Video CDN

In this project you will explore how **video content distribution networks (CDNs)** work.

There are two checkpoints and their grading policies are shown below:

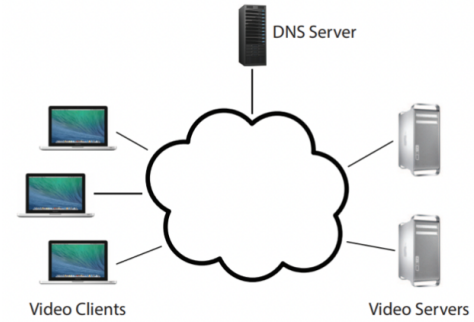


Figure 1: In the real world...

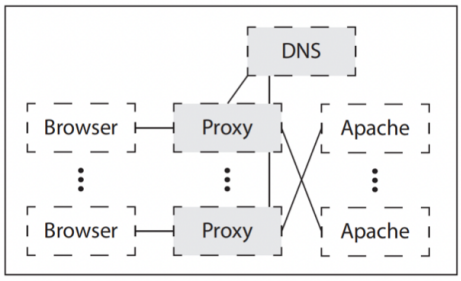


Figure 2: Your system.

Introduction

A content delivery network(CDN) refers to a geographically distributed group of servers which work together to provide fast delivery of Internet content. In particular, you will implement a video distribution systems that uses **adaptive bit rate selection** and **DNS redirect**. In bouns part, you are encouraged to **integrate danmaku system and comment system into your system** and do whatever you can think of to improve the functionality of the whole website.

Goal	Grade	Details
Video Distribution System	100	Implement the bit rate adaption, proxy and DNS
Bonus	15	Integrate danmaku system and comment system into our system
	5	Other improvements to the website

Figure 1 depicts (at a high level) what this system looks like **in the real world**. Since it is really complex, let's simplify things as figure 2 shows. A simplified CDN system works as follows:

1. Browser requests proxy for video as a video client does.
2. Proxy first issues a DNS query to resolve the service's domain name.
3. The CDN's authoritative DNS server selects the "best" content server for each particular client using specific algorithms.
4. Once the proxy receives the response from DNS server, it forwards the requests to apache server(act as a video server) and send the response from apache server back to client browser.
5. The video is encoded at multiple bit rates, as the proxy receives video data, it calculates the throughput of the transfer and requests the highest bit rate the connection can support.

Video Distribution System

Figure 2 shows the pieces of the system; those shaded in gray will be written by you. It contains four parts.

Browser. You will use an off-the-shelf web browser to play videos served by your CDN (via your proxy).

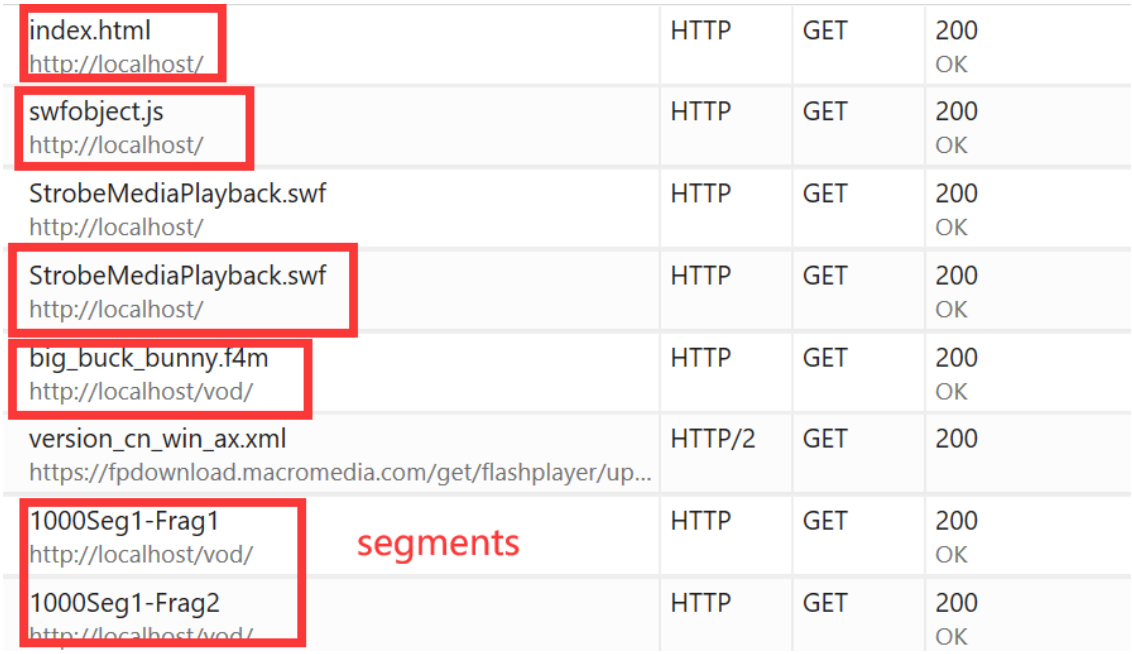
Proxy. You will implement a HTTP proxy with adaptive bit rate selection. The client requests chunks with standard HTTP GET requests; your proxy will intercept these and modify them to retrieve an appropriate bit rate your algorithm calculates. To simulate multiple clients, you will launch multiple instances of your proxy.

