Programming Project
CS 165, Spring 2020
Due: June 5, 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 servers to securely retrieve the desired file from a single remote server using a *consistent hashing* scheme and bloom filters (see below).

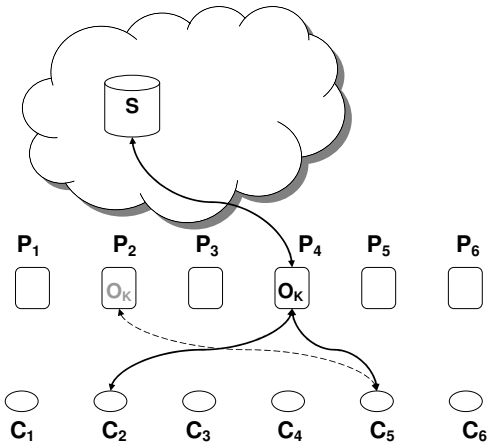

Figure 1: Highest Random Weight cache selection

1.1 Highest Random Weight Hashing

Let there be a set of proxies $P_1, P_2, \ldots, P_n$ that retrieve and cache objects from a remote server. Say that clients C_1, C_2, and C_5 all want to retrieve the same object (a file) O_k. If C_1 asked for O_k from proxy P_3, C_2 asked for it from P_4, and C_5 asked for it from P_2, there would be misses at P_2, P_3, and P_4, and O would be cached at all three proxies, wasting space. However, if C_1, C_2 and C_3 requested O_k from the same proxy, say, P_4, the first request for O would fetch it from the remote server, and cache it at P_4. Later requests would find O_k at P_4. A strategy that allows all clients to go to the same server for a given object is called a *consistent hashing* scheme.

You are to implement the Highest Random Weight (Rendezvous Hashing) scheme for consistent hashing. Clients first concatenate the object name with the proxy name for each of the n proxies, and hashing these n resulting strings to get n hash values. A client requests object O from the proxy P_k that yields the highest hash value. Since all clients see the same proxies, the highest hash value for O will result at all clients from the same proxy name.

1.2 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.

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 files it has cached locally. If the file specified in an incoming request has been seen previously, P_i should return the file from its cache. If the request is for a file never seen before, it should retrieve the file from the remote server.

2 Implementation details

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. Determines the identity of the proxy to contact to retrieve a particular file by computing a hash of the filename and the proxy name.
2. The client initiates a TLS handshake with the 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

The proxy executes as follows.

1. Waits for the client to initiate a TLS handshake.
2. Receives a request for filename from the client through the TLS connection.
3. Checks if the file has been requested in the past by querying the bloom filter for filename.
4. If the bloom filter returns 'yes', the file may have been requested in the past, and it might be present in the cache. If the file is present in the cache, read in the file and send it to the client over TLS. Then add the filename to the bloom filter.
5. If the bloom filter returns 'no' or if the file is not present in the cache, set up a TLS connection to the server , request for the filename and store it in the cache. Then read in the file and send it to the client over TLS. Finally, add the filename to the bloom filter.
6. 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 proxy 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. 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