# CS655 GENI Mini Project——Web Caching

Team members:
Yujing Chen (U70567267)
Mingyan Yang (U30215865)
Yicheng Li (U29503597)
Zilin Zhang (U87038789)

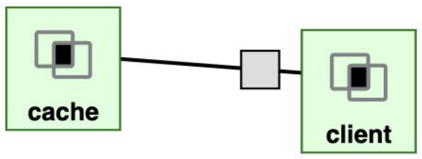
**GitHub link:** https://github.com/Yicli0512/CS655-GENI-Mini-Project.git
**GENI slice name:** project-WebCaching

## 1. Introduction / Problem Statement

To deeply understand the cache, our goal is to design an experiment on GENI nodes where one can show the benefit of having a web cache, automate the process of retrieving objects through the cache and without the cache. We mainly do our experiment on showing the cache can help increase the performance of the HTTP GET request. We explored some factors which relate to the latency(delay, loss rate and bandwidth). And we will send 50 requests to http://www.google.com and cache should help shorten latency. Also, study different aspects of web caching, e.g. max-age, expires headers and recent research work on caching(xCache, RFC).

## 2. Experimental Methodology

We use apache traffic server as the cache and direct requests to a cache. Using ATS, the client will send requests to the cache server. Then the cache will check. If the cache hits, it will get the request objects locally and return them to the client. If the cache misses, it will send the request to the requested server, get the objects and cache them.
First, we came up with the following network topology:



**Part A:** When we were exploring the performance without web cache, we made the client send 50 requests to http://www.google.com under different delays, loss rates and bandwidth. We recorded the total latencies of these requests. Also, we plot graphs to show our results and compare the results with or without cache.
**Part B:** When we were exploring the performance with web cache, the client will use the cache server as a proxy. It means all requests will be forwarded to the cache server first. We also send 50 requests to http://www.google.com under different delays, loss rates and bandwidth(the value is the same as part A). We also recorded the total latencies and compared with the results of Part A.

For both parts, the values of different delays are 0ms, 100ms, 200ms, 300ms, 400ms, 500ms. The values of different loss rates are 0, 0.02, 0.04, 0.06, 0.08, 0.1. The values of different bandwidths are 100kbit, 150kbit, 200kbit, 250kbit, 300kbit. And we write a script for both parts to automate the process of changing the value of factors, retrieving objects through cache and without cache.
(For the delay, we assume that the delay is mainly the delay between cache server and address that we request, the delay between client and cache server can be ignored.)

## 3. Results

### 3.1 Usage Instructions

See README file for detailed instructions. (For the experiment with cache, it needs to edit one file on the cache node and input the command on two nodes alternatively, so we can not put all the commands into one .sh file. Just follow our instructions.)
Please test under a good network environment.

**Part A**

In the Client node, just run *python withoutCache.py* in the root directory. This program would automatically set up different delays, loss rates and bandwidths and plot three graphs to describe the performance of the client.

**Part B**

In the Cache node, go to the traffic server folder and run traffic server:

```
cd /usr/bin
sudo ./traffic_server start
```

If you cannot acquire lock file, run the following command:

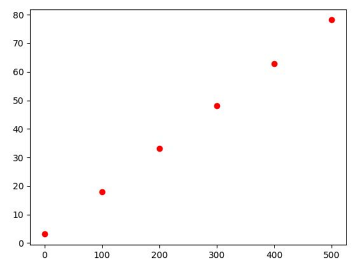
```
cd /var/run/trafficserver/
sudo rm server.lock
```

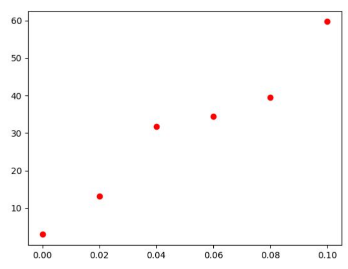
Then go back to the client node and run *python withCache.py.* This program would automatically set up different delays, loss rates and bandwidths and plot three graphs to describe the performance of the client.

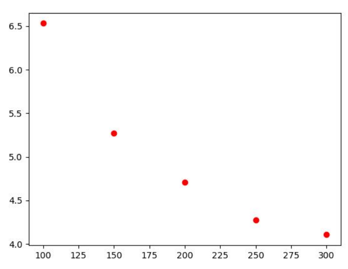
### 3.2 Analysis

**Part A**

Without using Cache, the total latency to finish the requests would increase linearly as the delay grows. The total time would increase as the loss rate increases generally. The total time would decrease as the bandwidth increases.



