How Do You "Tube"? Reverse Engineering the YouTube Video Delivery Cloud

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

ABSTRACT

In this paper we set out to "reverse-engineer" the YouTube video delivery cloud by building a globally distributed active measurement infrastructure. Through careful and extensive data collection, analysis and experiments, we deduce the key design features underlying the YouTube video delivery cloud. The design of the YouTube video delivery cloud 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. By mapping the video id space to the logical servers via consistent hashing and cleverly leveraging DNS and HTTP re-direction mechanisms, such a design leads to a scalable, robust and flexible content distribution system.

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. According to YouTube's own account [4], millions of users across geographically disparate areas in the world access YouTube each day, resulting in more than two billion video downloads per day. The number of videos hosted by YouTube grows by the clock – 24 hours worth of new video content uploaded per minute [4] – a testimony to its universal popularity and outreach.

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 [1] and subsequently re-structured the YouTube video delivery infrastructure, it was known that YouTube employed several data centers in US (see [5]) as well as third-party content delivery networks [2, 13] 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 [10] 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 re-structured and incorporated the YouTube delivery system into its own vast Internet infrastructure in the past few years, little is known how Google leverages its incredible engineering talents and resources to re-design and re-structure the YouTube video delivery infrastructure – which we will refer to as the *YouTube video delivery cloud* – to meet the rapidly growing user demands as well as (fairly high) user performance expectations.

This paper attempts to "reverse-engineer" the YouTube video delivery cloud through large-scale active measurement, data collection and analysis. The rationale behind our *reverse-engineering* 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 cloud. 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? Moreover, on one hand we believe that Google's YouTube video delivery cloud offers an example of the "best practices" in the design of an Internet-scale content delivery infrastructure. On the other hand, the design of YouTube video delivery cloud also poses some interesting and important questions regarding alternative architectural designs, cache placement, content replication and load balancing strategies, and so forth. It is with these goals in mind that we set out to "reverse-engineer" the YouTube video delivery cloud 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 cloud and the use of separate web vs. Flash video servers to serve the standard HTML content vs. video content (see Section 3.1) pose several challenges in *actively* measuring, and collecting data from the YouTube video delivery cloud. To address these challenges, we have developed a novel distributed active measurement platform with more than 1000 vantage points spanning five continents. As described in Section 3.2, our globally distributed measurement platform consists of three key components: 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 at our back-end compute clusters; ii) open recursive DNS servers to provide additional vantages and 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. This distributed active measurement platform enables us to collect nearly half a million YouTube videos[1], extensive YouTube DNS name-to-IP mappings at each vantage point, and detailed video playback traces such as HTTP logs (see Section 3.3 for a detailed description of the collected datasets).

Through careful data analysis and inference – especially by analyzing the relations among YouTube video ids, DNS names, and IP addresses – and by conducting extensive "experiments" to test and understand the behavior of the YouTube video delivery cloud, we are not only able to geolocate a large portion of YouTube video server and cache locations, but also to uncover and deduce the logical designs of the YouTube video id space, the DNS namespace structures and cache hierarchy, and how they map to the physical infrastructure and locations. We provide a high-level summary of the key findings regarding the YouTube design below, and refer the reader to Section 4 for more specifics. In Section 5, Section 6 and Section 7 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.

• The design of the YouTube video delivery cloud consists of three components: the *video id space*, multiple (anycast) video server DNS namespaces (as well as a separate (unicast) video server namespace) reflecting a multi-tiered cache hierarchy design, and a physical video server cache hierarchy with (at least) 38 primary locations, 8 secondary locations and 5 tertiary locations.

• YouTube uses a fixed length "flat" video id space, from which each video is assigned a "unique" identifier. YouTube employs a form of consistent hashing to map each video to one of 192 *logical* primary video servers, each identified by a unique DNS name. Such a fixed *video-to-video-server* mapping (at the logical level) makes it easy for YouTube web servers to generate URLs referencing videos that users are interested in without having to worry about where each video is located or currently cached.

• YouTube utilizes multiple (*anycast*) DNS namespaces, each representing a collection of *logical* video servers with certain specific roles and the tier of the cache hierarchy at which they reside. The use of these (anycast) namespaces allows YouTube to perform (coarse-grain) locality-aware server selection to reduce latency, and flexibly map each *logical* video server to multiple physical servers (IP addresses) at the same location or across multiple locations for load-balancing. Furthermore, by defining a strict ordering of these namespaces, YouTube can also handle cache misses, or take into account the popularity of videos, by tracking the role and tier of each (physical) server, and re-directing video requests (via HTTP re-direction) from, say, a server at the primary cache location, to a server at the secondary or tertiary cache location.

In a nutshell, by introducing multiple (tiered) DNS namespaces and employing a fixed mapping of the video id space to the DNS namespaces, YouTube clearly 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. For instance, YouTube can deploy additional (physical) video servers at either an existing or new primary cache location to meet user demands, or even change its server selection or load-balancing strategies, by simply adding new DNS name-to-IP address mappings or altering such mappings. Via HTTP request re-directions, it can avoid busy video servers or handle cache misses. On the other hand, the re-directions may lead to longer delays (as up to 7 re-directions may occur) or even a request failure (albeit rarely, see Section 7). All in all, Google's YouTube video delivery cloud design offers good lessons in the design of Internet-scale content delivery systems, but also poses many interesting research questions that require further study. We will further expand these points in Section 8.

[1]We discarded the actual video content, kept only the video identifiers (ids), namely, the URLs referencing the videos, their view counts, and other relevant information (e.g., titles of videos) and logs.

2. RELATED WORK

Before we delve into our work of reverse-engineering the YouTube video delivery cloud, we first 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 [6] 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 [8] 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. Another study [13] analyzed network traces for YouTube traffic at a campus network to understand benefits of alternative content distribution strategies. A more recent work [12] studies the impact of the YouTube video recommendation on the popularity of videos.

Perhaps most relevant to our work is the recent study carried in [5], where 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 re-structuring*. Another piece of relevant work is the study in [9], where the authors compare two design philosophies used in content distribution networks (CDNs) such as Akamai and Limelight, and conduct a comparative performance evaluation (e.g., delay and availability performance) of these CDNs through measurement. To the best of our knowledge, our work is the first study that attempts to reverse engineer the current YouTube video delivery cloud.

