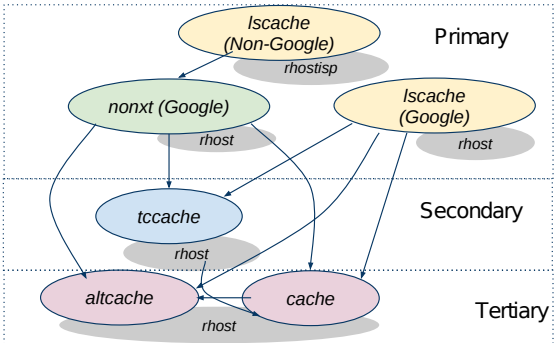


Figure 4.2: YouTube namespace hierarchy and redirection order.

As we show in Table 4.1 and elaborate further in Section 4.4, YouTube employs a number of DNS namespaces. Only DNS names belonging to the *lscache* namespace are generally visible in the URLs contained in the YouTube webpages; DNS names belonging to other namespaces only appear in URLs in subsequent HTTP redirection requests (see Section 4.5.3). We find that each *video id* is always mapped to a *fixed* hostname, out of the 192 possible names (*logical* servers) in the *lscache* namespace, regardless of location and time. For example, a video identified using the *video id* `MQCNuv2QxQY` always maps to v23.lscache1.c.youtube.com *lscache* name from all the PlanetLab nodes at all times. Moreover, when redirection happens, each *video id* is always mapped to a fixed hostname

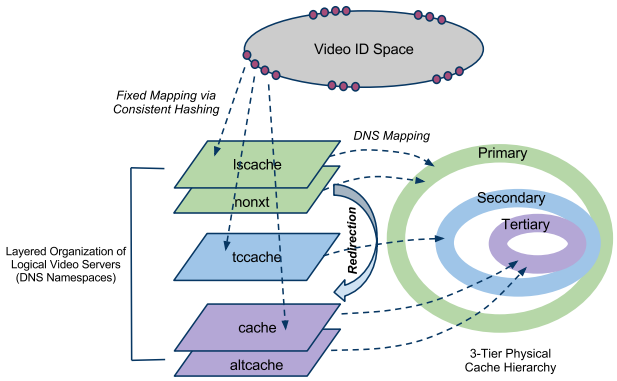


Figure 4.3: YouTube Architectural Design.

Table 4.1: Youtube *Anycast* (first five) and *Unicast* (last two) Namespaces.

DNS namespace	format	# hostnames	# IPs	# prefixes	# locations
lscache	v[1-24].lscache[1-8].c.youtube.com	192	4,999	97	38
nonxt	v[1-24].nonxt[1-8].c.youtube.com	192	4,315	68	30
tccache	tc.v[1-24].cache[1-8].c.youtube.com	192	636	15	8
cache	v[1-8].cache[1-8].c.youtube.com	64	320	5	5
altcache	alt1.v[1-24].cache[1-8].c.youtube.com	64	320	5	5
rhost	r[1-24].city[01-16][s,g,t][0-16].c.youtube.com	5,044	5,044	79	37
rhostisp	r[1-24].*isp*-city[1-3].c.youtube.com	402	402	19	13

(out of 192 names) in the *nonxt* or *tccache* namespace, and to a fixed hostname (out of 64 names) in the *cache* or *altcache* namespace. Moreover, we find that this fixed mapping from the video id space to anycast namespaces makes sure that the number of *video id*s that map to each anycast hostname are nearly equally distributed. To demonstrate this, we plot the number of *video id*s that map to each of the *lscache* hostnames in Figure 4.1. We see that there are approximately equal number of videos mapped to each of the *lscache* hostnames.

4.4 Cache Namespaces & Hierarchy

YouTube defines and employs a total of 5 *anycast* namespaces as well as two sets of *unicast* hostnames of the formats (*rhost* and *rhostisp*), respectively. Based on our datasets, these *anycast* and *unicast* names are resolved to a collection of nearly 6,000 IP addresses ("physical" video cache servers) that are distributed across the globe. Table 4.1 provides a summary of these namespaces, the number of IP addresses and locations they map to, and so forth.

Figure 4.4: Geographical distribution of YouTube Video Cache Locations.

We find that the 5 *anycast* and 2 *unicast* namespaces map essentially to the same set of IP addresses: about 93% of the 5,883 IP addresses have a (unique) *unicast* name associated with them. Second, 80% of the IP prefixes come from addresses assigned to Google/YouTube, while the remaining 20% of the prefixes coming from address space assigned to other ISPs such as Comcast and Bell-Canada (hereafter referred to as *non-Google* addresses/prefixes). The former have the *unicast* names of the form *rhost*, whereas the latter *rhostisp*. Clearly, the *unicast* names indicate that Google/YouTube have video caches co-located within other ISP networks (referred to as *non-Google* locations) as well as within its own (referred to as *Google* locations). The 3-letter city code provides a hint as to where the corresponding YouTube cache is located (at the granularity of a city or metro-area). To geo-locate and classify those IP addresses that do not have an associated *unicast* name in our datasets and to further validate the geo-locations of YouTube video caches, we conduct pair-wise round-trip measurements from each PlanetLab node to all of the YouTube IP addresses. Using these measurements as well as the round trip delay logs in the collected video playback traces, we perform geo-location clustering similar to the approach used by GeoPing [24]. This yields a total of 47 cache locations. We plot them (including both Google and non-Google cache locations) on a world map in Figure 4.4.

Based on the HTTP redirection sequences, there is a clear hierarchy among the 5 *anycast* namespaces, as shown in Figure 4.2: A video server mapped to a *lscache* hostname (in short, a *lscache* server) may redirect a video request to the corresponding *tccache* server, or directly to the corresponding *cache* server, but never the other way around. Based on these analyses, we deduce that the YouTube cache locations are organized into a *3-tiered* hierarchy: there are roughly *primary* cache locations geographically dispersed across the world (most are owned by Google, some are co-located within ISP networks);

there are 8 *secondary* and 5 *tertiary* cache locations in US and Europe and owned by Google only. We discuss each tier below in more details below.

• **Primary Video Caches.** The *lscache anycast* namespace consisting of 192 hostnames of the form *v[1-24].lscache[1-8].c.youtube.com* plays a key role in YouTube video delivery. These names are the ones that appear in the host name part of the URLs embedded in the HTML pages generated by YouTube *web* servers when users access the YouTube website. We note that the *lscache* namespace maps to both Google and non-Google primary cache locations The *nonxt anycast* namespace, also consisting of 192 hostnames of the form *v[1-24].nonxt[1-8].c.youtube.com*, maps to a subset of the IP addresses that the *lscache* namespace maps: namely, only those IP addresses belonging to Google (and thus with the *unicast* hostnames in the *rhost* namespace).

• **Secondary Video Caches.** The *tccache anycast* namespace, consisting of 192 hostnames of the form *tc.v[1-24].cache[1-8].c.youtube.com*, maps to a set of 636 IP addresses belonging to Google only. These IP addresses are mostly disjoint from the $4,999$ IP addresses that the *lscache* and *nonxt* namespaces map to, with a small number of exceptions. They all have a unique *rhost unicast* hostname, and are distributed at only 8 locations.

• **Tertiary Video Caches.** The *cache* and *altcache anycast* namespaces, both consisting of 64 hostnames of the form *v[1-8].cache[1-8].c.youtube.com* and *alt1.v[1-8].cache[1-8].c.youtube.com* and respectively, map to the same small set of 320 IP addresses belonging to Google only. These IP addresses all have a unique *rhost unicast* hostname, and are distributed at only 5 locations.

4.5 Video Delivery Dynamics

In this section we present our key findings on the mechanisms and strategies employed by YouTube to service user requests, perform dynamic load-balancing and handle potential cache misses. These are achieved via a combination of (coarse-grained) DNS resolution and a clever and complex mix of background fetch, HTTP re-directions and additional rounds of DNS resolutions.

Experimental Methodology. We divide the videos into two sets: i) *hot* videos which have a very high number of view counts (at least 2 million views); and ii.) *cold* videos

which have fewer than 100 view counts. We randomly select a video from both *hot* and *cold* sets and play them one by one, while the delay between two consecutive playback requests is modeled as a Poisson process with inter-arrival rate of 10 seconds. For each video playback request, we record the detailed logs including timestamps, redirection URLs (if any) and the IP addresses of the servers involved. In particular, we also examine the time difference between the time our client receives ACK for the HTTP GET request and the time the client sees the first packet of the HTTP response.

4.5.1 Locality-aware DNS Resolution

To characterize the granularity of locality-aware resolutions, we conduct the following analysis. For each PlanetLab node, we rank all 38 YouTube primary cache locations in the increasing order of round trip network delay and assign each YouTube location a rank in this order. Next, we consider the *lscache* hostname-to-IP addresses mappings and calculate how they are distributed with respect to the rank of the corresponding YouTube location for the given PlanetLab node. In Figure 4.5 we plot the number of PlanetLab nodes which have at least one of *lscache* hostnames mapped to an ith rank YouTube location. As seen in this figure, more than 150 PlanetLab nodes have at least one of the IP addresses at the closest YouTube location. Using our

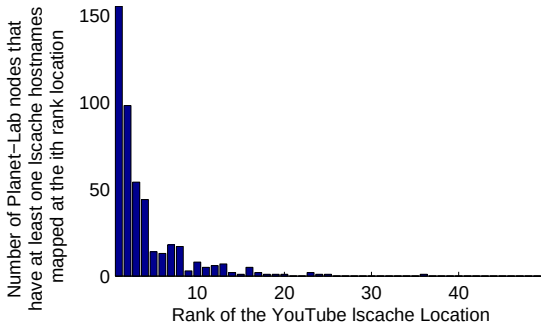


Figure 4.5: Locality aware DNS mappings for *anycast* hostnames.

DNS mapping data collected over several months, we also investigate whether YouTube adjusts the number of IP addresses mapped to each *lscache* hostname over time to, say, adapt to the changing loads at particular cache locations or regions of users. We create a *temporal matrix* of DNS name to IP address mapping matrix for each *lscache*

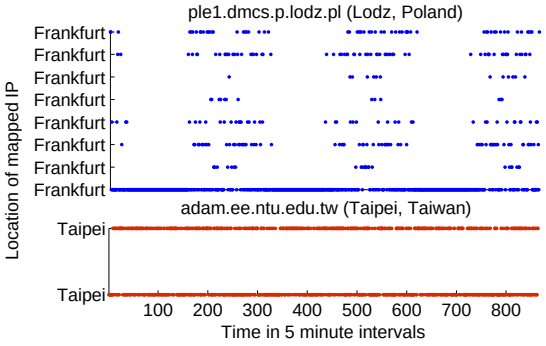


Figure 4.6: DNS resolution over time

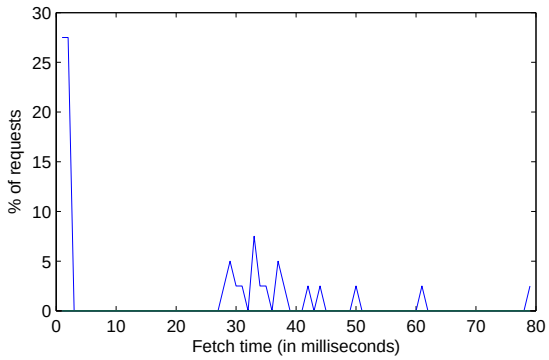


Figure 4.7: Fetch time distribution at a YouTube cache server.

hostname, where each row in the matrix represents the mappings of the hostname at a given time from all the PlanetLab nodes. Analysis of this matrix reveals two interesting aspects of the way YouTube DNS servers resolve *anycast* hostnames to IP addresses. First, we see that the hostname to IP address mappings may change over time. Based on how these mappings changed for PlanetLab nodes, we can put them into two distinct groups. In the first group of PlanetLab nodes, the mappings change during a certain time of the day, and the pattern repeats every day. In the second group, the set of IP addresses remains the same over time. Figure 4.6 provides an illustration: the top panel shows an example of the first group, while the bottom panel an example of the second group. In this figure: the X-axis represents the time which is divided in the intervals of 5 minutes each, and the Y-axis represents the mapped IP address. In the top panel, at the ple1.dmcs.p.lodz.pl PlanetLab node, one hostname is mapped to a fixed IP address

(belonging to the Frankfurt cache location) most of the time during the day; however, during the certain hours of the day we see a large number of distinct IP addresses for the same hostname. In the bottom panel, one hostname is always mapped to one of the two IP addresses (belonging to the Taipei cache location).

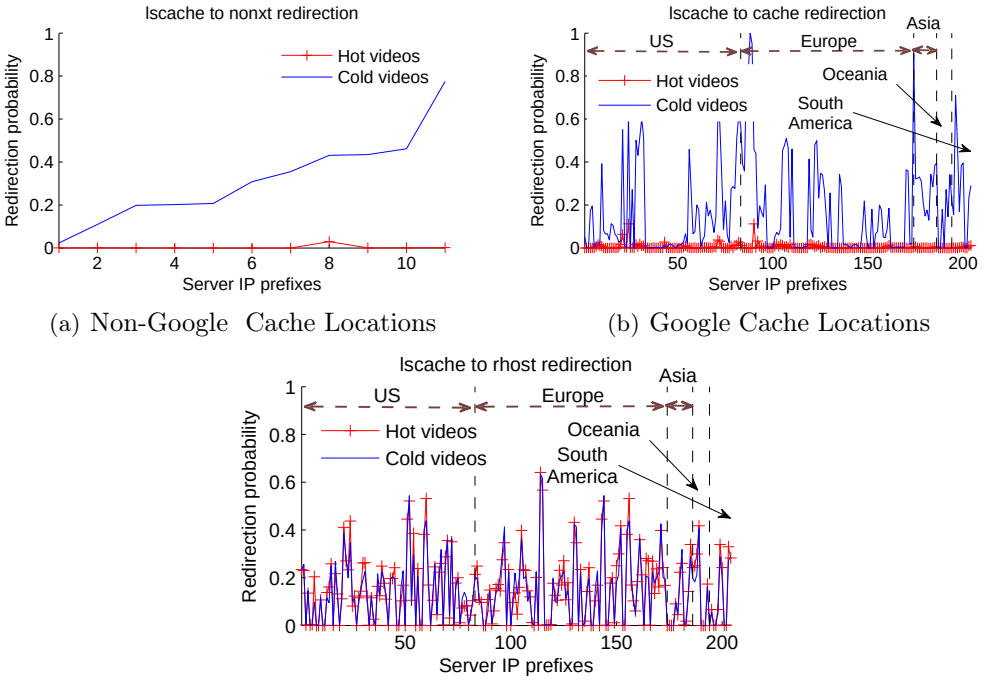
4.5.2 Handling Cache Misses via Backend Fetching

To handle cache misses, YouTube cache servers use two different approaches: (a) fetching content from the backend data center and delivering it to the client, or (b) redirecting the client to some other servers. We study the difference between the time the client receives the ACK for the GET request and the time that it receives the first packet for the HTTP response. We call this difference "fetch-time". This "fetch-time" indicates the time the server took after sending the ACK for the request and before it started sending the response. In our analysis, we can clearly put the fetch-times in two groups: few milliseconds and tens of milliseconds.

