Project Report - Final Stage

HIGHLY DEPENDABLE LOCATION TRACKER

Instituto Superior Técnico, Universidade de Lisboa

Afonso Paredes, ist189401
Rafael Poças, ist189527
Rafael Alexandre, ist189528

1 Introduction

This report complements the group 8 application for the problem proposed in the 2020/21 SEC course project - final stage.

The goal of this project was to design and implement a Highly Dependable Location Tracker. The system is composed of three types of nodes: clients, a centralized server and the Healthcare Authority (HA), which can also be regarded, to some extent, as a client node. Our algorithm is based on the course book algorithms, with some modifications specific to our problem.

2 1st Delivery improvements

In this delivery, the following improvements were made concerning phase 1 feedback:

- Inclusion of timestamps on the client to server read requests in order to prevent replay attacks.
- Verification, on the prover, of the witness' digital signature upon receiving a proof. Previously, prover would only send the incoming proofs to the server without any triage phase. On this final version, the prover first verifies the proof and only if the digital signature matches the content of the proof is the proof sent to the server replicas.
- Digital signatures included in location proof requests, adding integrity guarantees to the request itself.

3 Prover-Witness communication

The prover broadcasts a proof request containing the username and epoch of the prover. The witness that receives it will respond if the prover is close enough. The reply contains a list of proofs (each one of them uniquely targeted to one server replica of the system) that include the username of both the witness and the prover, the epoch that proof is for, and the coordinates of the witness, this last information encrypted with a session key. This session key is encrypted with the server's public key. This way, the prover will not be able to know the position of the witness, only relay it to the server. This reply also contains a digital signature to guarantee integrity, authenticity, and non-repudiation. After receiving the reply, the prover will make sure it comes from a witness that received the proof request and if it did, it will send that proof to the servers.

A man-in-the-middle could try to change some parameter in the location request (username or epoch) for his own benefit. However, all the parameters will have to be checked by the witness and, even if he does retrieve a proof, it will be deemed invalid by the server once it is submitted by the prover.

4 Client-Server communication

To ensure confidentiality, every message sent from the prover to the server will contain a session key encrypted with the server's public key and a digital signature of the message and the actual message will be encrypted with that session key.

Before even starting to send the witness proofs to the server, the prover first sends to the server replicas a location report request containing the username of the prover, the epoch and location that the prover will try to prove with the corresponding witness proofs. Each location report, with all its proofs , for a given user and epoch corresponds to a register. Each set of proofs are sent to the servers as soon as they arrives from the witness to the prover. When sending proofs to the servers, the prover also sends the encrypted secret key created by the witness.

When receiving a proof, the server will verify its digital signature to make sure the proof actually comes from the claimed witness and it will make sure it is actually for the prover that sent it and for the epoch of any location proof requested by the prover previously, a proof that reaches the server late will still eventually validate the location proof of that epoch. The server will also decrypt the witness location to make sure the witness was close enough to the prover for the proof to be valid. Once the server accepts a proof by a certain witness it will not accept any more proofs from that witness for that location report request, this prevents replay attacks.

Once the number of proofs received from the prover reaches the minimum, *fu*, the server finally accepts the location report and notify the prover.

Waiting for *fu* proofs ensures that the byzantine users could not send proofs from outside the prover's range, for example using a closer byzantine user to forward it. If we assumed that only the byzantine users in the provers range would be able to respond, then we could just wait for *fu'+1* proofs.

When the server accepts the report, the prover adds the server to its own list of servers that have acknowledged his location. Being *fs* the number of byzantine servers in the system and *3 fs+1* the total number of server replicas, it is needed a quorum of *2 fs+1* servers to have acknowledged the location report to make the prover proceed to the next epoch. This makes sure that there are always more correct processes with the write operation committed than byzantine ones, guaranteeing the system's good functioning.

Also, when receiving a location report or a valid proof the server will store it in a database, including their digital signatures, providing non-repudiation.

5 Reading operations

It is guaranteed that only the client (user or HA) that performs the read request can successfully get a response. The servers check if the given client is allowed to execute the operation by verifying the digital signature attached to message. In the case of the HA, it is the only client able to query the location of all users. Reading operations also send a random read identifier that needs to be returned in the response. This not only prevents replay attacks but also prevents late responses from previous reads to be accepted as a response for a current read.

5.1 ObtainLocationReport operation

When a client wants to obtain a user report he starts by broadcasting the read request. It is guaranteed that in a quorum of responses, at least one of them will contain the closed report with all the proofs. This means that we don't need to wait to receive the full quorum, when we receive the first valid location report with all the proofs we can move to the next phase. To guarantee atomic semantics the client then executes a write back where it broadcasts the value read and only returns after a quorum is written.

5.2 RequestMyProofs operation

This operation read from multiple registers of reports. When a client writes a proof it waits for a quorum of *2 f+1*, this means that when we receive a quorum of server responses, it is possible that it doesn't exist a server belonging to all the quorums, which means it will not have all the proofs. So we broadcast the read request and after we receive a quorum of responses we read them and get each unique (and valid) proof. A byzantine server cannot forge false proofs because they are signed by the user making the request.

5.3 ObtainUsersAtLocation operation

For the same reason as in the RequestMyProofs operation, we broadcast the read and wait for a quorum of responses. These responses now contain the reports and proofs of the users in the requested epoch and position. The client checks if the proofs are enough to prove the user report, to guarantee that the report is not forged by a byzantine server.

6 Denial of service combat mechanism

To prevent denial of service attacks, a prevention mechanism was put in place. This mechanism relies on proofs of work generated by the clients and verified on the servers. This proof of work is a number calculated by hashing incrementally the given request's data plus its timestamp until a specified number of zeros (difficulty) appear at the start of the generated hash. This task is trivial to verify on the server, but takes some effort from the client, possibly dissuading a DOS attacker. The timestamp is added to this hash in case the attacker makes a request with the same data, making the proof of work equal. By adding the timestamp, the hashed data is always different and, consequently, the proof of work. It was decided to add a proof of work to all of the client request because it was determined that all of them imposed a significant computational weight on the server that would justify the prevention of the DOS attack.

7 Conclusion

To finalize the report, it is important to mention the guarantees provided by the developed system. Regarding security, all messages between the server and the clients respect confidentiality, integrity, authenticity, non repudiation and freshness. Messages between clients respect integrity and freshness on the whole request while confidentiality is only assured on the transported locations. Having all the aforementioned mechanisms in place was necessary in order to provide a robust system against byzantine users and servers processes.