3. 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 front-end web servers and (front-end) Flash video servers. We then provide an overview of our distributed *active* YouTube mea-

surement and data collection platform.

3.1 YouTube Video Delivery Basics

YouTube is the most popular video sharing website in the world, where a large portion of the videos are *user-generated* content. Users can upload videos to YouTube in a number of formats, using various types of devices such as laptops, desktops, smart phones. YouTube converts videos into the Flash video format. 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). Users can also watch YouTube videos by clicking on YouTube video URLs embedded in other websites, emails, etc (without first going to the YouTube website). In addition, YouTube also allows cellphone users to watch YouTube videos using various platform-specific apps.

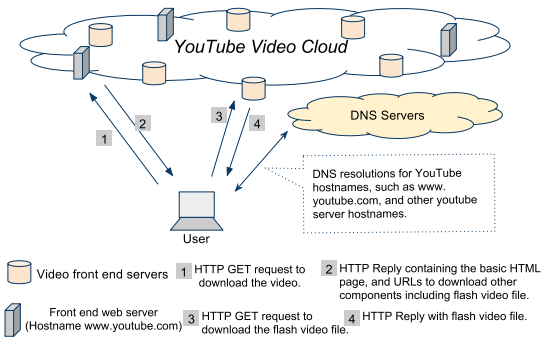

Figure 1: YouTube Typical Steps involved in YouTube Video Delivery.

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. As will be clear later, each YouTube video is identified by a URL. When a user goes to the YouTube website, or clicks on any URL of the form `http://www.youtube.com/watch?v=ABCDEFGHIJK` on an existing YouTube web page using her browser, the browser first resolves the hostname `www.youtube.com` using the local DNS server (LDNS). We refer to the YouTube servers (IP addresses) mapped to `www.youtube.com` as the *front-end web server*. The HTTP request from the user is then directed to one of the YouTube front-end 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 and related videos that the user is (or may be) interested in.

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 re-direct (via the HTTP request redirection) the video request to another video server, which may again re-direct 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 re-directions) 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 4).

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

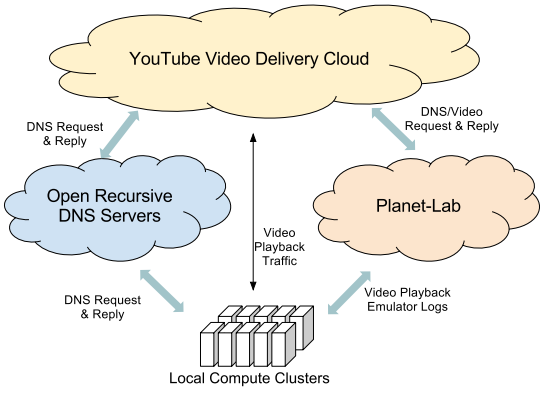

Figure 2: Our Active Measurement Platform: an Illustration.

[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 4.

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 (see Appendix A.4). Fig. 3 plots the geographical locations of the entities used in our active measurement and data collection platform: the nodes span the globe. 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 provided by and located at various ISPs and organizations (see Appendix A.2).

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 re-directions 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 [3], 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

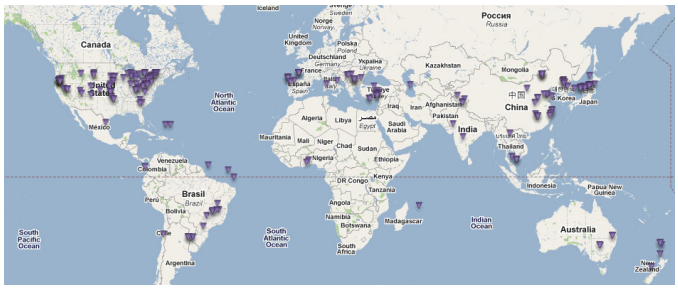

Figure 3: Geographical Distribution of PlanetLab Nodes and Open Recursive DNS Servers.

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 therefore 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 PlanetLab nodes to ensure the correctness of our platform (see Appendix A.5).

3.3 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 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 $1,314$ globally distributed vantage points, perform DNS resolutions from these vantage points, and record the entire playback processes including HTTP logs. This yields a col-

lection 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 and latency measurements from the PlanetLab nodes to the IP addresses, and so forth. Furthermore, we also extract the HTTP request re-direction sequences, analyze and model these sequences to understand YouTube re-direction 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 re-direction 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 re-direction decision process. (See Sections 6 and 7 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. Based on the current trend at each geographical area (e.g., a country), YouTube provides a (possibly different) listing of popular videos for that location, which are classified into several classes such as "Most Popular", "Most Viewed", "Trending", "Featured Videos", etc. We parsed the YouTube homepage HTML files 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). Combining the video lists from vantage points and all seeds, we compiled a final list of $434K$ unique videos, which represent videos popular at different geographical locations and their related videos. In addition to the URLs (or rather, the YouTube video identifiers extracted thereof) referencing the videos, we also recorded the (most recent) *view-count* listed in the web-page of each video and other relevant information. We created a few of our own videos and uploaded them to the YouTube website.

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 $1,314$ 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 re-direction 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 6). As mentioned earlier, 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.

Judicious use of resources. We took several steps to limit the load due to our experiments on the PlanetLab nodes and on the YouTube servers. We divided entire video list in chunks of 10000 videos each and played each chunk from multiple PlanetLab nodes. Our emulators running on PlanetLab nodes to "play" YouTube videos were configured to download not more than the first 1MB of a video. Additionally, the emulators were asked to sleep for 10 seconds between consecutive video playbacks.

(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 re-direction 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 4). 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.

4. SUMMARY OF KEY FINDINGS

In this section we provide a summary of our key findings

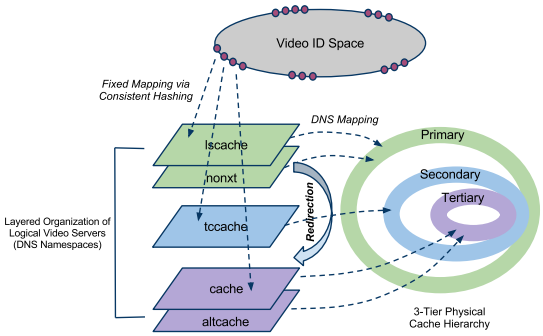

Figure 4: YouTube Architectural Design.

Figure 5: Geographical distribution of YouTube Video Cache Locations.

