Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 Keller Hall
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 11-012

Vivisecting YouTube: An Active Measurement Study

Vijay Kumar Adhikari, Sourabh Jain, Yingying Chen, and Zhi-li Zhang

July 11, 2011

Vivisecting YouTube: An Active Measurement Study

Vijay Kumar Adhikari, Sourabh Jain, Yingying Chen, and Zhi-Li Zhang
Department of Computer Science & Engineering, University of Minnesota
{viadhi,sourj,yingying,zhzhang}@cs.umn.edu

ABSTRACT

We build a distributed active measurement infrastructure to uncover the internals of the YouTube video delivery system. We deduce the key design features behind the YouTube video delivery system by collecting and analyzing a large amount of video playback logs, DNS mappings and latency data and by performing additional measurements to verify the findings. We find that the design of the 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. Further, YouTube employs a set of sophisticated mechanisms to handle video delivery dynamics such as cache misses and load sharing among its globally distributed cache locations and datacenters.

1. INTRODUCTION

Since its inception in 2005, YouTube has seen explosive growth in its popularity; today it is indisputably the world's largest video sharing site [16]. 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 (in the context of most recent Internet development). Before Google took over YouTube in late 2006 [7] and subsequently re-structured the YouTube video delivery infrastructure, it was known that YouTube employed several data centers in US [2] as well as third-party content delivery networks [9, 17] to stream videos to users. Since Google's take-over, YouTube has grown rapidly and became several-fold larger both in terms of users and videos. For instance, using inter-domain traffic collected in 2007 and 2009 at hundreds of ISPs across the world, the authors of a recent study [8] show that Google has become one of the top five *inter-domain* traffic contributors in 2009 (while not even among the top 10 in 2007); a large portion of Google's traffic can be attributed to YouTube. While it is widely expected that Google has incorporated the YouTube delivery system into its own vast Internet infrastructure in the past few years, little is known how Google leverages its engineering talents and resources to re-design and re-structure the YouTube video delivery infrastructure to meet the rapidly growing user demands as well as performance expectations.

This paper attempts to "reverse-engineer" the YouTube video delivery system through large-scale active measurement, data collection and analysis. The rationale behind our YouTube vivisection study is multi-fold. First of all, we are not simply interested in uncovering, e.g., where YouTube video cache servers or data centers are located, but more in the *design principles* underlying Google's re-structuring of the YouTube video delivery system. For instance, we are particularly interested in answering the following important design questions: i) how does YouTube design and deploy a *scalable* and *distributed* delivery infrastructure to match the geographical span of its users and meet varying user demands? ii) how does YouTube perform load-balancing across its large pool of Flash video servers (and multiple locations)? iii) given the sheer volume of YouTube videos which renders it too costly, if not nearly impossible, to replicate content at all locations, what strategies does YouTube use to quickly find the right content to deliver to users? and iv) how does YouTube handle the vastly differing popularity of videos in addressing the questions ii) and iii) above?

Answer to these questions are very important for future large-scale 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. It is with these goals in mind that we set out to uncover the internals of the YouTube video delivery system by building a globally distributed active measurement infrastructure, and to deduce its underlying design principles through careful and extensive data analysis and experiments. In the following we briefly outline our work.

The global scale of the YouTube video delivery system and the use of separate servers to service the stan-

dard HTML content vs. video content (see Section 2.1) pose several challenges in *actively* measuring, and collecting data from the YouTube video delivery system. To address these challenges, we have developed a distributed active measurement platform with more than 1000 vantage points spanning five continents. As described in Section 2.2, our globally distributed measurement platform consists of three key components: i) PlanetLab nodes that are used for crawling the YouTube website, performing DNS resolutions, and YouTube video playback; ii) open recursive DNS servers to provide additional vantage points to perform DNS resolutions; and iii) emulated YouTube Flash video players running on PlanetLab nodes and two 24-node compute clusters in our lab for downloading and "playing back" YouTube videos and for larges-scale data analysis. This distributed active measurement platform enables us to play half a million YouTube videos , and collect extensive YouTube DNS mappings at each vantage point, and detailed video playback traces.

Through careful and systematic data analysis and inference and by conducting extensive experiments to test and understand the behavior of the YouTube video delivery system, we not only geo-locate (a large portion of) YouTube video server locations and map out its cache hierarchy, but also uncover and deduce the logical design of the YouTube video delivery system and the key mechanisms it employ to handle delivery dynamics. We provide a high-level summary of the key findings regarding the YouTube design below, and refer the reader to Section 3 for more specifics.

• The design of the YouTube video delivery system consists of three components: a "flat" *video id* space, a *multi-layered* logical server organization consisting of five *anycast*[1] namespaces (and two *unicast* namespaces), and a *3-tiered* physical cache hierarchy with (at least) 38 primary locations, 8 secondary and 5 tertiary locations.

• YouTube assigns a fixed-length (unique) identifier to each video and employs a fixed hashing to map the *video id* space to the logical namespaces. Such a fixed *video-id-to-hostname* mapping makes it easy for YouTube web servers to generate URLs referencing videos that users are interested in without knowing where each video is located or currently cached.

• YouTube leverages (coarse-grained) *locality-aware* DNS resolution to service user video requests by mapping the (first-layer) logical hostnames to physical video servers (IP addresses) in *primary* cache locations that are reasonably close to users. To perform finer-grained *dynamic* load-balancing and handle cache misses, YouTube employs a clever and complex mix of dynamic HTTP redirections and additional rounds of DNS resolution (of *anycast* and *unicast* hostnames). More specifically, YouTube utilizes the layered *anycast* namespaces to redirect video requests across the physical cache hierarchy , and the *unicast* namespaces to redirect video requests from one video server to another within the same location or tier.

In a nutshell, by introducing multiple (layered) DNS namespaces and employing a fixed mapping of the video id space to the DNS namespaces, YouTube cleverly leverages the existing DNS system and the HTTP protocol to map its logical server cache hierarchy to the physical delivery infrastructure – this enables a scalable, robust and flexible design. In Section 4, Section 5 and Section 6 we present more details regarding how we derive these findings, including analysis performed, the methods used, and additional experiments conducted to verify and validate the findings. In Section 7 we will summarize the lessons learned, state how YouTube design differs from other CDNs, and briefly discuss what insights these lessons may shed on the design of large-scale Internet content delivery systems in general.

