h e p i a
Haute école du paysage, d'ingénierie
et d'architecture de Genève

Hes·SO GENÈVE
Haute Ecole Spécialisée
de Suisse occidentale

Lab 1: Developing and deploying a broadcast algorithm for a blockchain system on a Cloud infrastructure

September 29th 2021

Nabil Abdennadher

If you find a typo, no matter how small, please send an email to nabil.abdennadher@hesge.ch

We propose to design and implement a wave-based broadcast algorithm used to ensure communication in a blockchain system. The objective is not to implement a blockchain system itself, but to develop the communication layer used to send and receive transactions among the nodes that compose the blockchain system.

For the sake of simplicity, we assume that the blockchain system is composed of a set of identical "databases" which are stored on the nodes of the network. Each occurrence of the database is composed of a set of transactions. Each transaction has this structure:

1. Sender (from)
2. Receiver (to)
3. Amount (in monetary units)

The blockchain concept is simple: each node that generates or creates a transaction stores it in its database, then sends it to the other nodes, which in turn store it in their database. In case of any tempering, a comparison between versions of the same transaction stored on the nodes allows the anomaly to be detected. A transaction is assumed to be valid if the same version of this transaction is stored in a majority of the nodes.

A node can check if a given transaction is valid. For this purpose, it can send a request to all nodes and ask them if the version it holds locally is the same as the one stored on these nodes.

The blockchain system we propose to develop supports two functions:

1. ***create_transaction (trans)***: a node generates a transaction "*trans*", stores it in its database and sends it to all nodes of the network. This function relies on the broadcast by wave distributed algorithm.
2. ***rate = vote (trans)***: A node sends a request to all nodes to check if the transaction *trans*, stored locally, is the same as the ones stored in the other nodes. The result (rate) is a percentage representing the transactions which are the same as the one stored locally. This function relies on the broadcast by wave with ACK distributed algorithm. However, some adjustments are needed to implement this function.

Two additional functions will enable testing the blockchain system (they are not part of it):

1. ***fake (authentic_trans, fake_trans)***: this function, executed locally, simulates a tempering attempt. It replaces an authentic transaction (*authentic_trans*) by a fake one (*fake_trans*).
2. ***list_of_trans = list (node)***: list all transactions on a given node (local database). This function is executed locally.

There is no distributed processing in these two functions.

The goal of this Lab is twofold:

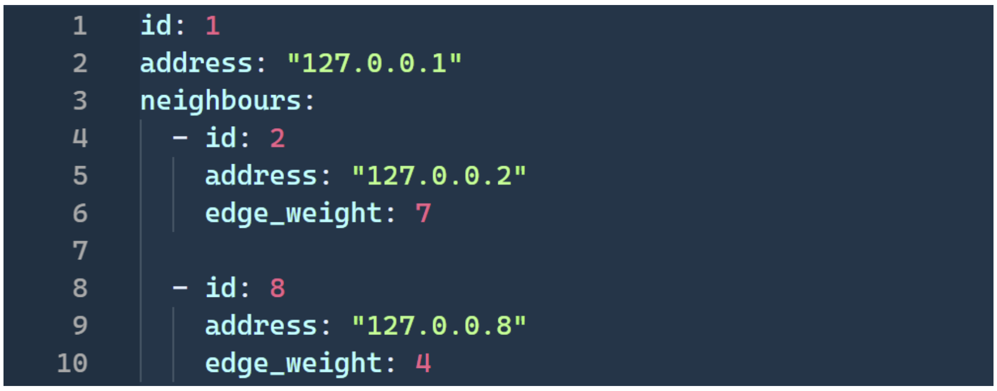
1. design and develop a socket server implementing these four functions.
2. design and develop a client program that invokes these functions. For this, the client program needs to know the IP address and the port on which the socket server is listening.

Client and server programs are different and must not be merged.

We assume that each node *x* only knows the identifiers (addresses) of its neighbours. When starting, each node *x* must read a text file: *neighbour-x.yaml*. The file contains the neighbours of the node x.

Format of the *neighbour-x.yaml* file

Each node is identified by an identifier (*id*), the port on which it listens (*port*) and a set of neighbours (*neighbours*). Each neighbour is a node which is represented by an identifier and the address/port on which it is listening. In the example below, the neighbours are deployed on the local machine. The "*edge_weight*" field is not used in this exercise.



```yaml
id: 1
address: "127.0.0.1"
neighbours:
  - id: 2
    address: "127.0.0.2"
    edge_weight: 7

  - id: 8
    address: "127.0.0.8"
    edge_weight: 4
```

Please, respect the format of this file.

Work schedule:

Steps	**Deliverable**
Step 1 1. Design of the communication protocol. 2. Data structure.	A one-page document (pdf format) describing the communication protocol and the data structure to use. Deadline: October 6th, 2021, 16h15
Step 2 Local deployment of the blockchain system. The whole system is deployed on one machine. All socket servers (nodes) are listening on the same IP but on different ports.	Two source codes (Python & Golang) + the neighbour file used to test your program (a network of minimum 10 nodes) Deadline: October 20th, 2021, 16h15
Step 3 Cloud deployment	A source code + the neighbour file used to test your program Deadline: October 27th, 2021, 16h15

Students must work in pairs. One will develop the program in Python, the other will develop in Go.

Assessment criteria

1. Client and server programs are different and must not be merged.
2. When needed, messages are sent in "parallel" (concurrence, multi-thread programming).
3. Each node must read the "neighbour" file as described above.
4. When receiving a message, the extraction of the sending node IP must be made through the functionalities offered by the sockets and not by including the IP of the sending node in the message itself.
5. The code must be optimised to run for large scale networks. This aspect will be explained in more detail by the teacher.
6. The system is deployed on a minimum of 6 nodes (represented by 6 AWS Amazon instances). This criterion is evaluated in step 3.

The validation of criteria 1, 2 and 3 is necessary for the validation of the lab.