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.

4.1 Overall Architecture

As schematically shown in Fig. 4, the design of the YouTube video delivery cloud consists of three major components: the *video id space*, the *multi-layered* organization of *logical* video servers via multiple *anycast* DNS namespaces, and a 3-tier *physical* server cache hierarchy with (at least) 38 *primary* locations, 8 *secondary* locations and 5 *tertiary* locations. Here by a *anycast* (DNS) namespace we mean that each DNS name is *by design*, mapped to multiple IP addresses ("physical" video servers).

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 5 for details). The total size of the YouTube video id space is effectively 64^{11}.

Multiple (Anycast) DNS Namespaces and Layered *Logical* Video Server Organization. YouTube defines multiple (*anycast*) DNS namespaces, each representing a collection of *logical* video servers with certain roles. Together, these (anycast) DNS namespaces form a *layered* organization of *logical* video servers. Logical video servers at each layer of this organization 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.

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. These *logical* layered DNS namespaces play an important role in dynamic HTTP request direction mechanism employed by YouTube. Last but not the least, we note that in general only the DNS names belonging to the *lscache namespace* are *visible* in the URLs or HTML pages referencing videos; whereas DNS names belonging to the other four namespaces occur mostly only in the URLs used in dynamic HTTP request re-directions during video playback.

Three-Tier (Physical) Server Cache Hierarchy and Their Locations. Using the YouTube IP addresses seen in our datasets, we are able to geo-map the YouTube "physical" video server cache locations, which are dispersed at five continents (see Fig. 5). From this analysis, we deduce that YouTube employs a 3-tier physical cache hierarchy with (at least) 38 *primary* locations, 8 *secondary* locations and 5 *tertiary* locations. Each location contains varying number of IP addresses ("physical" video servers), and there are some overlapping between the primary and secondary locations (e.g., at the Washington D.C. metro areas), where one "physical" video server may serve either as a "primary" or a "secondary" video server. Columns 4-6 show the total number of IPs, prefixes, and locations each DNS namespace is mapped[3]. In Section 6.3 we will provide some details regarding how we geo-map the YouTube physical cache locations.

Unicast Namespace. In addition, for each IP address (corresponding to a "physical" video server), YouTube also defines a *unicast* DNS name. Namely, there is a one-to-one between this DNS name and the IP address. We consider these unicast DNS names collectively as forming a *unicast* DNS namespace. As shown in Table 1, the unicast names have two formats, which we refer to as *rhost* and *rhostisp* formats. These unicast DNS names also play an important role in dynamic HTTP request re-directions (see below and Section 7).

4.2 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 form of consistent hashing, 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 cloud via *flexible*

[3]We note that because DNS names belonging to the *nonxt* namespace only show up in the URLs during dynamic HTTP request re-direction, we see fewer IP addresses, prefixes and locations than those mapped to the *lscache* namespace. This is likely due to the incompleteness of our datasets, as re-directions in general do not occur very frequently.

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

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 form of "consistent" hashing 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 DNS names of the *lscache* namespace and those of the *nonxt* namespace as well as those of the *tccache* namespace. The same also holds for the mapping between the *nonxt* and *tccache* namespace. There is also a fixed and consistent mapping between the *lscache* namespace and *cache* namespace, where 3 *lscache* DNS names are mapped to one *cache* DNS name. The same also holds for the mapping between the *lscache* namespace and *altcache* namespace, and the mappings between the *nonxt*, *tccache* namespaces and *cache* and *altcache* namespaces. 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.

DNS Resolution and (Coarse-grain) Locality-Aware Server Selection. The mapping between *logical* video servers (i.e., DNS names) and *physical* video servers (i.e., IP addresses) are done via DNS resolution. The mapping between DNS names to IP addresses are in general *many-to-many*: each (anycast) DNS names are generally mapped to multiple IP addresses; and multiple DNS names may be mapped to the same IP address. YouTube employs a (coarse-grain) *locality-aware* server selection strategy: depending on where the user request is originated (or rather the DNS request is originated), YouTube picks a primary video cache location that is (reasonably) "close" to the user, and resolves the requested *lscache* DNS name "randomly" to one of the IP addresses within that location. This server selection strategy is fairly *coarse-grained*, as the geographical area (where user requests come from) based on which this decision is made appears to be large, and the "granularity" of the DNS resolutions varies from geographical areas to areas: for instance, based on our measurement data, the number of IP addresses mapped to one DNS name belonging to the *lscache* namespace can vary from 9 to 626 IP addresses per PlanetLab node (see Section 6.4).

***Dynamic* HTTP Request Re-direction.** To perform finer-grain and dynamic load-balancing, or to handle cache misses, YouTube employs HTTP request re-direction mechanism (performed only by video servers as part of the Flash video object operations). Such a re-direction mechanism is especially useful and important, as YouTube always maps the user video request to a "physical" video server at a *primary* cache location (via DNS resolution of a *lscache* DNS name to an IP address). Since the size difference of the primary locations (in terms of the number of "physical" video servers or IP addresses) can be quite large, and the user demand is also likely to vary from one geographical area 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 can either directly fetch a "cold" video from another video server at a secondary or tertiary location (e.g., via the Google internal backbone network), or re-direct the request directly to another video server at a secondary or tertiary location.

YouTube cleverly utilizes the multiple (both anycast and unicast) DNS namespaces to perform dynamic load-balancing as well as to keep track of the re-direction process. There is a *strict ordering* among the anycast namespaces: referring to Fig. 12, only re-direction from a "higher" layer namespace to a "lower" layer namespace is allowed, e.g., from *lscache* to *nonxt*, but not the other way round; a re-direction can "jump" across multiple layers, e.g., from *lscache* to *tccache* or *cache* or *altcache*. In addition, re-directions from an *anycast* namespace to the *unicast* namespace and vice versa are also allowed, and can happen multiple times. However, when going from an *anycast* namespace to the *unicast* namespace and then back to another *anycast* namespace, the above ordering among the *anycast* namespaces must be observed. YouTube uses both a re-direction count and a tag to keep track of the re-direction sequences. Up to 9 re-directions may happen, although they are rarely observed in the video playback traces we collected.