Related Work. Before we delve into our work of reverse-engineering the YouTube video delivery system, we briefly discuss several pieces of related work. Most existing studies of YouTube mainly focus on user behaviors or the system performance. For instance, the authors in [3,4] examined the YouTube video popularity distribution, popularity evolution, and its related user behaviors and key elements that shape the popularity distribution using data-driven analysis. The authors in [6] investigate the (top 100 most viewed) YouTube video file characteristics and usage patterns such as the number of users, requests, as seen from the perspective of an edge network. Similarly, [12] compares multiple video service providers in terms of data center locations and user perceived delay. Another study [17] analyzed network traces for YouTube traffic at a campus network to understand benefits of alternative content distribution strategies. In a recent study carried in [2], the authors utilize the Netflow traffic data *passively* collected at various locations within a tier-1 ISP to uncover the locations of YouTube data center locations, and infer the load-balancing strategy employed by YouTube at the time. The focus of the study is on the impact of YouTube load-balancing on the ISP traffic dynamics, from the perspective of the tier-1 ISP. As the data used in the study is from spring 2008, the results reflect the YouTube delivery infrastructure *pre Google restructuring*. Another work that is similar to this paper is presented in [15]. In [15], Torres et al. examine data collected from multiple networks to uncover the server selection strategies YouTube uses. Our work differs significantly from [15] because the focus of our work is to

[1]In this paper by an *anycast* (DNS) namespace we mean that each (DNS) hostname in the space is – *by design*– mapped to multiple IP addresses ("physical" video servers). In contrast, a *unicast* hostname is mapped to a single (unique) IP address.

uncover the overall architecture (see Section 3), and not just how YouTube does server selection. 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. Initial results from this research also appear in [1].

2. YOUTUBE BASICS AND ACTIVE MEASUREMENT PLATFORM

In this section we first briefly describe the basics of YouTube video delivery, in particular, the roles of YouTube web servers and Flash video servers. We then provide an overview of our distributed *active* YouTube measurement and data collection platform.

2.1 YouTube Video Delivery Basics

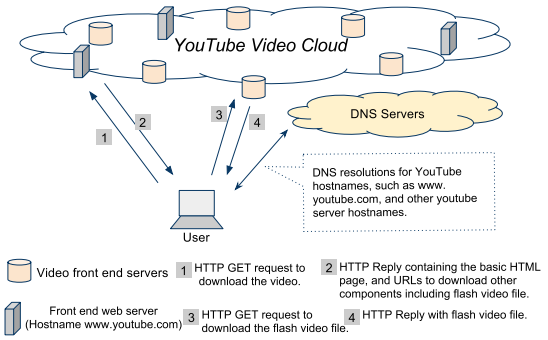


Figure 1: YouTube Typical Steps involved in YouTube Video Delivery.

Users typically go to the YouTube website and watch videos using a web browser equipped with the Adobe Flash Player plug-in, where videos are streamed from (individual) YouTube Flash video servers (separate from the YouTube web servers). In the following, we provide a brief overview of the sequence of steps involved when a user goes to the YouTube website and watches a video directly from the website. The steps are schematically shown in Fig. 1. Each YouTube video is identified by a URL (more specifically, the 11-literal string after `v=` is the *video id*, namely, `tObjCw_WgKs`, in the example URL below). When a user goes to the YouTube website, or clicks on any URL of the form `http://www.youtube.com/watch?v=tObjCw_WgKs` on an existing YouTube web page using her browser, the browser first resolves the hostname `www.youtube.com` using the local DNS server (LDNS). The HTTP request from the user is then directed to one of the YouTube web servers returned by the YouTube DNS system. The web server returns a HTML page with one or more *embedded* (i.e., invisible) URLs of certain forms, e.g., `v23.lscache5.c.youtube.com`, pointing to the Flash video server.

When the user clicks the playback button of an embedded Flash video object on the page (or when the video starts automatically), another round of DNS resolution occurs, resolving, say, `v23.lscache5.c.youtube.com`, to one of the many YouTube (*front-end*[2]) Flash video servers, which then streams the video to the user's browser. In fact, the YouTube front-end video server first resolved may redirect (via the HTTP request redirection) the video request to another video server, which may again redirect the request, until it finally reaches a video server that is able to stream the video to the user's browser. Hence multiple additional rounds of DNS resolutions (and HTTP redirections) may happen before the video is finally delivered to the user.

In summary, YouTube uses separate (web vs. Flash video) servers to deliver HTML webpages and videos to users. Furthermore, YouTube uses both DNS resolution and HTTP redirection (as part of the Flash video delivery operations) to select appropriate video servers for video delivery to users. Clearly, these strategies are needed so as to to take into account factors such as user locality, availability (and popularity) of videos at various video servers, the status of video servers (e.g., how busy a video server is), and so forth. This logical structure of YouTube video delivery as well as its globally distributed physical delivery infrastructure pose several challenges in *actively* measuring and collecting data from the YouTube delivery system. For example, one cannot simply crawl the YouTube website and perform DNS resolutions to uncover YouTube video servers, due to the Flash video objects used by video playback and multiple redirections among video servers (many of which are not directly visible on YouTube web pages) within a multi-tiered video server cache hierarchy deployed by YouTube (see Section 3).

2.2 Active Measurement Platform

The globally distributed video delivery infrastructure of YouTube necessitates a geographically dispersed (*active*) measurement platform. As shall be clear later, the YouTube (or rather, Google) DNS system takes into account the locality of users (or rather the local DNS servers issuing the DNS requests) to resolve its DNS names to IP addresses: when the YouTube DNS system receives a DNS request from a local DNS server (of a user), only IP addresses (of a web server or a video server) "close" to the local DNS server – purportedly also "close" to the user making a video request – are returned. Hence in order to uncover as many YouTube

[2]We refer to the YouTube Flash video servers with DNS-resolvable public IP addresses as the *front-end* video servers, as we understand that they are likely supported by multiple physical machines behind the scene, or even a number of back-end video server clusters. This is especially likely to be the case for YouTube secondary and tertiary video cache servers, see Section 3.

servers as possible, we need a large number of vantage points that are geographically dispersed to collect data.

