Vivisecting YouTube: An Active Measurement Study

Vijay Kumar Adhikari, Sourabh Jain, Yingying Chen, and Zhi-Li Zhang
University of Minnesota

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

I. INTRODUCTION

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

The rest of the paper is organized as follows. We describe the measurement infrastructure and collected data in Section II. In Section III, Section IV and Section V we present details regarding how we derive our findings, including the analysis performed, the methods used, and additional experiments conducted to verify the findings. We conclude the paper in Section VI.

This work is supported in part by the NSF grants CNS-0905037, CNS-1017092 and CNS-1017647.

Related Work. Most existing studies of YouTube mainly focus on user behaviors or the system performance. The authors in [2], [4] examined the YouTube video popularity distribution, popularity evolution, and its related user behaviors. 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.

In [1], Adhikari et al. uncover the YouTube data center locations, and infer the load-balancing strategy employed by YouTube at the time. In [7], the authors examine data collected from multiple networks to uncover the server selection strategies YouTube uses. To the best of our knowledge, our work is the first study that attempts to reverse engineer the current YouTube video delivery system to understand its overall architecture.

II. MEASUREMENT AND DATA

We developed a distributed active measurement and data collection platform consisting of 471 PlanetLab nodes that are distributed at 271 geographically dispersed sites and 843 open recursive DNS servers located at various ISPs and organizations. We also developed an emulated YouTube Flash video player in Python which performs the two-stage process involved in playing back a YouTube video: In the first stage, our emulated video player first connects to the YouTube’s website to download a web page, and extracts the URL referencing a Flash video object. In the second stage, after resolving the DNS name contained in the URL, our emulated video player connects to the YouTube Flash video server thus resolved, and follows the HTTP protocol to download the video, and records a detailed log of the process. In addition, our emulated YouTube Flash video player can be configured to use an open recursive DNS server (instead of the default local DNS server of a PlanetLab node) for resolving YouTube DNS names. This capability enables us to use the 843 open recursive DNS servers as additional vantage points.

We adopt a multi-step process to collect, measure, and analyze YouTube data. First, we crawl the YouTube website from geographically dispersed vantage points using the PlanetLab nodes to collect a list of videos, record their view counts and other relevant metadata, and extract the URLs referencing the videos. Second, we feed the URLs referencing the videos to our emulated YouTube Flash video players, download and “playback” the Flash video objects from the 471 globally distributed vantage points, perform DNS resolutions from these vantage points, and record the entire playback...
processes including HTTP logs. This yields a collection of
detailed video playback traces. Third, using the video playback
traces, we extract all the DNS name and IP address mappings
from the DNS resolution processes, analyze the structures
of the DNS names, and perform ping latency measurements
from the PlanetLab nodes to the IP addresses, and so forth.
Furthermore, we also extract the HTTP request redirection
sequences, analyze and model these sequences to understand
YouTube redirection logic.

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

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

**III. Video ID Space & Namespace Mapping**

YouTube references each video using a unique “flat” video
id, consisting of 11 literals. We refer to the collection of
all video ids as the video id space. While we find that the
literals in the first 10 positions can be one of the following
64 symbols: \{a-Z, 0-9, \., \-, \_\}, only 16 of these 64 symbols
appear in the 11th (last) position. Therefore, while the size
of the YouTube video id space is 64^{11}, the theoretical upper
bound on the number of videos in YouTube is 63^{11} \times 16, still
an astronomical number. Analyzing the 434K video ids in our
list, we find that they are uniformly distributed in the video id
space.