We find that when the cache server redirects the client the fetch-time is very small, generally about 3ms. We also see about the same fetch-time for most of the *hot* videos when the server actually serves the video. For most of the *cold* videos when they are not redirected, this lag is much higher, typically in tens of milliseconds and vary depending upon cache location. An example of the distribution is presented in Figure 4.7 which shows the distribution of fetch-times of one Google YouTube cache server observed from a fixed vantage point. There is a clear gap between the shorter and longer fetch times. We deduce that large fetch-time is the time it takes for the cache server to fetch the content from some backend data center (cf. [25]).

4.5.3 HTTP Redirections Dynamics

The video redirection logs reveal that HTTP redirections always follow a specific namespace hierarchy, as shown in Figure 4.2. Our analysis of video redirection logs also reveals that redirection probability highly depends on the popularity of the video. However, there were no significant evidences to show if the factors such as the location of the YouTube cache and time of the day influence the redirection probability. In Figure 4.8 we demonstrate how redirection probability is distributed for *hot* and *cold* at



(a) Non-Google Cache Locations

(b) Google Cache Locations

(c) Google Cache Locations

Figure 4.8: Comparison of redirection probabilities.

both Google and Non-Google locations. In these figures, x-axis represents the IP prefixes for the YouTube primary cache servers, which is sorted based on the region and then based upon the size of each location. The y-axis represents the probability of redirection to another namespace. As seen in Figure 4.8(a), at Non-Google locations, *cold* videos have much higher probability of being redirected to *nonxt* namespace than for the *hot* videos. In particular, around 5% of the requests to *hot* videos experience redirections as compared to more than 24% for the *cold* videos. Similarly, at Google cache locations, most of the requests to *cold* videos are redirected to *cache* hostnames (see Figure 4.8(b)). It indicates that these redirections are primarily done to handle cache misses by redirecting the users to the third tier directly. On the other hand, the redirection probability to *tccache* and *rhost* hostnames does not depend on the popularity of the video. As we see in Figure 4.8(c), the probability of redirection for *hot* and *cold* videos to *rhost* namespace is very similar at all the Google cache locations. Moreover, a closer inspection of redirection logs revealed that redirection *rhost* hostnames is used to

redirect the user to a different physical server at the same location, which is more than 99% of all the redirections to *rhost* namespace. This indicates that YouTube performs a very fine grained load balancing by redirecting the users from possibly a very *busy* server to a less busy server at the same location.

4.5.4 Delay due to Redirections

YouTube's use of HTTP redirections comes with a cost. In general, when the client is redirected from one server to another, it adds to the time before the client can actually start the video playback. There are three sources of delay due to redirections. First, each redirect requires the client to start a new TCP connection with a different server. Second, the client may need to resolve the hostname it is being redirected to. And finally, since the client is being redirected from a nearby location, the final server that actually delivers the video might be farther away from it which will add more delay in the video download time. To account for all these sources of delays and to compensate for the differences in video sizes, we analyze the total time spent to download 1MB of video data starting from the time the client sends HTTP GET requests to the first *lscache* server for a video. We refer to this time as *video initialization time*.

Figure 4.9 shows the CDF plot for the video initialization time observed by one of the PlanetLab nodes. As seen in this figure, HTTP redirection used by YouTube servers add a significant overhead to the video initialization time. In particular, our results show that on an average HTTP redirections increase the video initialization time by more than 33% in comparison to video initialization time when there are no redirections.

4.6 Summary

In this chapter we reverse-engineer the YouTube video delivery system by building a globally distributed active measurement platform and deduced the key design features of the YouTube video delivery system such as multi-tiered namespaces, hierarchical cache location and application-level redirections. While Google's YouTube video delivery system represents an example of the "best practices" in the design of a large-scale content delivery system, its design also poses several interesting and important questions regarding alternative system designs, cache placement, content replication and load balancing

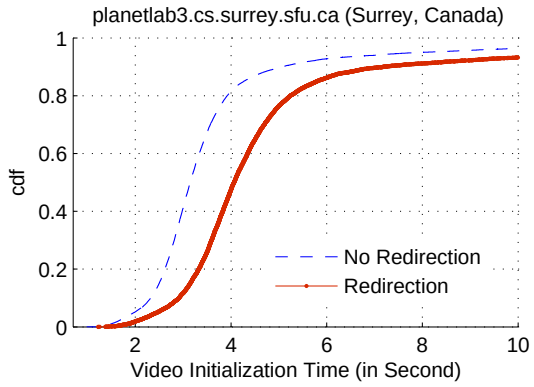


Figure 4.9: An example distribution of video initialization time.

strategies. Additionally, as the content quality and popularity increases, its homegrown delivery system is likely to encounter scalability challenges.

Chapter 5

Netflix and Hulu

Netflix is the leading provider of on-demand Internet video streaming in the US and Canada, accounting for 29.7% of the peak downstream traffic in US. Similarly, Hulu is one of most popular online video services serving primarily TV shows. Understanding the Netflix and Hulu architecture and their performance can shed light on how to best optimize their design as well as on the design of similar on-demand streaming services. In this chapter, we perform a measurement study of Netflix and Hulu to uncover their architecture and service strategy. We find that Netflix employs a blend of data centers and Content Delivery Networks (CDNs) for content distribution. We also perform active measurements of the three CDNs employed by Netflix to quantify the video delivery bandwidth available to users across the US. Finally, as improvements to Netflix's current CDN assignment strategy, we propose a measurement-based adaptive CDN selection strategy and a multiple-CDN-based video delivery strategy, and demonstrate their potentials in significantly increasing user's average bandwidth.

We also find that Hulu appears to distribute user requests among the CDNs in accordance with certain predetermined ratio. The selection of the "preferred" CDN for a given user does not seem to be taking into account the current condition of the network performance (between the user and the selected CDN). Further, Hulu frequently changes preferred CDNs for each user.

5.1 Netflix

Netflix is the leading subscription service provider for online movies and TV shows. Netflix attracts more than 23 million subscribers in the United States and Canada, and can stream out HD (High Definition) quality video with average bit rate reaching 3.6 Mbps. In fact, Netflix is the single largest source of Internet traffic in the US, consuming 29.7% of peak downstream traffic [26]. Its design and traffic management decisions have a substantial impact on the network infrastructure.

Designing such a large scale, fast growing video streaming platform with high availability and scalability is technically challenging. The majority of functions used to be hosted in Netflix's own data center. Recently, Netflix has resorted to the use of cloud services [27], Content Distribution Networks (CDNs), and other public computing services. Amazon AWS cloud replace in-house IT, and Amazon SimpleDB, S3 and Cassandra are used for file storage [27]. Video streaming is served out of multiple CDNs, and *UltraDNS*, a public DNS service, is used as its authoritative DNS servers. Microsoft Silverlight [28] is employed as the video playback platform for Netflix desktop users. The end result is amazing: Netflix manages to build its Internet video delivery service with little infrastructure of its own!

In this chapter we provide a detailed analysis of the Netflix architecture, which is designed to serve massive amounts of content by combining multiple third party services. This architecture can be considered as a possible blue print for a scalable, infrastructure-less content provider. We discuss the interaction between the components of this design including multiple CDNs and HTTP adaptive streaming, and analyze the algorithms used by Netflix that provide the glue to piece together the overall system. We also shed light on the implications the Netflix design decisions have on CDNs, the network and the end user experience both to understand its performance and to improve its design. In addition, based on our measurement results, we suggest new video delivery strategies that can further improve user experience by effectively utilizing multiple CDNs.

Despite the popularity of Netflix, surprisingly there have been very few studies looking into its streaming service platform. The authors of [29] investigate the Netflix security framework, while the authors of [30] focus on the rate-adaptation mechanisms employed by Silverlight player and experimentally evaluated the Netflix players. To the best of

our knowledge, we are the first to take a systematic look into the architecture of the Netflix video streaming together with an extensive measurement study of three CDNs it employs.

The main contributions of this chapter can be summarized as follows:

- We dissect the basic architecture of the Netflix video streaming platform by monitoring the communications between the Netflix player and various components of the Netflix platform. We collect a large number of Netflix video streaming manifest files to analyze how geographic locations, client capabilities, and content type affect the streaming parameters used by Netflix, such as content formats, video quality levels, CDN ranking, and so forth.

- We analyze how Netflix makes use of multiple CDNs under changing bandwidth conditions. We find that Netflix players stay attached to a fixed CDN even when the other CDNs can offer better video quality.

- We perform an extensive bandwidth measurement study of the three CDNs used by Netflix. The results show that there is significant variation in CDN performance across time and location.

- Finally, we explore alternative strategies for improving video delivery performance by using multiple CDNs. Our study shows that selecting the best serving CDN based on a small number of measurements at the beginning of each video session can deliver more than 12% bandwidth improvement over the static CDN assignment strategy currently employed by Netflix. Furthermore, using multiple CDNs simultaneously can achieve more than 50% improvement.

The chapter is organized as follows. Section 5.2 describes the architecture of Netflix video streaming system and CDN selection strategy. Section 5.3 presents our measurement study of the three CDNs. Section 5.4 explores the alternative strategies for CDN assignment in order to improve video delivery performance. Section 5.6 discusses the related work. Finally, Section 5.7 concludes the chapter and discusses the future work.

Table 5.1: Key Netflix Hostnames

Hostname	Organization
www.netflix.com	Netflix
signup.netflix.com	Amazon
movies.netflix.com	Amazon
agmoviecontrol.netflix.com	Amazon
nflx.i.87f50a04.x.lcdn.nflximg.com	Level 3
netflix-753.vo.llnwd.net	Limelight
netflix753.as.nflximg.com.edgesuite.net	Akamai

5.2 Netflix video streaming platform

We start the section with the overview of Netflix video streaming platform architecture. We dissect the architecture via traffic monitoring, DNS resolutions, and WHOIS[31] lookup. We then present the timeline of serving a single Netflix client as an example to illustrate the interplay between a Netflix player and various service components. We further collect a large number of video streaming manifest files using *Tamper Data* add-on[32], and analyze how geographic locations, client capabilities, and content types influence the streaming parameters. Finally, we focus on the Netflix CDN assignment strategy. Using *dummynet* [33] to strategically throttle individual CDN's bandwidth, we discover how Netflix makes use of multiple CDNs in face of bandwidth fluctuation.

5.2.1 Overview of Netflix architecture

To observe the basic service behavior, we create a new user account, login into the Netflix website and play a movie. We monitor the traffic during all of this activity and record the hostnames of the servers involved in the process. We then perform DNS resolutions to collect the canonical names (CNAMEs) and IP addresses of all the server names that the browser have contacted. We also perform WHOIS[31] lookups for the IP addresses to find out their owners. Table 5.1 summarizes the most relevant hostnames and their owners. Fig. 5.1 shows the basic architecture for Netflix video streaming platform. It consists of four key components: Netflix data center, Amazon cloud, CDNs and players.

• *Netflix data centers.* Our analysis reveals that Netflix uses its own IP address space for the hostname `www.netflix.com`. This server primarily handles two key functions:

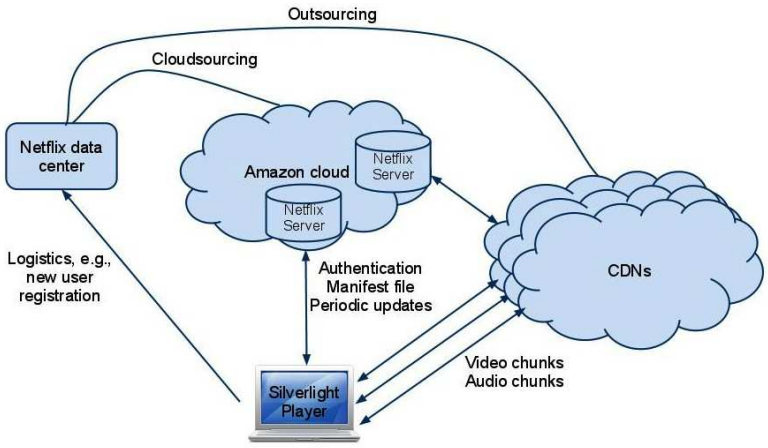


Figure 5.1: Netflix architecture

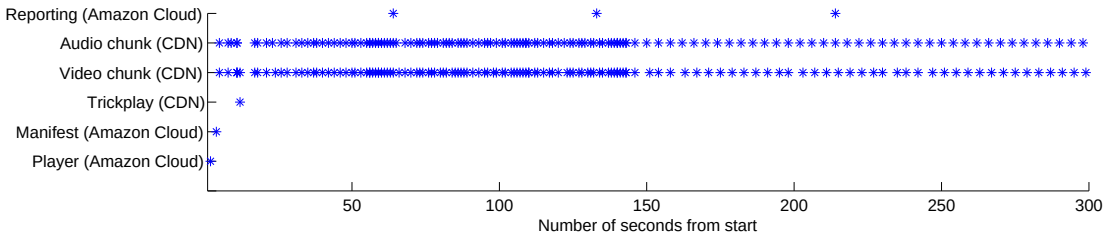


Figure 5.2: Timeline in serving a Netflix client

(a) registration of new user accounts and capture of payment information (credit card or Paypal account), and (b) redirect users to `movies.netflix.com` or `signup.netflix.com` based on whether the user is logged in or not respectively. This server does not interact with the client during the movie playback, which is consistent with the recent presentation from Netflix team [34].

• *Amazon cloud.* Except for `www.netflix.com` which is hosted by Netflix, most of the other Netflix servers such as `agmoviecontrol.netflix.com` and `movies.netflix.com` are served off the Amazon cloud [35]. [34] indicates that Netflix uses various Amazon cloud services, ranging from EC2 and S3, to SDB and VPC [35]. Key functions, such as content ingestion, log recording/analysis, DRM, CDN routing, user sign-in, and mobile device support, are all done in Amazon cloud.

• *Content Distribution Networks (CDNs).* Netflix employs multiple CDNs to deliver the video content to end users. The encoded and DRM protected videos are sourced in

Amazon cloud and copied to CDNs. Netflix employs three CDNs: *Akamai*, *LimeLight*, and *Level-3*. For the same video with the same quality level, the same encoded content is delivered from all three CDNs. In Section 5.2.4 we study the Netflix strategy used to select these CDNs to serve videos.

• *Players*. Netflix uses Silverlight to download, decode and play Netflix movies on desktop web browsers. The run-time environment for Silverlight is available as a plug-in for most web browsers. There are also players for mobile phones and other devices such as Wii, Roku, etc. This chapter, however, focuses on Silverlight player running on desktop PCs.

Netflix uses the DASH (*Dynamic Streaming over HTTP*) protocol for streaming. In DASH, each video is encoded at several different quality levels, and is divided into small 'chunks' - video segments of no more than a few seconds in length. The client requests one video chunk at a time via HTTP. With each download, it measures the received bandwidth and runs a *rate determination algorithm* to determine the quality of the next chunk to request. DASH allows the player to freely switch between different quality levels at the chunk boundaries.

5.2.2 Servicing a Netflix client

We now take a closer look at the interaction between the client web browser and various web servers involved in the video playback process. Fig. 5.2 shows the timeline along which the streaming service is provided to a desktop client, and indicates the involved server entities. The X-axis in this figure shows the time from the beginning of the experiment to 5 minutes and the Y-axis lists different activities. The client first downloads the Microsoft Silverlight application from `movies.netflix.com` and authenticates the user. After authentication, the player fetches the manifest file from the control server at `agmoviecontrol.netflix.com`, based on which it starts to download trickplay data and audio/video chunks from different CDNs. Client reports are sent back to the control server periodically. We describe further details of individual activities below.