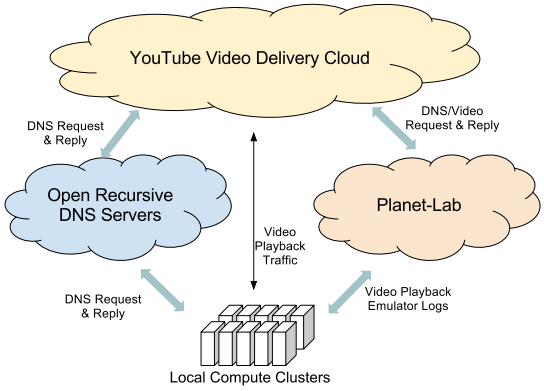


Figure 2: Our Active Measurement Platform.

To address the challenges posed by the YouTube global delivery infrastructure, we have developed a distributed active measurement and data collection platform consisting of the following three key components (see Fig.2): i) PlanetLab nodes that are used for both crawling the YouTube website, performing DNS resolutions, as well as functioning as proxy servers for YouTube video playback (see below); ii) open recursive DNS servers that are used for issuing DNS requests and verifying DNS resolution results; and iii) emulated YouTube Flash video players running on PlanetLab nodes and two 24-node compute clusters in our lab and a proxy web server architecture using the PlanetLab nodes for forwarding YouTube videos to our compute clusters for video playback. Our platform utilizes 471 PlanetLab nodes that are distributed at 271 geographical dispersed sites (university campuses, organization or companies), and 843 open recursive DNS servers located at various ISPs and organizations.

As alluded to earlier , simply crawling YouTube website and web pages to extract URLs that reference YouTube videos is insufficient; one needs to actually play those Flash videos to uncover the tiered video cache servers and the process of YouTube redirections among them. To circumvent this difficulty, we developed an emulated YouTube Flash video player in Python which emulates 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 object, and records a detailed log of the process. Note that during this process, multiple HTTP request redirections and DNS resolutions may be involved before the Flash video objects can be "downloaded" for playback. The detailed text-based logs recorded for each step of the process contain a variety of information such as the hostnames and URLs involved in each step, the HTTP request and response messages and their status codes, the basic HTML payload and timestamps for each of the steps. In particular, when there is a failure on the server side, for example, due to the file is not available temporarily, or the server is overloaded and fails to retrieve the video from an upstream or back-end server, the emulated player records the HTTP error code sent by the server.

In addition to playing back all the videos using the Flash video player emulator, We also play a large number videos using standard web browsers such as Mozilla Firefox and record all the HTTP and DNS transactions. We use this experiment to verify that our emulator was functioning correctly and the data we collected was not biased because of the use of an emulator. Due to the resource constraints on the PlanetLab nodes, we cannot run standard browsers directly on them to play those YouTube videos. Hence we configure the PlanetLab nodes to function as proxy servers using Squid [13], and run Firefox on two 24-node compute clusters in our lab. As proxy servers, the PlanetLab nodes forward all HTTP requests and replies to the browsers running on our back-end compute clusters; but to YouTube the video requests appear to come from the PlanetLab nodes that are geographically dispersed across multiple continents. This is crucial, as we need a large number of geographically dispersed client machines. 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, contact the YouTube video servers thus resolved to download and play back videos, and record a detailed log of the process. This capability enables us to use the 843 open recursive DNS servers as additional vantage points. Hence we have a total of $1,314$ globally distributed vantage points for active YouTube measurement and data collection.

Testing and Verification. Before we deployed the active measurement and data collection platform, we went through a careful testing and verification process to ensure the correctness of our platform. We selected a set of 85 geographically dispersed PlanetLab nodes as proxy servers and a list of several hundreds YouTube videos with varying popularity crawled from the YouTube website. We "manually" played back these videos in our lab machines using the PlanetLab nodes as proxy servers, and recorded a detailed log of each process. We compared these "manually" collected datasets with those collected via our emulated YouTube Flash video players (using the same set of PlanetLab nodes as proxy servers), verify the consistency of the manually and automatically collected datasets across different Planet-

Lab nodes to ensure the correctness of our platform.

2.3 Measurement Methodology and Datasets

Given the globally distributed active measurement platform described above, in this section we outline the methodology used (and the various steps involved) for collecting YouTube data, and describe the datasets thus collected. We conclude by briefly commenting on the completeness (or incompleteness) of the datasets. We make our data and code publicly available at http://networking.cs.umn.edu/youtubedata for the benefit of the community.

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. We repeat the entire process multiple times by extracting additional videos and playing back them, or repeating the process using the same set (or a subset) of videos.

Furthermore, based on our initial analysis of the YouTube DNS namespace structures and HTTP request redirection sequences, we have also conducted extensive "experiments" to further test and understand the behavior of the YouTube video delivery system. For instance, we uploaded our own videos on the YouTube website to test how YouTube handles "cold" (rarely viewed) videos. We also issued video download and playback requests to specific YouTube video servers to test the relations between the YouTube video id space and DNS namespaces as well as to understand the factors influencing the redirection decision process. (See Sections 5 and 6.3 for the discussion of some of these experiments.) In the following we summarize the main datasets we have collected and used in this study.

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. This yields a collection of detailed video playback traces and HTTP logs. These traces and logs play a critical role in our analysis and understanding the YouTube DNS namespace structures, video server cache hierarchy and HTTP request redirection logic and decision process.

YouTube DNS Names, IP Addresses and DNS-to-IP Resolution Mappings. Using the video playback traces, we extracted all the DNS name and IP address mappings from the DNS resolution processes at each vantage points. In total, we extracted $6,150$ YouTube DNS names from all the video playback traces. We repeated the DNS resolution processes multiple times at different times of the day using the PlanetLab nodes and open recursive DNS servers to test and verify the DNS-to-IP address mappings, to obtain additional mappings (if any), and to check for the completeness of the mappings. In total, we obtained a total of $5,883$ IP addresses for the YouTube DNS names we collected.

Other Data and Experiments. In addition to the above datasets, we also performed other measurements and collected additional data. For instance, we performed the round-trip-delay (RTT) measurements over time from each PlanetLab node used in our active measurement platform to each of the $5,883$ IP addresses (YouTube video servers), and collected several RTT datasets that are used for geo-locating the YouTube video servers (see Section 5). As mentioned earlier,

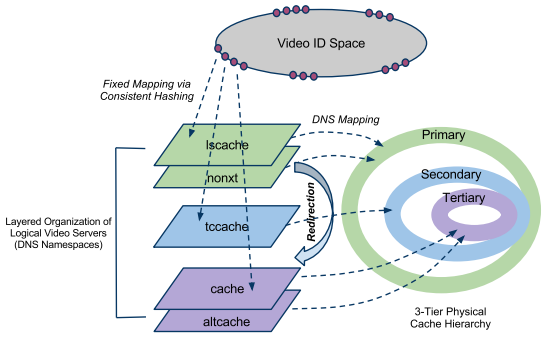


Figure 3: YouTube Architectural Design.

we also conducted extensive experiments to further test and understand the behavior of the YouTube video delivery system, and collected the relevant measurement data and logs for each experiment.