5. VIDEO ID SPACE

YouTube references each video using a "unique" URL string (we refer to it as *video id*) of 11 *literals*. As an example, consider a typical YouTube URL when watching a video:

`http://www.youtube.com/watch?v=tObjCw_WgKs`

In the above URL, the 11-literal string after `v=` is the video id, namely, `tObjCw_WgKs`. 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: the symbol 4 which appears in the *video id*'s in our list, but not the three symbols after it, namely, $5, 6, 7$; and so forth.

Further investigation and experiments reveal that while only 16 symbols appear to be used in the last position, the other 48 symbols are in fact also used for *identifying* videos, but appear to be *reserved* for some *internal* uses. In other words, these 48 symbols are generally not "visible" in *video id*'s appearing in the URLs or HTML pages presented to users. More specifically, we find that each YouTube video has *four* identifiers: for each of the 16 "visible" symbols in the last position, replacing it with one of the three "invisible" symbols following it, the resulting 11-literal string refers exactly to the same video. Consider the URL example above. Replacing the letter `s` in the last position of the *video id* with either `t`, `u` or `v` and then typing the new URL in a web browser, we would get the same video. Hence for *video id*, there are three *duplicate video id*'s. We speculate that these duplicate *video id*'s are used internally by YouTube to refer specifically to three additional replicas of each video.

We refer to the collection of all *video id*'s as the *video id space*. Based on the above discussion, 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 6. 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 too. Due to space limitation, we do not present the detailed results here.

Lastly, we study the distribution of popularity of videos in our list. Since there is no direct measure to quantify the popularity of a video, we rely on the number of times a video has been viewed, i.e., the *view-count*, as a metric to reflect its popularity. We plot the distribution of popularity as the view-count for the videos in Figure 7. As seen in this plot, the view-counts for these videos differ widely: some videos have more than 2 million views, whereas around $8,000$ videos have a view-count fewer than 100. Clearly, the popularity distribution shown in Figure 7 does not represent the true distribution of video popularity for all the YouTube videos, in particular in light that our crawling methodology introduces a bias towards more popular videos. Nevertheless, our collected list of videos represents a fairly good sample for videos with varying popularity, as shown in Figure 7.

6. CACHE NAMESPACES & HIERARCHY

In this section we describe the structure and organization of YouTube video caches. First, we describe several DNS namespaces used to refer to these cache servers, and how the mapping between *video id* namespace and the namespaces used by video cache servers is done. Finally, we describe how YouTube uses DNS infrastructure to direct the users to an IP address located in the vicinity of their location.

In order to identify the YouTube video cache architecture, we analyze the logs collected during our active measurement. These logs show that YouTube uses several different DNS namespaces to refer to video cache servers. As shown in Table 1 there are a total of 7 such unique DNS namespaces used in the video delivery, which are further classified as *anycast* and *unicast* namespaces.

6.1 Anycast DNS Namespace

Table 1 summarizes the 5 *anycast* namespaces used by YouTube to refer to video cache servers. Our detailed analysis of video playback logs show that these *anycast* namespaces can be divided into following three categories:

• **Primary Video Caches.** These are the hostnames embedded in the initial HTML file provided by the YouTube front-end web server to user, when a user accesses a video page. Using our analysis of initial HTML files for all the $434K$ collected from several vantage points, we found that there are a total of 192 such hostnames embedded in the main HTML file which refer to the servers hosting the Flash video objects. In addition, we see that all of these hostnames can be represented using the following regular expression: v[1-24].lscache[1-8].c.youtube.com. We also found that each *video id* maps to a unique *lscache* hostname out of 192 such names. E.g., a video identified using the *video id* `MQCNuv2QxQY` always maps to v23.lscache1.c.youtube.com *lscache* name from all the 1,314 vantage points at all times.

Since all the *video id*s are uniformly distributed in the flat video identifier space (see Sec 5), the number of *video id*s that map to each *lscache* hostname are also equally distributed. To demonstrate this we consider all the $434K$ *video id*s and plot the number of *video id*s that map to each of the *lscache* hostnames in Figure 8. As seen in this figure, there are approximately equal number of videos mapped to each of the *lscache* hostnames.

Furthermore, we found that there are $4,999$ IP addresses that map to *lscache* hostnames from all the measurement locations. Interestingly, there is another *anycast* namespace, constituted by another unique 192 DNS hostnames, which also maps to the same set of IP addresses as *lscache* hostnames. The hostnames in this space can be summarized using the regular expression: v[1-24].nonxt[1-8].c.youtube.com. As seen in Figure 4 there is one-to-one mapping between these two namespaces in terms of the DNS hostnames. E.g., the *anycast* hostname referred to by v1.lscache1. c.youtube.com maps to v1.nonxt1.c.youtube.com hostname, and therefore, all the *video id*s that map to v1.lscache1.c.youtube.com also map to v1.nonxt1.c.youtube.com and vice versa.

• **Secondary Video Caches.** Similar to primary video caches, YouTube also deploys a secondary video cache for better availability and reliability. Again, it uses *anycast* DNS namespace to identify hosts in this set, which is of the following form: tc.v[1-24].cache[1-8].c.youtube.com. In addition, this namespace maps to a relatively smaller set of IP addresses in total, and as we will explain later, these video caches are located at only 8 locations.

• **Tertiary Video Caches.** There is another layer of video cache that YouTube uses to serve the videos, which we refer to as *Tertiary Video Caches*. In our analysis of video playback logs we found that these are used as the final set of

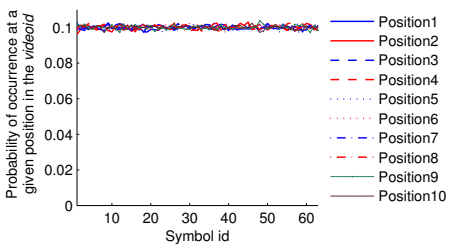

Figure 6: Distribution of *video id*'s with respect to first symbol

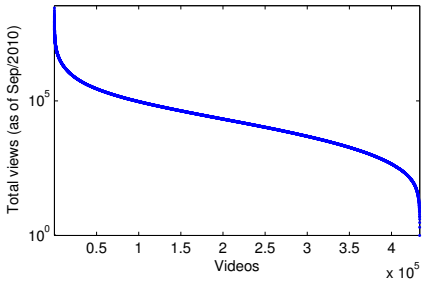

Figure 7: View counts for $434K$ videos

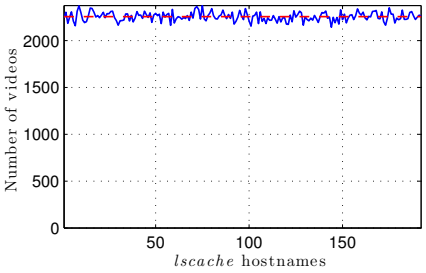

Figure 8: Number of videos mapped to each *lscache* hostname.

video caches. Unlike primary and secondary video caches, which are identified using 192 unique DNS hostnames, these tertiary video caches have a namespace constituted by only 64 DNS hostnames, which are represented using regular expression: v[1-8].cache[1-8].c.youtube.com. YouTube also uses an additional namespaces for these hosts which can be represented as: alt1.v[1-8].cache[1-8].c.youtube.com. This additional namespace is used for achieving dynamic load balancing using HTTP redirections (See Sec 7 for details).

