2019 CSEE 4119: Computer Networks

Project 1: Video CDN

Preliminary Stage Due: October 8, 2019, at 11pm
Final Stage Due: November 5, 2019, at 11pm
Updates:
2019-10-20: Fixed a link that was pointing to the wrong place. (in 3.2.4 fake-ip)

1. Overview **2**
1.1 In the Real World 2
1.2 Your System 3
1.3 Groups and collaboration policy 4

2. Preliminary stage **4**
2.1 Get your connections right. 4
2.2 Forwarding protocol. 5
2.3 Running the proxy 6
2.4 Test it out! 7
2.5 Final result 7
2.6 What to Submit for preliminary stage 7
2.7 Where to Submit 8

3. Final stage: Video Bitrate Adaptation **8**
3.1 Requirements 8
3.2 Implementation Details 9
3.2.1 Throughput Calculation 10
3.2.2 Choosing a Bitrate 10
3.2.3 Logging 11
3.2.4 Running the Proxy 11
3.3 What to Submit for final stage 12
3.4 Where to Submit 13
3.5 Possible plan of attack 13
3.6 Hints 14
3.7 Sample output 14

4 Development Environment **15**

4.1 Virtual Box 15
4.2 Starter Files (for final stage) 15
4.3 Network Simulation (for final stage) 17
4.4 Apache (for final stage) 18
4.5 Programming Language and Packages 18
7 Grading **19**
Academic integrity: Zero tolerance on plagiarism **19**

1. Overview

In this project, you will explore aspects of how streaming video works, as well as socket programming and HTTP. In particular, you will implement adaptive bitrate selection. The programming languages and packages are specified in the development environment section.

1.1 In the Real World

Figure 1 depicts (at a high level) what this system looks like in the real world. Clients trying to stream a video first issue a DNS query to resolve the service's domain name to an IP address for one of the content servers operated by a content delivery network (CDN). The CDN's authoritative DNS server selects the "best" content server for each particular client based on (1) the client's IP address (from which it learns the client's geographic or network location) and (2) current load on the content servers (which the servers periodically report to the DNS server).

Once the client has the IP address for one of the content servers, it begins requesting chunks of the video the user requested. The video is encoded at multiple bitrates. As the client player receives video data, it calculates the throughput of the transfer and monitors how much video it has buffered to play, and it requests the highest bitrate the connection can support without running out of video in the playback buffer.

1.2 Your System

Implementing an entire CDN is clearly a tall order, so let's simplify things. First, your entire system will run on one host; we're providing a network simulator (described in Development Environment) that will allow you to run several processes with arbitrary IP addresses on one machine. Our simulator also allows you to assign arbitrary link characteristics (bandwidth and latency) to the path between each pair of "end hosts" (processes). For this project, you will do your development and testing using a virtual machine (VM) we provide.

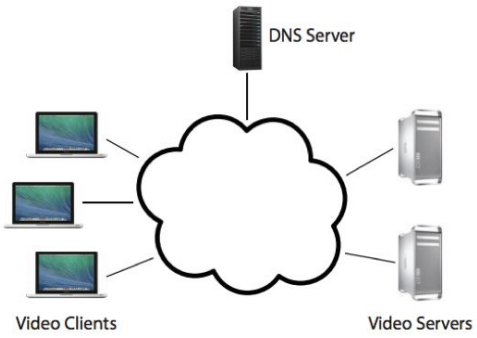


Figure 1: In the real world...

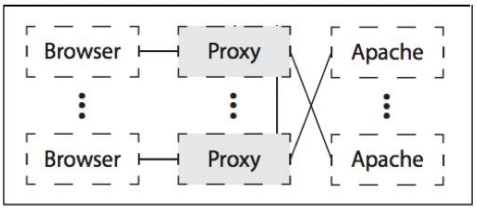


Figure 2: Your system.

Figure 3: System overview.

Browser. You'll use an off-the-shelf web browser (e.g. Firefox) to play videos served by your CDN (via your proxy).

Proxy. Rather than modify the video player itself, you will implement adaptive bitrate selection in an HTTP proxy. The player requests chunks with standard HTTP GET requests. Your proxy will intercept these and modify them to retrieve whichever bitrate your algorithm deems appropriate. To simulate multiple clients, you will launch multiple instances of your proxy. More detail in in the Video Bitrate Adaptation section.

