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 - Twin Cities
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 a fixed hashing and cleverly leveraging DNS and HTTP redirection mechanisms, such a design leads to a scalable, robust and flexible content distribution system.

1. INTRODUCTION

Given the traffic volume, geographical span and scale of operations, the design of YouTube's content delivery infrastructure is perhaps one of the most challenging engineering tasks (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 [3]) as well as third-party content delivery networks [2, 9] to stream videos to users. Since Google's takeover, 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 [6] show that Google has become one of the top five *inter-domain* traffic contributors in 2009; 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 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 user performance expectations.

This work is supported in part by the NSF grants CNS-0905037 and CNS-1017647, and the DTRA Grant HDTRA1-09-1-0050.

This paper attempts to "reverse-engineer" the YouTube video delivery cloud through large-scale active measurement, data collection and analysis. 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)? and 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? 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.

The global scale of the YouTube video delivery cloud poses 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. 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 geo-locate 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 describe the measurement infrastructure and collected data in Section 3. We provide a summary of the key findings regarding the YouTube design in Section 4. In 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.

978-1-61284-350-6/11/$26.00 ©2011 IEEE

2. RELATED WORK

Most existing studies of YouTube mainly focus on user behaviors or the system performance. For instance, the authors in [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 [5] investigate the YouTube video file characteristics and usage patterns such as the number of users, requests, as seen from the perspective of an edge network. Another study [9] analyzed network traces for YouTube traffic at a campus network to understand benefits of alternative content distribution strategies. A more recent work [8] 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 [3], 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*. To the best of our knowledge, our work is the first study that attempts to reverse engineer the current YouTube video delivery cloud.

3. MEASUREMENT PLATFORM

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

3.1. YouTube Video Delivery Basics

When a user visits, or clicks on any URL of the form `http://www.youtube.com/watch?v=ABCDEFGHIJK`, the web server returns an HTML page with *embedded* URLs of certain forms, e.g., `v1.lscache5.c.youtube.com`, pointing to the Flash video server responsible for serving that video. When the user clicks the playback button of an embedded Flash video object on the page, the browser resolves the hostname to get an IP address for the Flash video server, which then streams the video to the user's browser.

3.2. Active Measurement Platform

Our active measurement platform utilizes 471 PlanetLab nodes, and 843 open recursive DNS servers provided by and located at various ISPs and organizations. We also developed an emulated YouTube Flash video player in Python which emulates the process involved in playing back a YouTube video using HTTP. The emulator records detailed log of the process including multiple HTTP request redirections and DNS resolutions. 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 addition, our emulated YouTube Flash video player can be configured to use an open recursive DNS servers (instead of the default local DNS server of a PlanetLab node). 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.

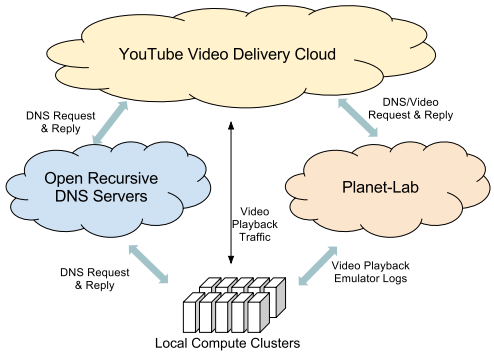

Fig. 1. Our Active Measurement Platform: an Illustration.

3.3. Measurement Methodology and 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, and record their view counts and other relevant metadata. We show the popularity of the videos in our list in Fig. 3. We note that because of our sampling method, a large percentage of the videos in our list are popular videos. To address this problem, we added a large number of videos with very low view counts to this list of 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. Third, we perform ping-based latency measurements from the PlanetLab nodes to all the observed IP addresses. Furthermore, we also extract the HTTP request redirection sequences, analyze and model these sequences to understand YouTube redirection logic.

4. SUMMARY OF KEY FINDINGS

In this section we provide a summary of our key findings regarding the design and operations of the YouTube global

Fig. 2. Geographical distribution of YouTube Video Cache Locations.

