**User Documentation**

**Summary**

[**1.Initialization**](#_ssvjao2daxg3)

[**2.Assumptions**](#_7zxdx1amzr03)

[**2.Description of the protocols**](#_8p7vur7mgval)

[**3.Implementation**](#_z5dkvxwkje2l)

[**4.Simulation results**](#_kp4kbivkeo6n)

# 

# 1.Initialization

Our implementation of the protocols requires at least python 3.10.

These are the following commands needed to run the algorithms implementation on the python environment:

* pip install pycryptodome
* pip install pycrypto
* pip install cryptography
* pip install pika
* pip install shamir

To run the simulation the following commands are needed:

* pip install netifaces
* pip install numpy
* pip install pandas
* pip install matplotlib
* pip install psutil
* pip install python-resources-0.3.tar.gz

# GABRIELE INSERISCI QUA LE ALTRE COSE DA CONFIGURARE PER RUNNARE LA SIMULAZIONE tipo questi comandi possono essere importati solo dopo alcuni comandi da terminale giusto?

from mininet.net import Mininet

from mininet.node import RemoteController

from mininet.node import CPULimitedHost, Host, Node

from mininet.topo import Topo

from mininet.node import OVSKernelSwitch, UserSwitch

from mininet.node import IVSSwitch

from mininet.cli import CLI

from mininet.log import setLogLevel, info

from mininet.link import TCLink, Intf

from mininet.util import ipAdd

Also it is required to install docker,rabbitmq from the official sites and running the following in the docker prompt:

* docker start
* docker run -d -p <ip\_address>:5672:5672 -p 15672:15672 rabbitmq:3.8.15-rc.2-management

The next step is to run the main\_s.py file to start the server(s) and then run the main.py file with this command line arguments:

* In the case of a non-broadcaster process:

python main.py <message to broadcast> <protocol number> <process type>

* In the case of a broadcaster process:

python main.py <protocol number> <process type>

# 2.Assumptions

In our implementation we assumed the following: -each process connects to a bootstrap server to gather its unique ID;

-the server(s) address(es) is(are) known to every process in the network;

- every process knows the ID of the broadcaster that is settled to one in our implementation;

-the broadcaster sends a message only to processes whose ids have been already received by itself before the broadcast primitive so eventually subsequent receivers that connect to the service after it has broadcasted the message will not receive it;

-the network is fully connected;

-every array that contains messages of a specific type is checked according to an interval of time(e.g every .001 s);

# 3.Description of the protocols

In this section will be stated all the four protocols implemented and considered in the simulations:

* Authenticated Links(AL):

This protocol takes advantage of the underlying authenticated link primitive between each pair of processes to exchange messages for guaranteed authenticity of the processes.The protocol specification is structured in this way:

In The first phase the sender broadcasts a message to all other processes in the network through a SEND message.

In the second phase, when a process receives a SEND message from the broadcaster, it sends to every process an ECHO message. So each process stores these messages to count them.

As soon as a process counts to (N+f)/2 +1 ECHO messages for the same message m, it enters phase three.

In the third phase each process that has received (N+f)/2+1 ECHO messages for a specific message m,broadcasts to all other processes a READY message and as soon as it receives a READY message it stores it. If a process counts to f+1 READY messages for the same message m it re-broadcasts them to all others. If a process counts to 2f+1 READY messages for the same message m and it has not already delivered that message m, it can deliver it.

* Authenticated Messages(AM):

This protocol takes advantage of the digital signature to digitally sign messages sent from a specific node for guaranteed authenticity and non-repudiation. The protocol specification is structured in this way:

1. Propose-Phase. The designated broadcaster 𝐿 with input 𝑣 sends ⟨propose, 𝑣⟩ to all parties.
2. Vote-Phase. When receiving the first proposal ⟨propose, 𝑣⟩ from the broadcaster, send a vote message for 𝑣 to all parties in the form of ⟨vote, 𝑣⟩𝑖 .
3. Commit-Phase. When receiving 𝑛 − 𝑓 signed vote messages for 𝑣, forward these vote messages to all other parties, commit 𝑣 and terminate.

* Hash-Based(HB):

This protocol takes advantage of the cryptographical hashing.In particular, the correctness of the algorithms rely on the collision-resistant property of the hash function used. Cryptographic hash functions are used widely in real-world applications.The protocol specification is structured in this way:

The source node simply sends a MSG message containing its identifier, message content m, and the sequence number h, to all the nodes. Following the convention, we assume that the source also sends the message to itself. Each node may receive five types of messages: ù

-MSG message: this must come directly from the source which contains the message content m. If the source identifier does not match the sender identifier, then the message is discarded.

– ECHO message: this message propagates information about a message already received by some node. ECHO messages contain the full content m. In the hash based algorithms, we only transmit H(m). This is the main reason that we are able to reduce bit complexity.

– ACC message: this message is used to declare to other nodes when a some node is ready to accept a message m. Again, instead of sending m with the ACC message, we send H(m).

– REQ messages: In our hash based approach, a node might not know the original message m, even after it has observed enough ACC(m) messages supporting it. Therefore, such a node needs to use REQ(H(m)) message to fetch the original message content from some non faulty node before accepting it.

-FWD messages: When a node is sent a REQ(H(m)) message, it replies with a FWD(m) message that contains the original message content of m.

* Erasure Code Based(EC):

This protocol takes advantage of the erasure coding technique that makes possible to decompose a message into shares and its reconstructions knowing a sufficient number of shares.The protocol specification is structured in this way:

EC will use [n, k] MDS code. To send a message m with sequence number h, it encodes the message m and then disseminates to each peer. The message is a tuple that contains the tag MSG, the source identifier s, corresponding coded element, and sequence number of the message h. To deal with asynchrony and failures, the algorithm is event-driven. First, upon receiving a coded element from the source, node i forwards an ECHO message along with the coded element. Second, upon receiving an ECHO message, node i decodes the message if it has received enough coded elements. The key design behind how crash-tolerant RB achieves the all-or-nothing property is that each peer needs to (pessimistically) help deliver the message to other peers.

# 4.Implementation

# 5.Simulation

# 

# 