(In)Completeness of Our Measurement Data. We conclude this section by briefly commenting on the completeness and incompleteness of the collected datasets. Clearly, the list of videos we collected represents only a small sample of all YouTube videos. In addition, due to the bias inherent in our crawling process, the videos collected tend to be popular videos (especially the initial video seeds used for the breadth-first search process), and those that are related to the popular videos. To partially correct this bias, we uploaded several videos of our own, the *explicit* purpose to test and understand how "cold" videos affect the YouTube delivery process, in particular redirection decisions. In terms of the YouTube DNS names and namespaces used for YouTube video delivery, we believe that they are (nearly) complete, with the possible exception of the *unicast* `r.xxx-xxx.c.youtube.com` *namespaces* (see Section 3). We are also confident that the YouTube IP addresses we collected represent a large majority of video servers and cache locations deployed by YouTube However, the design of the DNS namespaces and cache hierarchy makes it easier for YouTube to deploy additional video servers at existing and new locations to meet user demands. Nonetheless, we believe that any incompleteness due to the missing DNS name-to-IP-address mappings does not fundamentally affect the key findings of our study.

3. SUMMARY OF KEY FINDINGS

In this section we provide a summary of our key findings regarding the design and operations of the YouTube global video delivery system. This serves as the road map for the ensuing sections, where we will provide specifics as to how we arrive at these findings, including the data analysis and inference as well as experiments we have performed to verify the findings.

3.1 Overall Architecture

As schematically shown in Fig. 3, the design of the YouTube video delivery system consists of three major components: the *video id space*, the *multi-layered* organization of multiple *anycast* DNS namespaces (representing "logical" video servers), and a 3-tier "physical" server cache hierarchy with (at least) 38 *primary* locations, 8 *secondary* locations and 5 *tertiary* locations.

YouTube Video Id Space. Each YouTube video is "uniquely" identified using a "flat" identifier of 11 literals long, where each literal can be [A-Z], [0-9], - or _ (see Section 4 for details). The total size of the YouTube video id space is effectively 64^{11}.

Three-Tier (Physical) Server Cache Hierarchy and Their Locations. Using the YouTube IP addresses seen in our datasets, we geo-map the YouTube "physical" video server cache locations, which are dispersed at five continents (see Fig. 4). Through in-depth analysis of our datasets, we deduce that YouTube employs a 3-tier physical cache hierarchy with (at least) 38 *primary* cache locations, 8 *secondary* and 5 *tertiary* cache locations. About 10 of the primary cache locations are co-located within ISP networks (e.g., Comcast and Bell-Canada), which we refer to as *non-Google* cache locations. Each location contains varying number of IP addresses ("physical" video servers), and there are some overlapping between the primary and secondary locations.

Multi-Layered Anycast DNS Namespaces. YouTube videos and (physical) cache hierarchy are tied together by a set of (logical) *anycast* namespaces as well as *unicast* namespaces. YouTube defines five (*anycast*) DNS namespaces, which are organized in multiple layers, each layer representing a collection of *logical* video servers with certain roles. Logical video servers at each layer are mapped to IP addresses (of "physical" video servers residing) at various locations within a particular tier of the physical cache hierarchy. As shown in Table 1, there are a total of five *anycast* namespaces, which we refer to as, *lscache*, *nonxt*, *tccache*, *cache* and *altcache* namespaces; each namespace has a specific format. Columns 4-6 show the total number of IPs, prefixes, and locations each DNS namespace is mapped.

The first two namespaces, *lscache* and *nonxt*, contain 192 DNS names representing 192 *logical* video servers; and as will be shown later, they are mapped to the *primary* cache locations in the YouTube physical cache hierarchy. The *tccache* namespace also contains 192 DNS names representing 192 *logical* video servers; but they are mapped to the *secondary* cache locations in the YouTube physical cache hierarchy. The last two namespaces, *cache* and *altcache*, contain 64 DNS names representing 64 *logical* video servers; they are mapped to the *tertiary* cache locations in the YouTube physical cache hierarchy.

Unicast Namespaces. In addition, for each IP ad-

Table 1: Youtube *Anycast* (first five) and *Unicast* (last two) Namespaces.

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

Figure 4: Geographical distribution of YouTube Video Cache Locations.

dress (a"physical" video server), YouTube also defines a (unique) *unicast* DNS hostname. Namely, there is a one-to-one mapping between this hostname and the IP address. As shown in Table 1, the *unicast* hostnames have two formats; we refer to the collection of hostnames of each format as the *rhost* and *rhostisp* (unicast) namespaces. The *rhost* namespace covers the IP addresses (physical video servers) residing in Google cache locations, whereas *rhostisp* namespace covers those in non-Google cache locations

Last but not the least, we remark that only the hostnames belonging to the *lscache namespace* are generally visible in the URLs or HTML pages referencing videos. DNS names belonging to the other four *anycast* namespaces as well as the two *unicast* namespaces occur mostly only in the URLs used in dynamic HTTP request *redirections* during video playback. The five layered *anycast* namespaces and two *unicast* namespaces play a critical role in dynamic HTTP request redirection mechanisms employed by YouTube (see below and Section 6.3).

3.2 Mechanisms and Strategies

The introduction of the layered organizations of *logical* video servers via multiple namespaces enables YouTube to employ several mechanisms and strategies to i) map videos to *logical* video servers via a fixed mapping, and ii) map *logical* video servers to physical video servers at various locations of its physical cache hierarchy through both (semi-static) DNS resolution and (dynamic) HTTP redirection mechanisms. This leads to scalable and robust operations of the YouTube video delivery system via *flexible* strategies, and allows YouTube to, for instance, effectively perform load balancing and handle cache misses.