As we show in Table I and elaborate further in Section IV,
YouTube employs a number of DNS namespaces. Only DNS
names belonging to the lscache namespace are generally
visible in the URLs contained in the YouTube webpages; DNS
names belonging to other namespaces only appear in URLs in
subsequent HTTP redirection requests (see Section V-C). We
find that each video id is always mapped to a fixed hostname,
out of the 192 possible names (logical servers) in the lscache
namespace, regardless of location and time. For example, a
video identified using the video id MQCNuv2QxQY always maps
to v23.lscache1.c.youtube.com lscache name from all the
PlanetLab nodes at all times. Moreover, when redirection
happens, each video id is always mapped to a fixed hostname
(out of 192 names) in the nonxt or tccache namespace, and to
a fixed hostname (out of 64 names) in the cache or altcache
namespace. Moreover, we find that this fixed mapping between
the video id space to anycast namespaces makes sure that the
number of video ids that map to anycast hostname are
nearly equally distributed. To demonstrate this, we plot the
number of video ids that map to each of the lscache
hostnames in Figure 1. We see that there are approximately equal number of
videos mapped to each of the lscache hostnames.

**IV. Cache Namespaces & Hierarchy**

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

We find that the 5 anycast and 2 unicast namespaces map
equally to the same set of IP addresses: about 93% of the
5,883 IP addresses have a (unique) unicast name associated
with them. Second, 80% of the IP prefixes come from ad-
dresses assigned to Google/YouTube, while the remaining 20%
of the prefixes coming from address space assigned to other
ISPs such as Comcast and Bell-Canada (hereafter referred to as
non-Google addresses/prefixes). The former have the unicast
names of the form rhost, whereas the latter rhostisp. Clearly,
the unicast names indicate that Google/YouTube have video
caches co-located within other ISP networks (referred to as
non-Google locations) as well as within its own (referred to
as Google locations). The 3-letter city code provides a hint as
to where the corresponding YouTube cache is located (at the
granularity of a city or metro-area). To geo-locate and classify
those IP addresses that do not have an associated unicast
name in our datasets and to further validate the geo-locations
of YouTube video caches, we conduct pair-wise round-trip
measurements from each PlanetLab node to all of the YouTube
IP addresses. Using these measurements as well as the round
trip delay logs in the collected video playback traces, we
perform geo-location clustering similar to the approach used
by GeoPing [6]. This yields a total of 47 cache locations.
We plot them (including both Google and non-Google cache
locations) on a world map in Figure 4.

Based on the HTTP redirection sequences, there is a clear
hierarchy among the 5 anycast namespaces, as shown in
Figure 2: A video server mapped to a lscache hostname (in
short, a lscache server) may redirect a video request to the
The YouTube cache locations are organized into a 3-tiered hierarchy: there are roughly primary cache locations geographically dispersed across the world (most are owned by Google, some are co-located within ISP networks); there are 8 secondary and 5 tertiary cache locations in US and Europe and owned by Google only. We discuss each tier below in more details below.

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

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

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

V. VIDEO DELIVERY DYNAMICS

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

**Experimental Methodology.** We divide the videos into two sets: i) **hot** videos which have a very high number of view counts (at least 2 million views); and ii) **cold** videos which have fewer than 100 view counts. We randomly select a video from both **hot** and **cold** sets and play them one by one,
while the delay between two consecutive playback requests is modelled as a Poisson process with inter-arrival rate of 10 seconds. For each video playback request, we record the detailed logs including timestamps, redirection URLs (if any) and the IP addresses of the servers involved. In particular, we also examine the time difference between the time our client receives ACK for the HTTP GET request and the time the client sees the first packet of the HTTP response.

