STOR

Onion Routing Service

1. Introduction

When a user sends requests to the Internet, each outgoing request contains metadata with the user's identity, location, request destination, and other information. Anyone observing the network can see this information, which can be problematic in situations where the user does not want to be associated with their requests.

2. Proposed Solution

We will be implementing STor, an implementation of onion routing. STor aims to provide fault tolerant and highly available anonymous browsing services that allow users to request static web pages. Clients will be able to browse HTTP webpages without the worry of being tracked. We will be using RPC over TCP as our main mode of communication between Routers, Coord, and Clients. Moreover, we will be using OCheck, a modified version of the FCheck UDP heartbeat service for the Coord to detect Router failures.

We have numerous Routers that will behave as the STor network where the Clients' requests will be passing through. Each message will pass through exactly three Router hops, similar to Tor. Each Router will be coordinated by a centralized node we will call Coord.

3. Implementation

The system consists of Coord, Routers, and Clients.

3.1. Coord

The Coord acts as the directory node that maintains the list of Routers currently connected and available. To simplify the design, we decided that there will only be one coord with the assumption that it will never fail.

Coord begins by initializing OCheck to begin the monitoring process. The OCheck library will return:

- **notifyCh**: informs the Coord of Router failures and their routerId so that Coord can remove it from the Router directory
- **circuitCh**: informs the Coord how many circuits each Router is currently handling, such that the Coord can update its Router load mapping. This is important for Router load balancing.

Afterwards, the Coord will be waiting for Routers to send join requests via an established RPC endpoint. Upon receiving a Router join request, Coord will store that Router's information in its directory. Also, within the join request, the Router will pass on its IP endpoint for Coord to dial into its OCheck. With the Router's IP endpoint, the Coord will register the Router with OCheck, such that OCheck can begin monitoring the new Router. After at least 3 Routers have joined, the Coord will open a new RPC endpoint to allow the Client to request relay Routers.

The relay Routers provided to the Client will be based on the Routers' active chain count to distribute the load equally. Basically, we search through the Coord's internal map containing the router's active chains, select the three Routers with the lowest active chains and return them to the Client. We decided to use active chain counts over total chain counts (which counts the total number of circuits the Router has joined in its lifetime), because a Router can be used numerous times before but has no traffic at the current moment, so it makes more sense to use active chain count. Furthermore, a newly joined Router would be overloaded if we used the total chain count.

Afterwards, Routers can join and exit the system at any time such that we will always have at least 3 Routers connected. When a Router exits the system or fails, OCheck will notify the Coord and Coord will remove said Router from its directory. We decided to remove it from the directory to prevent it from returning invalid addresses to the Client.

3.2. OCheck

This is simply the FCheck UDP heartbeat service with a few modifications to allow it to:
- Dynamically add Routers to be monitored by the Coord.
- Monitor multiple Routers at the same time.
- Send extra data in each ACKs to the monitoring node to tell it the number of active chains that the current node is a part of.

One alternative solution we had was to simply increment the chain count as we send the chain of Routers to the Client. However, we decided against this solution, because a Router failure during the Client request process would make the Coord think the Router is being used when it isn't. Also, since we had the FCheck library available from A3, we thought it would be reasonable to piggyback the active chain count data on FCheck to create OCheck. Lastly, we decided to floor the min RTT to one second to reduce the frequency of packets being sent.

3.3. Routers

The Routers act as the relay in the network to forward the Client's request to the end web server.

The Router begins by generating RSA public and private keys to exchange with the Coord. At the same time, OCheck is initialized, generating an IP address for the Coord to listen to. Next, the Router contacts the Coord with the aforementioned information to register itself to the Coord's directory. Once the exchange of information is complete, the Router is ready to accept connections from Clients to relay onionized messages.

Before a Client sends a Circuit Init request to the Router, the telescoping protocol needs to be performed - a procedure where the Client exchanges their shared keys with the Router. Depending on where in the circuit it is, the Router will receive a message from either another Router or from the Client directly. The message will contain the Client's shared key and is encrypted with the Router's public key. Subsequently, the Router will store the shared key in its cache to be used later. Afterwards, the Router will also receive an init request for the Router after it, so the Router will decrypt it using the previously stored shared key and forward the message to the next Router address listed in the message.

One problem that arose in this implementation was discerning whether the init message is encrypted with AES or RSA. A solution we came up with was to wrap the encrypted payload with a message that states whether the payload is encrypted in AES or RSA in plaintext. It may be a security risk as the encryption method is in plaintext, but we decided that the risk should be minimal considering the fact that AES and RSA are popular and reliable security standards.