***Fixed* Mapping between Video Id Space and *Logical* Video Servers (Anycast DNS Namespaces).** YouTube adopts a fixed mapping to map each video id uniquely to one of the 192 DNS names in the *lscache* namespace. In other words, the video id space is uniformly divided into 192 sectors, and each *lscache* DNS name – representing a *logical* video cache server – is responsible for a fixed sector. This *fixed* mapping between the *video id* space to the *lscache* DNS namespace (*logical* video servers) makes it easier for individual YouTube front-end *web servers* (www.youtube.com) to generate – *independently and in a distributed fashion* – HTML pages with embedded URLs pointing to the relevant video(s) users are interested in, regardless where users are located or how logical servers are mapped to physical video servers or cache locations. Furthermore, there is also a fixed and consistent mapping between the (anycast) namespaces. For example, there is one-to-one mapping between the 192 hostnames of the *lscache* namespace and those of the *tccache* namespace. The same also holds for the mappings between other *anycast* namespaces (see Section 5 for details). These fixed mappings make it easy for each (physical) video server to decide – given its logical name – what portion of videos it is responsible for serving.

(Coarse-grained) Locality-Aware Video Cache Selection via DNS Resolution. YouTube employs *locality-aware* DNS resolution to serve user video requests regionally by mapping *lscache* hostnames (logical video servers) to physical video servers (IP addresses) residing in *primary* cache locations reasonably close to users. This server selection strategy is fairly *coarse-grained*: based on our study, DNS queries for each *lscache* hostname from more than vantage points, are mapped to approximately 75 IP addresses distributed across the 38 primary cache locations; where each PlanetLab node site generally sees only one IP address per *lscache* hostname at a time (with some variations across the PlanetLab node sites or over time, see Section 6.1 for details).

Background fetch and *Dynamic* HTTP Request Redirection. The DNS resolution mechanism, while locality-aware, is generally agnostic of server load or

whether a server has the requested video in cache or not. Since the size difference of the primary cache locations can be quite large (ranging from 9 to 626 IP addresses per location) and the user demand is also likely to vary from one region to another, *dynamic load-balancing* is needed. Further, the cache size at each location may also differ significantly, and videos cached at each location can change over time (e.g., due to the differing popularity of videos), cache misses are inevitable: depending on how busy a video server at the primary location, it may either directly fetch (e.g., via the Google internal backbone network) the missed video in the background from another video server which has the video cached and serve the client directly, or redirect the request to another video server at a secondary or tertiary location. Our analysis and experiments show that more than 18% times, a user video request is redirected from a primary video cache server selected via DNS *lscache* name resolution to another video server.

YouTube employs a clever and complex mix of dynamic HTTP redirections and additional rounds of DNS resolution (of *anycast* and *unicast* hostnames) to perform finer-grained dynamic load-balancing and to handle cache misses. For instance, our investigation shows that YouTube utilizes the layered *anycast* namespaces to redirect video requests i) from one location to another location (especially from a non-Google primary cache location to a Google primary cache location via the use of *nonxt* namespace, also one tertiary cache location to another via the use of the *altcache* namespace); and ii) from a Google cache location in one tier to another tier (the primary to secondary or tertiary, or the second to tertiary via the use of the *tccache*, *cache* and *altcache* namespaces). There is a *strict ordering* as to how the *anycast* namespaces are used for redirection (see Fig. 7). At each step of the redirection process, the corresponding *anycast* hostname is resolved to an IP address via DNS. YouTube also utilizes the *unicast* namespaces to dynamically redirect a video request from one video server to a specific server usually (more than 90% of times) *within the same cache location*, and occasionally in a different location. The use of the layered *anycast* (and *unicast*) namespaces enables to enforce an strict ordering and control the redirection process. On the other hand, each redirection (and DNS resolution) process incurs additional delay: YouTube uses both a redirection count and a tag to keep track of the redirection sequences. Up to 9 redirections may happen, although they are rarely observed in the video playback traces we collected.

4. 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*. In this section we briefly discuss how YouTube videos are distributed in this *video id* space using a list of YouTube *video id*s we have collected. We also study how YouTube maps *video id*s to the (anycast) DNS namespaces – especially the *lscache* namespace corresponding to its primary cache locations.

4.1 YouTube Video Ids and Their Distribution

Each YouTube video is "uniquely" identified using a "flat" identifier of 11 literals long. 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. For instance, around $27K$ out of the $434K$ YouTube *video id*s in our list have 0 as the last symbol, but the three symbols after 0, namely, $1, 2, 3$, do not appear in the last position in any of the *video id*s in our list. The same observation holds for other symbols. In other words, 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*. To demonstrate this, we calculate the frequency of each symbol at any of the first 10 positions in the *video id*s, and the results are shown in the Figure 5. In this figure, the x-axis represents the 64 symbols, and y-axis is the probability of occurrence of each symbol at one of the first 10 positions. We see that each symbol has roughly the same probability of appearing at any position in the *video id*s. The same distribution holds regardless of the popularity of videos. We further verify this finding by collecting a large number of additional *video id*s as well as by randomly sampling the *video id* space by querying the YouTube data API.

4.2 Video Id to Anycast Hostname Mapping

As mentioned in Section 3, YouTube employs multiple *anycast* namespaces (see Table 1) to *logically* represent its (front-end) Flash video servers or caches that are geographically dispersed. In this section we explore the relation between YouTube videos (or rather, *video id*s) and these namespaces, in particular, the *lscache* namespace corresponding to video servers located at its primary cache locations. In other words, given a video requested by a user, we investigate how YouTube decides what DNS name representing a *logical* video server is responsible to deliver that video.

We first note that 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 6.3). We perform a systematic analysis of the YouTube web-

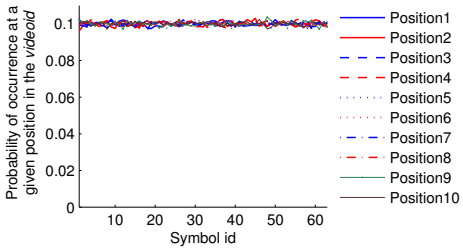


Figure 5: Distribution of *video id*s with respect to first symbol

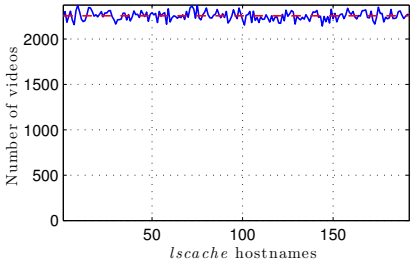


Figure 6: Number of videos mapped to each *lscache* hostname.