A. Locality-aware DNS Resolution

To characterize the granularity of locality-aware resolutions, we conduct the following analysis. For each PlanetLab node, we rank all 38 YouTube primary cache locations in the increasing order of round trip network delay and assign each YouTube location a rank in this order. Next, we consider the ls.cache hostname-to-IP addresses mappings and calculate how they are distributed with respect to the rank of the corresponding YouTube location for the given PlanetLab node. In Figure 5 we plot the number of PlanetLab nodes which have at least one of ls.cache hostnames mapped to an ith rank YouTube location. As seen in this figure, more than 150 PlanetLab nodes have at least one of the IP addresses at the closest YouTube location. Using our DNS mapping data collected over several months, we also investigate whether YouTube adjusts the number of IP addresses mapped to each ls.cache hostname over time to, say, adapt to the changing loads at particular cache locations or regions of users. We create a temporal matrix of DNS name to IP address mapping matrix for each ls.cache hostname, where each row in the matrix represents the mappings of the hostname at a given time from all the PlanetLab nodes. Analysis of this matrix reveals two interesting aspects of the way YouTube DNS servers resolve anycast hostnames to IP addresses. First, we see that the hostname to IP address mappings may change over time. Based on how these mappings changed for PlanetLab nodes, we can put them into two distinct groups. In the first group of PlanetLab nodes, the mappings change during a certain time of the day, and the pattern repeats every day. In the second group, the set of IP addresses remains the same over time. Figure 6 provides an illustration: the top panel shows an example of the first group, while the bottom panel an example of the second group. In this figure: the X-axis represents the time which is divided in the intervals of 5 minutes each, and the Y-axis represents the mapped IP address. In the top panel, at the ple1.dmcs.plodz.pl PlanetLab node, one hostname is mapped to a fixed IP address (belonging to the Frankfurt cache location) most of the time during the day; however, during the certain hours of the day we see a large number of distinct IP addresses for the same hostname. In the bottom panel, one hostname is always mapped to one of the two IP addresses (belonging to the Taipei cache location).

B. Handling Cache Misses via Backend Fetching

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

We find that when the cache server redirects the client the fetch-time is very small, generally about 3ms. We also see about the same fetch-time for most of the hot videos when the server actually serves the video. For most of the cold videos when they are not redirected, this lag is much higher, typically in tens of milliseconds and vary depending upon cache location. An example of the distribution is presented in Figure 7 which shows the distribution of fetch-times of one Google YouTube cache server observed from a fixed vantage point. There is a clear gap between the shorter and longer fetch times. We deduce that large fetch-time is the time it takes for the cache server to fetch the content from some backend data center (cf. [3]).

C. HTTP Redirections Dynamics

The video redirection logs reveal that HTTP redirections always follow a specific namespace hierarchy, as shown in Figure 2. Our analysis of video redirection logs also reveals that redirection probability highly depends on the popularity of the video. However, there were no significant evidences to show if the factors such as the location of the YouTube cache and time of the day influence the redirection probability. In Figure 8 we demonstrate how redirection probability is distributed for hot and cold at both Google and Non-Google locations. In these figures, x-axis represents the IP prefixes for the YouTube primary cache servers, which is sorted based on the region and then based upon the size of each location. The y-axis represents the probability of redirection to another namespace. As seen in Figure 8(a), at Non-Google locations, cold videos have much higher probability of being redirected to nonxt namespace than for the hot videos. In particular, around 5% of the requests to hot videos experience redirections as compared to more than 24% for the cold videos. Similarly, at Google cache locations, most of the requests to cold videos are redirected to cache hostnames (see Figure 8(b)). It indicates that these redirections are primarily done to handle cache misses by redirecting the users to the third tier directly. On the other hand, the redirection probability to tccache and rhost hostnames does not depend on the popularity of the video. As we see in Figure 8(c), the probability of redirection for hot and cold videos to rhost namespace is very similar at all the Google cache locations. Moreover, a closer inspection of redirection logs revealed that redirection rhost hostnames is used to redirect the user to a different physical server at the same location, which is more than 99% of all the redirections to rhost namespace. This indicates that YouTube performs a very fine grained load balancing by redirecting the users from possibly a very busy server to a less busy server at the same
D. Delay due to Redirections

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

Figure 9 shows the CDF plot for the video initialization time observed by one of the PlanetLab nodes. As seen in this figure, HTTP redirection used by YouTube servers add a significant overhead to the video initialization time. In particular, our results show that on an average HTTP redirections increase the video initialization time by more than 33% in comparison to video initialization time when there are no redirections.

VI. SUMMARY

In this paper we reverse-engineer the YouTube video delivery system by building a globally distributed active measurement platform and deduced the key design features of the YouTube video delivery system. While Google’s YouTube video delivery system represents an example of the “best practices” in the design of a large-scale content delivery system, its design also poses several interesting and important questions regarding alternative system designs, cache placement, content replication and load balancing strategies.

REFERENCES