To send the request, the client will send an AES encrypted blob to the first Router in the circuit which upon reception will decrypt a layer to receive the address of the next router and another AES encrypted blob. This process will continue until the final Router which after decryption will receive the final web server address and the unencrypted request. It will then send this request to the web server. When it receives a response from the web server, the final Router will encrypt it with AES and pass it back to the previous Router and continue the process until it reaches the Client.

3.3.1. Router Definitions

Guard Router: the first Router in the chain
Exit Router: the last Router in the chain

3.3.2. Failure

If the Router fails, its OCheck will send a failure notification to Coord which will lead to the Coord removing it from its directory.

Also, given a circuit of Routers 1→2→3, if Router 3 fails in the middle of circuit initialization or request relay, it will be detected by Router 2 which will create an error message and encrypt it with its AES key and pass it back to Router 1. Router 1 will encrypt it with AES again and pass it back to the client who will decrypt the entire thing and detect the error. We did this to hide the fact that there was an error from everyone but the Client.

3.3.3. Teardown

When the Client is finished with a request, it will send a teardown message to the Routers. When the Router receives the teardown message, it will clear out the information cached relevant to the Client. Afterwards, it will forward the teardown message to the subsequent Router. The message is encrypted similarly to web server requests. The AES shared keys for each router on the chain will be removed for that particular Client. Even if the communication fails, the Router will clear out the cache automatically after first contact with the Client after x amount of time, if it has not been cleared already. These two ways ensure the cache will be cleared regardless of failures.

3.4. Client

The Client will be run locally on the user's machine to intercept requests.

The Client begins by starting a web server that listens for requests made to the address "localhost:port/web_address" where

- port: the port specified in the config
- web_address: the final web server that the user wants to go to

Upon receiving a request, the Client will contact the Coord to get a set of Routers and their RSA Public Keys. After generating an AES Shared Key for each Router, it will exchange it with the Routers using the telescoping protocol described above (under Routers).

After the AES keys have been exchanged, the Client can now relay its messages. First, the message will be encrypted with Router 3's AES key, then that will be encrypted with Router 2's AES key and then Router 1's AES key to create the onionized message. This onionized message will then be sent to Router 1 for relaying.

When it gets back a response from Router 1, it will decrypt it first with Router 1's AES key, then Router 2's AES key, then Router 3's AES key to obtain and render the raw web server response.

Finally after the request is handled and rendered, the Client will send a teardown signal to the Routers to clean up resources.

3.4.1. Failures

The Client will never fail since it is local to the user's machine. If the user's machine fails, then the application will also fail; fate sharing.

If any error occurs during circuit init or message relaying between the Routers, the Client will receive an AES encrypted error and request a new set of Routers and RSA keys from the Coord to use instead

4. How the system works/doesn't work

4.1. How the system works

The user starts by launching the client, then they can connect to it via the browser as mentioned in the previous section. Once the user inputs the website they want to visit, the Client will communicate with Coord and the Routers accordingly to transmit the request across the internet. Afterwards, the user should see the web page displayed on their browser. We can guarantee that the message requests will be encrypted throughout the network except for the last hop from the exit Router to the web server. Also, we can guarantee that the IP address of the Client will be anonymized.

4.2. How the system doesn’t work

With our implementation of onion routing, the user cannot access the website they want to visit by inputting its address directly into the browser’s address bar. The user must instead connect to the Client as mentioned in the previous section.

No requests can be made through the STor network if there are less than three Routers alive.
The system does not hide the contents of the request from the destination; rather, it can only guarantee that the request cannot be easily associated with the user making it.

5. Evaluation

5.1. IP anonymity

We need to ensure and validate that the Client IP is anonymous to Routers, and most importantly the destination.

When we fetch the html page, we are logging the IP address of the perceived sender of the request from the destination web server’s perspective. Instead of the client’s IP address, it will see the exit router’s IP address, ensuring IP anonymity.

5.2. New Routers

We need to verify that new Routers that join the network are utilized.

Via our stress tests, which make multiple sequential requests on delay. During these requests, we can join Routers and validate via the OnionRingCreated traces that they are being included in the new circuits of Routers.

5.3. Fair Router usage

To ensure that traffic spreads reasonably across all Routers, Coord chooses Routers to return to the Client based on the Active Chain count (ACC). The ACC is the number of chains that the Router is currently a part of. Coord will receive this information through OCheck monitoring.

When the Client requests a chain, Coord will return the Routers that have the smallest count of ACC. We verified this behaviour by tracing: the counts for each Router right before Coord chooses a set; the set of Routers that Coord chooses; and, actions for incrementing and decrementing ACC. We observed the following:

- When Router R was chosen, all Routers with ACC less than R’s ACC were also chosen
- There were exactly 3 unique Routers per set

Note that the ACC values used are the values that Coord has access to at the moment of the Client chain request, which may not be exactly equal to the actual ACC since there may be delays in OCheck ACC update notifications.