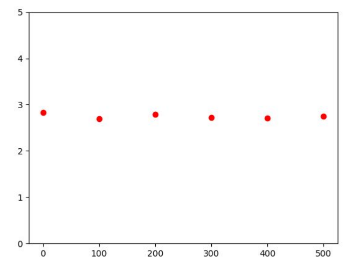


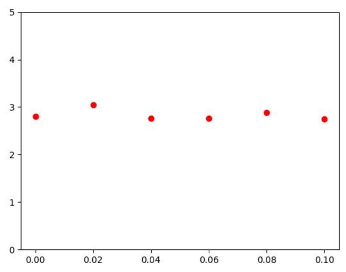


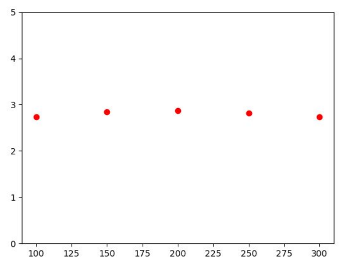
Total time(seconds) VS delays(ms)        Total time(seconds) VS loss rate        Total time(seconds) VS bandwidth(kbits)

**Part B**

Using cache technology, the total latency will not be affected by the change of delays, loss rates and the bandwidth. The request objects are cached in the cache and we do not need to send requests to the outer network from the cache server. We can see the total time is stable and stays in a reasonably small range. Basically, we can tell that the total time is a constant number.







Total time(seconds) VS delays(ms)  Total time(seconds) VS loss rate  Total time(seconds) VS bandwidth(kbits)

## 4. Conclusion

Our result shows that web cache can significantly increase the performance of HTTP GET requests. Without cache, different delays, different loss rates will make the latency longer with larger value and smaller bandwidth will also make the latency longer. However, the latency is stable with cache and the total latency has been much shortened. Also, it reduces network traffic since much traffic is handled in the local network and we do not go through the out network due to the cache.

For further work, we can explore more factors which affect the latency without cache and compare the experiment results with the results with cache.

## 5. Study of Web Caching

### 5.1 Headers related to the caching

The **expires header** specifies a fixed date/time for the expiration of a cached resource. For example, Expires: Sat, 13 May 2017 07:00:00 GMT signals that the cached resource expires on May 13, 2017 at 7:00 am GMT. The expires header is ignored when a cache-control header containing a max-age directive is present. The **max-age** request directive defines, in seconds, the amount of time it takes for a cached copy of a resource to expire. After expiring, a browser must refresh its version of the resource by sending another request to a server. There are also other contents. The **no-cache** directive means that a browser may cache a response. The **no-store** directive means browsers aren’t allowed to cache a response and must pull it from the server each time it’s requested. The **public** response directive indicates that a resource can be cached by any cache. The **private** response directive indicates that a resource is user specific, it can still be cached, but only on a client device.The **vary** header determines the responses that must match a cached resource for it to be considered valid.

### 5.2 xCache

xCache is a cloud-managed distributed web caching architecture that proactively manages the cacheable web content at the extreme edge of slow networks, which caused by developing contexts that normally experience high end-to-end server latencies, have limited edge bandwidth and face poor performance due to the complexity of web pages. The xCache architecture consists of a set of Edge Caches(ECs) in close proximity to the users. It functions as ordinary cache but they are centrally monitored and managed by a Cloud Controller(CC) and are periodically updated with web objects.  Also, they gather access logs and periodically send the aggregated statistics to the CC. CC optimizes the content of each EC by maximizing the EC's hit rates while minimizing the bandwidth used to update them. The CC has two key components for this, Page Profiler, which determines the list of active pages to profile and to learn the object-group representation of each active page  and Cache Manager, which maintains the content of ECs. The paper described more details but it's difficult to discuss all here. Also, the paper has provided some evaluation of the performance and compares with the conventional cache in some aspects.

**5.3 RFC**

Request For Comments (RFC) is a series of documents scheduled by number. The file collects information about the Internet, as well as software files for UNIX and the Internet community.
RFC 7324 is an Internet Standards Track document about caching of Hypertext Transfer Protocol (HTTP/1.1). This document defines HTTP caches and the associated header fields that control cache behavior or indicate cacheable response messages. This document also standardized error handling, how to store responses in caches,  how to construct responses from caches and header field definitions, which can be divided into Age, Cache-Control, Expires, Pragma and Warning. It also contains IANA considerations and security considerations. It is a useful guidance for people who want to understand the cache deeply.

**6. Division of Labor**

Zilin Zhang: The design of network topology and configurations, the testing part and data collection part.
Mingyan Yang: The design of network topology and configurations, the testing part and data collection part.
Yujing Chen: Mainly finished the coding part. Study aspects of web caching.
Yicheng Li: Mainly finished the coding part. Study aspects of web caching.
The report is finished by all of us.