2022 CSEE 4119: Computer Networks
Project 1: Video CDN

Deadlines and updates

Google Cloud Setup Due: September 30, 11:59pm
Preliminary Stage Due: October 14, 11:59pm
Final Stage Due: November 8, 11:59pm (projects are normally due on Friday, but we are giving you the long weekend)
Updates:
2022/09/26: None yet
2022/10/22: What to Submit

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.

We will do this in 3 stages:

1. Due **September 30, 11:59pm**. Set up your environment: BEFORE DEVELOPING YOUR PROXY, you must set up your Google cloud account and request access to the class virtual machine image using the process described in 'Setting up Google Cloud'. **This should be the first thing you do!** (You don't even need to read the rest of this document first.)
2. Due **October 14, 11:59pm**: Preliminary stage: building a simple proxy
3. Due **November 8, 11:59pm**: Final stage: building a proxy that implements adaptive bitrate streaming

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 *netsim* (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.

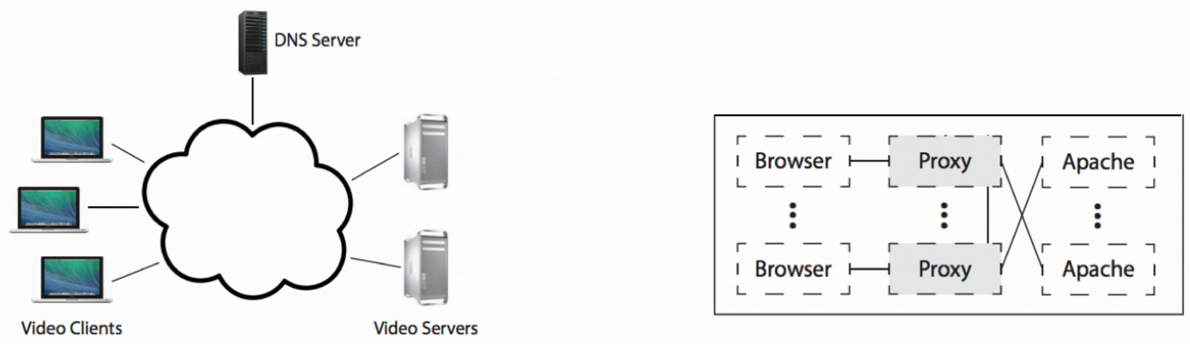


Figure 1: In the real world...

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.

Web Server. Video content will be served from an off-the-shelf web server (Apache). More detail is in the Development Environment section. 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 (plus the initial set up to get you ready for the 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.

1.4 Submission and slip days

You will submit your writeup and your proxy to Gradescope. The assignment is due on the specified due date and time. Any submission past the deadline will be subject to the late policy described in the syllabus. To avoid last minute problems (with a scanner, network, etc), please submit your assignment well in advance. Please also specify your use of slip days on Gradescope.

2. Setting up Google Cloud

Follow the tutorial (see more details in the section on the Virtual Machine) to set up your VM instance. The VM instance is where you should write and test your code. **This is the first thing you should do.** There are two stages for this tutorial:

1. Create your Google Cloud project:
    1. Claim Google Cloud credits, and create a Google Cloud project.
    2. Find your Google API service account and submit it to the form. Please do so before the due date.
2. We will grant access to an image with the project files, so you can create your VM instance from the image.
    1. We will grant access to the image to students who have submitted the form on a daily basis (so the earlier you submit the form, the earlier you can start).
    2. Once you have access to the image, you can create a properly configured VM instance.

3. Preliminary stage

The preliminary stage was meant to help you understand socket programming and prepare you for building a proxy between web browsers (clients) and video servers.

However, the requirements are different enough that you may want to use your preliminary stage code as a roadmap rather than using it as starter code. Either way, please read the preliminary stage description again, if you have trouble building a proxy.

4. 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.