Silverlight player download and user authentication

Video playback on a desktop computer requires the Microsoft Silverlight browser plug-in to be installed on the computer. When the user clicks on "Play Now" button, the browser

downloads the Silverlight application and then that application starts downloading and playing the video content. This small Silverlight application is downloaded for each video playback.

Netflix manifest file

Netflix video streaming are controlled by instructions in a manifest file that the Silverlight client downloads. The Netflix manifest file provides the DASH player metadata to conduct the adaptive video streaming. The manifest files are client-specific, i.e., they are generated according to each client's playback capability. For instance, if the user player indicates it is capable of rendering h.264 encoded video, h.264 format video is included in the manifest file. If the player indicates that it can only play back .wmv format, only .wmv format video is included.

The manifest file is delivered to end user via SSL connection and hence the content of the file cannot be read over the wire using packet capture tools such as *tcpdump* or *wireshark*. We use Firefox browser and *Tamper Data* plug-in to extract the manifest files. The extracted manifest file is in XML format and contains several key pieces of information including the list of the CDNs, location of the trickplay data, video/audio chunk URLs for multiple quality levels, and timing parameters such as time-out interval, polling interval and so on. The manifest file also reveals interesting information on the Netflix system architecture. For instance, they show that Netflix uses three CDNs to serve the videos. Different ranks are assigned to different CDNs to indicate to the clients which CDN is more preferred than others. A section of one of the manifest files is shown in Fig. 5.3, where Level3 is listed as the most preferred CDN for this client. We will conduct more elaborate experiments and discuss more details of the manifest files later in this section.

Trickplay

Netflix Silverlight player supports simple trickplay such as pause, rewind, forward and random seek. Trickplay is achieved by downloading a set of thumbnail images for periodic snapshots. The thumbnail resolution, pixel aspect, trickplay interval, and CDN from where to download the trickplay file are described in the manifest file. The trickplay

```
<nccp:cdns>
    <nccp:cdn>
        <nccp:name>level3</nccp:name>
        <nccp:cdnid>6</nccp:cdnid>
        <nccp:rank>1</nccp:rank>
        <nccp:weight>140</nccp:weight>
    </nccp:cdn>
    <nccp:cdn>
        <nccp:name>limelight</nccp:name>
        <nccp:cdnid>4</nccp:cdnid>
        <nccp:rank>2</nccp:rank>
        <nccp:weight>120</nccp:weight>
    </nccp:cdn>
    <nccp:cdn>
        <nccp:name>akamai</nccp:name>
        <nccp:cdnid>9</nccp:cdnid>
        <nccp:rank>3</nccp:rank>
        <nccp:weight>100</nccp:weight>
    </nccp:cdn>
</nccp:cdns>
```

Figure 5.3: CDN list in manifest file

interval for the desktop browser is 10 seconds, and multiple resolutions and pixel aspects for trickplay are provided.

Audio and video chunk downloading

As shown in Fig. 5.2, audio and video contents are downloaded in chunks. Download sessions are more frequent at the beginning so as to build up the player buffer. Once the buffer is sufficiently filled, downloads become periodic. The interval between the beginning of two consecutive downloads is approximately four seconds - the playback length of a typical chunk.

The manifest file contains multiple audio and video quality levels. For each quality level, it contains the URLs for individual CDNs, as shown in Fig. 5.4.

User experience reporting

After the playback starts, Netflix player communicates periodically with the control server `agmoviecontrol.netflix.com`. Based upon the keywords such as "/heartbeat"

```
<nccp:bitrate>560</nccp:bitrate>
<nccp:videoprofile>
   playready-h264mpl30-dash
</nccp:videoprofile>
<nccp:resolution>
   <nccp:width>512</nccp:width>
   <nccp:height>384</nccp:height>
</nccp:resolution>
<nccp:pixelaspect>
   <nccp:width>4</nccp:width>
   <nccp:height>3</nccp:height>
</nccp:pixelaspect>v
<nccp:downloadurls>
   <nccp:downloadurl>
      <nccp:expiration>131xxx</nccp:expiration>
      <nccp:cdnid>6</nccp:cdnid>
      <nccp:url>http://nflx.i.../...</nccp:url>
   </nccp:downloadurl>
   <nccp:downloadurl>
      <nccp:expiration>131xxx</nccp:expiration>
      <nccp:cdnid>4</nccp:cdnid>
      <nccp:url>http://netflix.../...</nccp:url>
   </nccp:downloadurl>
   <nccp:downloadurl>
      <nccp:expiration>131xxx</nccp:expiration>
      <nccp:cdnid>9</nccp:cdnid>
      <nccp:url>http://netflix.../...</nccp:url>
   </nccp:downloadurl>
</nccp:downloadurls>
```

Figure 5.4: Video downloadable for one quality level

and "/logblob" in the request URLs and the periodicity of the communication, we conjecture that they are periodic keep alive messages and log updates. However, the actual messages that we have extracted by using *Tamper Data* do not appear to be in clear text and hence we cannot verify it further.

5.2.3 Manifest file analysis

A manifest file is delivered over the SSL connection. We use *Tamper Data* plug-in for Firefox browser to read the file. Since the manifest files contain a wealth of information and shed lights on the Netflix strategies, we conduct a large scale experiment by collecting and analyzing a number of manifest files. We are interested in understanding how

geographic locations, client capabilities, and content type (e.g., popular vs unpopular, movies vs TV shows) may impact the streaming parameters. We use six different user accounts, 25 movies of varying popularity, age and type, four computers with Mac and Windows systems at four different locations for this experiment. From each computer, we log into Netflix site using each of the user accounts and play all of the movies for few minutes to collect the manifest files. In addition to using client machines located in different geographies, we also configure those client browsers to use Squid proxy servers running on ten PlanetLab nodes hosted by US universities in different geographic regions to collect additional manifest files.

CDN ranking and user accounts

Netflix manifest files rank CDNs to indicate which CDNs are preferred. CDN ranking determines from which CDN the client downloads the video and may affect user perceived video quality. We analyze the collected manifest files to understand the factors that affect the rankings of the CDNs. For this analysis, we build a table that lists CDN ranking for each combination of user account, client computer (or PlanetLab proxy), movie ID and time of day for several days. Analysis of this table suggests that the CDN ranking is only based upon the user account. For a given user account, the CDN ranking in the manifest file remains the same irrespective of movie types, computers, time and locations. Furthermore, for the same movie, computer, location and around the same time, two different users may see different CDN rankings. We also observe that the CDN ranking for each user account remains unchanged for at least several days. As we show in measurement results in the next section, such assignment of ranking seems to be independent of available bandwidth from each CDN.

Audio/Video bit rates

Netflix serves videos in multiple formats and bit rates. When a Netflix client requests for the manifest file from Netflix, the client indicates the formats of the content it can play. Netflix server then sends back a manifest file based upon the client request. For instance, Netflix client running on an older computer (Thinkpad T60 with Windows XP) and a newer computer (Macbook Pro with Snow Leopard) have different capabilities and receive different video downloading format and bit rates.

Based on the client capabilities, the server sends URLs for the video and audio chunks in the returned manifest files. In general, manifest files contain information about video chunks encoded in bit rates between 100Kbps to 1750Kbps (and 2350Kbps and 3600Kbps for videos available in HD) for the manifest files sent to the newer computer. We see that videos available in HD can be served in up to 14 different bit rates whereas non-HD content can be served in up to 12 different bit rates. We also note that Netflix clients do not try all possible available bit rates when trying to determine the optimal playback rate.

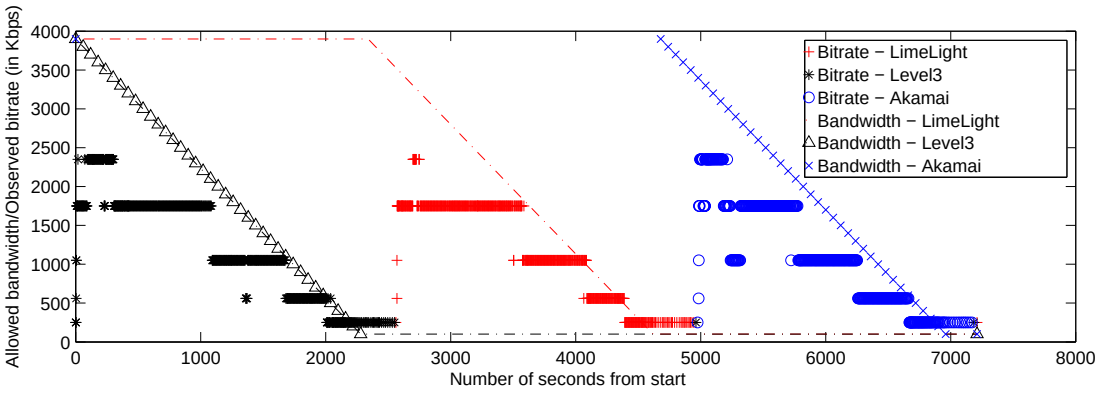


Figure 5.5: CDN switching

5.2.4 CDN selection strategy

We have seen that a Netflix client can choose different video bit rates and different CDNs for video downloading. In this section we conduct experiments to help understand how Netflix make such choices when bandwidth is dynamic. We play a single movie from the beginning. Once the playback starts, we gradually throttle the available bandwidth of the top ranked CDN in the manifest file. We use *dummynet* to throttle the inbound bandwidth to the client.

At the beginning, servers from each CDN are allowed to send data at 3,900Kbps. After every minute, we reduce the available bandwidth for the current CDN by 100Kbps till it reaches 100Kbps. At that point we start throttling the next CDN in the same way and so on. We plot our observation in Fig. 5.5. In this figure, the X-axis shows the time starting from the beginning of playback. The Y-axis shows both the throttled bandwidth

and the playback rate. In this instance, Level3, Limelight and Akamai CDNs are ranked first, second and third respectively. The client starts downloading video chunks from the first CDN. In the beginning, it starts from a low bit rate and gradually improves the bit rate in a probing fashion. As we lower the available bandwidth for the first CDN while leaving the other CDNs intact, we notice something interesting. Instead of switching to a different CDN, which is not throttled, the client keeps lowering the bit rate and stays with the first CDN. Only when it can no longer support even the very low quality level (i.e. when the available bandwidth for the first CDN reaches 100Kbps), it switches to the second CDN. It repeats almost the same behavior as we leave the first CDN at 100Kbps and gradually lower the available bandwidth for the second CDN while leaving the third CDN intact. In general, the Netflix client appears to stay with the same CDN as long as possible even if it has to degrade the quality level of the playback.

5.3 CDN Performance Measurement

Figure 5.6: Best CDN at each vantage point

We have observed in the previous section that Netflix CDN ranking is tied to each user account and remains unchanged over many days. Even a change in the user geographic location does not trigger any CDN ranking change. Such assignment strategy naturally leads to the following questions:

- How does each CDN perform? Can the selected CDN server consistently support the bandwidth needed for high quality streaming?

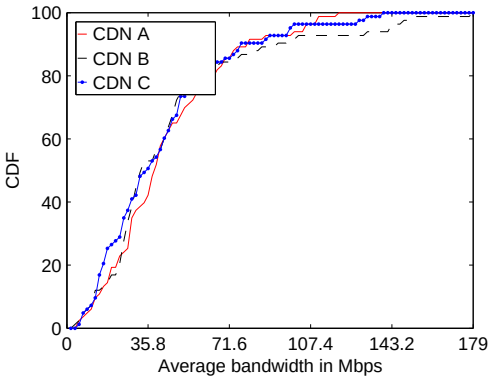


Figure 5.7: CDF of average bandwidth at PlanetLab nodes

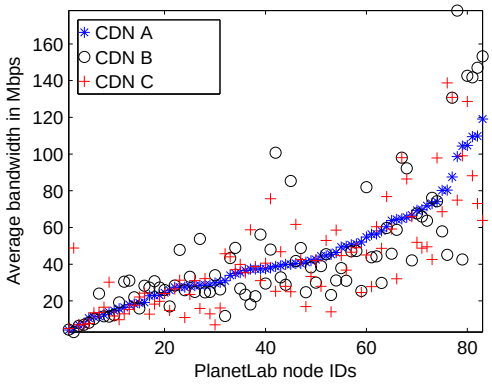


Figure 5.8: Average bandwidth at PlanetLab nodes over the entire period

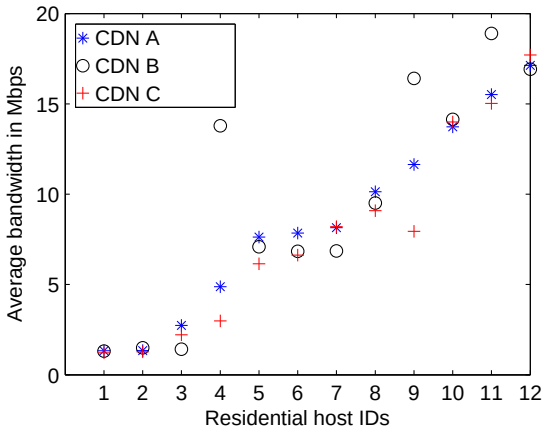


Figure 5.9: Average bandwidth at residential networks over the entire period

- How do different CDNs compare in terms of performance? Is any CDN clearly better or worse than others?
- How far is the current Netflix assignment strategy from "optimal"?
- Is it possible to improve the assignment strategy to support higher delivery bandwidth?

In the next two sections, we attempt to address the above questions by conducting extensive measurement experiments for the three CDNs used by Netflix from 95 vantage points across the United States.

We measure the bandwidth throughput between each vantage point and a given CDN server by downloading multiple video chunks from the CDN server. Video file URLs are collected for all three CDNs from manifest files. Here we take advantage of the fact that the URLs in the manifest remain valid for several hours from the time the manifest file is generated, and the validity of the URLs are not tied to client IP address. Furthermore, the byte "range" of the download can be adjusted without affecting the URL validity. Once we extract the URLs for the three CDNs, we "replay" the GET request from all vantage points with byte range modified so that we download video chunks of the same size.

Similar to the actual Netflix video playback, when GET requests are sent from a vantage point, the hostnames in the URLs are resolved by DNS server, which returns the IP address of the edge server assigned by the CDN. To ensure the measured bandwidth of three CDNs are comparable, we send GET requests to three CDNs in round-robin order within a short duration. More specifically, measurement is repeated in multiple "rounds", with each round lasting 96 seconds. A round is further partitioned into four "slots", with 24 seconds for each slot. The first three slots of each round correspond to three CDNs, respectively, and we download video chunks of size 1.8MByte. The last slot of each round is for a "joint" measurement for all CDNs, i.e., we send GET requests to the three CDNs simultaneously, each requesting video chunks for 0.6MByte data. We intend to find out how much total bandwidth one can get if all three CDNs are used simultaneously. We pick the size of the chunks and length of "slots" based upon multiple trial measurements. In our trials, we find that these numbers make sure that different experiments do not interfere with each other and chunk size is sufficiently large so that we can have a good estimate of the bandwidth. We also send keep-alive messages to each server every second when no data is transferred to make sure that the TCP session is alive and sender window size does not drop.