5.4. Encryption

Request and response payloads are being traced at every Router to observe that they are encrypted. We validated this by tracing the request body in plaintext on the Exit Router and compared it to the encrypted payloads elsewhere.

5.5. Periodic circuit refresh

To prevent timing analysis attacks, every request and re-request from any Client will be met with a brand new circuit.

We traced all circuits received by the Client and verified their uniqueness.

6. Demo script

Disclaimer: 19cabfe242325b0de9d135aa8573618f289710d7 was the commit that we used in the demo. Changes after this commit are related to tracing for ShiViz and do not affect the functionality of the system.

6.1. Stage 1 - Normal operation scenarios

6.1.1. Golden path/normal operation

The golden path for our system is to retrieve a static webpage with a get request, with one Client or multiple Clients concurrently. We start by running the tracing server, Coord, Router 1 - 3. We would then have two scenarios, the first scenario is with one Client, and the second scenario is with three Clients. We would make sure that in both scenarios, every scenario will receive the requested webpage.

6.1.2. Stress tests

For the stress tests, we will issue a series of requests from all three Clients. We first start by running the tracing server, Coord, Router 1 - 6. Afterwards, we will start all three clients and run our bash scripts which will execute ten `wget <Client addr:port>/<website addr:port>`. We would make sure that all three Clients receive all ten webpages each, totalling 30 files generated.

6.2. Stage 2 - Failure scenarios

6.2.1. Base failure test

We will test the most basic case of Router failure by starting 6 Routers and failing 3 Routers immediately without any Client requests in-flight. We can verify that the system still works by sending a request after the failures and verifying that the webpage is returned successfully.

6.2.2. Guard, Relay, and Exit failure tests

We can test that our system handles failures at different points of the routing process by failing a Router during each of the three stages: circuit initialization, relaying the request and relaying the web server response. For each of these failures, we check that the Client handles the failures by requesting a new set of Routers from the Coord and that the webpage is always eventually returned successfully.

6.2.3. Orthogonal failures

We can test that the failures of Routers that do not currently directly play a role in request relays do not affect the normal operation of our system. We do this by first sending a request, then failing all Routers that are not part of the circuit, and verifying that the request is unaffected by the failures.

6.3. Stage 3 - Fairness (Handling new node joins as well)

6.3.1. Fair Router usage

Tracing will be used to verify Routers are used fairly, and that circuits have no cycles. We can test this by making concurrent requests. We first start the tracing server, then the Coord, and the three Routers, making sure that each Router has a sufficient delay (e.g., 3 seconds). Then, we start with three clients and request a static webpage from all Clients. We can use the command `wget <Client1 addr:port>/<website addr:port>` if we want to request a webpage from the first Client. Afterwards, we will start three more additional Routers, and have one of the clients perform another request to a static webpage. By the end, we should ensure that the onion rings returned are based on the correct ACC.

7. Conclusion

With an onion routing solution, IP anonymity can be achieved at the cost of increased latency. Although we were able to preserve the anonymity of the Client's IP address, we could not achieve complete privacy for the Client as there were many other channels of attacks that a malicious person could use to seek out more information regarding the Client. Including sniffing the exit Routers where messages would be received in plain-text. As well as time-analysis attacks where the attackers sniff the guard and exit Routers to figure out which requests belong to which Client based on the timing. Nevertheless, using STor is a good start towards privacy.