There is consistent one-to-one mapping between the hostnames in primary and secondary video cache namespace. Since there are only 64 hostnames in the tertiary video cache namespace, hence there is a three-to-one mapping between the hostnames in primary/secondary video cache namespaces and tertiary video cache namespaces.

As mentioned earlier, our analysis showed that there is a consistent mapping between a *video id* and the corresponding *anycast* hostname. It reveals that YouTube uses a hashing based mechanism to consistently map a *video id* to a fixed set of *anycast* hostnames. In order to verify this, we requested *anycast* hostnames which were not responsible to serve the given set of videos repeatedly. In all such attempts, these *anycast* hostname redirected our request to the correct hostname corresponding to the *video id*.

6.2 Unicast DNS Namespace

In addition to several *anycast* namespaces used by YouTube to refer to video cache servers, YouTube also uses a *unicast* namespace to identify individual video servers. Since, these *unicast* DNS hostnames map to a unique IP address irrespective of the user location, it helps in redirecting the user to a specific server during the dynamic load balancing process. The DNS hostnames that constitute *unicast* DNS namespace are one 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 PoP co-located with YouTube caches, and *city* is a 3 letter code representing the city name, which usually refers to the nearest airport.

While examining the video playback logs we found that there is a key difference between the *unicast* hostnames and the *anycast* in terms of their role in the video playback. Our analysis revealed that these *unicast* hostnames do not have a fixed *video id* to hostname mapping. In fact, any of these hosts can serve any of the videos. Therefore, they act as relay or reverse proxy servers for YouTube videos. In order to further verify our finding, we requested these *unicast* hostnames to play any randomly selected video from our list of $434K$ and found that most of the time these hosts actually served all the videos.

Using the video playback logs, we extracted a total of $5,446$ unique *unicast* hostnames. Based upon the embedded three letter city code and the optional ISP short name, we found that these hosts are distributed in 47 unique geographical locations. In addition, we found the IP addresses corresponding to this *unicast* DNS hostnames have almost complete overlap with the IP addresses seen during the resolution for the *anycast* namespaces from all the vantage points. Our analysis of these overlapping IP address space revealed another interesting finding, that the *unicast* hostnames seen for different *anycast* namespace IP addresses have slightly different structures. E.g. all the IP addresses appeared during the resolution of tertiary video cache namespaces, had a corresponding *unicast* hostname represented using: r[1-24].*city*[01-16]**t**[0-16].c.youtube.com. While the *unicast* hostnames overlapping with primary and secondary video cache server IP addresses were of the form, r[1-24].*city*[01-16]**s**[0-16].c.youtube.com or r[1-24].*isp-city*[1-3].c.youtube.com, and r[1-24].*city*[01-16]**g**[0-16].c.youtube.com respectively.

6.3 Geo-mapping YouTube Cache Locations

We resolved both *unicast* and *anycast* DNS hostnames using several vantage points, and extracted a total of $5,883$ unique IP addresses. We then geolocate each of these IP addresses using our hybrid geolocation methodology. Due to space limitation, we only describe the key ideas behind our geolocation framework here.

In order to geolocate YouTube IP addresses, first of all, we leverage the large number *unicast* hostnames extracted using the video playback logs. As described earlier, each of these hostnames have 3-letter city codes embedded, which represent the nearest airport code for the corresponding YouTube location. We further verified using round trip delay measurements that these embedded city codes are indeed correct. To perform this analysis, we chose the PlanetLab nodes which are located near the cities corresponding to each airport code seen. Using these nodes near each airport location, we performed round trip measurements to the corresponding YouTube IPs, and found that measured delay is fairly small, usually in the order of 1-2ms.

In the second step, we used the round trip delay logs for all the YouTube IP addresses collected using 471 PlanetLab nodes. We use a basic idea similar to the approach used by GeoPing [11]. In this approach, 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 was already known using *unicast* hostnames. Also, in several cases we found that there were multiple such IP addresses in the clusters, for which the location was already known. In all such instances the location for these multiple IP addresses were always the same. In the end, we had only three clusters in which we had no IP addresses with the corresponding *unicast* name. Based on the nearest PlanetLab nodes for these clusters in terms of the round trip delay, we classified them into a coarser level geographical location (see Appendix A.1 for details).

Finally, based on the above methodology we geolocate all the 5,883 YouTube IP addresses extracted using DNS resolutions of all the YouTube hostnames. We extracted 47 locations where YouTube has its various caches. Furthermore, we found that primary caches are distributed in 38 locations, secondary caches in 8 and tertiary caches in 5 locations with some locations hosting overlapping cache hierarchy. We plot these extracted YouTube locations on a world map in Figure 5. We summarize the distribution of various YouTube hostnames in terms of the geographical location in Appendix A.1.

6.4 Locality Aware DNS Resolution

YouTube uses DNS-based location awareness to direct users to a nearby cache locations. As an example, in Table 2, we show DNS resolutions for v1.lscache1.c.youtube.com hostname performed from 5 different PlanetLab nodes. As seen in this table, based upon the location of the PlanetLab node, these hostnames mapped to an IP address at a nearby YouTube location.

In order to verify if DNS based resolutions for lscache hostnames are location aware at all the locations, we conducted following experiment. For each PlanetLab node, we order all 47 YouTube locations in the increasing order of round trip network delay and assign each YouTube location a rank in this order. Next, we consider *lscache* hostname to IP addresses mapping for each of the PlanetLab nodes, and see how they are distributed with respect to the rank of the corresponding YouTube location for the given PlanetLab node. E.g., if DNS resolutions of *lscache* hostnames results in to a set of 192 unique IP addresses at a given PlanetLab node, then we see how many of these are located at rank 1 YouTube location for the PlanetLab node, and so on. In Figure 9 we plot the number of PlanetLab nodes which had 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 with respect to network delay. While, only a very small number of nodes have their one of the *lscache* hostnames mapped to farther locations.