NOTE: You should test your proxy within a Google Cloud VM built on the provided image. Refer to the tutorial about how to create such a VM, and **be sure to copy the codes to your own user directory by running 'cp -r /home/sy3011/csee_4119_abr_project ~/'.**

4.1 Requirements

1. *Implement basic 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. *Evaluate your basic video adaptation function.* Running the dumbbell topology (via netsim) will 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. See Sample output for details. Now
    1. (Place yourself in ~/**csee_4119_abr_project/netsim**) Start playing the video through each proxy. Run the topo1 events file and direct **netsim.py** to generate a log file: **sudo ./netsim.py -l <log-file> ../topos/topo1 run** -- after 1 minute, stop video playback and kill the proxies.
    2. (Place yourself in ~/**csee_4119_abr_project/plot**) 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>**.
    3. Repeat this process for alpha = .1, .5, .9. Generate 9 plots (from (b)) and compile them into a single PDF. In the PDF, include a brief discussion of the tradeoffs and trends when varying alpha. We are looking for a paragraph or two, at most.

We will grade based on two files: a file named **proxy** (with no extension) that **includes all your codes,** and a pdf file named **writeup.pdf** that includes all your figures and analysis.

4.2 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.

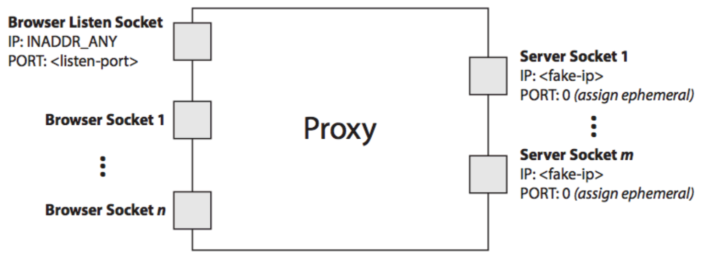


Figure 4: Your proxy should listen for browser connections on `INADDR_ANY` on the port specified on the command line. It should then connect to web servers on sockets that have been bound to the proxy's fake IP address (also specified on the command line).

4.2.1 Simple ABR Algorithm

The simple ABR algorithm looks at bandwidth available to a client, and requests bitrates that the connection can likely handle. Hence, there are two functions: throughput calculation and bitrate requests.

4.2.1.1 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 = Btf- ts

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

Tcurrent = Tnew + (1 -)Tcurrent

The constant $0 \le \alpha \le 1$ 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.

4.2.1.2 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 ".mpd" 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 labeled bitrate by modifying the HTTP request's Request-URI. Video chunk URIs are structured as follows:

/path/to/video/bunny_<bitrate_label>bps/BigBuckBunny_6s<num>

For example, suppose the player requests chunk 2 of the video Big Buck Bunny at bitrate label 45514:

/path/to/video/bunny_45514bps/BigBuckBunny_6s2

To switch to a higher bitrate, e.g., bitrate label 1006743, the proxy should modify the URI like this:

/path/to/video/bunny_10067431bps/BigBuckBunny_6s2

IMPORTANT: When the video player requests **big_buck_bunny_6s.mpd**, you should instead return **big_buck_bunny_6s_nolist.mpd**, which is also stored in servers. The latter does not list the available bitrates (actually, it only lists a single bitrate label, 1000, therefore you should expect the browser to always send you queries with 1000 as the desired bitrate label for any sequence). Your proxy should, however, fetch **big_buck_bunny.mpd** for itself (i.e., don't return it to the client) so you can parse the list of available encodings as described above. **Don't hardcode the bitrates!**

Hint: You can find the path to video by reading section 5.4 carefully. To better understand how a video player requests video chunks, point the Firefox browser in your VM instance to http://localhost, right click on the page, and choose **Inspect > Network** and take a look at HTTP GET requests.

4.2.2 Logging

We require that your proxy create a log of its activity in a very particular format for scoring and graphing. After each request, your proxy 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.

actual bitrate: The *actual* bitrate your proxy requested for this chunk in Kbps (NOT LABELED).

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).