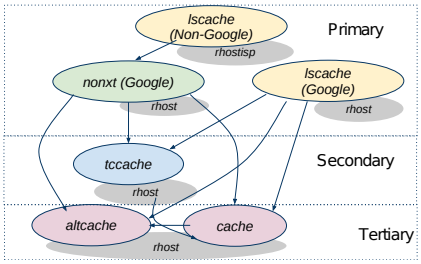


Figure 7: YouTube namespace hierarchy and redirection order.

pages, HTTP logs and video playback traces to investigate the relation of YouTube *video id*s and namespaces. 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 (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. We also perform experiments to verify these findings by *directly* requesting videos from different hostnames. When we attempt to download a video from a different *anycast* hostname than the one the video id is mapped to, we are always redirected to the correct *anycast* hostname. As an aside, we find that in contrast we can request any video from any of the *unicast* hostnames. Hence the mappings between *video id*s and the *unicast* namespaces are not fixed or unique.

Moreover, we find that YouTube employs a fixed mapping from the video id space to anycast namespaces such 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 6, using the collection of *video id*s we have collected. We see that there are approximately equal number of videos mapped to each of the *lscache* hostnames. We conclude that YouTube employs a *fixed* mapping which maps *video id*s uniformly to the anycast namespaces.

There are several advantages of using such a fixed mapping. First of all, the web servers that handle user requests can *independently* generate URLs referencing individual videos that users are interested in, without knowledge of where videos are stored and which video servers should be used to service the requested videos. Such a mapping also guarantees that, at least at the *logical* level, video loads are distributed (nearly) uniformly, as each anycast hostname (a *logical* video server) is responsible for a (roughly equal) sector of the flat *video id* space (recall that *video id*s are uniformly distributed in this space). This enables YouTube to employ a combination of locality-aware DNS, HTTP redirection (both among and across anycast and unicast namespaces) to perform load-balancing and handle cache misses among the (physical) video servers (see Section 6 for details).

5. CACHE NAMESPACES & HIERARCHY

YouTube define and employ 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 6000 IP addresses ("physical" video cache servers) that are distributed across the globe. Table 1 provides a summary of these namespaces, the number of IP addresses and locations they map to, and so forth. We utilize a combination of geo-mapping methods to geo-locate YouTube cache locations. Furthermore, we explore the correlations between the *anycast* and *unicast* namespaces and IP addresses they resolve to and investigate the relations among the *anycast* and *unicast* namespaces themselves to uncover and unroll the YouTube 3-tier cache hierarchy. In the following we present our methods and findings.

5.1 Geo-mapping YouTube Cache Locations

Using our initial datasets as well as performing additional DNS resolutions over time using a large number of vantage points, we collect a total of 5,883 unique IP addresses that the *anycast* and *unicast* names are resolved to. We employ a combination of heuristic geo-mapping methods to geolocate each of these IP addresses and classify them into 47 distinct cache locations. Due to space limitation, in the following we briefly describe the key ideas behind our geolocation framework here.

First of all, 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, based on the ownership of the IP address prefixes (WHOIS lookups [5]), the IP addresses can be sepa-

rated into two groups: 80% of the IP prefixes come from addresses assigned to Google/YouTube, 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*. More specifically, these two *unicast* namespaces are of the following two forms:
a) r[1-24].*city*[01-16][s,g,t][0-16].c.youtube.com,
b) r[1-24].*isp-city*[1-3].c.youtube.com.
Here *isp* represents the short name for the ISP (e.g., bell-canada) to which the corresponding IP address belong, and *city* is a 3 letter code representing the city name, which usually refers to the nearest airport, (e.g., syd).

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). We verify these city locations by performing round trip delay measurements, and find that these embedded city codes indeed point to the correct locations. We also verify that the caches residing within the ISP address spaces are indeed located inside the said ISPs' network (and at the right city locations) by performing traceroutes to those addresses.

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 [10]. We consider the delay between an IP address and a set of PlanetLab nodes (vantage points) as the feature vector representing the IP address. Next, we cluster all these IP addresses using k-means clustering algorithm, and use euclidean distance between the feature vectors as a distance measure. We assign each cluster a location, if we have at least one IP address in that cluster for which the location is already known using its *unicast* hostname. We have only three clusters in which no IP addresses in the cluster have an associated *unicast* name. Based on the nearest PlanetLab nodes for these clusters in terms of the round trip delay, we classify them into a coarser level geographical location. 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.

5.2 Unveiling the YouTube Cache Hierarchy

We perform extensive and in-depth analysis of our datasets, in particular, the DNS resolution logs and video playback traces, to investigate the mappings between YouTube namespaces and IP addresses and to uncover the relations between the 5 *anycast* namespaces as well as between the *anycast* and *unicast* namespaces. Based on the HTTP redirection sequences (see Section 6.3 for a more detailed analysis), there is a clear hierarchy among the 5 *anycast* namespaces, as shown in Fig. 7: 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. A *tccache* video server may redirect a video request to the corresponding *cache* sever but it never redirects it back to a *lscache* server; and a *cache* video server may redirect a video request to the corresponding *altcache* sever, but never the other way around. Furthermore, there is one-to-one correspondence between the IP addresses that the 5 *anycast* namespaces map to and the *unicast* namespaces (the shaded spaces in Fig. 7, as we will see in Section 6.3), YouTube utilizes *unicast* hostnames for redirecting video requests from one server to another server within the same *anycast* namespace (and the *nonxt* namespace is used for redirections from non-Google cache locations to Google cache locations). 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. In our datasets, the 192 *lscache* hostnames map to a total of 4,999 IP addresses belonging to both Google and other ISPs. Based on our geo-location clustering, these IP addresses are distributed in 38 locations. We refer to these cache locations to which the *lscache* namespace maps to as the YouTube *primary* cache locations, as user video viewing requests are first handled by the (Flash) video servers (IP addresses) in these locations. We note that the *lscache* namespace maps to both Google and non-Google primary cache locations[3].

Interestingly, 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

[3]Non-Google locations host only YouTube *primary* video caches.

that the *lscache* namespace maps: namely, only those IP addresses belonging to Google (and thus with the *unicast* hostnames in the *rhost* namespace). In other words, the *nonxt anycast* namespace covers exactly the set of IP addresses belonging to Google primary cache locations. More in-depth analysis reveals that *nonxt* hostnames only appear in the URLs of HTTP redirection requests generated by video servers located in *non-Google* primary cache locations. Hence we deduce that the *nonxt anycast* namespace is only used by video servers in non-Google primary cache locations to redirect a user video request for service at a Google primary cache location (see Section 6.3).

