Tomás' Contributions

By: Tomás Fonseca (tfonseca@kth.se)

Introduction

For this project, I focused on how the Networking would work for the Onion Routers. For this, I had to carefully examine the TOR Documentation and understand how it should be implemented for our specific project, i.e. which thinks should be kept and which should be simplified.

TOR Documentation

Based on the documentation, I understood that the main problems regarding the Onion Router were going to be the circuit management and the communication with other hosts.

According to the documentation, connections should be made used TCP, which was established with the rest of the members of the project. Additionally, TOR has many details regarding congestion control, efficient communication and also provided many commands, which we decided to filter to keep this project as concise and understandable as possible. Therefore, I filtered the information, proposed it to my colleagues and we decided on the following:

1. The Onion Routers would focus on one proxy at a time, disregarding parallelism
2. The only commands implemented: would be for circuit creation, circuit extension and TCP connection establishment with the end-point server
3. A circuit would work as a "path" for the packets to follow. So, for instance, if the proxy wants to send a packet to the server, the packet would follow the following path: Proxy -> Router1 -> Router2 -> Server. Each connection would have a circuitID (so that proxy could use router1 for many paths).

With this in mind, I began designing how the Onion Router would implement these features, which I will describe in the following section.

Onion Router implementation

Circuit Management

For this part, I implemented a Circuit class, which I elabore later, and created a dictionary which kept the relation "IP" to "Circuit Class", e.g., `{"192.168.0.3": Circuits}`.

The circuit class consists itself on another dictionary. This would keep the relation "CircuitID" to "Info", i.e. `{"circID": {"addr": outgoingAddr, "outgoingCircID": outgoingCircID}}`. The reason to keep this dictionary in a class was to also implement certain methods that would facilitate the project.

There are more methods that were created to support a larger project, however they are not used. So, to sum them up, `findAvailableCircID` would be used to find what circuit ID could be used for a host with which we already have a circuit setup (useful if we want the same router for more than one circuit, for instance), while `addNewEntry` would be used to create a new circuit with a host which which we already have a circuit. Therefore, the truly useful methods are `__init__`, which initializes the first circuit with a new host, and `addOutGoingConnection`, which will receive an existing circuit with the given host and what host the packets should be forwarded to. Consequently, if we are adding a host to which we forward the packets to, when we receive the response from the forwarded packets, we also need to forward it back. So, this function is called in pairs, so that the response can go Server -> Router2 -> Router1 -> Proxy.

The circuit class is used frequently, but the most important moments that I believe are worth stressing are:

1. When the router receives a connection creation request, and thus the circuit is initiated
2. When the router receives a circuit extension request, and so the existing circuit is "linked" to the new host and a new circuit is created for the new host
3. When the router has to forward a packet and, since the packet will got to a new sub-path of the whole circuit, the circuitID must be changed (since each connection between hosts has a unique circuitID)

General Networking

As for the networking, I made sure that the router would only process one TCP connection from a proxy at a time, although the application could be scalable. Whenever the router has to send a message to another host, say, the other router or the server, I simply open a TCP connection, send the message and then close it. To clarify, each router treats the first connection it receives as the main one which is kept during the whole program. Therefore, the first router will keep the proxy's connection as the main, while the second router will keep the first routers connection.

In regards to the packets, I also contributed for how they should be built, implementing methods which would either break a message into its components and changing important values (such as the circuitID), and also functions that would create a new packet. Likewise, I also created the logic for what should happen when a packet with a specific command was received, after it was decrypted by my colleague when it was needed.

Finally, in regards to the networking, I also attempted to make the project a bit more realistic by keeping a dictionary of streams. This is used by the last router to know to which TCP stream it should forward a packet to (much like a circuit, the same router can be used as a gateway to the same server by many proxies). However, for simplicity's sake, this was not fully implemented nor tested.