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

The design of 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.

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 _. The total size of the YouTube video id space is effectively 64^{11}. Analyzing the $434K$ video ids we collected, we find that they are uniformly distributed in the video id space.

Anycast DNS Namespaces and Layered *Logical* Video Server Organization. Based upon the request redirection mechanism deiscussed later, we observe that 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. As shown in Table 1, there are a total of three *anycast* namespaces, which we refer to as *lscache*, *tccache*, and *cache* namespaces; each namespace has a specific format [1].

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. 2). We next see what logical hostnames map to what location to classify the physical cache locations. 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. In Section 5.3 we will provide some details regarding how we geo-map the YouTube physical cache locations.

Unicast Namespace. In addition, for each IP address, YouTube also defines a *unicast* DNS name. Namely, there is a one-to-one between this DNS name and the IP address. As shown in Table 1, the unicast names have two formats, which we refer to as *rhost* and *rhostisp* formats.

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 fixed hashing, and ii) map *logical* video servers to physical video servers at various locations of its physical cache hierarchy through both DNS resolution and HTTP redirection mechanisms.

***Fixed* Mapping between Video Id Space and *Logical* Video Servers (Anycast DNS Namespaces).** YouTube adopts a fixed 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 *tccache* namespace, and a three-to-one maping between these namespaces and *cache*. 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

[1] We also encountered, although very rarely, slightly different formats for *lscache* and *cache* namespaces where hostnames had *nonxt* and *altcache* in them respectively

Table 1. Youtube *Anycast* and *Unicast* 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
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
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

(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, YouTube picks a primary video cache location that is "close" to the user, and resolves the requested *lscache* DNS name to one of the IP addresses within that location.

***Dynamic* HTTP Request Re-direction.** To perform finer-grain and dynamic load-balancing, or to handle cache misses, YouTube employs HTTP request redirection mechanism. Such a redirection mechanism is especially useful and important, as YouTube always maps the user video request to a "physical" video server at a *primary* cache location. 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 redirection process. There is a *strict ordering* among the anycast namespaces, only redirection from a "higher" layer namespace to a "lower" layer namespace is allowed, e.g., from *lscache* to *cache*, but not the other way round; a redirection can "jump" across multiple layers, e.g., from *lscache* to *tccache* or *cache*. YouTube uses both a redirection count and a tag to keep track of the redirection sequences.

5. 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* space 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.

5.1. Anycast DNS Namespace

Table 1 summarizes the *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 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. We also found that each *video id* maps to a unique *lscache* hostname out of 192 such names. For instance, 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, 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 4. As seen in this figure, there are approximately equal number of videos mapped to each of the *lscache* hostnames.

Based upon the location of IP addresses that hostnames in this namespace map to, we see 38 primary cache locations. We note that as we increased the number of vantage points, we increasingly uncovered additional primary cache locations.

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. *Tertiary Video Caches* show up as the final set of video caches during redirections in our playback logs. Unlike primary and secondary video caches, these tertiary video caches have a namespace constituted by only 64

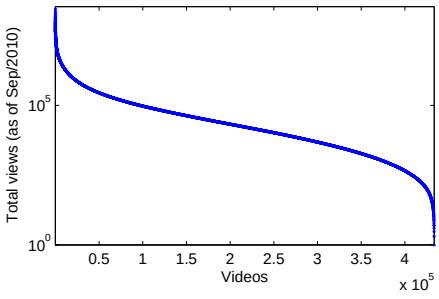

Fig. 3. View counts for $434K$ videos

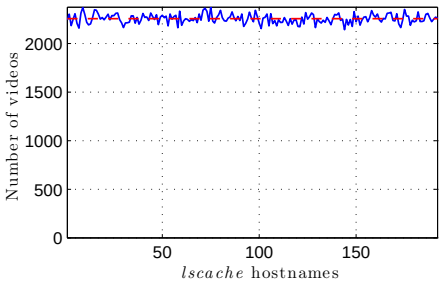

Fig. 4. Number of videos mapped to each *lscache* hostname.