Web Server. Video content will be served from an off-the-shelf web server (Apache). More detail in Development Environment. As with the proxy, you can run multiple instances of Apache on different fake IP addresses to simulate a CDN with several content servers. However, in the assignment, rather than using DNS redirection like a CDN would, the proxy will contact a particular server via its IP address (without a DNS lookup). A possible (ungraded) future extension to the project could include

implementing a DNS server that decides which server to direct the proxy to, based on distance or network conditions from a proxy to various web servers.

The project is broken up into two stages:

- In the preliminary stage, you will implement a simple proxy that sequentially handles clients and passes messages back and forth between client and server without modifying the messages.
- In the final stage, you will extend the proxy to implement the full functionality described above, with the proxy modifying HTTP requests to perform bitrate adaptation.

1.3 Groups and collaboration policy

This is an individual project, but you can discuss it at a conceptual level with other students or consult Internet material (excluding implementations of Python proxies), as long as the final code and configuration you submit is completely yours and as long as you do not share code or configuration. Before starting the project, be sure to read the collaboration policy at the end of this document.

2. Preliminary stage

You will be implementing a simple proxy that accepts client connections sequentially (i.e. handles a client, and, once it disconnects, takes care of the next client). In later stages, your proxy will be required to handle client connections concurrently.

In this preliminary stage, the proxy does NOT need to modify any messages it receives, as it will just relay the messages back and forth. In later stages, you will enhance your proxy to modify messages in order to perform adaptive bitrate selection.

2.1 Get your connections right.

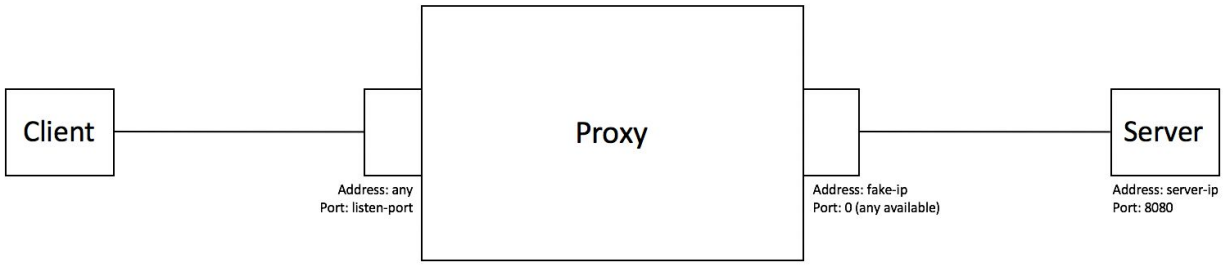
Your proxy should accept connections from clients and then open up another connection with a server (see How to run the proxy). Once both connections are established, the proxy should forward messages between the client and server.

You should implement this in two steps:

a. Establish a connection with a client:
   Your proxy should listen for connections from a client on any IP address on the port specified as a command line argument (see How to run the proxy). Your

proxy should accept multiple connections from clients. It is not required to handle them concurrently for now. Simply handling them one by one, sequentially, will be enough.

b. Establish a connection with a server:
   Once the proxy gets connected to the client, it should then connect to the server. The server IP is provided as a command line argument. As for the port number, use 8080. Make sure to close connections to the client and server when either of them disconnects.



(Figure 4) Preliminary structure

2.2 Forwarding protocol.

The “messages” that the proxy forwards follow a particular structure (for example, in HTTP, you know that there is a header and a body). This structure is important, as the recipient of the message can know where to look to get a specific piece of information. For the preliminary stage, we keep our message structure very simple:

(Figure 5) Structure of a message

The message has a body, and an End Of Message (EOM) symbol that indicates the end of the message. In our case, we define our EOM as the the new line character ‘\n’. Please note that detecting ‘\n’ is different from detecting the slash character ‘\’ and the letter ‘n’..

A message has to be fully received by the proxy before being forwarded to the other side.

Here is how the forwarding protocol works in our case:

1. The proxy gets a message from the client and forwards it to the server
2. The proxy expects a response from the server, gets it, and forwards it to the client

An important thing to notice here is that there is no asynchronous forwarding (i.e., the proxy doesn't simply forward any message, it first waits for a message from the client, and then waits for a response from a server). In other terms, the proxy shouldn't forward a message coming from a server before getting one from the client.