The measurement is conducted for two hours between 8 to 10pm CST, from June 8, 2011 to June 26, 2011. Based on downloading time, we calculate the instantaneous bandwidth (i.e., throughput for each GET request), the one-day average bandwidth (average bandwidth during the two hour period), and average bandwidth (over entire measurement study). These metrics allow us to examine CDN performance at multiple timescales.

We have conducted experiments from both residential sites and PlanetLab nodes. There are 12 residential sites, 10 in New Jersey, 1 in Minnesota, and 1 in California. The residential sites spread across 5 different service providers. To cover a wider range of geographic locations, we also choose 83 PlanetLab nodes spread across the United States as additional vantage points. We ensure that all selected PlanetLab nodes are lightly loaded so that the nodes themselves do not become the bottleneck and the measurement results reflect the actual bandwidth that can be supported by the CDN server and the network.

The rest of this section attempts to address the first two questions on CDN performance. We will further investigate the other two questions on performance improvement in the Section 5.4. We use CDN A, B, and C to denote the three CDNs without particular order in the rest of the discussion.

5.3.1 Overall CDN performance

Fig. 5.6 shows the locations of all vantage points in our experiments as well as the CDN with highest average bandwidth at each vantage point during the measurement period. As the result indicates, no CDN clearly outperforms the others. In addition, Fig. 5.7 shows the CDF (*Cumulative Distribution Function*) of average bandwidth at the PlanetLab nodes over the entire measurement period. The available bandwidth at different PlanetLab nodes varies significantly from location to location, ranging from 3Mbps to more than 200Mbps. The CDF curves of three CDNs, however, are close to each other, indicating similar overall performance. Figures 5.8 and 5.9 further show the average bandwidth at individual locations for PlanetLab nodes and residential sites, respectively. The location index is sorted in the ascending order of CDN A's average bandwidth. CDN bandwidth measured at PlanetLab nodes appear to have much higher than that of residential sites in general. This is because most PlanetLab nodes are located in universities, which typically have better access links. This also implies that in most cases, the last mile is still the bottleneck for streaming video. However, even the residential sites with relatively low bandwidth, e.g. home 1 and 2 in Fig. 5.9, can support 1.3Mbps on average, enough for standard definition (SD) videos.

It is also interesting to note that home sites 4, 9, and 11 see significantly different average bandwidth from different CDNs. In particular, CDN B outperforms all others

by a large margin. We find that these three homes use the same service provider. It is conceivable that CDN B has a better presence in this provider's network.

5.3.2 Daily bandwidth variation

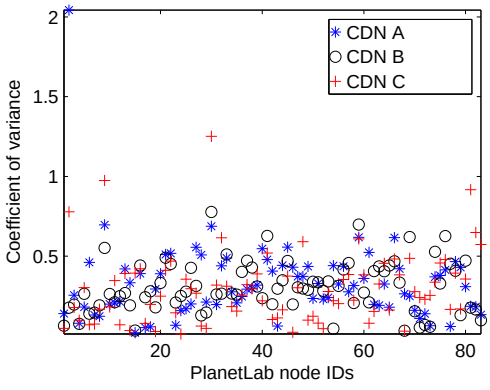


Figure 5.10: Coefficient of variance for the one-day average at PlanetLab nodes

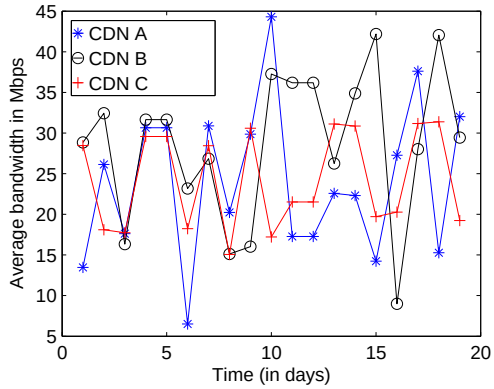


Figure 5.11: One-day average bandwidth at a PlanetLab node over time

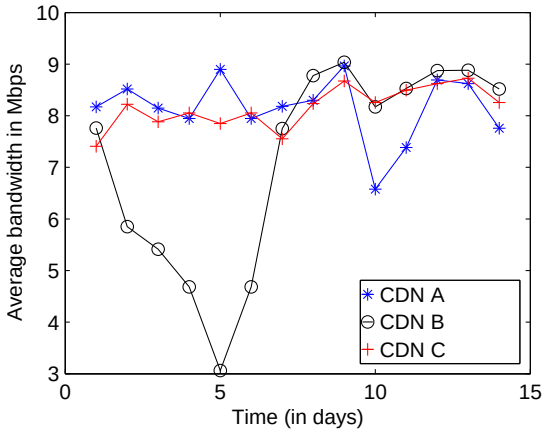


Figure 5.12: One-day average bandwidth over time at residential site 7

Next we examine the bandwidth variation at different sites from different CDNs over different timescales. We compute the coefficient of variance of the daily average bandwidth at all PlanetLab nodes by computing the ratio of the standard deviation to the mean at each of the locations. Fig. 5.10 shows the coefficient of variance for the

one-day average bandwidth at different PlanetLab nodes over multiple days. We indeed see high coefficient of variance at most nodes. The average coefficient of variance is 0.33, 0.30, and 0.30 for CDN A, B and C, respectively. At most locations, there is a significant variation in daily bandwidth for all three CDNs. We show a few representative locations in Figures 5.11, 5.12 and 5.13, which plot the one-day average bandwidth over the measurement period at one PlanetLab node and two residential sites, respectively. The results show significant variation of average bandwidth on a daily basis.

Furthermore, Figures 5.11, 5.12 and 5.13 show that the performance ranking of different CDNs also change over time. Although the lowest CDN bandwidth across all three nodes is still above 3Mbps, sufficient to support standard definition (SD) levels, the significant variation in bandwidth and ranking of CDNs indicates a good potential to further increase bandwidth for future higher quality video delivery if better CDN selection strategy is used.

5.3.3 Variation in instantaneous bandwidth

We further investigate the instantaneous bandwidth variation during two hours of video playing. This is important since a DASH player constantly monitors the available bandwidth to decide which quality level of video to download. The small time scale bandwidth may significantly impact the Netflix users' viewing experience as two hours is a typical length of movie. Figures 5.14, 5.15 and 5.16 show the comparison of three CDNs for the same PlanetLab node and residential nodes. Although the variance is still significant, there is a "pattern" in the bandwidth change. For example, bandwidth for CDN B in Fig. 5.14 alternates between two levels, one around 35 Mbps and one around 20 Mbps. The average coefficient of variation for two hour period is 0.19, 0.21 and 0.18 respectively for CDNs A, B and C respectively for residential sites.

5.4 Alternate Video Delivery Strategies

From the measurement study, we observe that Netflix statically assigns a CDN to users for extended period of time. Although all three CDNs are available, each user only uses one in most cases. Other CDNs appear to serve only as backups and are used only if current CDN server cannot support even the lowest video quality. On the other hand,

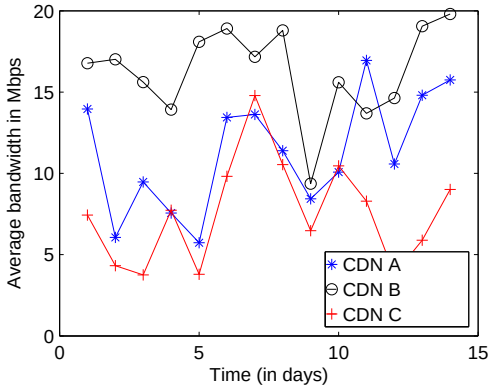


Figure 5.13: One-day average bandwidth over time at residential site 9

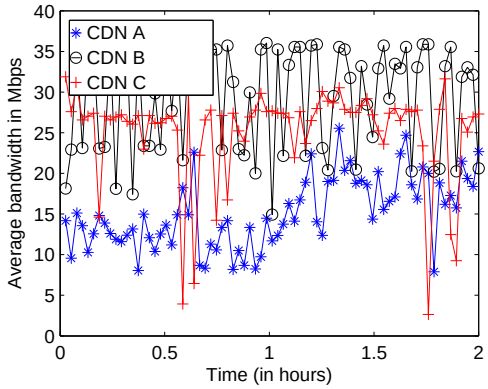


Figure 5.14: Instantaneous bandwidth at a PlanetLab node

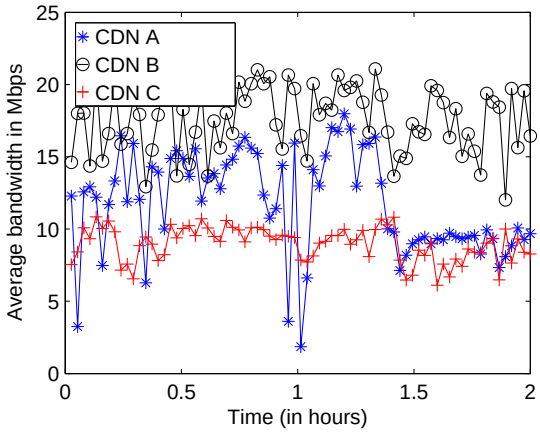


Figure 5.15: Instantaneous bandwidth at residential site 7

our study also shows that the available bandwidth on all three CDNs vary significantly over time and over geographic locations. For instance, as shown in Fig. 5.6, out of 83 PlanetLab locations, CDNs A, B, and C perform best at 30, 28 and 25 locations, respectively. The measurement study of residential hosts shows similar results. If users are tied to a bad CDN choice, their video viewing quality may suffer even though other CDNs can provide them with more satisfying experience. In addition to improving experience for "unlucky" users, exploring potential ways of increasing video delivery bandwidth may also open doors for new bandwidth-demanding services in future, e.g., 3D movies or multiple concurrent movies in the same household.

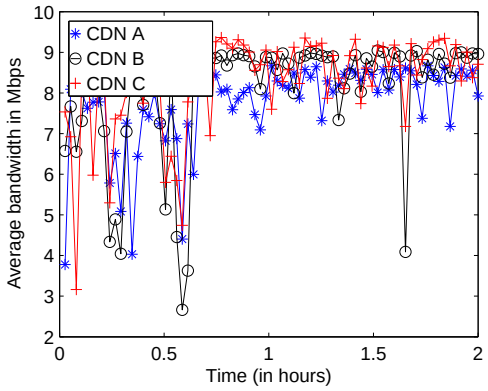


Figure 5.16: Instantaneous bandwidth at residential site 9

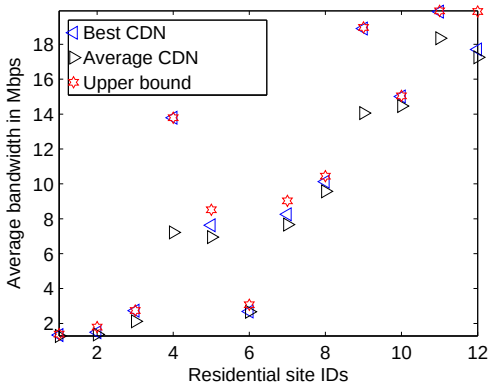


Figure 5.17: Average bandwidth and the upper bound at residential sites

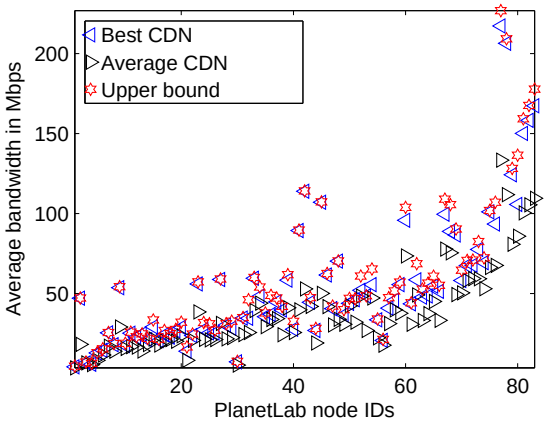


Figure 5.18: Average bandwidth and the upper bound at PlanetLab nodes

In this section, we first determine how much room there is for further improvement. In other words, if we could have the optimal CDN selection strategy in theory, how much better it would be compared to current static assignment. We then explore two alternative strategies for CDN assignment that can easily be used in practice, and demonstrate we can indeed significantly increase the bandwidth for video delivery to users despite the simplicity of such strategies.

5.4.1 Room for improvement

Given the instantaneous bandwidth trace, the optimal CDN selection strategy is to choose the top CDN at each point of time. Although this cannot be done in practice since we do not know the instantaneous bandwidth beforehand, this theoretical optimal strategy allows us to find out the highest bandwidth each client can receive if the best (one) CDN is used at any given point of time. We refer to the average bandwidth achieved by the optimal strategy as the *upper bound average bandwidth*.

Fig. 5.17 and Fig. 5.18 show the average bandwidth of three CDNs and the upper bound average bandwidth for residential sites and PlanetLab nodes respectively. Here we use the average bandwidth over all three CDNs to reflect the static assignment strategy. The actual assignment may of course be better or worse depending on which CDN gets selected, but this gives the expected value. We also show the bandwidth if one top CDN, i.e., the one with highest average bandwidth is selected. For the majority of the sites, the upper bound is much better than the average CDN case, and close to the top CDN case. In particular, the upper bound is 17% and 33% better than the average case for residential sites and PlanetLab nodes respectively, indicating there is significant room for improvement. Assigning users to top CDN is only 6% to 7% worse than the theoretical optimal case. This indicates that if we can estimate which CDN is likely to perform best in next couple hours, we can achieve average bandwidth that is fairly close to the *upper bound average bandwidth*.

5.4.2 Measurement based CDN selection

Since selecting the top CDN for users gives good performance, we next study how to identify the top CDN effectively. We propose to have the player conduct the instantaneous bandwidth measurement multiple times at the beginning, and assign users the best-performing CDN for the rest of the movie. Fig. 5.19 shows the effect of number of measurements on performance. As reference, two straight lines show the ratio of the CDN average bandwidth over top CDN bandwidth for all PlanetLab and residential nodes, respectively. In both cases we calculate the average CDN bandwidth over all locations, time, and CDN providers, so they reflect the expected CDN performance, assuming the three CDNs are equally likely to be chosen in the static CDN assignment strategy. The

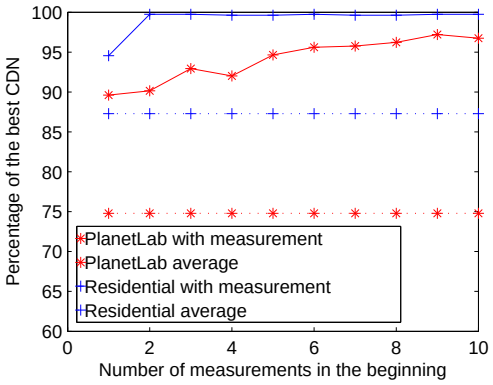


Figure 5.19: Effect of number of measurements

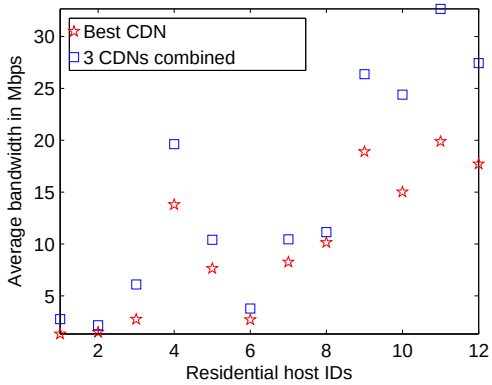


Figure 5.20: Best CDN vs three combined CDNs for residential hosts

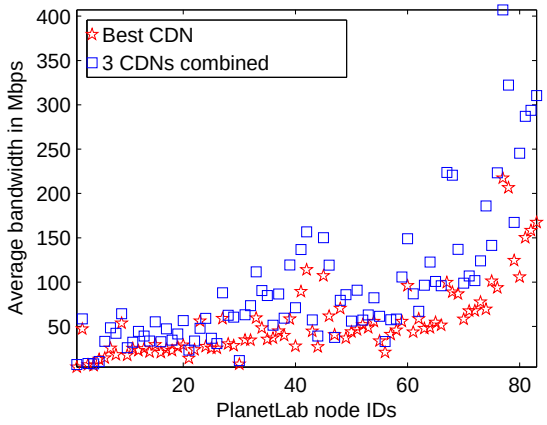


Figure 5.21: Best CDN vs three combined CDNs for PlanetLab nodes

other two curves are ratio of average bandwidth using measurement based CDN selection strategy over that of using top CDN for both PlanetLab nodes and residential sites. Using a small number of measurements (≥ 2), the measurement based strategy delivers more than 12% improvement over the static CDN assignment strategy. Although the average improvement is moderate, for certain users the improvement is significant, e.g., more than 100% for residential host 4. Given this method is very straightforward and easy to implement, we believe this is a favorable approach for improving video delivery.