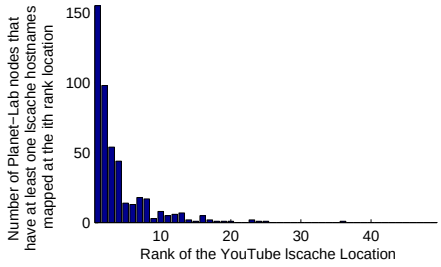

Fig. 5. PlanetLab nodes with delay based rank of the YouTube location.

DNS hostnames, which are represented using regular expression: v[1-8].cache[1-8].c.youtube.com. 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. Unlike primary cache locations, the number of secondary and tertiary cache locations did not increase as we added new vantage points.

Table 2. DNS resolutions for v1.lscache1.c.youtube.com

PlanetLab node	Resolved IP	IP location
adam.ee.ntu.edu.tw	202.169.174.208	Taipei
chimay.infonet.fundp.ac.be	74.125.10.144	Amsterdam
cs-planetlab4.cs.surrey.sfu.ca	74.125.107.16	Seattle
dannan.disy.inf.uni-konstanz.de	173.194.18.70	Frankfurt
ds-pl3.technion.ac.il	173.194.18.6	Frankfurt

5.2. 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. 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.

5.3. Geo-mapping YouTube Cache Locations

In order to geolocate YouTube IP addresses, first of all, we leverage the large number of *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 correct.

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 [7]. 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 are able to geolocate all the YouTube IP addresses to 47 different cities spread across the globe. 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 2.

5.4. 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 location-awareness in DNS resolutions, we performed the following analysis. 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. In Figure 5 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. Only a very small number of nodes see that the mapped IP addresses are from farther locations.

6. HTTP REDIRECTIONS

YouTube uses HTTP based redirection to achieve dynamic load-balancing. For instance, if the *lscache* server responsible for a video cannot serve the requested video, it sends a HTTP 302 response back to the client. This response tells the client to go to another server to download the video from. If the host corresponding to the HTTP redirect URL, again, can not provide the video for some reason, it sends another HTTP 302 response to the client to ask it to try yet another hostname. 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.

To better understand the patterns in HTTP redirections, we carefully examined the video playback logs corresponding to such cases. Our analysis of these logs reveals several interesting patterns in these redirections and the mechanisms used to avoid HTTP redirection loops.

The first key finding here is that HTTP redirections for any given video follow a very specific namespace hierarchy. A *lscache* video cache server may re-direct a video request to a corresponding *tccache* video cache server , or directly to a corresponding *cache* video cache server. Likewise, a *tccache* video cache server may only re-direct a video request to a corresponding *cache* video cache server but not to a *lscache* video server. In addition, we found that these HTTP redirections may also involve several *unicast* hostnames (such as *rhost* and *rhostisp*) as well. However, the redirection to one of these hostnames only occurs from one of the *anycast* hostnames. 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 (*tccache* and *cache* use the tags `&st=tcts` and `&st=ts` respectively). These mechanisms ensures that the HTTP redirection always follow the cache hierarchy and does not create a redirection loop.

As an ultimate protection against the possible redirection loops, redirection URL also contain a "redirection counter", which is incremented by one every time a video request is forwarded from one anycast namespace to another anycast namespace. When the redirection counter reaches 4 and the corresponding *anycast* YouTube host can not serve the video, then it simply fails with a HTTP 503 error code. 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 HTTP 503 error code.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we built a globally distributed active measurement platform to reverse engineer YouTube video delivery framework and uncovered and geo-located YouTube's 3-tier physical video server hierarchy, and deduced the key design features of the YouTube video delivery cloud. 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 redirection decision logic and process, and how they are affected by video popularity, that still require in-depth analysis and additional experiments. Our current ongoing work, in part, attempts to answer these questions.

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.

8. 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] 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.

[4] 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.

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

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

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

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

[9] 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.