4.2.3 Running the Proxy

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

```
cd ~/csee_4119_abr_project/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]. Use 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)

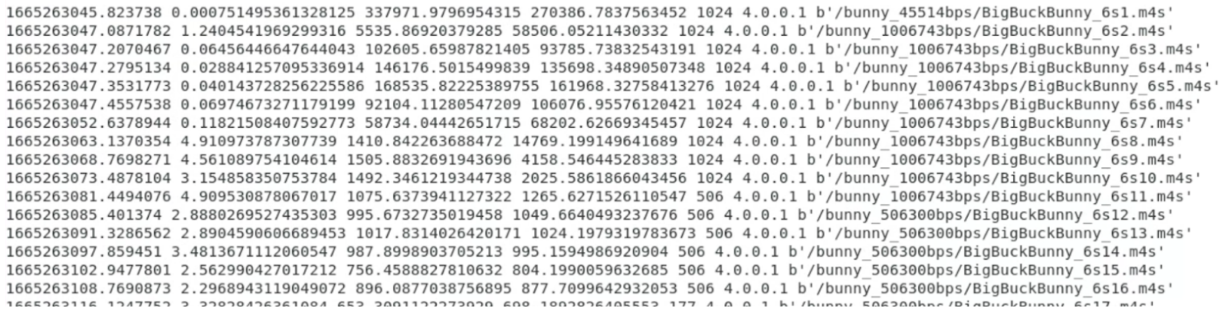
web-server-ip: Your proxy should accept an argument specifying the IP address of the web server from which it should request video chunks. It can only be one of the servers' IP addresses under the network topology you specified (see Network Simulation).

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.)

4.2.4 Sample output

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

1. Place yourself in ~/**csee_4119_abr_project/netsim**. Start netsim of topology 1 by running (make sure to run with sudo to start)
   sudo ./netsim.py ../topos/topo1 start
2. Place yourself in ~/**csee_4119_abr_project/proxy**. Start your proxy server by running
   ./proxy log1.txt 0.5 8000 1.0.0.1 4.0.0.1



