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:
2019-09-21 None yet

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.

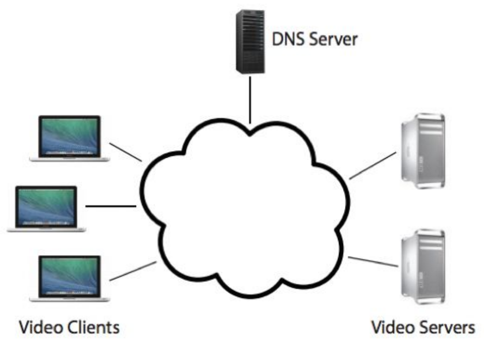


Figure 1: In the real world...

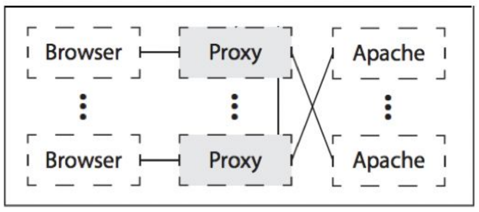


Figure 2: Your system.

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.

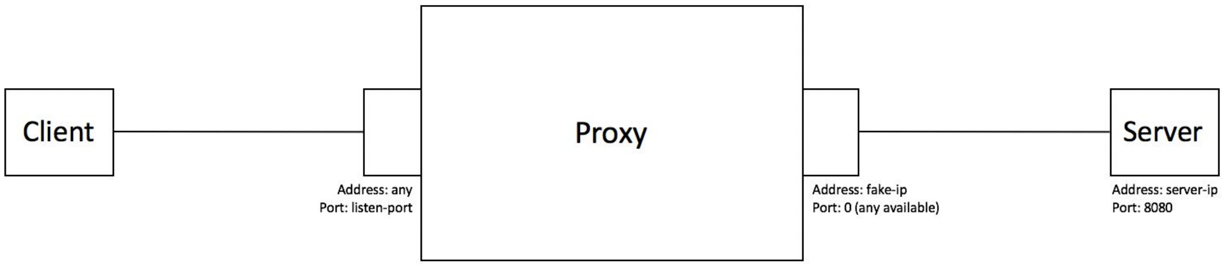
Your proxy should accept connections from clients and then open up another connection with a server (see How to run the proxy). Once both connections are established, the proxy should forward messages between the client and server.

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.

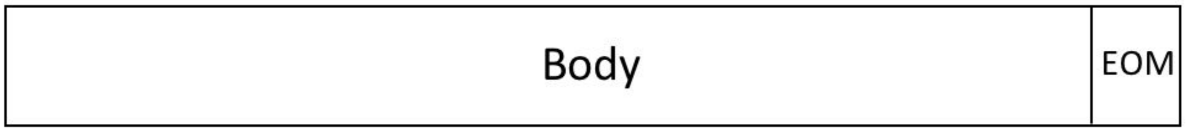
b. Establish a connection with a server:
Once the proxy gets connected to the client, it should then connect to the server. The server IP is provided as a command line argument. As for the port number, use 8080. Make sure to close connections to the client and server when either of them disconnects.



(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.

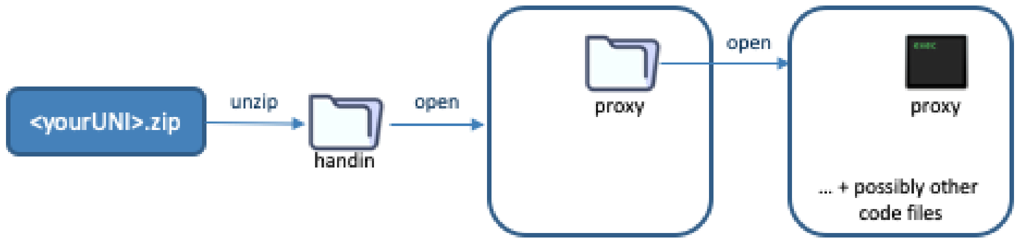
Note that this version simply forwards messages with the structure specified in Figure 2. The next stage will have a different message structure: the HTTP message structure, and you'll have to adapt your proxy based on your knowledge of HTTP messages.

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

To be released later.

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.