Programming Project
CS 165, Fall 2020
Due: December 10, 2020

1 Overview

You will work on this project in pairs. You are to implement a secure proxy application which uses the TLS protocol to provide simple authentication and secure file transmission. Your program is to allow a set of clients to interact with a group of proxy caches to securely retrieve the desired file from a single remote server using a *consistent hashing* scheme and Bloom filters (see below).

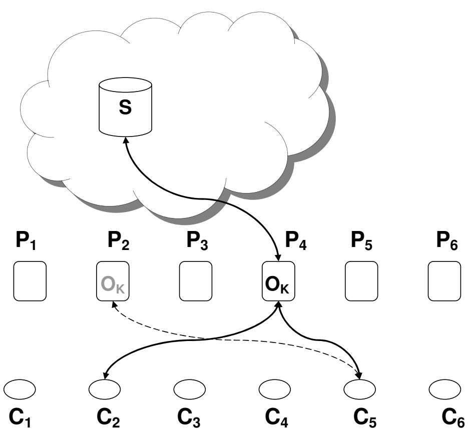

Figure 1: Highest Random Weight cache selection

The system consists of a set of clients $C_1, C_2, \ldots, C_n$ each of which requests objects stored in a server S. To reduce both server load and client latency, objects are fetched from S and cached in a collection $P_1, P_2, \ldots, P_m$ of *proxy caches.* A client does not go to the server S directly, but requests it from a proxy, which is selected using the *Rendezvous Hashing* method.

Some objects are *black-listed* as security risks by local administrative policy, and may not be accessed by clients. Each proxy maintains (1) a local cache of objects, and (2) a way to check for *black-listed objects.* When a client requests an object (say, O_k), the proxy first checks if O_k is black-listed. If it is black-listed, it refuses the request. Otherwise, it checks its local cache for O_k. If O_k is in the cache, it returns O_k to the client. Otherwise, it fetches O_k from the server S, caches it locally, and returns it to the client.

1.1 Selecting Proxies Using Rendezvous Hashing

As in Figure 1, let $P_1, P_2, \ldots, P_6$ be the proxies retrieving and caching objects from a server S. Say clients C_1, C_2, and C_5 all want the same object O_k. If C_1 asked P_3 for O_k, C_2 asked P_4, and C_5 asked P_2, there would be misses at P_2, P_3, and P_4, and O_k would be retrieved from S by all three proxies and cached locally, wasting both time and space.

However, if C_1, C_2 and C_3 were to ask for O_k from the same proxy, say, P_4, the first request for O_k would fetch it from S, and cache it at P_4. Later requests would find O_k at P_4. A strategy that allows all clients pick the same server for a given object is called a *consistent hashing* scheme.

You are to use the *Rendezvous Hashing* (Highest Random Weight) scheme, which works as follows. Say we have m proxies $P_1, P_2, \ldots, P_m$. A client that wants object O proceeds as follows:

1. Concatenate the object name O with each proxy name $P_1, P_2, \ldots, P_m$ to get the m strings $s_1, s_2, \ldots, s_m$, where $s_1 = O.P_1,\ s_2 = O.P_2,\ \ldots,\ s_m = O.P_m$.
2. Hash these strings $s_1, s_2, \ldots, s_m$ to get m hash values $h(s_1), h(s_2), \ldots, h(s_m)$.
3. Pick the highest hash value $h(s_k)$. Request object O from the proxy P_k that yielded $h(s_k)$.

All clients see the same proxy names $P_1, P_2, \ldots, P_m$, so they all agree on the same P_k for O.

1.2 Identifying Forbidden Objects Using Bloom Filters

Bloom filters are probabilistic data structures used for approximate set membership queries. A Bloom filter F consists of m 1-bit cells and k hash functions $h_1, h_2, \ldots, h_k$. All cells are initially zero. Two operations are allowed:

- To *insert* an item x to F, we set the k cells at indexes $h_1(x), h_2(x), \ldots, h_k(x)$.
- To *query* if an item y is present in the set of elements that have been inserted in F, we check cells at indexes $h_1(y), h_2(y), \ldots, h_k(y)$. If all these cells are set, the element is *probably* present in F, otherwise, the element is definitely not in F.