```
1665263045.823738 0.000751495361328125 337971.9796954315 270386.7837563452 1024 4.0.0.1 b'/bunny_45514bps/BigBuckBunny_6s1.m4s'
1665263047.0871782 1.2404541969299316 5535.86920379285 58506.05211430332 1024 4.0.0.1 b'/bunny_1006743bps/BigBuckBunny_6s2.m4s'
1665263047.2070467 0.06456446647644043 102605.65987821405 93785.73832543191 1024 4.0.0.1 b'/bunny_1006743bps/BigBuckBunny_6s3.m4s'
1665263047.2795134 0.028841257095336914 146176.5015499839 135698.34890507348 1024 4.0.0.1 b'/bunny_1006743bps/BigBuckBunny_6s4.m4s'
1665263047.3531773 0.040143728256225586 168535.82225389755 161968.32758413276 1024 4.0.0.1 b'/bunny_1006743bps/BigBuckBunny_6s5.m4s'
1665263047.4557538 0.06974673271179199 92104.11280547209 106076.95576120421 1024 4.0.0.1 b'/bunny_1006743bps/BigBuckBunny_6s6.m4s'
1665263052.6378944 0.11821508407592773 58734.04442651715 68202.62669345457 1024 4.0.0.1 b'/bunny_1006743bps/BigBuckBunny_6s7.m4s'
1665263063.1370354 4.910973787307739 1410.842263688472 14769.199149641689 1024 4.0.0.1 b'/bunny_1006743bps/BigBuckBunny_6s8.m4s'
1665263068.7698271 4.561089754104614 1505.8832691943696 4158.546445283833 1024 4.0.0.1 b'/bunny_1006743bps/BigBuckBunny_6s9.m4s'
1665263073.4878104 3.154858350753784 1492.3461219344738 2025.5861866043456 1024 4.0.0.1 b'/bunny_1006743bps/BigBuckBunny_6s10.m4s'
1665263081.4494076 4.909530878067017 1075.6373941127322 1265.6271526110547 506 4.0.0.1 b'/bunny_1006743bps/BigBuckBunny_6s11.m4s'
1665263085.401374 2.8880269527435303 995.6732735019458 1049.6640493237676 506 4.0.0.1 b'/bunny_506300bps/BigBuckBunny_6s12.m4s'
1665263091.3286562 2.8904590606689453 1017.8314026420171 1024.1979319783673 506 4.0.0.1 b'/bunny_506300bps/BigBuckBunny_6s13.m4s'
1665263097.859451 3.4813671112060547 987.8998903705213 995.1594986920904 506 4.0.0.1 b'/bunny_506300bps/BigBuckBunny_6s14.m4s'
1665263102.9477801 2.562990427017212 756.4588827810632 804.1990059632685 506 4.0.0.1 b'/bunny_506300bps/BigBuckBunny_6s15.m4s'
1665263108.7690873 2.2968943119049072 896.0877038756895 877.7099642932053 506 4.0.0.1 b'/bunny_506300bps/BigBuckBunny_6s16.m4s'
1665263116.1247752 3.32828426361084 653.3091122273929 698.1892826405553 177 4.0.0.1 b'/bunny_506300bps/BigBuckBunny_6s17.m4s'
```

(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
sudo ./netsim.py -l netsim_log.txt ../topos/topo1 run

You may run **./proxy log2.txt 0.5 8001 2.0.0.1 3.0.0.1** along with step 2 and point the web browser also to http://localhost:8001 to initiate and to use a second proxy.

4.3 What to Submit for the final stage

You will submit two files to two questions on Gradescope separately.

- Submit **writeup.pdf** to Q2 Writeup - This should contain the plots and analysis described in Requirements.