7. HTTP REDIRECTIONS

YouTube uses HTTP based redirection to achieve dynamic load-balancing to handle cache misses. If the *lscache* server responsible for a video cannot serve the requested video due to some reason such as high load, it sends a HTTP 302 response back to the client. The header tells the client to go to another hostname and download the video from there. However, if the host corresponding to the HTTP redirect URL in the HTTP 302 response can not provide the video due to some reason, it sends another HTTP 302 response to the client to ask it to try yet another hostname, otherwise, it sends the video to the client. This redirection mechanism allows YouTube to perform dynamic load-sharing among its geographically distributed physical resources. In the following, we discuss the specific mechanisms used in these HTTP based redirections, their correlation with video popularity and the performance implications.

7.1 Redirection Sequence Analysis

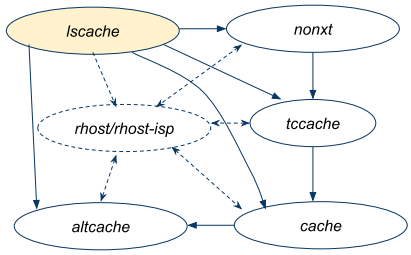

Figure 12: Redirection hierarchy.

To better understand the patterns in HTTP redirections, we carefully examined the video playback logs, where HTTP 302 code was seen during the playback of the video. Our analysis of these logs reveals several interesting patterns in these redirections and the mechanisms used to avoid HTTP redirection loops. These redirection loops may occur when a set of YouTube hosts redirect the user using HTTP 302 code among each other in a circular fashion.

The first key finding here is that HTTP redirections for any given video follow a specific namespace hierarchy. A *lscache* video cache server in one of the 38 primary cache locations may re-direct a video request to a corresponding *nonxt* video cache server within the same primary cache location or to another (typically "close-by") primary cache location; it may also re-direct it directly to a corresponding *tccache* video cache server in one of the 8 secondary video cache locations, or directly to a corresponding *cache* video cache server in one of the 5 tertiary video cache locations. Similarly, *nonxt* video cache server in one of the 38 primary cache locations may re-direct a video request to a corresponding *tccache* video cache server in one of the 8 secondary video cache locations, or to a corresponding *cache* video cache server in one of the 5 tertiary video cache locations. It never re-directs a video request to a *lscache* video cache server, regardless of video and location. Likewise, a *tccache* video cache server in one of the 8 secondary cache locations may only re-direct a video request to a corresponding *cache* video cache server in one of the 5 tertiary video cache locations. Lastly, a *cache* video cache server in one of the 5 tertiary cache locations may only re-direct a video request to a corresponding *altcache* video cache server in *another* tertiary video cache location (usually Europe to US and US to Europe). Fig. 12 shows the direction in which HTTP redirections can be sent from one hostname to another.

In addition, we found that these HTTP redirections also involve several *unicast* hostnames (such as *rhost* and *rhostisp*) as well. However, the redirection to one of these hostnames

Table 2: Different IP address resolutions for v1.lscache1.c.youtube.com

PlanetLab node location	Resolved IP	Resolved IP location
Taipei, Taiwan (adam.ee.ntu.edu.tw)	202.169.174.208	Taipei
Namur, Belgium (chimay.infonet.fundp.ac.be)	74.125.10.144	Amsterdam
Surrey, British Columbia, Canada (cs-planetlab4.cs.surrey.sfu.ca)	74.125.107.16	Seattle
Konstanz, Germany (dannan.disy.inf.uni-konstanz.de)	173.194.18.70	Frankfurt
Haifa, Israel (ds-pl3.technion.ac.il)	173.194.18.6	Frankfurt

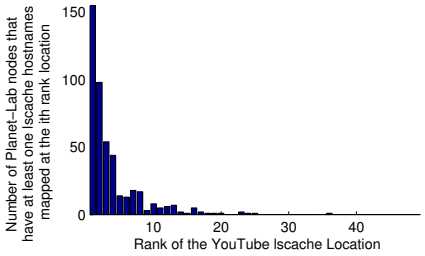

Figure 9: Distribution of PlanetLab nodes with respect to the delay based rank of the YouTube location.

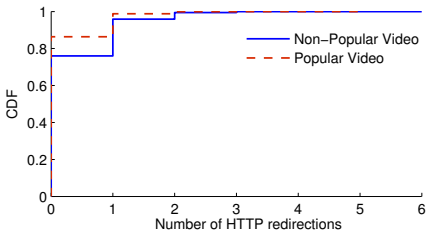

Figure 10: Comparison of number of redirections for popular and non-popular videos.

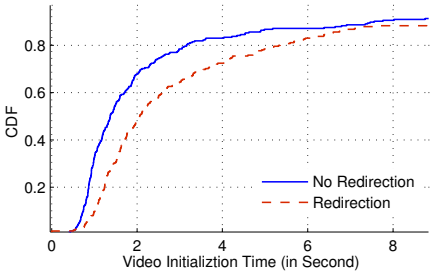

Figure 11: Comparison of video initialization time.

only occurs from one of the *anycast* hostnames only. In the event when an *anycast* hostname redirects the user to one of the *unicast* hostnames, then it also updates the redirection URL with a 'tag' corresponding to the tier of the forwarding *anycast* hostname. E.g., whenever an *lscache* host forwards the video request to *rhost* host, it adds a unique tag `&st=lc` in the redirection URL, which represents that request was forwarded from one of the host in *lscache* hostnames. Similarly, when *tccache* hosts forward the request to *unicast* hosts then they always append the tag `&st=tcts` in the redirection URL, and *cache* appends the tag `&st=ts` before forwarding the request to a *unicast* host. This mechanism ensures that the HTTP redirection always follow the hierarchy shown in Figure 12, where maximum number of times a user can be redirected is limited to 4 if we exclude the *unicast* hostnames on the path, and 9 if we include the *unicast* hostnames as well.