5.4.3 Using multiple CDNs simultaneously

In previous sections, we have assumed that only one CDN can be used at a time. However, since Silverlight player downloads video and audio content in chunks, it is possible to use all three CDNs simultaneously. For instance, the player can download three different chunks in parallel from three different CDNs to obtain larger bandwidth. Since the design of a HTTP adaptive streaming protocol that can best utilize multiple CDNs is out of the scope of this chapter, we try to see if multiple CDNs can be used, whether they can offer higher aggregated throughput for end users.

Fig. 5.20 and Fig. 5.21 compare the average bandwidth using top CDN and the average bandwidth obtained by combining three CDNs for residential and PlanetLab nodes, respectively. We see that combining all three CDNs can significantly improve the average bandwidth. Specifically, the aggregate bandwidth obtained by combining all 3 CDNs is greater than the bandwidth of the single best CDN by 54% to 70% for residential sites and PlanetLab nodes, respectively.

5.5 Hulu

Hulu employs multiple CDNs to serve its content. A question that naturally arises is: how does Hulu use the three CDNs? For instance, are all three CDNs used during one video playback session, or just one CDN at a time? How does Hulu decide which CDNs to pick – is it based on performance of the CDNs for a given client, or on overall performance of the CDNs across all users or are they selected at random? Is the choice static or dynamic? We attempt to answer such questions in this section.

5.5.1 Rate and CDN adaptation

We analyze the captured packet traces to find that during normal playback, when the network condition is relatively stable, Hulu uses only one CDN server throughout the duration of a video. But interestingly, it usually switches to a different CDN server for the next video.

To further understand how network conditions affect player behavior and CDN selection strategy, we conduct the following experiment using a tool called `dummynet`[33].

In the beginning, servers from each CDN are allowed to send data at 1501 Kbps. At the end of every minute, we reduce the available bandwidth for the current active CDN by 100 Kbps till it reaches 1 Kbps. As we lower the available bandwidth for the current CDN while leaving the other CDNs intact, we notice that instead of switching to a different CDN, which is not throttled, the client keeps lowering the bit-rate and stays with the original CDN. This indicates that Hulu adapts to changing bandwidth by adjusting the bit-rates and continues to use the same CDN server as long as possible. Only when the current CDN server is unable to serve the lowest possible bit-rate, it switches to a different CDN server.

In summary, Hulu prefers to stay with the same CDN server than to keep maintaining maximum achievable bandwidth and video quality for the user. As a result, if a user is assigned to a CDN experiencing degraded performance at the beginning of a video, that user is most likely to remain with the same CDN for the duration of the video. We next try to explore how CDN selection is made for each video playback.

5.5.2 Status Reporting

Several factors can potentially affect CDN selection: bandwidth between the client and servers of different CDNs, past performance history of different CDNs, and non-technical reasons such as pricing and business contracts. We first try to find out what information is available to Hulu for making such decision.

From the packet trace, we find that the Hulu player sends periodic reports to a server that includes detailed information about the status of the client machine at that time, the CDN servers for video content and advertisements, and any problems encountered in the recent past. These periodic status reports are sent to `t.hulu.com` which maps to the same single IP address from all the locations in US. Using WHOIS[31] queries, we learn that the corresponding IP address, 208.91.157.68, is allocated to Hulu. Examples of detailed performance information contained in the periodic reports include: video bit-rate, current video playback position, total amount of memory the client is using, the current bandwidth at the client machine, number of buffer underruns, and number of dropped frames. When the client adapts bit-rate due to changed network conditions, the periodic reports also include details on why the bit-rate was changed. For instance, one of the messages reads “Move up since avg dropped FPS $0 < 2$ and bufferLength $>$

10". In summary, it appears that Hulu has sufficient performance data for CDN selection for any given request, if they choose to do it based upon user experience.

5.5.3 CDN Selection Strategy

We try to infer how much, if any, of the performance data contained in the status reports influences CDN selection.

Hulu clients follow the manifest files they receive from the server to decide which CDN server to request video content from. Since Hulu encrypts the manifest file sent to the client, it is not easy to read the contents of the manifest files from the network traces. Therefore, we collect and read the manifest files by using a tool called `get-flash-videos`[36]. A small section of the content of an example Hulu manifest file is shown in Fig. 5.22. The last line in the figure shows Hulu's CDN preference in that manifest file. When we examine CDN preferences in a few manifest files, we observe that the preferred CDN server included in the manifest file seems very dynamic. For instance, when we make two requests simultaneously (or within a few seconds) for the same video, the preferred CDNs for those two requests can be different.

To better understand the CDN selection strategy employed by Hulu, we request one manifest file every second for the same video from the same computer for 100 seconds. In Figure 5.23, we show how CDN preference changes very frequently. In this figure, X-axis shows the time and the Y-axis shows the three CDNs. Each '*' in the plot represents the CDN selected for a given request. Since the network conditions on the tested Hulu client is fairly stable during the experiment, the above result indicates that Hulu CDN selection is not based on instantaneous network conditions.

To further understand the impact of various factors such as client location, video and time on CDN selection, we use the `get-flash-videos` tool to collect manifest data for 61 different videos of different genres, length, popularity and ratings available on Hulu from 13 different locations across the United States over multiple days (up to 24 days at one of the locations). The client machines on these locations are connected to residential bradband networks or business high speed Internet services. They also cover a number of different ISPs including Comcast, AT&T, Verizon and CenturyLink.

For a given video at a given location and time, we download the manifest file 100 times, with 1 second interval between two consecutive downloads. We call such 100

```
'title' => 'Soldier Girls',
'tp:Ad_Model' => 'longform',
'tp:Frame_Rate' => '25',
'dur' => '4991138ms',
'tp:enableAdBlockerSlate' => 'true',
'tp:Aspect_Ratio' => '4x3',
'tp:BugImageURL' => '',
'tp:researchProgram' => '',
'tp:comScoreId' => '',
'tp:hasBug' => 'false',
'tp:defaultBitrate' => '650_h264',
'tp:primarySiteChannelNielsenChannelId' => '71',
'tp:CP_Promotional_Link' => '',
'tp:CPIdentifier' => 'ContentFilm',
'tp:Primary_Category' => 'Documentary and Biography',
'tp:adType' => '',
'tp:fingerPrint' => 'csel3_prod_iad115',
'tp:CP_Promotional_Text' => '',
'tp:Segments' => 'T:00:11:54;22,T:00:26:29;09,
  T:00:38:49;27,T:00:57:37;18,T:01:03:15;02,
  T:01:17:12;03',
'tp:adTypePlus' => 'SponsoredFilm',
'tp:Tunein_Information' => '',
'tp:distributionPartnerComScoreId' => '3000007',
'tp:secondarySiteChannelNielsenId' => '38',
'tp:cdnPrefs' => 'level3,akamai,limelight',
```

Figure 5.22: A section of Hulu manifest file

consecutive downloads an *experiment*. Each downloaded manifest file assigns one CDN as preferred CDN. We count the number of times each CDN is preferred for each experiment. We refer to the percentage of times that a CDN is preferred in an experiment as *preference percentage*. This preference percentage essentially reflects the likelihood for a CDN to be selected by the clients.

Overall CDN preference Figure 5.24 shows the distribution of preference percentage for the three CDNs based on results for all videos, locations, and time. The three curves representing the three CDNs are very close to Gaussian distributions. The mean preference percentage for Limelight, Akamai and Level3 are 25, 28 and 47, respectively. Level3 is the preferred CDN 47% of times, much more than the other two CDNs.

CDN preference over different locations Figure 5.25 shows CDN preference observed from clients at different geographic locations. These 13 locations span different

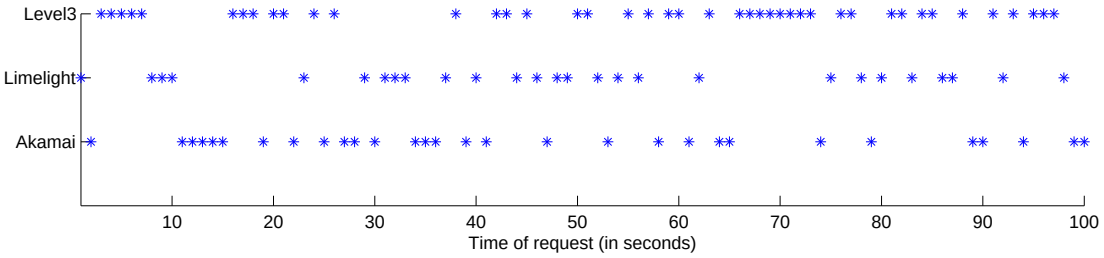


Figure 5.23: CDN preference change in a short interval

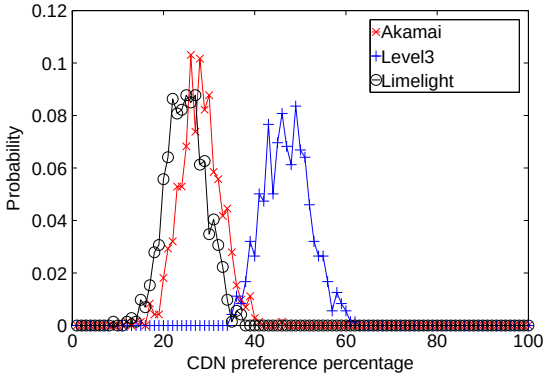


Figure 5.24: Overall CDN preference distribution

cities across eight US states. For this analysis, we combine data for all the videos collected at the same location and calculate the average preference percentage for each location. We observe that different CDNs have different popularity but the popularity do not change over different locations.

CDN preference for different videos Figure 5.26 shows CDN preference for different videos. Here we aggregate the experiments for each video across location and time and calculate its average preference percentage. The small variation in preference percentage across different videos indicate CDN preference is independent of which video is being served.

CDN preference over time Figure 5.27 shows CDN preference change over different days at the same location. This result is based on 24 days of experiments at a single location. Each data point represents the average preference percentage over all videos on each day for a given CDN. The results for for other locations (not shown here) are very similar. We observe that the CDN preferences do not change over time either.

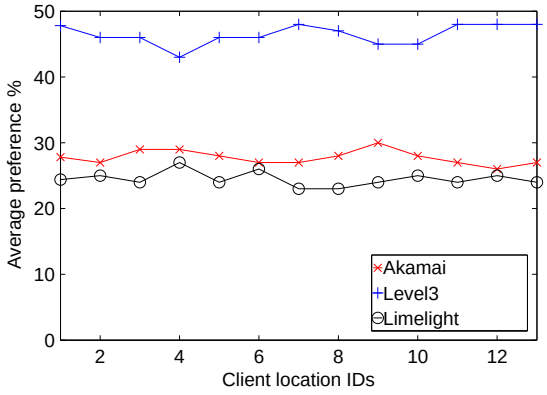


Figure 5.25: CDN preference from geographic regions

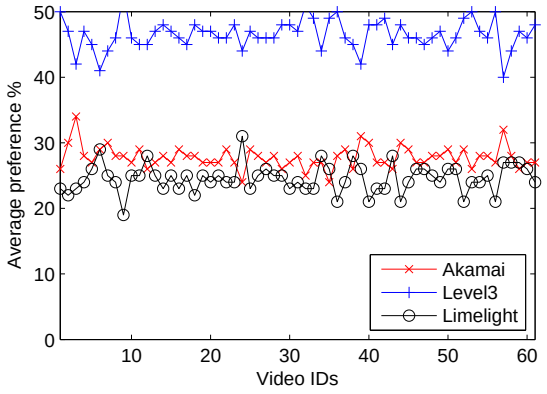


Figure 5.26: CDN preference for different videos

In summary, we conclude that Hulu selects the preferred CDN randomly following a fixed latent distribution for each of the playback requests. On average, one CDN (Level3) is preferred more than others, but such selection preference does not seem to depend on instantaneous network conditions. It is also evident that CDN selection is not affected by client location at which the video is played. And the selection does not change over the 24 days that we measured. We conjecture that such CDN preference is most likely based on pricing and business arrangements and is not dependent upon instantaneous bandwidth or past performance history of the CDNs.

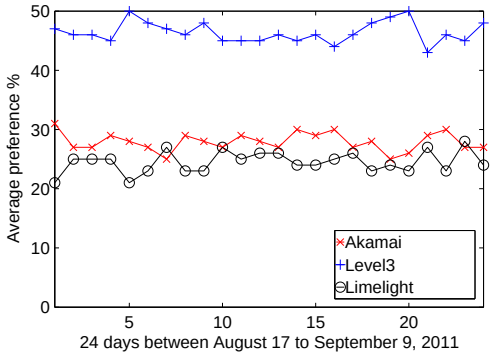


Figure 5.27: CDN preference over time

5.5.4 CDN Server Distribution

Hulu server selection consists of two steps: (a) selecting a CDN provider, and (b) selecting a specific CDN server to serve the content. In this section, we study how the three CDNs utilize their server resources to serve different types of clients (desktop clients vs. mobile clients) and Hulu advertisement. We find that while three CDNs are used to serve desktop clients, only two CDN are used to serve mobile clients and Hulu advertisement. We further examine the number of CDN servers employed by individual CDNs and these servers' locations. We find that different CDNs employ varying number of servers at different locations. In addition, the allocated servers for Hulu desktop clients, mobile clients, and for the advertisement are different and do not overlap with each other, suggesting CDNs may employ dedicated servers for different services. Finally, we briefly compare the CDN service for Hulu with that for Netflix and Amazon.

We find that all three CDNs use locality-aware DNS resolutions. We analyze the packet traces collected by playing a large number of videos from multiple geographical locations and extract the hostnames of the servers. The same hostname may be mapped to different IP addresses when queried from different locations. In order to better understand how and where the video content is served by each CDN, we next try to identify all servers in each CDN and also cluster them based on locations. Besides the IP addresses extracted from packet traces, we resolve the extracted hostnames from 100 PlanetLab[37] nodes in the US. We also obtain a list of publicly available open recursive resolvers[38] and employ 700 of them (inside the US) to resolve these hostnames.