2.3 Running the proxy

You should create an executable Python script called **proxy** inside the proxy directory (see below for a description of the development environment), which should be invoked as follows:

cd ~/project1-starter/proxy
./proxy <listen-port> <fake-ip> <server-ip>

listen-port: The TCP port your proxy should listen on for accepting connections from the client.

fake-ip: Your proxy should bind to this IP address for outbound connections to the server (Edit 2018/10/10: this used to incorrectly say "to the client," but it should be "to the server"). You should not bind your proxy listen socket to this IP address— bind the listen socket to receive traffic to the specified port regardless of the IP address. (i.e. by calling mySocket.bind(("", <listen-port>))

Important note: The above is a pretty unusual thing to do. You might think "in lecture, we saw that the client socket doesn't bind, and now, you are telling us to bind the proxy's outbound socket when connecting to the server". However, it is necessary for the Final Stage's network simulator to work properly, and so we will add it in this stage.

server-ip: The IP address of the server

See instructions for making your script executable in the section Hand In.

2.4 Test it out!

You can test parts Get your connections right and Forwarding protocol of your proxy implementation by using the netcat tool (**nc** or **netcat**, which is installed in the VM) presented in class, using both a netcat client and a netcat server. You should be able to send a message from the client and see it appear on the server side. Then, any response sent from your server should also appear on the client. For the fake-ip, you can indicate 127.0.0.1 (localhost) when testing with netcat instances that are created on your machine.

2.5 Final result

Your proxy should be able to forward a message from a client to the server, and in turn, forward the response from the server back to the client. It should support back-and-forth messages until one side closes the connection. After the connection is closed, it should be able to accept a new connection from a client. You should be able to test the behavior of your application by creating netcat instances.

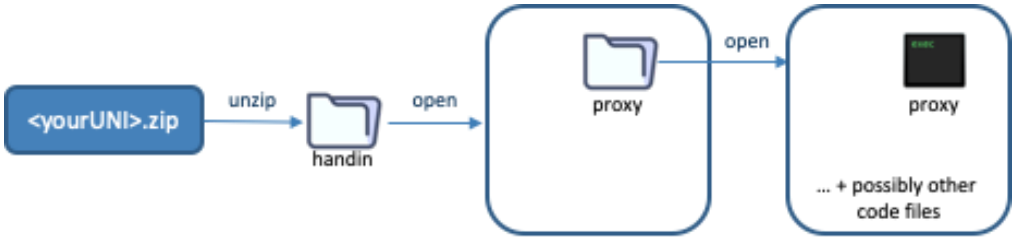
Note that this version simply forwards messages with the structure specified in Figure 2. The next stage will have a different message structure: the HTTP message structure, and you'll have to adapt your proxy based on your knowledge of HTTP messages.

2.6 What to Submit for preliminary stage

PLEASE PAY ATTENTION TO THE HANDIN STRUCTURE, AS EVEN A TYPO WILL CAUSE THE GRADER TO BREAK, WHICH CAN MAKE YOU LOSE 10 POINTS.
You will submit your project as a zipped file named **<yourUNI>.zip.** Unzipping this file should give us a directory named **handin** which should ***only*** contain the following:

- **proxy** — A directory named **proxy** containing only your source code. The code that you want to execute should be an executable named **proxy**, as described in 2.3 How to run the proxy**.** To make the code executable, follow these steps:
    1. Add '#!/usr/bin/env python' to the top of your **proxy** Python file

2. Run ‘chmod 755 proxy’ to ensure that the file has the correct permissions to be executable by us.



(Figure 6) Preliminary stage submission file structure.

You may organize your code within the **proxy** directory as you see fit. Part of your grade may be based on how understandable/organized/well-explained your code is, but we do not require any particular organization as well as it is well-organized.

2.7 Where to Submit

You will submit your code to CourseWorks**.** If you have any questions about it, please let us know ASAP.

