Programming Project
CS 165, Spring 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).

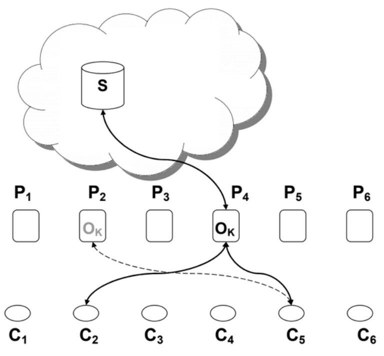

Figure 1: Highest Random Weight cache selection

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. Let us say that clients C_1, C_2, and C_3 all want to retrieve the same object (a file) O. If C_1 asked for O from proxy P_{k1}, C_2 asked for it from P_{k2}, and C_3 asked for it from P_{k3}, there would be misses at P_{k1}, P_{k2}, and P_{k3}. On the other hand, if C_1, C_2 and C_3 were to all ask for O from the same proxy, say, P_k, the first request for O would cause O to be retrieved from the remote server, and cached at P_k. Later requests would therefore see a hit at P_k for O. 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 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 to answer set approximate membership queries. A bloom filter F consists of m cells and k hash functions $h_1, h_2, \ldots, h_k$. Two operations are allowed:

- To *insert* an item x to F, we update 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 $h_1(y), h_2(y), \ldots, h_k(y)$ for updates. If at least one cell is found to be updated, the element is *probably* present in the set, otherwise, the element is definitely not.

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 the set. However, no false negatives are allowed, i.e. if a query returns 'no', the element is definitely not in the set.

Each proxy server P_i should maintain a bloom filter F_i to check if the P_i has seen a file request in the past. If the request has been fulfilled in the past, P_i should return the file from its cache, or retrieve the file from the remote server otherwise. You may assume that a proxy never deletes cached objects.

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 might 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 assignment the assignment should be the *original work* of the team. You might discuss the concepts and the libtls library with other students but sharing the code between teams will result in you failing the assignment. We will use automated tools to check for cooperation.

4 References

1. Bob Beck's libTLS tutorial.

2. LinuxConf AU 2017 slides.

3. On Certificate Authorities.

4. Official libtls documentation