As an ultimate protection against the possible redirection loops, redirection URL during the HTTP redirections also contain a "redirection counter", which is incremented by one every time a video request is forwarded to another host[4]. This redirection counter appears in the redirection URL as "`?redirect_counter=n`", here n represents the current value of the redirection counter, which is in the range [1-4]. Since there are maximum of 4 redirections possible among the *anycast* hosts based on the hierarchy shown in Figure 12, the maximum value for redirection counter is also 4. In case, the redirection counter reaches 4 and the corresponding *anycast* YouTube host can not serve the video, then it either sends an HTTP response with 503 HTTP error code, or forwards the user to one of the *unicast* host which might serve the video. However, upon failure to serve the video a *unicast* hostname sends 503 HTTP error code to the user. To confirm that maximum value for the redirection counter is 4, we modified the redirection URLs to have a redirection counter greater than 4, and asked several hosts to serve the video. In all such cases, we found that either the host served the video, or sent the 503 HTTP error code. During our experiments using modified URLs with redirection counter greater than one, we never encountered a case when the hosts redirected the user to another host.

[4]The redirect counter is incremented by one only when a *anycast* hostname forwards the request to another host, however, *unicast* hostnames do not increment this counter

7.2 Redirection Probability

YouTube hosts millions of videos using its YouTube video delivery cloud. Clearly, it is not possible to store all the videos at every location. Similarly, the caching policies used by YouTube may affect the availability of a video at a given location based upon its popularity in the corresponding geographic region. In the following, we consider the HTTP redirection probability for *hot* and *cold* videos, and analyze how the popularity of the videos affect the redirection probability.

For these analysis, we select *hot* videos using top 1200 videos based on the number of views for them from our initial list of $434K$. Similarly 1200 videos with smallest view counts in the list are considered as *cold* videos. Next, we extracted the video playback logs for these *hot* and *cold* videos and analyze the probability of HTTP redirections for both.

In Figure 10, we show the distribution of number of HTTP redirections for each of the *hot* and *cold* videos as a CDF plot. In this figure, x-axis represents the number of redirections for the video, and y-axis shows the cumulative probability. As seen in this figure *hot* videos see much smaller number of redirections compared to *cold* videos. In particular, 87% times *hot* videos are directly served without any HTTP redirections whereas only 78% times *cold* videos were directly served from the *lscaches*.

On the other hand, we do not see any conclusive evidence that suggests that the size of the cache locations or their geographic location correlate with the number of times cache servers in that location redirect a user request.

7.3 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 11 shows the CDF plot for the average video initialization time at each PlanetLab node for the video playback requests which involved HTTP redirections (Redirection) and the requests which were directly served by the *lscache* hosts (No Redirection). As seen in this figure, the video initialization time is significantly higher in case of HTTP redirections. In particular, only 35% video playback requests with no redirections have a video initialization time greater than 2 second, while in case of redirections more than 50% instances of video playbacks result in a video initialization time of more than 2 second.

8. CONCLUSIONS AND FUTURE WORK

In this paper we set out to reverse-engineer the YouTube video delivery cloud 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 cloud. In particular, we reveal that YouTube define multiple layers of DNS namespaces to present a *logical* video server cache hierarchy, with each layer having a fixed number of logical video servers and representing servers of certain roles. This design plays a critical role that has led to the success of the YouTube video delivery cloud in flexibly meeting user demands as well as performance expectations. All in all, we are confident that we have uncovered the major design features of the YouTube video delivery cloud. Nonetheless, there are still specific questions, such as the precise YouTube re-direction decision logic and process, that still require in-depth analysis and additional experiments.

While Google's YouTube video delivery cloud represents an example of the "best practices" in the design of such Planet-scale systems, its design also poses several interesting and important questions regarding alternative architectural designs, cache placement, content replication and load balancing strategies, especially in terms of user perceived performance. In addition, the YouTube video delivery cloud design is clearly confined and constrained by the existing Internet architecture. Understanding the pros and cons in the YouTube video delivery cloud 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, nor the precise status of candidate physical servers. Re-directions may be necessary, prolonging the response time. For instance, up to 9 re-directions may occur (albeit rarely); request failure may also occur when the re-direction 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.

9. REFERENCES

[1] Google to acquire youtube for $1.65 billion in stock. `http://www.google.com/intl/en/press/pressrel/google_youtube.html`.

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

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

[4] Youtube fact sheet. `http://www.youtube.com/t/fact_sheet` [Online; accessed 26-August-2010].

[5] V. K. Adhikari, S. Jain, and Z. Zhang. YouTube Traffic Dynamics and Its Interplay with a Tier-1 ISP: An ISP Perspective. In *IMC '10*. ACM, 2010.

[6] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *IMC '07*. ACM, 2007.

[7] DNSServerList. A free open recursive dns server list. `http://www.dnsserverlist.org`.

[8] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. Youtube traffic characterization: a view from the edge. In *IMC '07*. ACM, 2007.

[9] C. Huang, A. Wang, J. Li, and K. Ross. Measuring and evaluating large-scale CDNs (withdrawn). In *Proc. of IMC*, 2008.

[10] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet inter-domain traffic. In *SIGCOMM '10*. ACM, 2010.

[11] V. N. Padmanabhan and L. Subramanian. An investigation of geographic mapping techniques for internet hosts. In *SIGCOMM '01*, 2001.

[12] R. Zhou, S. Khemmarat, and L. Gao. The Impact of YouTube Recommendation System on Video Views. In *IMC '10*. ACM, 2010.

[13] M. Zink, K. Suh, Y. Gu, and J. Kurose. Characteristics of youtube network traffic at a campus network - measurements, models, and implications. *Comput. Netw.*, 2009.

APPENDIX

A. MINOR COMMENTS

In this appendix we describe several results, findings and additional details that we could not include the main section due to space limitation.

A.1 YouTube Cache Locations

The geographical locations for YouTube caches we described our experiments are shown in Table 3. In this table, we present the 3-letter code representing the city for each of the location. These are the same codes that YouTube uses in the *unicast* hostnames. In most cases, the codes represent the nearest airport from that city. We also present a list of YouTube cache locations which were co-located with ISP PoPs in this table. In addtion, the locations for which we could not map the IP addresses to a city are indicated with an asterisk(*), and represent coarser level geographical location.

Table 3: List of geographical locations for Youtube caches.