3. Final stage: Video Bitrate Adaptation

Many video players monitor how quickly they receive data from the server and use this throughput value to request higher or lower quality encodings of the video, aiming to stream the highest quality encoding that the connection can handle. Rather than modifying an existing video client to perform bitrate adaptation, you will implement this functionality in an HTTP proxy through which your browser will direct requests.

3.1 Requirements

(1) *Implement video adaption functionality in your proxy.* Your proxy should calculate the throughput it receives from the video server and select the best bitrate for the connection. See Implementation Details for details.

(2) *Explore the behavior of your proxy.* Once your proxy is running, launch two instances of it on the “dumbbell” topology (topo1) we provide.
./netsim.py ../topos/topo1 start

(3) Running the dumbbell topology will also create two servers (listening on the IP addresses in topo1.servers: 3.0.0.1:8080 and 4.0.0.1:8080); **you should direct one proxy to EACH server.** Now:

1. Start playing the video through each proxy.

2. Run the topo1 events file and direct **netsim.py** to generate a log file: **./netsim.py -l <log-file> ../topos/topo1 run**
3. After 1 minute, stop video playback and 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 (see Throughput Calculation). Compile your 9 plots, labelled clearly, into a single PDF named writeup.pdf, along with a brief (1-3 paragraphs) discussion of the tradeoffs you make as you vary α. We're not looking for a thorough, extensive study, just observations.

3.2 Implementation Details

You are implementing a simple HTTP proxy. It accepts connections from web browsers, modifies video chunk requests as described below, opens a connection with the web server's IP address, and forwards the modified request to the server. Any data (the video chunks) returned by the server should be forwarded, unmodified, to the browser.

Your proxy should listen for connections from a browser on any IP address on the port specified as a command line argument (see below). Your proxy should accept multiple concurrent connections from web browsers by starting a new thread or process for each new request. When it connects to a server, it should first bind the socket to the fake IP address specified on the command line (note that this is somewhat atypical: you do not ordinarily bind() a client socket before connecting). Figure 4 depicts this.

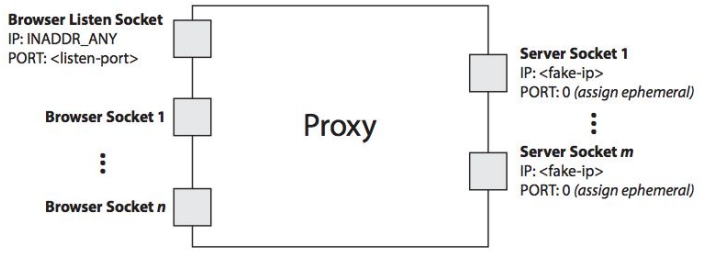


Figure 4: Your proxy should listen for browser connections on `INADDR_ANY` on the port specified on the command line. It should then connect to web servers on sockets that have been bound to the proxy's fake IP address (also specified on the command line).

3.2.1 Throughput Calculation

Your proxy could estimate each stream's throughput once per chunk as follows. Note the start time, t_s, of each chunk request (i.e., include "time" and save a current timestamp using time.time() when your proxy receives a request from the player). Save another timestamp, t_f , when you have finished receiving the chunk from the server. Now, given the size of the chunk, B, you can compute the throughput, T, your proxy saw for this chunk (to get the size of each chunk, parse the received data and find the "Content-Length" parameter) :

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

To smooth your throughput estimate, your proxy should use an exponentially-weighted moving average (EWMA). Every time you make a new measurement (T_{new}), update your current throughput estimate as follows:

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

The constant $0 \leq \alpha \leq 1$ controls the tradeoff between a smooth throughput estimate (α closer to 0) and one that reacts quickly to changes (α closer to 1). You will control α via a command line argument. When a new stream starts, set $T_{current}$ to the lowest available bitrate for that video.

3.2.2 Choosing a Bitrate

Once your proxy has calculated the connection's current throughput, it should select the highest offered bitrate the connection can support. For this project, we say a connection can support a bitrate if the average throughput is at least 1.5 times the bitrate. For example, before your proxy should request chunks encoded at 1000 Kbps, its current throughput estimate should be at least 1.5 Mbps.

Your proxy should learn which bitrates 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 **<media>** element, whose **bitrate** attribute you should find.