CDN Servers for Hulu Desktop Clients

Through experiments we find that Akamai and Level3 only use one hostname each, while Limelight uses 1,000 different hostnames for delivering Hulu video content to desktop clients (Table 5.2). We also observe that Limelight uses last three digits of the internal ID of a video to decide what hostname is responsible to serve that video. For instance, a video with ID 50061220 is mapped to host `hulu-220.fcod.llnwd.net`. Such mapping provides some limited means of balancing load among different servers because each of the 1000 hostnames expect to responsible for an equal number of videos. However, since different videos have widely differing popularity, this approach can not be expected to equally divide load among different servers.

Table 5.2: CDN servers for Hulu

CDN	Hostname(s)	# IPs	Clusters
Akamai	`cp39466.edgefcs.net` (Desktop)	1178	40
	`httpls.hulu.com` (Mobile)	450	38
	`ads.hulu.com` (Advertisement)	454	39
Limelight	`hulu-{000-999}.fcod.llnwd.net` (Desktop)	868	9
	`ll.a.hulu.com` (Advertisement)	18	9
Level3	`hulufs.fplive.net` (Desktop)	48	10
	`httpls-1.hulu.com` (Mobile)	125	10

After we obtain the set of unique IP addresses for each CDN, we then measure ping latencies to all IP addresses. Level3 IP addresses serving Hulu content do not respond to ping probes. We therefore measure latencies to other "nearby" IP addresses. Assuming IP addresses from a "/26" subnet will likely to placed nearby, we try to ping other IP addresses in the same "/26" network as the original IP addresses when they do not respond to ICMP ping probes. Based on ping latencies to all the IP addresses from hundreds of PlanetLab nodes, we cluster the IP addresses in different groups. These clusters roughly represents the server locations from which the CDNs are serving Hulu content. The number of such clusters are also shown in Table 5.2. In summary, Akamai uses the largest number of IPs (1178) and clusters (40), while the number of clusters for Limelight and Level3 is much smaller, at 9 and 10 clusters respectively. Interestingly, the number of IPs used by Limelight is close to that of Akamai, and is much larger than the IP addresses used by Level3. These observations we made for servers serving Hulu

content are also consistent with the previous studies [9, 39] on these three large CDNs' architecture.

CDN Servers for Mobile Clients (for HuluPLus)

We find that only two CDNs are used to serve HuluPlus subscribers on iOS devices: Akamai and Level3. These two CDNs use only one hostname each: `httpls.hulu.com` for Akamai and `httpls-1.hulu.com` for Level3. These canonical names are in turn mapped to other hostnames and finally to IP addresses. We find that Akamai and Level3 have 450 and 125 unique IP addresses, respectively. However, these IP addresses are different from IP addresses serving Hulu content to desktop browsers. On the other hand, the number of clusters is roughly the same as that for Hulu desktop CDN servers for both CDNs.

CDN Servers for Hulu Advertisements

We observe that CDNs and servers used for advertisements and regular video content are generally independently assigned. Hulu advertisements are only served from Akamai and Limelight CDNs. Two hostnames are used: `ads.hulu.com` for Akamai and `ll.a.hulu.com` for Limelight. `ll.a.hulu.com` is mapped to 2 IP addresses at each of the 9 clusters of Limelight CDN. Likewise, `ads.hulu.com` is mapped to approximately 450 IP addresses. There is again no overlap between servers for advertisements and regular content. The number of clusters is roughly the same as that for regular content.

Overall, we observe that all CDNs employ locality-aware DNS resolution, while their server distribution and load balancing strategies differ in some cases. Akamai has many more distinct server IP addresses and locations compared to Level3 and Limelight. Note that this does not necessarily translate into more servers or better performance since there can be multiple physical servers behind the same IP address, and also performance for video delivery mostly depends on throughput. A number of these results have been previously reported for generic CDN studies[9, 39, 40]. In contrast to these studies, we primarily focused on CDNs, hostnames and server IPs that serve Hulu content and advertisements. Additionally, Akamai and Level3 only rely on DNS to do load balancing; while Limelight also tries to distribute load by hashing video IDs into hostnames. We also observe that numbers of CDNs and servers for Hulu mobile clients and advertisements

are smaller compared to those for regular videos served to desktop clients. This is most likely because the latter incurs significantly more traffic.

5.6 Related Work

Several recent works have been done in analyzing different aspects of Netflix video streaming. Akhshabi et al. have studied several video streaming players including Netflix player and investigated how the streaming clients react to bandwidth changes [30]. The measurement is done mostly from one fixed location. Pomelo has also presented an interesting analysis of Netflix security framework [29]. This work was done before Netflix is migrated into Amazon cloud. Unlike the previous work, we investigate a broader set of components in Netflix video delivery system, and focus on how the player interacts with different CDNs. To achieve this, we conduct more extensive measurement from multiple geo locations.

Recent work has also been done for other streaming platforms [41, 2]. Krishnappa et al. have studied Hulu streaming with emphasis on improving performance using prefetching and caching [41]. Adhikari et al. build a measurement infrastructure by using PlanetLab nodes with the goal to understand the YouTube system architecture [2]. Unlike our work, such works do not cover behavior of multiple CDNs.

Extensive work has been done to study CDNs such as Akamai, Limelight and YouTube [42, 43, 2]. But most work has been focusing on measurement of latency and does not cover the scenario where the client interacts with multiple CDNs.

Many techniques have been proposed to measure available bandwidth on a network path before, such as pathchar [44], pathload [45] and FabProbe [46]. However, they are not suitable for our study for two reasons. First, both pathchar and pathload require control at the target machine of the measurement. Second, all such tools only measure the in-path bandwidth and they cannot capture possible bandwidth shaping at the server side. Additionally, using our method more accurately reflects the download speed over HTTP than other generic methods.

5.7 Summary

In this chapter, we perform active and passive measurements to uncover the overall architecture of Netflix and Hulu. Since Netflix and Hulu use multiple Content Delivery Networks (CDNs) to deliver videos to its subscribers, we measure the available bandwidth of employed CDNs, and investigate its behavior at multiple time scales and at different geographic locations. We observe that neither Netflix nor Hulu takes into account the current network conditions when selecting a CDN.

For Netflix, we find that conducting light-weighted measurement at the beginning of the video playback and choosing the best-performing CDN can improve the average bandwidth by more than 12% than static CDN assignment strategy, and using all three CDNs simultaneously can improve the average bandwidth by more than 50%. This can be very beneficial for future bandwidth-demanding services such as 3D movies. We find that although individual CDNs are doing fairly well, the client experience is primarily dependent upon how well the CDN selection is made and if we could make the clients smarter. Additionally, even commercial CDNs have scalability challenges. Our geolocation analysis of CDN servers show that even the larges CDNs have a limited number of cache locations.

Chapter 6

Open CDN Architecture

Building upon the results of recent works on understanding large scale content distribution systems, we propose a first step in the direction of an open CDN architecture that allows for better scalability and performance. The two key ingredient of this proposal are to let any willing ISP to participate as CDNs and instrument client software to make decisions based upon measurements. We also provide a proof of concept implementation using PlanetLab infrastructure.

6.1 Introduction

The current Internet architecture was primarily designed to facilitate communication between pairs of end hosts. Therefore, as the growth of large scale content distribution is reaching new heights, content providers are bringing forth a number of different solutions such as geographically distributed large data centers, multi tiered caches, use of third party content distribution systems (CDNs), and using protocols such as DNS in ways not originally intended for.

A number of recent studies aimed at reverse engineering existing large scale content distribution systems show that there is significant room to improve such architectures[1, 2, 23, 3, 4]. A major finding of such studies is that the success of any large scale content distribution system is primarily tied to the success of the CDNs, either homegrown (such as YouTube) or commercial. However, commercial CDNs have their own limitations on how big they can grow and developing a homegrown solution is not practical or

economical for most of content providers. Moreover homegrown CDNs face the same scalability challenges as the commercial ones. This naturally leads to the idea of an open CDN ecosystem, where content creators (or providers) let any ISP become a CDN. This allows for virtually unlimited growth for the CDNs as any ISP can participate in content distribution. This idea is promising not only from technical point of view, but it also makes a business case. Users get to download content from their closest ISPs and ISPs in turn can get a new source of revenue. It has been shown in a number of works how closeness to the cache server improves user perceived performance[25]. Additionally, low overhead measurement performed by clients can improve user-perceived quality significantly. With these in mind, in this chapter, we propose an open CDN architecture that allows any ISP to participate in content distribution any any content provider and shares intelligence between control providers and clients. We also provide a proof of concept implementation. This proposal is a first step towards a direction that will simplify the large-scale content distribution architectures, improve user perceived quality and offer incentive to the ISPs.

We make two contributions in this chapter. We first use the lessons learnt from recent studies to design a highly scalable content distribution architecture. We next provide a proof of concept implementation for the architecture.

Related works This chapter builds upon the findings of a number of current works trying to understand how large scale content distribution through reverse-engineering. Current and original YouTube architectures are explored in [1, 2, 23] using active and passive measurements. Similarly, multi-CDN delivery architecture for Netflix and Hulu are reported in [3, 4]. A survey of different types of CDNs is presented in [47].

Our proposal differs from other approaches such as CDN federation[48]. In our proposal, the intellingence is shared between content providers and the clients. ISPs (or CDNs) do not need to collaborate with other ISPs (or CDNs). It also differs from CP CDNs such as Netflix OpenConnect [49] because instead of using different infrastructure for each of the different content providers, our proposal uses a single shared infrastructure for all the content providers and also gives more control to the ISPs.

The rest of this chapter is organized as follows. The high level architecture of open CDN is presented in Section 6.2. In Section 6.3, we present a proof of concept implementation to show the viability of this architecture. We provide preliminary results of

experiments on our implementation in Section 6.4. We summarize and conclude the chapter in Section 6.5.

6.2 An open CDN architecture

In this section we lay out the details of our proposed open CDN architecture that allows for better scalability and performance. The two key ingredient of this proposal are to let any willing ISP to participate as CDNs and instrument client software to make decisions based upon measurements. In what follows, we will use the following three key participants: clients, ISPs and content-providers. Content-providers are the source of the content that the clients are interested in and ISPs provide the network connectivity between the clients and the content providers.

As we explain below, in this architecture, the intelligence to make server selection is shared between the content providers and the clients. Content providers use past history of the ISPs to suggest them to the clients whereas the clients evaluate current network conditions of the suggested ISPs to decide on which ISP to download a piece of content from. Although, the ISPs are not part of the shared intelligence, they still have incentives for participating and providing the best possible service because this architecture opens up a new revenue stream for them.

6.2.1 Motivation for open CDN architecture

As recent works aimed at understanding large scale content distribution have shown, the success of such systems depend primarily upon two factors: (1) the scale of the CDN, either homegrown or commercial, and (2) the server selection (including CDN selection, for multi-CDN content distribution) mechanisms.

The scale of the CDN is critical because if a client can communicate with nearby cache servers, it generally reduces the latency and also increases reliability of the communication. As we have seen in case of original YouTube, a small number of data centers coupled with proportional load balancing resulted in clients communicating with faraway servers. Therefore, the more widespread the cache servers, the better the user perceived performance. We see that clearly when comparing the performance of current YouTube architecture which uses a large number of cache locations. However, individual CDN

have limits on how big they can grow.

Secondly, the server/CDN selection is also very important. This primarily means the system should be able to find an available server with high bandwidth which is very close to the user. As observed in the study of Netflix, a static assignment of CDNs to client resulted in suboptimal performance as perceived by users. Similar results are reported for Hulu which selects CDNs randomly without taking any performance metrics into consideration. Even in case of YouTube, when the initial server selection was not done carefully, the clients had to follow one or more redirections to actually find a server that can serve them. Studies of Netflix also show that with a small number of client side measurement, we can significantly improve the user perceived quality levels of the video content.

Overall, these recent studies find that the mix of static mappings, dynamic and location-aware DNS and application level redirections result in a highly reliable architecture. This coupled with a very highly scalable CDN architecture and client software that can make decisions based upon measurements will result in scalable and high performance content distribution architecture. In this section, we propose an open CDN architecture that uses these key lessons learnt from the recent findings.

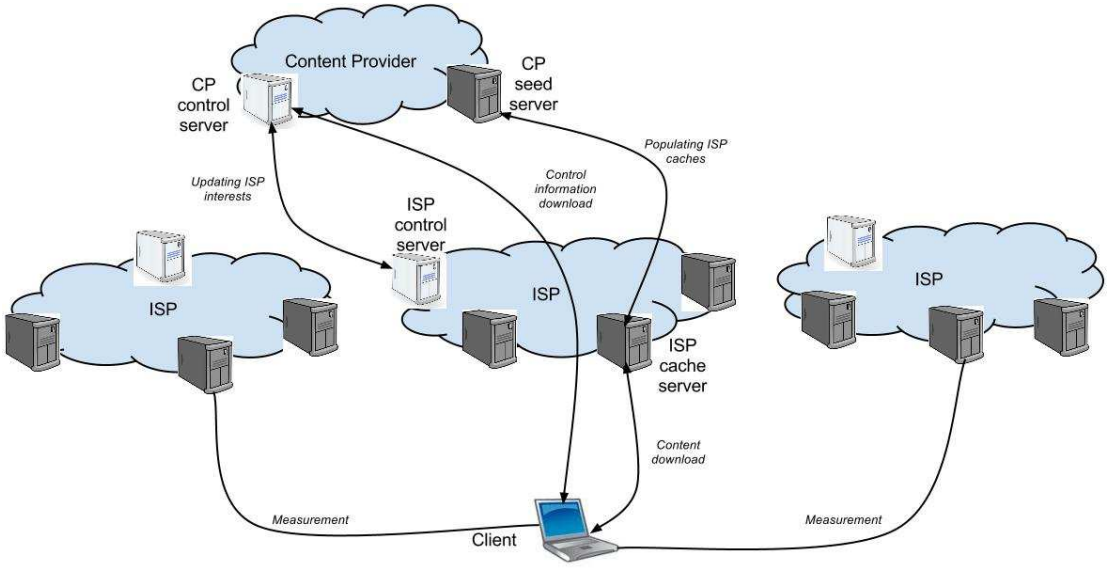


Figure 6.1: An open CDN architecture

6.2.2 Requirements

The proposal outlined in this section is a very generic protocol. It is not specifically tied with any specific control or delivery protocols. However, there are a number of requirements that needs to be met for this architecture to work. The first requirement is of chunked delivery. To allow clients to switch servers as needed and to adapt the quality of the video as the network conditions change over time, the delivery protocol needs to allow for self-contained chunks such as those in MPEG-DASH. At a minimum, the delivery protocol should allow for clients to only parts to of a content similar to HTTP byte range request. Additionally, since the aim is to allow any ISP to be able to serve content from any content provider, the protocols used must be standard and open.

6.2.3 Description of the proposed architecture

There are three key players in this open CDN architecture: (1) the content providers, (2) the ISPs and (3) the clients. We describe the whole architecture by describing the roles of each of these three key players. An overview of the architecture is presented in Figure 6.1.