- Submit **proxy** to Q3 Proxy - Please add '#!/usr/bin/env python3.10' to the top of your Python file so that we can easily make it executable. (Please add exactly the same characters. We won't be able to execute the file if there is an extra space or slash. Please also notice that the name is proxy, not proxy.py. **All your codes must be in this single file.**)

4.4 Where to Submit

You will submit your code to Gradescope**.** You will be asked how many slip days you would like to use for the final stage. 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.

4.5 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, throughput estimators, and ABR algorithms as described in Implementation Details.

4.6 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. You may have to 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. **Even when using incognito mode, your browser will still build up a cache, if you don't close the browser.**

5 Development Environment

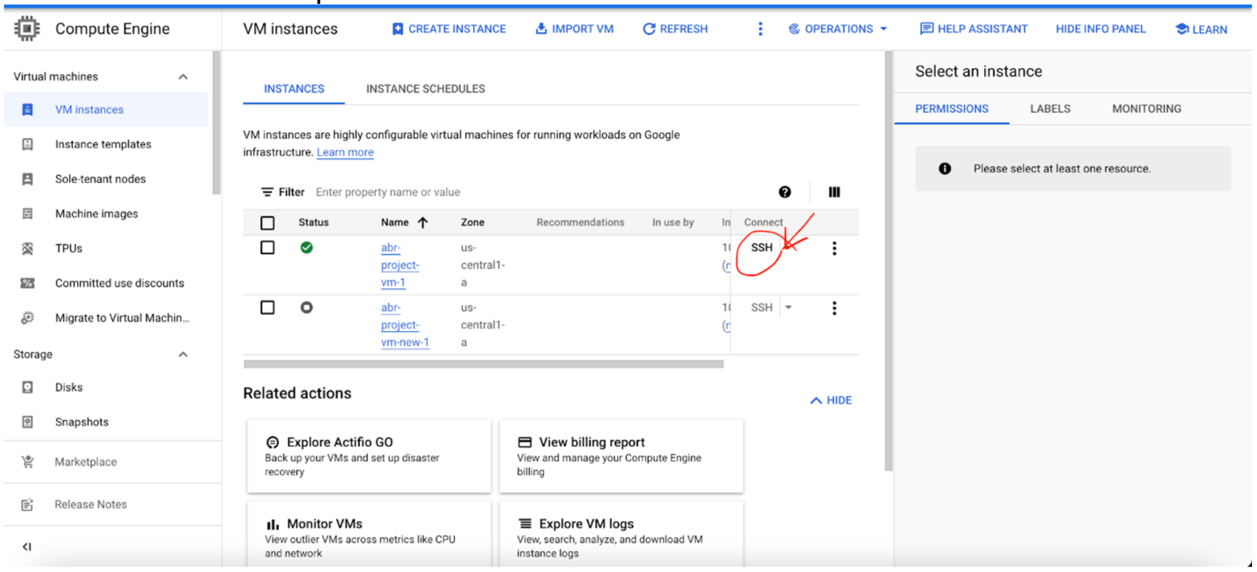
For the project, we are providing a virtual machine (VM) pre-configured with the software you will need. We strongly recommend that you do all development and testing in this VM; your code must run correctly on this image as we will be using it for grading. For example, some students in previous years 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.

5.1 Virtual Machine

We provide an image on GCP (Google Cloud Platform) and you can create a virtual machine (VM) instance based on it. Please follow the tutorial to set up your VM instance and do all your testing there.

In the event you need to move files between the VM and your computer (e.g., for submission), there are a few options.

1. (*recommended*) Create a (private) Github repository with the relevant project files. You can push to your repository from the VM and access the files from anywhere.
2. If you SSH into your VM instance using the GCP default option (the SSH button on the same row of the instance), on the top of your SSH window, there are two arrow buttons which upload and download files.



VM instances are highly configurable virtual machines for running workloads on Google infrastructure. Learn more

	Status	Name	Zone	Recommendations	In use by	In	Connect
☐	✓	abr-project-vm-1	us-central1-a			1((r	SSH
☐	●	abr-project-vm-new-1	us-central1-a			1((r	SSH

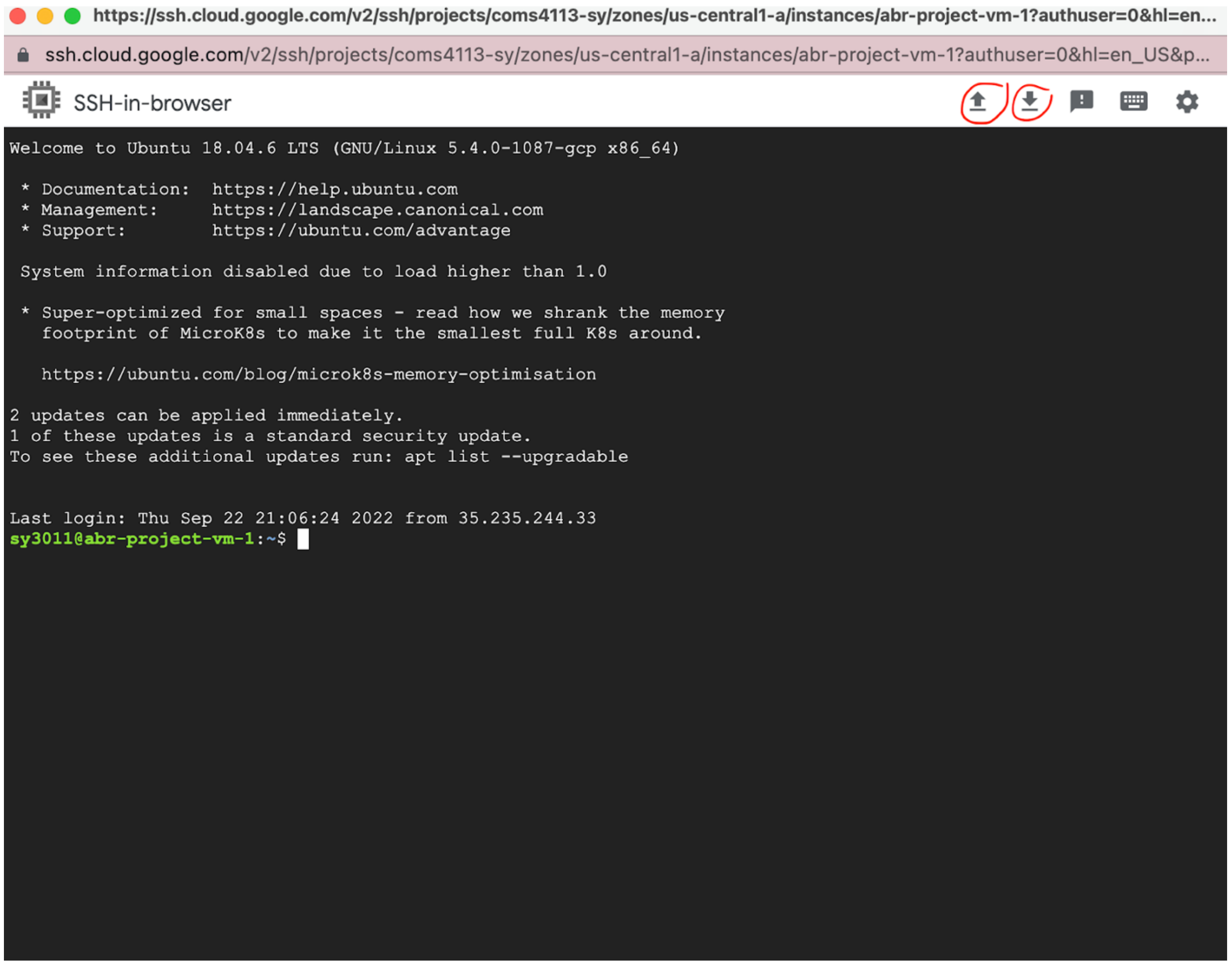
Explore Actifio GO
Back up your VMs and set up disaster recovery

View billing report
View and manage your Compute Engine billing

Monitor VMs
View outlier VMs across metrics like CPU and network

Explore VM logs
View, search, analyze, and download VM instance logs

Please select at least one resource.


https://ssh.cloud.google.com/v2/ssh/projects/coms4113-sy/zones/us-central1-a/instances/abr-project-vm-1?authuser=0&hl=en...
ssh.cloud.google.com/v2/ssh/projects/coms4113-sy/zones/us-central1-a/instances/abr-project-vm-1?authuser=0&hl=en_US&p...
SSH-in-browser