Your proxy replaces each chunk request with a request for the same chunk at the selected bitrate (in Kbps) by modifying the HTTP request's Request-URI. Video chunk URIs are structured as follows:

/path/to/video/<bitrate>Seq<num>-Frag<num>

For example, suppose the player requests fragment 3 of chunk 2 of the video Big Buck Bunny at 500 Kbps:

/path/to/video/500Seg2-Frag3

To switch to a higher bitrate, e.g., 1000 Kbps, the proxy should modify the URI like this:

/path/to/video/1000Seg2-Frag3

IMPORTANT: When the video player requests **big_buck_bunny.f4m**, you should instead return **big_buck_bunny_nolist.f4m**, **which is also stored in servers. The latter does not list the available bitrates (actually, it only lists a single bitrate, 1000, therefore you should expect the browser to always send you queries with 1000 as the desired bitrate for any sequence).**. Your proxy should, however, fetch **big_buck_bunny.f4m** for itself (i.e., don't return it to the client) so you can parse the list of available encodings as described above.

3.2.3 Logging

We require that your proxy create a log of its activity in a very particular format. After each request, it should append the following line to the log:

<time> <duration> <tput> <avg-tput> <bitrate> <server-ip> <chunkname>

time: The current time in seconds since the epoch.

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 link in Kbps.

avg-tput: Your current EWMA throughput estimate in Kbps.

bitrate: The bitrate your proxy requested for this chunk in Kbps.

server-ip: The IP address of the server to which the proxy forwarded this request

chunkname: The name of the file your proxy requested from the server (that is, the modified file name in the modified HTTP GET message).

3.2.4 Running the Proxy

You should create an executable Python script called **proxy** under the proxy directory, which should be invoked as follows:

cd ~/project1-starter/proxy
./proxy <log> <alpha> <listen-port> <fake-ip> <web-server-ip>

log: The file path to which you should log the messages described in **Logging**.

alpha: A float in the range [0, 1]. Uses this as the coefficient in your EWMA throughput estimate.

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

fake-ip: Your proxy should bind to this IP address *for outbound connections to the web servers.* The fake-ip can only be one of the clients' IP addresses under the network topology you specified (see Network Simulation).

The main reason why we are doing this is because netsim emulates a network topology where it can manipulate throughput on links between end-hosts. If we bind the outbound socket to this fake-ip, then we can be sure that packets sent between the proxy and the server traverse ONLY the links set by netsim. (and here is why you want your packets to traverse netsim links only)

web-server-ip: Your proxy should accept an argument specifying the IP address of the web server from which it should request video chunks. It can only be one of the servers' IP addresses under the network topology you specified (see Network Simulation).

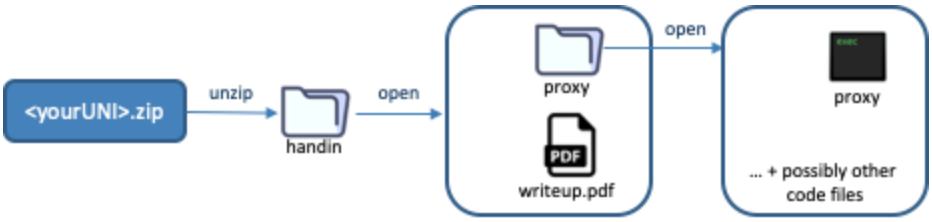
To play a video through your proxy, point a browser on your VM to the URL **http://localhost:<listen-port>/index.html.** (You can also configure VirtualBox's port forwarding to send traffic from **<listen-port>** on the host machine to **<listen-port>** on your VM; this way you can play the video from the web browser on the host.)

3.3 What to Submit for final stage

You will submit your project as a zipped file named <**yourUNI>.zip.** Unzipping this file should give us a directory named **handin** which should ***only*** contain the following:

- **writeup.pdf —** This should contain the plots and analysis described in Requirements.

- **proxy** — A directory named **proxy** containing only your source code. The code that you want to execute should be an executable named **proxy**, as described in Running the Proxy**.** To make the code executable, follow these steps:
    - Add '#!/usr/bin/env python' to the top of your Python file
    - Run 'chmod 755 proxy' to ensure that the file has the correct permissions to be executable by us.
    - You may organize your code within this directory as you see fit. If you use multiple files, please include a plain text ASCII file **README** describing what is in each file.



(Figure 5) Final stage submission file structure.