Content providers

Content providers maintain two sets of servers: (a) CP control servers and (b) seed servers. Both sets of servers can be outsourced to other providers, if needed.

CP control servers. The control servers that the content providers will maintain will receive and send control information to and from the ISPs as well as the clients. They will maintain two distinct interfaces for the ISPs and the client.

The ISP interface will be used to communicate with the ISPs that includes the ISP interests in serving some of the content to some of the clients. Specifically, whenever an ISP is interested in serving some content owned by the content provider, the ISP will use this interface to inform the content provider of its intent. Similarly, the ISPs use the same interface to inform the content provider when it decides to stop serving the content. The content provider has to maintain a database of all such ISP interests and has to update the database as ISPs update their interests. Additionally, when an ISP shows interest in serving an piece of content, the control server provides the ISP with

the URL for the seed server where the ISP can download the content from.

The client interface is used to communicate with the client details about servers that can serve the requested content. The clients can request such information periodically and the CP control server responds with the most recent information it has in the database. In general, the goal of the CP control server is to provide the client with the details of cache servers that have high bandwidth and are close to the clients. Although we do not discuss specific methods that CPs can employ in this chapter, CPs should be able to use any approach they chose. For instance, the CP control server can lookup BGP routing path to the client and see what ISP can be close to the client. This interface can optionally also be used by the client to inform the CP control server if any of the cache servers suggested by the control servers encounter problems.

Seed servers. The seed servers maintained by the content providers store all of their content. These servers also serve to seed the ISP that want to cache and serve some of the content. These seed servers do not interact directly with clients. In case of DRM content, the seed servers either store content encrypted offline or they encrypt the content on the fly before sending them to the ISP CDNs. However, at no time any of the DRM content is sent in clear to the ISPs.

ISPs

In this architecture, ISPs become the most important players. Any ISP that has enough resources should be able to act as a CDN for any content provider. The ISPs have to make a number of decisions based upon the server and bandwidth resources they have. These ISPs will have to maintain two distinct sets of servers: (a) ISP control servers, and (b) cache servers.

ISP control server. ISP control servers only talk to the CP control servers. These servers will inform the CP control servers what content the ISP is interested in serving and optionally for which sets of clients. For instance, an ISP might only be interested in caching and serving content for its own clients. They also receive URLs for the seed servers from where the ISP cache servers can download the content they want to cache.

Cache server. ISP cache servers server client requests by serving the content from their cache. The cache servers from the most critical part of the whole architecture because these servers are what the users primarily interact with. The primary role of

these servers can be described in two steps. Populate the caches by downloading the content from the seed servers, and serve those content to the client when requested. There are many different ways in which these cache servers can be maintained so that they share load among themselves, and are able to handle cache misses and flash crowds. These cache server structures can be modeled after commercial CDNs or they can be modeled after YouTube like multi-tiered hierarchy. However, we propose that the ISPs maintain their cache servers and the communication between them in a way that is similar to what YouTube does. This includes hierarchical cache server organization, dynamic and location-aware DNS and use of application-level redirects. Our suggestion is based upon the observations that most of the details of commercial CDNs is not available and this approach employed by YouTube has shown to be having very good reliability and load sharing and flash-crowd handling. More details of current YouTube architecture is available in [2, 23]. Whether the ISPs model their cache structure like commercial CDNs or YouTube or some other way, at some level, some cache servers must be able to serve the client by fetching the content from the seed servers. Although, cache servers can redirect the client to some other cache servers in the same ISP, they cannot redirect a client to any server outside of the ISP. If, for any reason, none of the cache servers can serve a given request, the ISP simply "fails" and lets the client handle the problem.

One important aspect of cache servers is naming. As CP control servers need to inform the clients which servers to contact for any given content, naming the cache servers in a consistent way is critical. Moreover, since static mapping of content to a pool of cache server names can act as a first attempt of load balancing, ISPs need to decide how many cache server names to use. In this regard, we propose that ISP cache servers use names of the form
`content-provider-`*`num`*`.isp-name.com`. The number of such cache servers can be decided by each of ISPs and communicated to the content providers. However, a much simpler approach could be to use the fixed number of cache server names for each of the ISPs. For instance, each ISP can maintain 1000 cache server names as in the case of Limelight CDN. As an example, Comcast can maintain 1000 hostnames for Hulu content from hulu-000.comcast.com to hulu-999.comcast.com. The case where some ISPs might want to use more or less servers can be handled easily using DNS servers. For instance,

if an ISP wants to use only 10 IP addresses for a content provider, it can use DNS to map a groups of 100 hostnames to a single IP address. Similarly, if an ISP wants to deploy 2000 IP addresses for a content provider, it can use round robin DNS to map a hostname to two IP addresses.

Clients

Clients in this architecture represent the users who want to consume the content provided by the content providers. The communication between the client and CP control server is fairly simple. The client contacts the CP control server to obtain the control information that lets the client identify the ISP cache servers that can serve the content it is interested in. Once it fetches the control information, it then contacts the ISP cache servers to download the content. However, as we have seen in the study of Netflix[3], content providers might not have the best visibility and the suggested servers in the control information might not offer the best possible performance. Therefore, clients in our architecture can optionally perform their own measurements in tandem with the control information provided by the CP control server to make the final server selection. Therefore, in addition to downloading from a cache server, clients might be performing additional measurements to aid in server selection in this architecture. For instance, clients might attempt to download partial contents from all of the suggested server in the control information and then decide on the best server to use for the remaining time. Moreover, these clients can communicate with the CP control servers periodically to receive the most recent list of recommended servers. In rare cases when the whole ISP fails for some reason, clients will select some other suggested ISP and, if needed, will request new control information.

6.2.4 Other components

Content providers who serve DRM content can use the same architecture. The only addition to this architecture that will be required is a DRM server. The DRM keys are exchanged between the content provider server and the clients directly and ISPs or CDNs are not involved in it at all. We do not focus on DRM issues or other security concerns in this chapter.

Similarly, we do not focus on billing and accounting problems associated with this architecture in this chapter. There are existing literature on how billing and accounting can work in such environment.

In summary, this proposed architecture aims to expand the content distribution to include the participation all of the willing ISPs. This does not concern much about the internal working of the ISP cache hierarchy except for the consistency of external interfaces to CP control server. The server selection intelligence is divided between CP control server and the clients. We also note that this architecture is incrementally deployable. Not all ISPs have to start participating in it for it to be deployed. Additionally, early participants will start benefitting from it immediately after deployment.

6.3 A proof of concept implementation

We provide high-level view of the open CDN architecture in Section 6.2. As a proof of concept, in this section we provide details an example implementation of such architecture using PlanetLab[37] infrastructure.

6.3.1 Overview

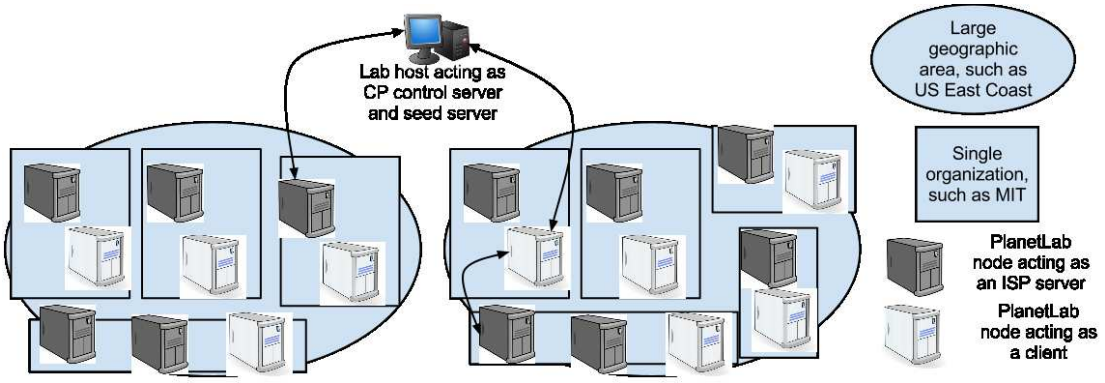


Figure 6.2: A proof of concept implementation

The architecture proposed in Section 6.2 is fairly general. We use MPEG-DASH protocol in our implementation as MPEG-DASH is becoming the standard method for large-scale content delivery. However, the general architecture can be adapted to deliver

Field	Description
isp_id	The ISP that is interested in serving
content_id	The content that the ISP is interested in serving
client_prefixes	The client prefixes that the ISP is willing to serve

Table 6.1: Fields in table maintained by CP control server

other types of content using other protocols as well. The control protocol used by MPEG-DASH is referred as Media Presentation Description (or MPD). In our implementation, we use standard MPEG-DASH protocol as much as possible, supplementing it additional features as needed. Figure 6.2 depicts the the setup of our implementation. We provide details in the following sections.

6.3.2 Content provider

We include only one content provider in our implementation. One of our lab computer runs both the CP control server software and the seed server software. Our content provider has three different content items in four different quality levels. We use data made available by ITEC[50]. We copy and store the datasets used in our experiment on the CP seed server running Apache web server.

The MPD server, which is the CP control server in our implementation, communicates with the ISP control servers as well as the clients. Communication between the clients and the MPD server uses standard protocol. We use a HTTP-based protocol for control communication between the CP and the ISP control servers. To store data on which ISPs are interested in serving which pieces of content and to which sets of clients, the control server maintains an SQLite database table. The fields of the database tables are described in Table 6.1. This table is updated when an ISP shows interest in serving new content or for a new set of users. The content provider assumes that the ISPs are willing to serve any client unless the they explicitly specify a set of client prefixes they want to serve.

In our experiments, the MPD file included at most three ISPs. Each ISP is represented by a single PlanetLab nodes. One of the challenges in an open CDN architecture is to identify what ISPs can best server and given client for any given request. In this implementation we use a rough geographic proximity to the client as a measuring stick.

When a client requests MPD for any content, the CP control server follows the following algorithm. First, it will check to see if there is any PlanetLab node acting as an ISP in the same organization as the one the client belongs to. If so, this ISP will be included in the MPD. In this case, we are trying to simulate the case where the last hop ISP is participating in the content distribution. Next, it will check to see if there are any PlanetLab nodes in the same geographic region as the client. For instance, the if the client PlanetLab node is located in the US East Coast region, the MPD will include randomly selected ISPs from the same region such that the total number of included ISPs is not more than three. In this case, we are trying to simulate a big regional ISP is participating in content distribution. After these steps, it the number of ISPs included in the MPD is still less than three, randomly selected ISPs interested in serving the content will be added to the MPD to make sure three ISPs are included in the MPD.

6.3.3 ISPs

In our implementation, each ISP consists only of a single PlanetLab node. The same node hosts the ISP control server software as well as the cache server software. We have at most one node in any PlanetLab site that can act as an ISP. The ISP cache servers are powered by lighttpd web server[51]. We randomly select 10% of the ISPs to serve only one or two content items. The rest of the ISPs serve all three content items owned by the content provider. In the beginning, these PlanetLab nodes communicate with the CP control server and express their interest in serving the content items they decide to serve. To simplify the experiment, these ISPs do not change their interest during the course of the experiment.

6.3.4 Clients

We pick hundreds of PlanetLab nodes to act as clients. These clients do not actually play back any media as such. The primary task they perform is to request the MPD from the CP control server, conduct initial measurements against the ISPs suggested in the MPD document and then download the pieces of content. A high level of pseudocode for the client is presented in Pseudocode 1. The clients keep a counter that records the buffer they can maintain based upon the playback time of each chunk. These clients monitor

Pseudocode 1 Clients: main loop

```
request MPD
best isp ← evaluate isps
quality level ← best possible level
while there are more chunks to download do
    if time for periodic MPD fetch then
        request MPD
        best isp ← evaluate isps
    end if
    (status, time) ← download next chunk for current quality level from best isp
    if time < time_low_threshold & buffer_size > buffer_low_threshold then
        if quality level is not best possible level then
            increase quality level
        end if
    else if time > time_high_threshold & buffer_size < buffer_high_threshold then
        if quality level is not worst possible level then
            decrease quality level
        end if
    end if
end while
```

download bandwidth and adapt to a lower quality or switch ISP servers. Additionally, clients perform measurements of the bandwidth of the recommended servers in the most recent MPD every five minutes and change ISPs if needed. The pseudocode for evaluating ISPs recommended by the MPD servers are presented in Pseudocode 2. The clients in our implementation also refresh the MPD files periodically. We also implement a protocol similar to the communication protocol between ISP and CP control servers.

6.4 Preliminary evaluation

We describe a proof of concept implementation of our open CDN proposal in Section 6.3. In this section, we discuss some of the preliminary findings from that implementation.

We do not induce any ISP failures in our implementation. The PlanetLab nodes are prone to failures. Part of our implementation is to study how the clients react to such failures. However, when a PlaneLab node acting as a client fails, we discard all data from that client collected during the experiment.

Pseudocode 2 Clients: evaluate ISPs

```
best time = infinity
best isp = unknown
for all isp ∈ MPD do
    (status, time) ← download next chunk for current quality level from isp
    if status is failed then
        report isp as failed to CP control server
        remove isp from MPD
    else
        if time < current best then
            best isp ← isp
            best time ← time
        end if
    end if
end for
return best isp
```

One of the first metric we look at is the download time. We compute the time to download a 4 second chunk for the highest quality bit rate. We find that the average time to download for a client when there is a node acting as an ISP in the same organization is less than 40% of the time when there is none. This supports the claim that user-perceived performance generally increases when content is being served by nearby caches.

Next, we examine how often the clients decide to switch the ISP they are currently downloading the content from. For this experiment, we look at logs collected at more than 100 PlanetLab nodes acting as clients, "playing back" a one hour long movie. In this experiment, each chunk is approximately 4 seconds of play back. We plot the data in Figure 6.3. We see that although a large number of clients did not switch ISPs at, there were other clients that changed the best CDN very frequently over that download period. This shows that client side measurements are very helpful in server selection.

6.5 Summary

In this chapter we presented a scalable and high performance content distribution architecture. In addition, this architecture is economically more sustainable as it opens up new sources of revenue for the ISPs. This architecture is both incrementally deployable

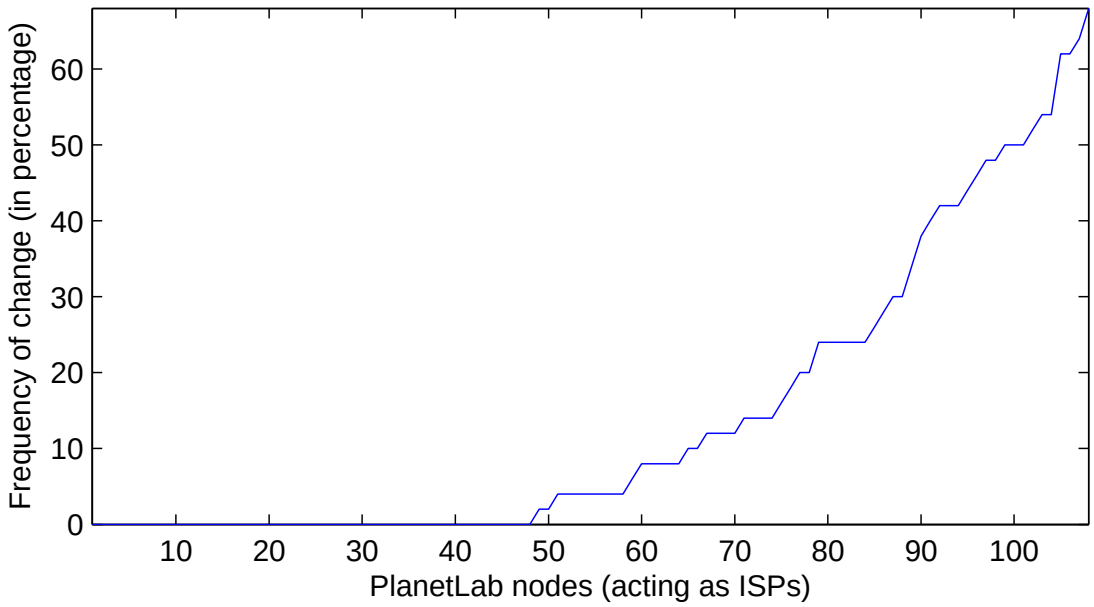


Figure 6.3: Number of changes in best ISP

and beneficial to early participation. We also provide a proof of concept implementation using our lab computers and PlanetLab nodes.