```
Welcome to Ubuntu 18.04.6 LTS (GNU/Linux 5.4.0-1087-gcp x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 System information disabled due to load higher than 1.0

 * Super-optimized for small spaces - read how we shrank the memory
   footprint of MicroK8s to make it the smallest full K8s around.

   https://ubuntu.com/blog/microk8s-memory-optimisation

2 updates can be applied immediately.
1 of these updates is a standard security update.
To see these additional updates run: apt list --upgradable


Last login: Thu Sep 22 21:06:24 2022 from 35.235.244.33
sy3011@abr-project-vm-1:~$
```

3. Launch your remote desktop. You can use sidebar options to upload and download files. Alternatively, send the files via the Internet (email, Google Drive). Firefox is installed on the VM.

5.2 Starter Files *(for final stage)*

After following the directions to copy the starter files, you will find the following files in **~/csee_4119_abr_project/.** We may update the files as we release bug fixes, which we will announce on ED. Assuming these fixes are completely contained to this directory (as opposed to the image), we will release them via Git. Use the command ‘git pull origin master’ to fetch changes.

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.

plot

grapher.py A script to produce plots of link utilization, fairness, and smoothness from log files. (See Requirements.)

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 (same as topo1, just with different available bandwidth)

5.3 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 (make sure to run with sudo):

sudo ./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:

sudo ./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:

sudo ./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 even when **netsim.py** finishes running.

5.4 Apache

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/html**; we have put sample video chunks here for you.

5.5 Programming Language and Packages

This project must be implemented in **Python 3.10.** Your VM instance comes with Python version 3.10.7, which is the version we will use to test your code. If this choice of language poses a significant problem for you (i.e., you have never used Python before), please contact the instructors.

To run python 3.10.7 on VM instance, please use the following command:

```
python3.10
```

To install python packages for version 3.10.7 on VM instance, please use:

```
python3.10 -m pip
```

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

sys, socket, threading, select, time, re, numpy

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 Ed Discussion ***and*** a TA or the professor explicitly responds to your request approving the use. We will maintain a pinned Ed 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 Ed 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.

6 Grading

Your grade will consist of the following components:

VM setup (2 points)

Proxy (68 points)

- Preliminary stage proxying [15 pts]
- Final stage proxying runs on browser [10 pts]
- Final stage proxy - implementing EWMA throughput estimator & bitrate adaptation [43 pts]
- Code executes as instructed [-10 pts if we have to manually debug things]

Writeup (20 points)

- Plots of utilization, fairness, and smoothness for {0.1, 0.5, 0.9}[10 pts]
- Discussion of tradeoffs for varying [10 pts]

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 Ed) 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.

This assignment 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 or ask people who took the class in previous years for help. 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. Helping other students violate the policy (for example, letting them look at your code) is a violation, even if you completed the code yourself. Please do not publish your code or make it available to future students -- for example, please do not make your code visible on Github. Uploading course materials to sites such as CourseHero, Chegg or Github is academic misconduct at Columbia (see pg 10).

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 Ed.

You are not allowed to use GitHub Copilot, Kite, or similar tools. **Using AI-assisted coding tools (such as GitHub Copilot) is considered academic dishonesty**. Since "AI-assisted" can be somewhat vague, I will clarify that "AI-assisted tools" includes, but is not limited to, any coding tools that require access to the internet or more than 2MB of data for semantic analysis (we are setting the size bar rather high because the GCC executable is inexplicably large; the intention is that you should not be performing analyses using a sizable dataset).

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.