3.4 Where to Submit

You will submit your code to CourseWorks**.** If you have any questions about it, please let us know ASAP. You are expected to perform tests apart from the ones we give to you.

3.5 Possible plan of attack

1. Familiarize yourself with the netsim and network topology. Make sure you can play the videos by pointing your web browser directly to the web server's IP address.
2. Set up the connection between the web browser and the proxy. Make sure any request from your browser will be forwarded to the proxy.
3. Set up the connection between the proxy and the web server. Make sure the proxy can get the video content from web servers and can parse the content for the information you need.
4. Set up the combined connections as described in Implementation Details and test it thoroughly under different topology.

3.6 Hints

1. Open a new thread for each new connection for concurrent connections.
2. The proxy outbound socket has to wait before it's ready to read. Use select.select() to achieve that.
3. The package re can be helpful to parse the content.
4. **Remember to clear your web browser cache after each test.** Otherwise, the request won't be forwarded to your proxy but responded to by the web browser cache directly. Another way to do so is to always use **incognito mode**.

3.7 Sample output

You can get a piece of output like the sample by the following steps:

1. Start netsim of topology 1 by running
   ./netsim.py ../topos/topo1 start
2. Start your proxy server by running
   ./proxy log1.txt 0.5 8000 1.0.0.1 4.0.0.1

```
1509240972 4.92396211624 257 2263.0 1000 4.0.0.1 /vod/1000Seg9-Frag50
1509240978 4.23925685883 258 2062.0 1000 4.0.0.1 /vod/1000Seg9-Frag51
1509240983 4.94494390488 259 1882.0 1000 4.0.0.1 /vod/1000Seg9-Frag52
1509240989 4.94519901276 263 1720.0 1000 4.0.0.1 /vod/1000Seg9-Frag53
1509240996 5.64342808723 256 1574.0 1000 4.0.0.1 /vod/1000Seg9-Frag54
1509241004 6.35390591621 251 1441.0 1000 4.0.0.1 /vod/1000Seg10-Frag55
1509241007 2.8359951973 283 1325.0 500 4.0.0.1 /vod/500Seg10-Frag56
1509241009 1.43203115463 263 1219.0 500 4.0.0.1 /vod/500Seg10-Frag57
1509241011 1.4340171814 315 1129.0 500 4.0.0.1 /vod/500Seg10-Frag58
1509241013 1.4304690361 285 1044.0 500 4.0.0.1 /vod/500Seg10-Frag59
1509241014 1.44487190247 263 966.0 500 4.0.0.1 /vod/500Seg10-Frag60
1509241016 1.39674711227 329 903.0 500 4.0.0.1 /vod/500Seg11-Frag61
1509241019 1.44435095787 299 842.0 500 4.0.0.1 /vod/500Seg11-Frag62
1509241021 1.43053507805 361 794.0 500 4.0.0.1 /vod/500Seg11-Frag63
1509241024 2.84328579903 269 742.0 500 4.0.0.1 /vod/500Seg11-Frag64
1509241026 1.43109798431 249 692.0 100 4.0.0.1 /vod/100Seg11-Frag65
1509241028 1.3797750473 270 650.0 100 4.0.0.1 /vod/100Seg11-Frag66
1509241029 1.43085694313 246 610.0 100 4.0.0.1 /vod/100Seg12-Frag67
1509241030 0.682588815689 474 596.0 100 4.0.0.1 /vod/100Seg12-Frag68
1509241033 0.744397878647 400 576.0 100 4.0.0.1 /vod/100Seg12-Frag69
1509241034 0.724179983139 365 555.0 100 4.0.0.1 /vod/100Seg12-Frag70
1509241035 0.720443964005 341 534.0 100 4.0.0.1 /vod/100Seg12-Frag71
1509241037 0.718963861465 409 521.0 100 4.0.0.1 /vod/100Seg12-Frag72
1509241038 0.731734037399 378 507.0 100 4.0.0.1 /vod/100Seg13-Frag73
1509241040 0.730190038681 427 499.0 100 4.0.0.1 /vod/100Seg13-Frag74
1509241043 1.43884992599 278 477.0 100 4.0.0.1 /vod/100Seg13-Frag75
```

(sample log1.txt output)

3. Start to play the video through your proxy by pointing the web browser to **http://localhost:8000**
4. Generate the network events by running
   ./netsim.py -l netsim_log.txt ../topos/topo1 run

4 Development Environment

For the project, we are providing a virtual machine (VM) pre-configured with the software you will need. We strongly recommend that you do all development and testing in this VM; your code must run correctly on this image as we will be using it for grading. For example, some students on previous years decided to write their code on their Windows environment, which changed the control characters to **CLRF (Unix uses LF, thus our grader could not run their code)**. Please make sure your code uses **LF**. This section describes the VM and the starter code it contains.