Cache tier	Locations
Primary (isp)	aapt-syd, alestra-mty, ascc-tpe bellcanada-yyz, comcast-lga, grnet-ath jordantelecom-amm, plix-waw, rogers-yyz seabone-pmo, singtel, snap-chc tata-mrs, telstraclear, tpnet-waw
Primary	ams, arn, atl, ber, bom, bru, bud, cbf dfw, eze, fra, gru, ham, hkg, iad, kul lax, lga, lhr, maa, mad, mia, mil, mrs nrt, nuq, ord, par, per, prg, sea, sin sjc, svo, syd, tpe, yyz Canada*, Mexico*, Europe*
Secondary	ams, fra, iad, lga, lhr, mia, par, sjc
Tertiary	fra, bru, iad, cbf, nuq

A.2 Open recursive DNS server

For our study, we have collected a list of 1,348 open recursive DNS servers from various online sources. A detailed list of some of these DNS servers and their country-wise locations is available at [7]. Before we use them in our measurement platform, we first verify the correctness of these open recursive DNS servers by resolving a number of hostnames on our campus network. During this verification process, we found 12 of these servers to be malfunctioning, e.g., they returned incorrect IP addresses for these hostnames, 41 were too slow in responding (e.g., taking more than 5 seconds to resolve each hostname), and 452 did not respond to DNS queries at all. From the initial list of 1,348 open recursive DNS servers, we winnowed down to a list of 843 healthy DNS servers.

A.3 Coverage of YouTube IPs and locations

In this section we argue that although we might not have covered every single IP address or location used by YouTube, we have definately covered a significantly large portions of them.

We believe that we have covered *all* of the secondary and tertiary cache locations and their IP addresses. This is because we could have obtained all of IP addresses and locations for those caches from less than 50 vantage points. Deploying additional 1000+ vantage points did not uncover any new secondary or tertiary hostnames or IP addresses.

We have uncovered a significantly large number of YouTube *rhost* (and *rhostisp*). As we keep on adding new vantage points, the number of newly seen such hostnames becomes very small. This suggests that there are not many hostnames that we could have uncovered by deploying additional vantage points. Since these hostnames have unique mappings for IP addresses, we also uncovered equally large number of IP addresses. In addtion, YouTube is officially present in 24 countries, and in our result we have found at least one cache location for each of these countries except for South Africa. However, it can be easily discovered by addding vantage points around or in South Africa.

Additionally, we also made use of the fact that the /24 prefixes that were discovered by resolving the YouTube hostname were mostly consecutive prefixes. For the prefixes that were "close-by" the seen prefixes, we verified that they were not part of the YouTube video delivery framework by trying to connect at port 80 on those IP addresses and requesting the "/" document. For instance, we see all /24 prefixes between 74.125.0.0/24 to 74.125.16.0/24 except 74.125.1.0/24, 74.125.5.0/24 and 74.125.13.0/24 and we verified that those three were indeed not being used for YouTube video delivery.

Table 4: Comparison of failures in video delivery for popular and non-popular videos.

HTTP error code	Popular	Non-popular
500 Internal Server Error	0.0002%	0.0038%
502 Bad Gateway	0.0102%	0.0256%
503 Service Unavailable	0.0019%	0.0048%
Total	0.0123%	0.0342%

A.4 Compute Clusters

Here, we provide a brief description of two local compute clusters that we used a controller for our active measurement testbed. Our first compute clusters consists of 24 Sun Fire X2200 M2 x64 nodes. Each node had 2 AMD quad core processors, a total of 16GB RAM and 500GB of storage space. On the other second compute cluster consist of 24 PowerEdge 6950 compute nodes. Each node in this cluster had four dual-core CPUs with 16GB RAM and 146 hard disk. We also had 6 RAID storages attached to these clusters. Each of these RAID has a storage capacity of 4 TB.

A.5 Data cleaning

During our verification process we found that the three PlanetLab nodes located in China return obviously incorrect IP addresses for YouTube DNS names. Apparently, to block access to YouTube the local or regional DNS servers within China simply return "random" IP addresses when asked to resolve YouTube DNS names! We therefore excluded these "erroneous" (or other malfunctioning) PlanetLab nodes from use in our platform.

A.6 Video Playback Failures

While video delivery network delivered most of the videos without any failures, during rare instances it failed to serve the videos. We describe the distribution of failures for popular (hot) and non-popular videos in Table 4. As seen in this table, though instances of failures are failure small, still considering YouTube delivers around 2 billion videos per

day, these number may reflect around couple of thousand failures in serving the videos. However, those problems appeared temporary. When we initiate the request again for the same video immediately after the failure, videos were served without any failures.

A.7 Distribution of Copyrighted Videos

YouTube uses a token based authentication approach for ensuring compliance with copyright laws. In this approach, in addition to providing the basic html content, the front end web servers also generate a token based signature for each video request. This tokens embeds the user IP address, which is used to determine the location of the user, which can be presented to any of the video server to download the videos. An interesting aspect of this set up is that any of the video servers or proxy relays can serve the video if video request also includes a valid token based signarture generated by the front end server.

For example copyright laws in US prohibited the distribution of the certain videos on YouTube for US based hosts, however, these videos were successfully played by PlanetLab nodes in Asia. This means if a user, identified using an IP address located in US goes to a video server located in Chicago (US) to play the video then video server verifies if the token generated for the corresponding user is valid to download the video file, otherwise, it denies the request. However, the same video server will serve the video file if requested by another user located outside of US and presented with a valid token generated by the front end.

Such a design has two key advantages. First, front end web servers do not need to keep the list of users, their location based on the IP address and the videos that they are allowed to watch. Therefore, it helps in simplifying the overall design. Second, since YouTube caches can store any content irrespective of their geographical locations, it helps in load sharing when a cache serving a different geographic region sees high traffic demand. Hence, it allows more flexibility in terms of the placement of the video caches and *proxy relay*s and does not require geography based segregation for caching/storing the videos.

A.8 Tidbits

- We accidentally discovered another YouTube *anycast* namespace, of the format `tc.v[1-24].lscache[1-8].c.youtube.com`, but never seen in the actual data we haven collected. This appears to be a back-up namespace for the *lscache* namespace, but not used externally.
- We also found that YouTube uses a different domain name for its caches located at different location. As described in Table 1, where we show it uses the domain name `c.youtube.com` for all the hosts, in some instances it also used the domain name `googlevideo.com` for the same sets of hosts.