Note that due to hash collisions, Bloom filters may have *false positives*, i.e. a query may return 'yes' even for elements that are not present in F. However, no false negatives are possible. If a query returns 'no', the element is definitely not in F.

2 Implementation details

To keep things simple, you may assume that proxies do no cache management. That is, a proxy P_i never deletes any object once it has been cached. Each proxy server P_i should maintain a Bloom filter F_i to track the black-listed files that map to proxy P_i (it need not track black-listed objects that map to other proxies).

If the object O specified in an incoming request is not black-listed, P_i should check its cache for O. If it is in the cache, P_i returns O. Otherwise, P_i retrieves O from S, caches it locally, and returns it to the client.

The starter code given to you is the same as one handed out in the labs. You are given a simple client and server that communicate in plaintext over a socket. `libtls` is included in the given repository. You are to use the `libtls` C API to make the client, the proxy servers and the remote server use TLS for all connections. Please read through the included README.md and the lab assignment for more hints on implementation details.

2.1 Client side steps

The client executes as follows.

1. Reads object names from a file, and determines which proxy to ask for each object, using Rendezvous Hashing.
2. Initiates a TLS handshake with the selected proxy. You may assume that the client already has a CA root certificate (*root.pem*) required to authenticate the server.
3. Sends the filename securely to the proxy.
4. Receives and displays the contents of the file requested.
5. Closes the connection.

The client application should be executed as follows:

```
your_application_name -port proxyportnumber filename
```

2.2 Proxy side details

Each proxy P_i executes as follows.

1. Creates a Bloom filter F_i that uses 5 hash functions. There will be no more than 30,000 forbidden objects. The false positive rate for the Bloom filter is to be no more than 1%.
2. Reads black-listed objects from a file, and enters into F_i all objects that map to P_i.
3. Waits for a client to initiate a TLS handshake.
4. Receives a request for filename from the client through the TLS connection.
5. Checks if the requested file is black-listed, using F_i. If it is black-listed, deny the request.
6. Otherwise, check the local cache for the file. If the file is present in the cache, read in the file and send it to the client over TLS.
7. If the file is not in the cache, set up a TLS connection to the server, request the filename, and store it in the cache. Then read in the file and send it to the client over TLS.
8. Closes the connection.

The proxy application should be executed as follows:

```
your_application_name -port portnumber -servername:serverportnumber
```

2.3 Server side details

The server executes as follows.

1. Wait for a proxy to initiate a TLS connection. (You may assume that all proxies already have the CA's root certificate (*root.pem*) required to authenticate the server).
2. Receives a request for filename from the proxy through the TLS connection.
3. Sends the file securely to the proxy over the TLS connection.

The server application should be executed as follows:

```
your_application_name -port portnumber
```

3 Requirements

1. All applications should:
   (a) display console messages after each step
   (b) check errors during the communication of the two parties and display appropriate message indications for the specific error identified prior, during and after the connection
2. Since you will most likely be implementing the clients, proxies and server all on the same machine please organize the information for each client, proxy and the server in a separate directory on the file system.
3. You should use C to implement your application, and your code should be clearly written and well documented. Using C++ is allowed, but please remember that calling C code from C++ and vice versa may not be straightforward due to linking issues. You may want to look up the "extern C" directive. Please write a README file with your code. You should turn in your code on iLearn.
4. Although you are allowed work in pairs to complete this project, the project should be the *original work* of the 2-person team. You may discuss the project concepts and the `libtls` library with other students, but sharing code between teams will result in you failing the assignment. We will use automated tools to check for cooperation.
5. Each team should also submit detailed information what each member of the team contributed to the project.

4 References

1. Bob Beck's libTLS tutorial.
2. LinuxConf AU 2017 slides.
3. On Certificate Authorities.
4. Official libtls documentation