Web Server. Video content will be served from an off-the-shelf web server (Apache). For the same reason as proxy, you will run multiple instances of Apache to simulate a CDN with several content servers.

DNS Server. You will implement a simple DNS server(supporting only a small portion of actual DNS functionality). Your server will respond to each request with the "best" server for that particular client.

1. Implementing a Proxy

1. Your proxy should listen for connections from a browser on the port specified as a command line argument (see 4).
2. Resolves the web server's DNS name(**specified as you want**), and modifies video chunk requests (see 2.2).
3. Forwards the modified request to the apache server, the requests browser issues are listed as follows:



4. Any data (the video chunks) returned by the proxy should be forwarded, unmodified, to the browser.
5. Your proxy should have the ability to **accept multiple concurrent connections**. The reason is that the browser may open multiple TCP connections to download objects (chunks in our case) in parallel. Similarly, one instance of proxy should **open multiple connections to one server**.
6. You should also design a strategy to let proxy exit properly, such as after transferring the whole video, or after a predefined duration without any new browser requests, or other occasions that suitable for your system.

IMPORTANT: In the above picture, we can see that before requesting all video chunks, client will first requests a big_buck_bunny.f4m. We can parse this file to get the available bit rates in apache server. To avoid client to do bit rate selection itself, the proxy should modify the request url to get **big_buck_bunny_nolist.f4m** for client and left big_buck_bunny.f4m for itself.

2. Video Bit rate Adaptation

You will implement this functionality in the HTTP proxy you implemented. It involves two steps:

2.1 Throughput estimation

The goal of bit rate adaptation is to **choose the best rate for each video chunk**. The choice is based on throughput estimation.

To calculate the throughput for a particular chunk, you should note the start time **ts** of the request and the end time **tf** we you finished receiving the chunk from the server, suppose **B** is the length of the video chunk, then the throughput is :

$$T = \frac{B}{t_f - t_s}$$

The total throughput estimation should be an Exponentially Weighted Moving Average(EWMA). Every time you make a new measurement (T_{new}), update your current throughput estimate(EWMA) as follows:

$$T_{current} = \alpha T_{new} + (1 - \alpha) T_{current}$$

You will control α via a command line argument. When a new server is used, e.g., for a new stream, set $T_{current}$ to the lowest available bit rate for that video and server.

2.2 Adaptive bit rate selection

Once your proxy has calculated the connection's current throughput, it should select the highest offered bit rate the connection can support. **For this project, we assure a connection can support a bit rate if the average throughput is at least 1.5 times the bit rate.**

Your proxy should learn which bit rates are available for a given video by **parsing the manifest file** (the ".f4m" initially requested at the beginning of the stream). The manifest is encoded in XML; each encoding of the video is described by a element, whose bit rate attribute you should find. **And at the beginning, your proxy should request the lowest bit rate.**

Your proxy replaces each chunk request with a request for the same chunk at the selected bit rate (in Kbps) by **modifying the HTTP request's Request-URI**. Video chunk URIs are structured as `/path/to/video/<bitrate>Seq<num>-Frag<num>`

For example, suppose the player requests a chunk that corresponds to fragment 3 of sequence 2 of the video Big Buck Bunny at 500 Kbps: `/path/to/video/500Seg2-Frag3`.

3. DNS

3.1 Load Balancing

DNS load balancing, which configures the CDN's authoritative DNS server to resolve a single domain name to one out of a set of IP addresses belonging to replicated content servers, can spread the load of the video service across a set of servers.

There are many algorithms could achieve DNS load balancing, like round-robin and shortest path. In this project, you are required to implement the DNS load balancer based on the round-robin algorithm.

3.1.1 Round-Robin

You should implement the Round-Robin strategy based on a DNS server which could communicate with the proxy server. In our scenario, apache servers are running on different ports specified by the file in **docker_setup/netsim/servers**, your dns server should read the file from here. **Note that what you read should consistent with what you have launched.** Then, the balancer will return suitable port in the list as the response to each request, and ,cycle back to the beginning when the list is exhausted. The port retured should be transffered to the proxy server.