• **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.[4] They all have a unique *rhost unicast* hostname, and are distributed at only 8 locations, 4 in Europe and 4 in US. We refer to these cache locations that the *tccache* namespace maps to as the *secondary* cache locations: they are the locations where video servers in Google primary cache locations often redirect user video requests to, and they occasionally also redirect user video requests to the tertiary cache locations described below.

• **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, 2 in Europe and 3 in US. We refer to these cache locations that these two namespaces map to as the *tertiary* cache locations: they are the last resort for YouTube HTTP video redirections; an *altcache* hostname is used when a video server in one tertiary cache location wants to redirect a request to a different tertiary cache location. This implies that Google DNS servers are configured in such a manner that while a *cache* hostname, e.g., *v1.cache1.c.youtube.com*, is mapped to one tertiary cache location (that is perhaps relatively close to the user issuing the DNS query), the corresponding *altcache* hostname (i.e., *alt1.v1.cache1.c.youtube.com*) is always mapped to a different tertiary cache location (that is perhaps farther away from the user). See Section 6.1 for more in-depth analysis of YouTube DNS resolution.

6. 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. Before we discuss our findings, we first briefly discuss the experiments we have designed to measure, uncover and deduce these mechanisms and strategies and to validate our findings.

Experimental Methodology. To analyze and understand the YouTube video deliver, we have designed a series of experiments to learn how various factors such as video popularity, YouTube cache location and size, and time of the day affect these redirections. For the video playbacks, we divide the videos into two sets: i) *hot* videos which have a very high number of view counts (at least 2 million views) including all of the top trending videos published on the YouTube's homepage; and ii.) *cold* videos which have few than 100 view counts and never appeared as the top trending videos on YouTube's homepage. 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 modelled as a Poission process with inter-arrival rate of 10 seconds. For each of 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. We repeat these experiments several times using different sets of hot and cold videos, and conduct them over a period of several months. We perform extensive and in-depth analysis of the DNS-to-IP resolution data and video playback traces and HTTP logs collected from these experiments to analyze, deduce and uncover the YouTube video dynamics, in particular, the key factors affecting the fine-grained dynamic HTTP re-directions.

6.1 Locality-aware DNS Resolution

YouTube employs "locality-aware" DNS resolution to serve user video requests regionally by directing them to primary cache locations that are reasonably close to users. Using our datasets, we find that each hostname in the primary *lscache* namespace is mapped to more than 75 unique IP addresses distributed in the distributed across the 38 primary cache locations. Each PlanetLab node site generally sees only one IP address per *lscache* hostname at a time, with some variations across the PlanetLab node sites or over time (see below).

To characterize the granularity of locality-aware resolutions, we conduct the following analysis. For each PlanetLab node, we rank all 38 YouTube primary cache

[4]A few YouTube cache locations function both as primary and secondary cache locations.

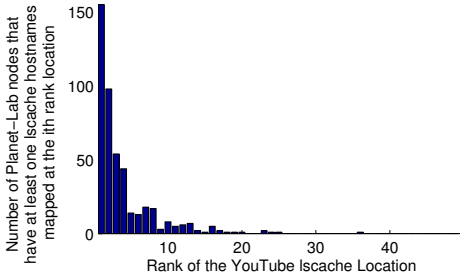


Figure 8: Locality aware DNS mappings for *anycast* hostnames.

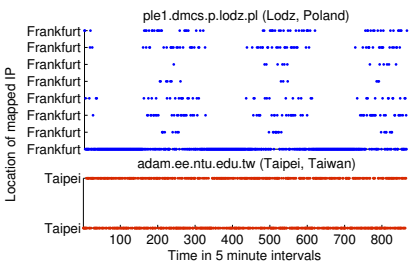


Figure 9: DNS resolution over time

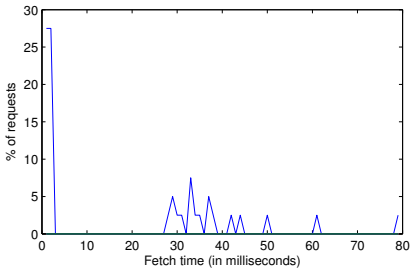


Figure 10: Fetch time distribution at a YouTube cache server.

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. For example, if the DNS resolutions of all 192 *lscache* hostnames yield a set of 192 unique IP addresses at a given PlanetLab node, we compute how many of these belong to the rank-1 YouTube location from the PlanetLab node, and so on. In Fig. 8 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 (in terms of round-trip time measurements). Only a very small number of PlanetLab nodes have all the *lscache* hostnames mapped to farther locations. This analysis shows that YouTube DNS servers generally map each *anycast* hostname to a nearby YouTube primary cache location.

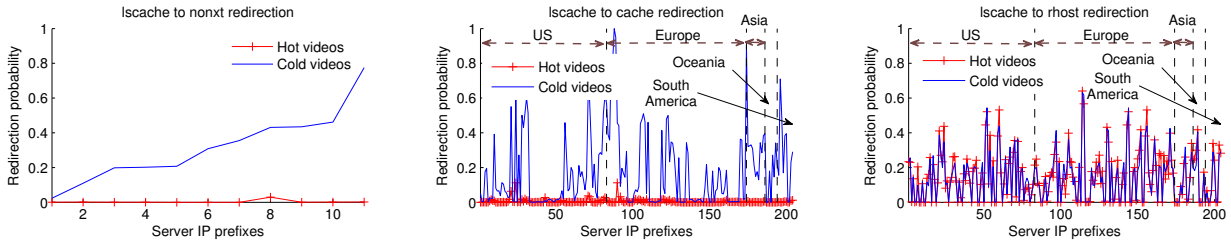
Using our DNS mapping data collected over several months, we also investigate whether YouTube adjusts the number of IP addresses mapped to each *lscache* hostname over time to, say, adapt to the changing loads at particular cache locations or regions of users. To analyze this, we create a *temporal matrix* of DNS name to IP address mapping matrix for each *lscache* 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 9 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 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).

6.2 Handling Cache Misses via Backend Fetching

To handle cache misses, YouTube cache servers use two different approaches: (a) fetching content from the backend datacenter 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 10 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 datacenter (cf. [11]).

6.3 HTTP Redirections Dynamics



(a) Non-Google Cache Locations (b) Google Cache Locations (c) Google Cache Locations

Figure 11: Comparison of redirection probabilities.

