SEC project - part 3

Group 2, Alameda:
Daniel Trindade, 76349
Fernando Marques, 76419
Ricardo Abreu,76370

In this final part of the project we went from having a standalone server to multiple servers. This decision was made so that we can now tolerate (some) byzantine servers. We will refer to the number of faulty servers as f. From this f, we launch N replicas such that N = 3f + 1.
Because of this, we changed our communication protocol between the client and the replicas to behavior like a byzantine regular register (1,N).

When issuing a read or a write request from the client to the servers, we broadcast it to all N replicas. When a replica receives a write request it checks if this request is more recent than the last one he has stored for that client (does this using a timestamp) and if so updates the corresponding value. After this the replica acknowledges the request back to the client who in turn has to wait for at least (N+f)/2 acks for the write to be considered successful.
When a replica receives a read request, it returns the correct block back to the client. When reading, a client has to wait for (N+f)/2 responses and then choose the highest value from those returned as the block that must be returned as result of the read operation.
This was the standard approach to implement a byzantine (1,N) read/write regular register. We now present some optimizations that are specific for the context of our file system.
It's worth noting that our ContentBlocks are self-verifying (its id is the content's hash), which means that we do not need to wait for more than (N+f)/2 responses from the replicas when reading (like we do for the PKBlocks). Due to this fact, we simply check the first received ContentBlock against the expected one. This expected block is obtained from the PKBlock - note that when reading a PKBlock, we still wait for more than (N+f)/2 responses. This way, we can be assured that the PKBlock's data is up to date, and use it to verify the ContentBlocks id (which in turn verifies its content). Due to this optimization, we can also modify the write protocol when dealing with ContentBlocks. If we assume that only f servers are byzantine, and we only need to read 1 (correct) response from a replica when dealing with ContentBlocks, we need only to write a given ContentBlock to f+2 replicas. This minimizes the number of replicas that need to hold a given ContentBlock and as such, we can also introduce some load balancing when writing ContentBlocks so that the f+2 servers we write to are different for each execution.

The regular Register algorithm uses a perfect-links layer to support its communication between the client and server.
To implement this, we rely on the resending and non-duplication properties of the TCP protocol. To fully implement a perfect links abstraction we still needed to guarantee point-to-point authentication and message freshness. In regards to the first aspect, we have the server generate a Secret Key which is later used by itself and the clients who which to communicate with it. We use this shared secret to generate a Message Authentication Code for every message sent through the network. This MAC will assure both the integrity of the messages and the authentication of the parties involved in sending/receiving them. This is due to the fact that only the server and the clients know the secret. For simplicity we have the Client and the Server read this shared secret from a .txt file. There are no security guarantees regarding the exchange of this secret.
To guarantee message freshness, the caller inserts a challenge (128 bit random number) inside the MAC and the callee answers with that same challenge inside the response's MAC.

As said in the beggining we assume that only f replicas can be byzantine. Our implementation relies on the fact that a client can not be byzantine. If we assume a byzantine client, we can imagine a scenario where this client makes himself look like a server and the victim wouldn't notice it. He could do this because he has access to the shared secret and we provide no guarantees on the secure distribution of this secret.