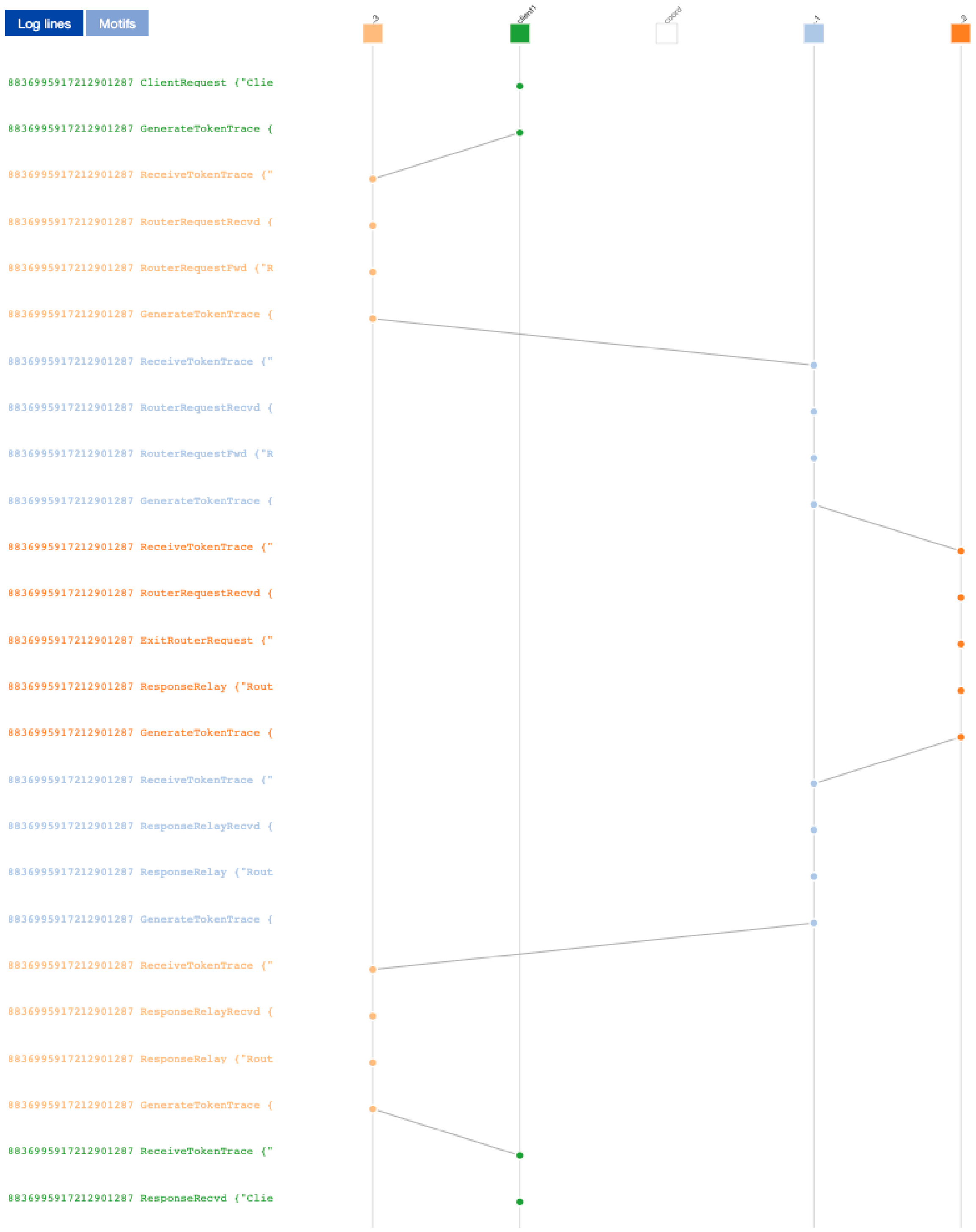
The next steps for STor if we had more time would be to decentralize the Coord/directory node to provide better fault tolerance and load balancing. We could also have the Coord select Routers that are closer to the Client to improve overall latency. To improve the user experience, the Client could be implemented as a proxy to handle more complex websites and support POST requests.

8. Reference(s)

[1] https://www.onion-router.net/Publications/tor-design.pdf

9. ShiViz

Diagram of a HTTP Request



```
8836995917212901287 ClientRequest {"Clie
8836995917212901287 GenerateTokenTrace {
8836995917212901287 ReceiveTokenTrace {"
8836995917212901287 RouterRequestRecvd {
8836995917212901287 RouterRequestFwd {"R
8836995917212901287 GenerateTokenTrace {
8836995917212901287 ReceiveTokenTrace {"
8836995917212901287 RouterRequestRecvd {
8836995917212901287 RouterRequestFwd {"R
8836995917212901287 GenerateTokenTrace {
8836995917212901287 ReceiveTokenTrace {"
8836995917212901287 RouterRequestRecvd {
8836995917212901287 ExitRouterRequest {"
8836995917212901287 ResponseRelay {"Rout
8836995917212901287 GenerateTokenTrace {
8836995917212901287 ReceiveTokenTrace {"
8836995917212901287 ResponseRelayRecvd {
8836995917212901287 ResponseRelay {"Rout
8836995917212901287 GenerateTokenTrace {
8836995917212901287 ReceiveTokenTrace {"
8836995917212901287 ResponseRelayRecvd {
8836995917212901287 ResponseRelay {"Rout
8836995917212901287 GenerateTokenTrace {
8836995917212901287 ReceiveTokenTrace {"
8836995917212901287 ResponseRecvd {"Clie
```

Nodes from left to right: Router 3, Client, Coord, Router 1, Router 2.
[The full logs can be found in the GitHub repository.]