YouTube employs a clever and complex mix of dynamic HTTP redirection mechanisms and additional rounds of DNS resolutions to perform fine-grained dynamic load balancing, handle cache misses, and so forth. The video redirection logs reveal that HTTP redirections always follow a specific namespace hierarchy, as shown in Fig. 7. We examine the key factors affecting when redirections are performed, e.g., video popularity, cache location and size, server load and time-of-the day etc., and present the results. Our analysis of video redirection logs 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 Fig. 11 we demonstrate how redirection probability is distributed for *hot* and *cold* at 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 Fig. 11(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 Fig. 11(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 11(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 much fine granular load balancing by redirecting the users from possibly a very *busy* server to a less busy server at the same location.

6.4 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 HTTP session 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 12 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 son an average HTTP redirections increase the video initialization time by more than 33% in comparison to video initialization time for no redirection scenarios.

7. LESSONS LEARNED & INSIGHTS GAINED

In this paper we set out to reverse-engineer the YouTube video delivery system by building a globally distributed active measurement platform. Through careful and extensive data collection, measurement and analysis, we have uncovered and geo-located YouTube's 3-tier physical video server hierarchy, and deduced the key design features of the YouTube video delivery system. As expounded earlier in the paper, YouTube introduces an elaborate layered logical namespace structure and employs a number of clever and complex mix of DNS resolutions and HTTP redirection techniques to service user requests regionally while perform dynamic load-

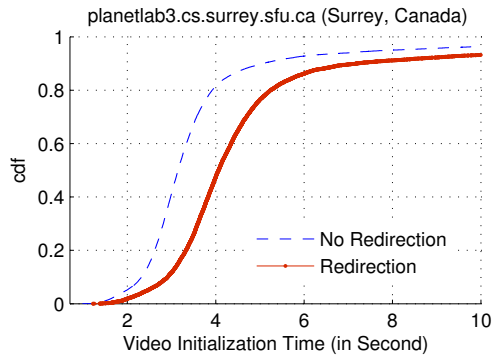


Figure 12: An example distribution of video initialization time.

balancing and effectively handle cache misses. This multi-layered logical namespace (logical server) organization separating the video (*id*) space and physical cache hierarchy also offers YouTube enormous flexibility. For instance, it can easily deploy additional (physical) video servers at either an existing or new primary cache location to meet user demands; it can even change its server selection or load-balancing strategies, without significant changes to its software substrate. All these can be done by simply adding new DNS name-to-IP address mappings, or altering how such mappings are performed. Additionally, most aspects of the YouTube video delivery system do not require any "global" coordination. Local load sharing only requires information sharing inside a cache location, and the hostname of the server in higher tier of the cache hierarchy can be easily computed using the static mapping without any knowledge of the load etc on those servers. Although YouTube's architecture shares some similarity (such as the use of location-aware DNS resolution) with Akamai and other CDNs, there are several parts of the architecture that make YouTube video delivery system different. For instance, Akamai [14] usually relies on very short TTL (generally 20 seconds) for DNS resolutions whereas YouTube uses much longer TTL of 5 minutes and uses HTTP redirections to achieve fine-grain load balancing. The use of static mapping between objects and hostnames and among multiple namespaces is also not common in other CDNs. We believe that these intelligent design choices play a role in enabling YouTube to flexibly meet user demands as well as performance expectations.

While Google's YouTube video delivery system represents an example of the "best practices" in the design of 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 strategies. In addition, the YouTube design is clearly confined and constrained by the existing Internet architecture. Understanding the pros and cons in the YouTube design also provides valuable insights into the future Internet architecture designs. For instance, the use of the DNS system for mapping the YouTube logical video servers to the physical cache locations is at best approximate, as it lacks the accurate user location information. Redirections are often necessary, prolonging the response time; request failure may also occur occasionally when the redirection upper bound is reached. All these point to many exciting research questions and important future directions that are worthwhile to be pursued. In light of the increasing popularity of large-scale video distribution – not only relatively short-duration, YouTube-like videos but also full-length, DVD-quality videos – and the likely dominance of video streaming/downloads in the Internet traffic, coupled with the emergence of cloud computing and services, we believe that successfully addressing these challenges are critical to both the design of large-scale content delivery systems as well as the development and evolution of the future Internet architecture.

8. REFERENCES

[1] V. K. Adhikari, S. Jain, Y. Chen, and Z.-L. Zhang. Reverse Engineering the YouTube Video Delivery Cloud. In *HotMD'11*.

[2] V. K. Adhikari, S. Jain, and Z.-L. Zhang. YouTube Traffic Dynamics and Its Interplay with a Tier-1 ISP: An ISP Perspective. In *IMC'10*.

[3] M. Cha et al. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *IMC'07*.

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

[5] L. Daigle. WHOIS Protocol Specification. RFC 3912, Internet Engineering Task Force, Sept. 2004.

[6] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. Youtube traffic characterization: a view from the edge. In *IMC '07*.

[7] Google. Google To Acquire YouTube for $1.65 Billion in Stock. `http://www.google.com/intl/en/press/pressrel/google_youtube.html`.

[8] C. Labovitz et al. Internet inter-domain traffic. In *SIGCOMM'10*.

[9] R. Miller. Google-YouTube: Bad News for Limelight? `http://www.datacenterknowledge.com/archives/2006/10/13/google-youtube-bad-news-for-limelight/`.

[10] V. N. Padmanabhan et al. An investigation of geographic mapping techniques for internet hosts. In *SIGCOMM'01*.

[11] A. Pathak et al. Measuring and evaluating tcp splitting for cloud services. In *PAM'10*.

[12] M. Saxena, U. Sharan, and S. Fahmy. Analyzing video services in web 2.0: a global perspective. In *NOSSDAV'08*.

[13] Squid. Squid: Optimising web delivery. `http://www.squid-cache.org`.

[14] A.-J. Su, D. R. Choffnes, A. Kuzmanovic, and F. E. Bustamante. Drafting behind akamai. *SIGCOMM'06*.

[15] R. Torres et al. Dissecting Video Server Selection Strategies in the YouTube CDN. In *ICDCS'11*.

[16] YouTube. YouTube statistics. `http://www.youtube.com/t/press_statistics`.

[17] M. Zink et al. Characteristics of youtube network traffic at a campus network - measurements, models, and implications. *Comput. Netw.*, 2009.