4.1 Virtual Box

The virtual machine disk (VMDK) we provide was created using VirtualBox, though you may be able to use it with other virtualization software. VirtualBox is a free download for Windows, OSX, and Linux on **https://www.virtualbox.org.** And please download the VM instance here, and then import it to your own VirtualBox. Please set the number of processors according to your host machine (By selecting your imported VM image and go to *Settings-->System-->Processor*).

We've already set up an admin account:

Username: networks

Password: csee4119

4.2 Starter Files *(for final stage)*

You will find the following files in **/home/networks/project1-starter.**

common Common code used by our network simulation and LSA generation scripts.

lsa

lsa/genlsa.py Generates LSAs for a provided network topology. (***LSAs are not used in this version of the project, so you can ignore them.)***

netsim

netsim/netsim.py This script controls the simulated network; see Network Simulation.

netsim/tc setup.py This script adjusts link characteristics (BW and latency) in the simulated network. It is called by netsim.py; you do not need to interact with it directly.

netsim/apache setup.py This file contains code used by netsim.py to start and stop Apache instances on the IP addresses in your .servers file; you do not need to interact with it directly.

grapher.py A script to produce plots of link utilization, fairness, and smoothness from log files. (See Requirements.) (not applicable for preliminary stage)

topos

topos/topo1

topos/topo1/topo1.clients A list of IP addresses, one per line, for the proxies. (Used by netsim.py to create a fake network interface for each proxy.)

topos/topo1/topo1.servers A list of IP addresses, one per line, for the video servers. (Used by netsim.py to create a fake interface for each server.)

topos/topo1/topo1.dns A single IP address for your DNS server. (Used by netsim.py to create a fake interface for the DNS server.) However, in this project you will ignore DNS and let your proxy connect to one of the video server directly by IP address.

topos/topo1/topo1.links A list of links in the simulated network. (Used by genlsa.py.)

topos/topo1/topo1.bottlenecks A list of bottleneck links to be used in topo1.events. (See §4.3.) (not applicable for preliminary stage)

topos/topo1/topo1.events A list of changes in link characteristics (BW and latency) to "play." See the comments in the file. (Used by netsim.py.) (not applicable for preliminary stage)

topos/topo1/topo1.lsa A list of LSAs heard by the DNS server in this topology. You can ignore it for this project.

topos/topo1/topo1.pdf A picture of the network.

topos/topo2

4.3 Network Simulation *(for final stage)*

To test your system, you will run everything (proxies, servers, and DNS server) on a simulated network in the VM. You control the simulated network with the **netsim.py** script. You need to provide the script with a directory containing a network topology, which consists of several files. We provide two sample topologies; feel free to create your own. See Starter Files for a description of each of the files comprising a topology. Note that **netsim.py** requires that each constituent file's prefix match the name of the topology (e.g. in the **topo1** directory, files are named **topo1.clients, topo1.servers**, etc.).

To start the network from the **netsim** directory:

./netsim.py <topology> start

<topology> is the path of the topology file, e.g. ../topos/topos1 for topology 1

Starting the network creates a fake network interface for each IP address in the **.clients**, **.servers** files; this allows your proxies, Apache instances to bind to these IP addresses.