Our proposal makes use of the best parts of current architectures. It is based upon the reliability and performance of current commercial and homegrown CDNs. Our approach is different from current multi-CDN content distribution in that we allow for very dynamic participation for ISPs in content distribution. Instead of pre-arranged CDN participation, this architecture lets ISPs join and leave content distribution based upon their resource availability and need over time. These changes also require our architecture to include more dynamic communication between content providers and ISPs interested in participating in content distribution. Because ISPs participation is dynamic, we also needed more frequent control communication between the content providers and the clients so that the clients can have the most up-to-date information about the servers available to serve them.

To further expand this work, there are two future directions. The first one is to expand on some of the details on how ISPs can decide when to participate in content distribution. For instance, will it make more sense for ISPs to join the distribution when they realize that there customers are being served from some faraway ISPs/CDNs?

Additionally, we can compare different approaches the content providers can take to identify the best ISP to serve any given request. Second future direction is to deploy a more realistic implementation and compare it with existing systems.

Chapter 7

Conclusion and Discussion

In this thesis, we studied how large scale content distribution works in the Internet today. We also studied the challenges they face and we identified a number of solutions to these problems. In addition to understanding how current content distribution architectures work, we also analyzed the impacts of such systems on one of the major players in the Internet, the Internet service providers (ISPs). For instance, when we study a content distribution system, we paid close attention to how any specific design choice made by the content providers affect the ISPs.

The original YouTube architecture was fairly simple[1]. They deployed six large data centers in the US and the content was replicated in all of the data centers. Therefore, users could be served from any of the six data centers. The system was designed in such a way that the front end web server serving any given request directed a user in a proportional load balancing way. This meant that the probability of any given request to be routed to a datacenter was proportional to the size of the data center. The location of the datacenter or the user requesting a video was not a factor in routing the request. This approach was well suited to handle flash crowds but did not offer the best performance.

From the ISPs point of view, since YouTube did not even try to be location aware, the ISPs had to carry traffic between users and data centers that are far away. This obviously increased traffic load on ISP backbones. To understand the impact of such architecture, we also studied several "what if" questions exploring alternate strategies that allowed ISPs to obtain an estimate of YouTube's traffic and prepare for the effects in YouTube's content distribution architecture.

In contrast, in the current YouTube architecture, location awareness is one of the top priorities[2]. The current architecture defines several namespaces and maps video IDs to those namespaces. Using DNS those namespaces are mapped to a large number of YouTube cache locations. They group the cache locations as primary, secondary and tertiary. Additionally, YouTube also enlists help of a large number ISPs who host Google Global Cache inside their networks. Using location aware DNS YouTube directs a given request to a "primary" cache server that is supposed to be close to the user. In case of cache misses, the client is then redirected to a higher tier cache location. Except for the ISPs participating in Google Global Cache, all of the video delivery infrastructure employed by YouTube is its own. The large number cache locations and hierarchical structure, coupled with location aware and dynamic DNS allowed a very high performance as perceived by the user. This architecture handles flash crowds like phenomena primarily by using HTTP redirections. The current YouTube architecture could be considered better than the original one from an ISPs point of view. First, although not always accurate, DNS based localization is helpful. Therefore, most of the time the distance between clients and the content cache in minimized and that reduces the load on ISP backbones as most of the content delivery remains local. However, YouTube's homegrown delivery network has scalability challenges.

The third system we studied is Netflix. Netflix content delivery architecture differs significantly from that of YouTube because of its use of commercial CDNs[3]. Netflix employs three CDNs: Akamai, Limelight and Level3. It also uses adaptive streaming to deliver videos as opposed YouTube's use of progressive download. The system from Netflix side is simple. For any request, Netflix picks a CDN for the request and the CDN handles the rest, including localizing the client, locating closest server and delivering the actual content. One of the major findings in this work is that Netflix does not use network conditions or past performance of the CDNs in making the CDN selection. It maps users to CDNs and uses the same static mapping to make a CDN selection for the user irrespective to time of day, content or network conditions.

Looking again from the ISPs perspective, we see the same problems with non optimal DNS based geolocation as well as HTTP redirections employed by the CDNs. In addition, Netflix's static CDN selection likely increases the probability of clients communicating with far away cache servers which result in increased load for ISPs.

The final content delivery system we studied is the one employed by Hulu. We studied Hulu because as opposed to other three systems, Hulu's primary protocol is RTMP (and not HTTP)[4]. In many cases, Hulu's architecture resembles that of Netflix as they both enlist the services of the same commercial CDNs. Similar to Netflix, Hulu does not use network conditions in making CDN selection decisions. Hulu's CDN selection tries to make sure that each of the CDN gets to serve their share of content based upon some commercial agreements. In this work, we also geolocate and uncover the commercial CDN infrastructure.

Finally, we presented a scalable and high performance content distribution architecture. It is aimed at scaling content distribution to the whole Internet and use smarter client software that can make decisions based upon measurements in tandem with the content providers. In addition, this architecture is economically more sustainable as it opens up new sources of revenue for the ISPs. This architecture is both incrementally deployable and beneficial to early participation. We also provide a proof of concept implementation using our lab computers and PlanetLab nodes.

7.1 Future directions

There are two future directions that can be expanded upon this thesis. The first one is to expand upon the open CDN architecture idea. For instance, exploring details on how ISPs can decide when to participate in content distribution is one possible expansion. Will it be better for ISPs to join the distribution when they realize that there customers are being served from some faraway ISPs/CDNs? What are the ways in which a content providers can select the best ISP for any given request? Additionally, we can compare different approaches the content providers can take to identify the best ISP to serve any given request. Moreover, deploying a more realistic implementation and comparing it with existing systems will also help shed more light on the areas of improvement.

Second future direction is to study if we can change the Internet architecture as a whole and start with a clean slate. Currently, the Internet architecture is not well suited large-scale content distribution. Large scale content distribution will be more natural when content is considered a first class citizen. Therefore exploring new clean slate design ideas such as CCN[52] might be another future direction.

References

[1] Vijay Kumar Adhikari, Sourabh Jain, and Zhi-Li Zhang. YouTube Traffic Dynamics and Its Interplay with a Tier-1 ISP: An ISP Perspective. In *IMC*, 2010.

[2] Vijay Kumar Adhikari, Sourabh Jain, Yingying Chen, and Zhi-Li Zhang. Vivisecting YouTube: An Active Measurement Study. In *INFOCOM'12 Mini-conference*. IEEE, 2012.

[3] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Matteo Varvello, Volker Hilt, Moritz Steiner, and Zhi-Li Zhang. Unreeling Netflix: Understanding and Improving Multi-CDN Movie Delivery. In *INFOCOM'12*. IEEE, 2012.

[4] Vijay Kumar Adhikari, Yang Guo, Fang Hao, Volker Hilt, and Zhi-Li Zhang. A tale of three CDNs: An active measurement study of Hulu and its CDNs. In *Computer Communications Workshops (INFOCOM WKSHPS), 2012 IEEE Conference on*, pages 7–12. IEEE, 2012.

[5] Vijay Kumar Adhikari and Zhi-Li Zhang. Feel free to cache: enabling ISPs for content distribution. Technical report, University of Minnesota, 2012.

[6] Erik Nygren, Ramesh K. Sitaraman, and Jennifer Sun. The akamai network: a platform for high-performance internet applications. *SIGOPS Oper. Syst. Rev.*, 44(3):2–19, August 2010.

[7] W. Norton. The evolution of the US Internet peering ecosystem. Equinix White Papers, 2004.

[8] Rupa Krishnan, Harsha V. Madhyastha, Sridhar Srinivasan, Sushant Jain, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. Moving beyond end-to-end path

information to optimize CDN performance. In *IMC '09*, pages 190–201, New York, NY, USA, 2009. ACM.

[9] Cheng Huang, Angela Wang, Jin Li, and Keith W. Ross. Measuring and evaluating large-scale CDNs (Paper withdrawn). In *IMC '08*, pages 15–29, New York, NY, USA, 2008. ACM.

[10] V. Jacobson, C. Leres, and S. McCanne. Tcpdump Man Page. *URL http://www.tcpdump.org/tcpdump_man. html*, 2003.

[11] Meeyoung Cha et al. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *IMC*, 2007.

[12] X. Cheng, C. Dale, and J. Liu. Statistics and social network of youtube videos. In *Proc. of IEEE IWQoS*, 2008.

[13] Phillipa Gill, Martin Arlitt, Zongpeng Li, and Anirban Mahanti. Youtube traffic characterization: a view from the edge. In *IMC '07*. ACM, 2007.

[14] Michael Zink, Kyoungwon Suh, Yu Gu, and Jim Kurose. Characteristics of youtube network traffic at a campus network - measurements, models, and implications. *Comput. Netw.*, 53(4), 2009.

[15] Mohit Saxena, Umang Sharan, and Sonia Fahmy. Analyzing video services in web 2.0: a global perspective. In *NOSSDAV '08: Proceedings of the 18th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 39–44, New York, NY, USA, 2008. ACM.

[16] J. Dilley, B. Maggs, J. Parikh, H. Prokop, R. Sitaraman, and B. Weihl. Globally distributed content delivery. *IEEE Internet Computing*, pages 50–58, 2002.

[17] A. Vakali and G. Pallis. Content delivery networks: Status and trends. *IEEE Internet Computing*, 7(6):68–74, 2003.

[18] Zakaria Al-Qudah, Seungjoon Lee, Michael Rabinovich, Oliver Spatscheck, and Jacobus Van der Merwe. Anycast-aware transport for content delivery networks. In *WWW '09*, pages 301–310, New York, NY, USA, 2009. ACM.

[19] T. Brisco. DNS Support for Load Balancing. RFC 1794, April 1995.

[20] T. Will. Introduction to the Singular Value Decomposition. *La Crosse, WI*, 2003, 2003.

[21] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot. Traffic matrix estimation: Existing techniques and new directions. *ACM SIGCOMM Computer Communication Review*, 32(4):174, 2002.

[22] R. Teixeira, A. Shaikh, T. Griffin, and J. Rexford. Dynamics of hot-potato routing in IP networks. *SIGMETRICS Perform. Eval. Rev.*, 32(1):307–319, 2004.

[23] R. Torres et al. Dissecting Video Server Selection Strategies in the YouTube CDN. In *ICDCS*, 2011.

[24] Venkata N. Padmanabhan et al. An investigation of geographic mapping techniques for internet hosts. In *SIGCOMM*, 2001.

[25] Yingying Chen, Sourabh Jain, Vijay Kumar Adhikari, and Zhi-Li Zhang. Characterizing roles of front-end servers in end-to-end performance of dynamic content distribution. In *IMC*, 2011.

[26] Sandvine. Global Internet Phenomena Report, Spring 2011. `http://www.sandvine.com/news/global_broadband_trends.asp`, 2011.

[27] Adrian Cockroft, Cory Hicks, and Greg Orzell. Lessons Netflix Learned from the AWS Outage. Netflix Techblog, 2011.

[28] Microsoft Silverlight. `http://www.microsoft.com/silverlight/`.

[29] Pomelo LLC. Analysis of Netflix's security framework for 'Watch Instantly' service., 2009.

[30] S. Akhshabi et al. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *MMSys*, 2011.

[31] L. Daigle. WHOIS Protocol Specification, 2004.

[32] Tamper Data. `addons.mozilla.org/en-US/firefox/addon/tamper-data`.

[33] Dummynet. http://info.iet.unipi.it/~luigi/dummynet/.

[34] Adrian Cockroft. Netflix Cloud Architecture. Velocity conference, 2011.

[35] Amazon Web Services. http://aws.amazon.com.

[36] get-flash-videos, A command line program to download flash videos. http://code.google.com/p/get-flash-videos/, 2011.

[37] Planetlab | an open platform for developing, deploying, and accessing planetary-scale services. http://www.planet-lab.org.

[38] Public DNS Servers. http://sites.google.com/site/kiwi78/public-dns-servers.

[39] Balachander Krishnamurthy, Craig Wills, and Yin Zhang. On the use and performance of content distribution networks. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, IMW '01, pages 169–182, New York, NY, USA, 2001. ACM.

[40] Bernhard Ager, Wolfgang Mühlbauer, Georgios Smaragdakis, and Steve Uhlig. Comparing dns resolvers in the wild. In *Proceedings of the 10th annual conference on Internet measurement*, IMC '10, pages 15–21, New York, NY, USA, 2010. ACM.

[41] Dilip Kumar Krishnappa, Samamon Khemmarat, Lixin Gao, and Michael Zink. On the feasibility of prefetching and caching for online tv services: a measurement study on hulu. In *Proceedings of the 12th international conference on Passive and active measurement*, PAM'11, pages 72–80, Berlin, Heidelberg, 2011. Springer-Verlag.

[42] Ao-Jan Su et al. Drafting behind akamai: inferring network conditions based on cdn redirections. *IEEE/ACM Trans. Netw.*, 2009.

[43] Cheng Huang, Angela Wang, Jin Li, and Keith W. Ross. Understanding hybrid cdn-p2p: why limelight needs its own red swoosh. In *NOSSDAV*, 2008.

[44] A.B. Downey. Using pathchar to estimate internet link characteristics. In *ACM SIGCOMM CCR*, 1999.

[45] Manish Jain and Constantinos Dovrolis. Pathload: A Measurement Tool for End-to-End Available Bandwidth . In *PAM'02*, March 2002.

[46] Daniele Croce, Taoufik En-Najjary, Guillaume Urvoy-Keller, and Ernst W. Biersack. Fast available bandwidth sampling for adsl links: Rethinking the estimation for larger-scale measurements. In *PAM '09*, pages 67–76, 2009.

[47] M. Pathan and R. Buyya. A taxonomy of CDNs. *Content delivery networks*, pages 33–77, 2008.

[48] M. Day, B. Cain, G. Tomlinson, and P. Rzewski. A Model for Content Internetworking (CDI), 2003.

[49] Netflix Open Connect Content Delivery Network. `https://signup.netflix.com/openconnect`.

[50] Download | ITEC - Dynamic Adaptive Streaming over HTTP. `http://www-itec.uni-klu.ac.at/dash/?page_id=6`.

[51] lighttpd fly light. `http://www.lighttpd.net`.

[52] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 1–12, New York, NY, USA, 2009. ACM.