4. Running Details

Running the apache servers:

```
./netsim.py servers start -s servers/2servers
#will open two server port, in this situation, you should read servers/2servers
in your dns server.
```

Running the proxy: proxy.py should be invoked as follows, note that not all arguments are functional at each test.

```
./proxy <log> <alpha> <listen-port> <dns-port> [<default-port>]
```

log The file path to which you should log the messages.

α A float in the range [0, 1]. Uses this as the coefficient in your throughput estimate.(see 4.1)

listen-port The TCP port your proxy should listen on for accepting connections from your browser.

dns-port UDP port DNS server listens on.

default-port Your proxy should accept an optional argument specifying the port of the web server from which it should request video chunks. If this argument is not present, your proxy should obtain the web server's port by querying your DNS server.

To play a video through your proxy, point a browser to the URL:

```
http://localhost:<listen-port>/index.html
```

4.1 Log Message

After each request, it should append the following line to the log:

```
<time> <duration> <tput> <avg-tput> <bitrate> <server-port> <chunkname>
```

time The start time accurate to the second of proxy requesting for current chunk.

duration A floating point number representing the number of seconds it took to download this chunk from the server to the proxy.

tput The throughput you measured for the current chunk in Kbps.

avg-tput Your current EWMA throughput estimate in Kbps.(see 2.2.1)

bit rate The bitrate your proxy requested for this chunk in Kbps.

server-port The port number of the server to which the proxy forwarded this request.

chunkname The name of the file your proxy requested from the server (Seq-Frag).

5. Tasks Summary

Task 1: Implement the basic proxy

Task 2: Implement bit rate adaptation

Task 3: Implement DNS server

Task 4: Explore the behavior of your proxy. Once your proxy is working, launch two instances of it. Then run two servers on two different ports. you should direct one proxy to each server. Now:

1. Run the servers direct netsim.py to generate a log file:

```
$ python3 netsim.py servers start -s docker_setup/netsim/servers/2servers -l
log_file_path
```

2. Start playing the video through each proxy.
3. After 1 minute, kill the proxies.
4. Gather the netsim log file and the log files from your proxy and use them to generate plots for link utilization, fairness, and smoothness. Use our grapher.py script to do this:

```
$ ./grapher.py <netsim-log> <proxy-1-log> <proxy-2-log>
```

Repeat these steps for α = 0.1, α = 0.5, α = 0.9. Compile your plots, labelled clearly, into your report, along with a brief (1-3 paragraphs) discussion of the trade-offs you make as you vary α.

Bonus

You are encouraged to **integrate danmaku system and comment system into our system** and do whatever you can think of to improve the functionality of the whole website!

We explain danmaku system as follows:

Danmaku is a popular form of comment. When watching videos or live shows, you can send a danmaku on the certain timestamp or in real-time. Generally, you will see your danmaku flying from one side of the screen to the other side (e.g., right to left).
In this project, you can implement a simple danmaku system as what you did in PA2. As you query video contents from server, the server will also send back the danmakus together with video and they will be shown on screen. Besides, send new danmakus in video page and record it so that we will see it any time when we watch the video. Make sure that all clients can receive danmakus correctly.

Explore other parts as far as you can :) !

Hand In

report.pdf — This should contain the plots and analysis described above. Please also clearly illustrate what you've done, especially how to exit your proxy, and work division. Add screenshots or codes if needed.

src.zip — A directory named **src** containing your source code. At least contains proxy.py and dns.py

Grading

Task	Points	Details
Compile and Output	15	Correct running format and successfully compile. Successfully set up environment.
Log File	5	Your proxy can generate log file with right format.
Basic Proxy	15	Receive requests from browser and forward them to servers. Forward the chunks to servers. Accept multiple concurrent connections. Open multiple connections to servers. Exit properly.
Bit Rate Adaption	20	Throughput estimation. Bit rate selection.
DNS	15	Implement the communication between the proxy server and the DNS server.
Round-Robin	10	Achieve load balancing based on the Round-Robin strategy.
Bonus	20	
Report	20	

Development Environment

For this project, we are providing a docker image pre-configured with the software you will need. You can choose other environment, but we strongly **recommend** that you do all development and testing in this image. **Your code must compile and run correctly on this image as we will be using it for grading.**

For clarity, we put environment description into GitHub repos. You can check their readme for more info.

Video Distribution System:https://github.com/Nancyzxy/CS305-proj

Please post your questions in issues. If there are some necessary updates for our repo, we will announce at QQ group.

Contact Us

11912039@mail.sustech.edu.cn 郑鑫颖

11912935@mail.sustech.edu.cn 刘思语

11812419@mail.sustech.edu.cn 江宇辰

11812636@mail.sustech.edu.cn 周颖泉

11812704@mail.sustech.edu.cn 王硕

Reference

CMU 15-441/641: Computer Networks, Fall 2019

cmu-abr-proxy

cmu-load-balancer