To stop it once started (thereby removing the fake interfaces), run:

./netsim.py <topology> stop

To facilitate testing your adaptive bitrate selection, the simulator can vary the bandwidth and latency of a link designated as a bottleneck in your topology's **.bottlenecks** file. (Bottleneck links must be declared because our simulator limits you to adjusting the characteristics of only one link between any pair of endpoints. This also

means that some topologies simply cannot be simulated by our simulator.) To do so, add link changes to the **.events** file you pass to **netsim.py**. Events can run automatically according to timings specified in the file or they can wait to run until triggered by the user (see **topos/topo1/topo1.events** for an example). When your **.events** file is ready, tell **netsim.py** to run it:

./netsim.py <topology> run

Note that you must start the network before running any events. You can issue the run commands as many times as you want without restarting the network. You may modify the **.events** file between runs, but you must *not* modify any other topology files, including the **.bottlenecks** file, without restarting the network. Also note that the links stay as the last event configured them even when **netsim.py** finishes running.

4.4 Apache *(for final stage)*

You will use the Apache web server to serve the video files. **Netsim.py** automatically starts an instance of Apache for you on each IP address listed in your topology's **.servers** file. Each instance listens on port 8080 and is configured to serve files from **/var/www**; we have put sample video chunks here for you.

4.5 Programming Language and Packages

This project must be implemented in **Python 2.7**. If this choice of language poses a significant problem for you (i.e. you have never used python before), please contact the instructors.

For this project, you are allowed to use the following python packages:

sys, socket, thread, select, time, re

Using an unallowed package in your code may result in no credit being given. Other than the packages listed, you may only use the package if you ask on Piazza ***and*** a TA or the professor explicitly responds to your request approving the use. We will maintain a pinned Piazza post titled "Project 1 List of Approved (and Disallowed) Packages", so please check that post before posting your request. If you would like to use a package

not mentioned here and are unsure if it would be acceptable, please **add a *new* followup discussion** under the above mentioned pinned post on Piazza at least 3 days in advance of the project deadline.

In your followup discussion, you must mention the package name, the package version, and a link to the official repository of the package (e.g. http://pypi.python.org/pypi). TAs will examine your request and determine if the package is allowed and add it to the list of allowed/disallowed packages.

7 Grading

Your grade will consist of the following components:
Proxy (70 points)

- Preliminary stage proxying
- HTTP proxying
- Proxy - throughput estimation (EWMA)
- Adaptive bitrate selection
- Code executes as instructed

Writeup (20 points)

- Plots of utilization, fairness, and smoothness for $\alpha \in \{0.1, 0.5, 0.9\}$
- Discussion of tradeoffs for varying α

Style (10 points)

- Code thoroughly commented
- Code organized and modular
- README listing your files and what they contain

Please make sure your code runs as an executable and as described in Running the Proxy (Preliminary Stage, Final Stage) before submitting. Code that does not run may receive a zero on the assignment.

Academic integrity: Zero tolerance on plagiarism

The rules for Columbia University, the CS Department, and the EE Department (via SEAS: 1 and 2) apply. It is your responsibility to carefully read these policies and ask the professor (via Piazza) if you have any questions about academic integrity. Please ask the professor before submitting the assignment, with enough time to resolve the issue before the deadline. A misunderstanding of university or class policies is not an excuse for violating a policy.

This class requires closely obeying the policy on academic integrity, and has zero tolerance on plagiarism for all assignments, including both projects/programming assignments and written assignments. By zero tolerance, we mean that the minimum punishment for plagiarism/cheating is a 0 for the assignment, and all cases will be referred to the Dean of Students.

Unless explicitly stated otherwise on the assignment itself, assignments must be completed individually. For programming assignments, in particular, you must write all the code you hand in yourself, except for code that we give you as part of the assignments. You are not allowed to look at anyone else's solution (including solutions on the Internet, if there are any), and you are not allowed to look at code from previous years. You may discuss the assignments with other students at the conceptual level, but you may not write pseudocode together, or look at or copy each other's code. Please do not publish your code or make it available to future students -- for example, please do not make your code visible on Github. You may look at documentation from the tools' websites. However, you may not use external libraries or any online code unless granted explicit permission by the professor or TA. For written (non-programming) answers, if you quote material from textbooks, journal articles, manuals, etc., you **must** include a citation that gives proper credit to the source to avoid suspicion of plagiarism. If you are unsure how to properly cite, you can use the web to find references on scientific citations, or ask fellow students and TAs on Piazza.

For each programming assignment, we will use software to check for plagiarized code.

Note: You *must* set permissions on any homework assignments so that they are readable only by you. You may get reprimanded for facilitating cheating if you do not follow this rule.