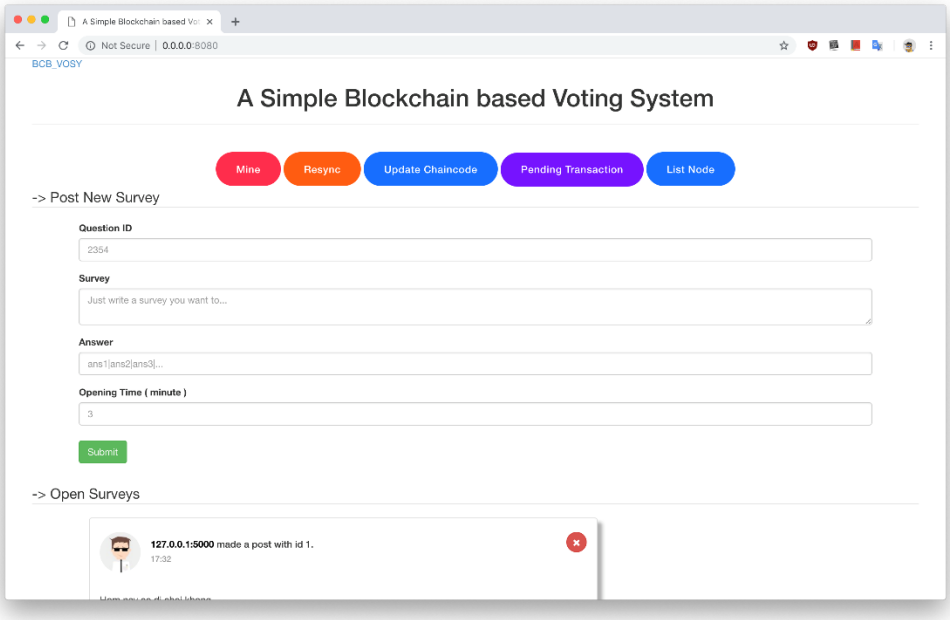
Content

Chapter 1: Setting and running Instructions

1.1. How it looks



1.2. Setting instruction

To understand, read system architecture This project can run separately by python or use docker-compose

1.2.1. Setting by Docker-compose

NOTE!!! It can only be used in linux, or maybe window. I have some problem with macOS. I use remote_addr to identify user. But every request from outside of docker having the same remote address is 172.18.0.1. It cause `network_mode: "bridge"` is default in Docker. If you have any problem with request ip address, try to uncomment `network_mode: "host"` in `docker-compose.yml`. It just works in Linux. Docker in macOS have some limited and I cannot find any good solution. If you have any idea, please report to me. Thank you.

Prerequisites

You need to install `docker` and `docker-compose` before

Running

In first machine

Follow this command:

```
docker-compose build
docker-compose up
```

You can run in background:

```
docker-compose up -d
```
You can stop this application:

```
docker-compose stop
```
Or down (delete container):

```
docker-compose down
```

In second machine

You have to provide IP address of machine 1 in `.env` file.
```
ORDERER_IP=192.168.43.162
CA_IP=192.168.43.162
PEER_IP=192.168.43.162
```
Then run command

```
docker-compose -f docker-compose-peer-only.yml build
docker-compose -f docker-compose-peer-only.yml up
```
1.2.2. Setting by Python command

Prerequisites

It needs `python`, `pip` to run. Install requirements
```
pip install -r requirements.txt
```

Running

In first machine

You need to run 4 app `orderer.py` `certificate_authority.py` `peer.py` `vosy.py` (if you don't need front-end in this machine, you don't need to run `vosy.py`) . You can run each app on different machines but need to provide ip address for it.
```
python bcb_server/orderer.py
```
Certificate authority need to know aleast 1 orderer. so if is not default value `0.0.0.0`, you need to pass orderer ip address to certificate_authority by argument `--orderer`
```
python bcb_server/certificate_authority.py
```
Peer need to know aleast 1 orderer and 1 certificate_authority so you need to pass orderer ip address and ca ip address to peer by argument `--orderer` and `--ca`
```
python bcb_server/peer.py
```
Vosy need to know aleast 1 peer so you need to pass peer ip address to vosy app by argument `--host`
```
python vosy_app/vosy.py
```
for example, with window users, ip address `0.0.0.0` is not avalible, so you need to run in localhost, so you have to follow this command in 4 cmd:
```
python bcb_server/orderer.py
python bcb_server/certificate_authority.py --orderer 127.0.0.1
python bcb_server/peer.py --orderer 127.0.0.1 --ca 127.0.0.1
python vosy_app/vosy.py --host 127.0.0.1
```

In second machine

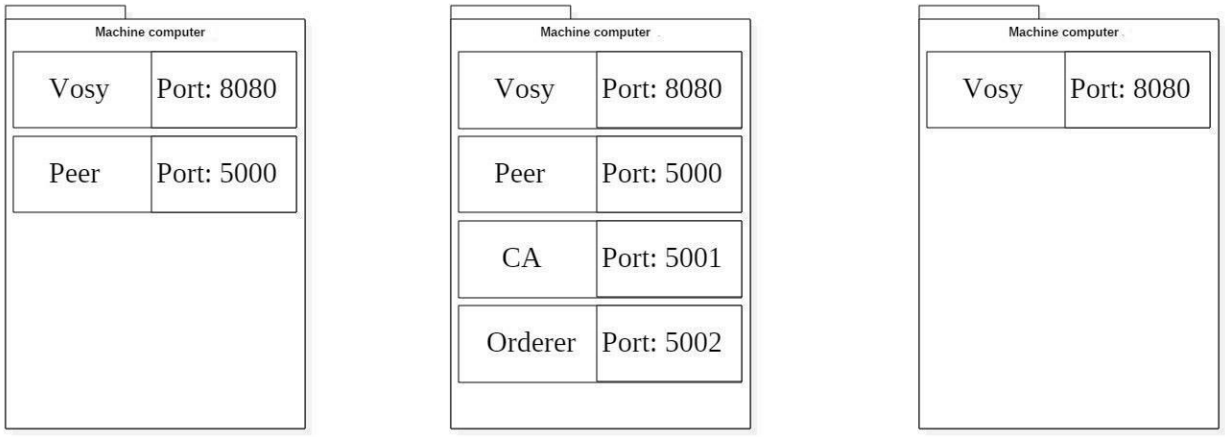
You just need to run `peer.py` and `vosy.py` but you need to provide Lan IP address `orderer.py` and `certificate_authority.py` run in machine 1. In my case, it is `192.168.43.162`

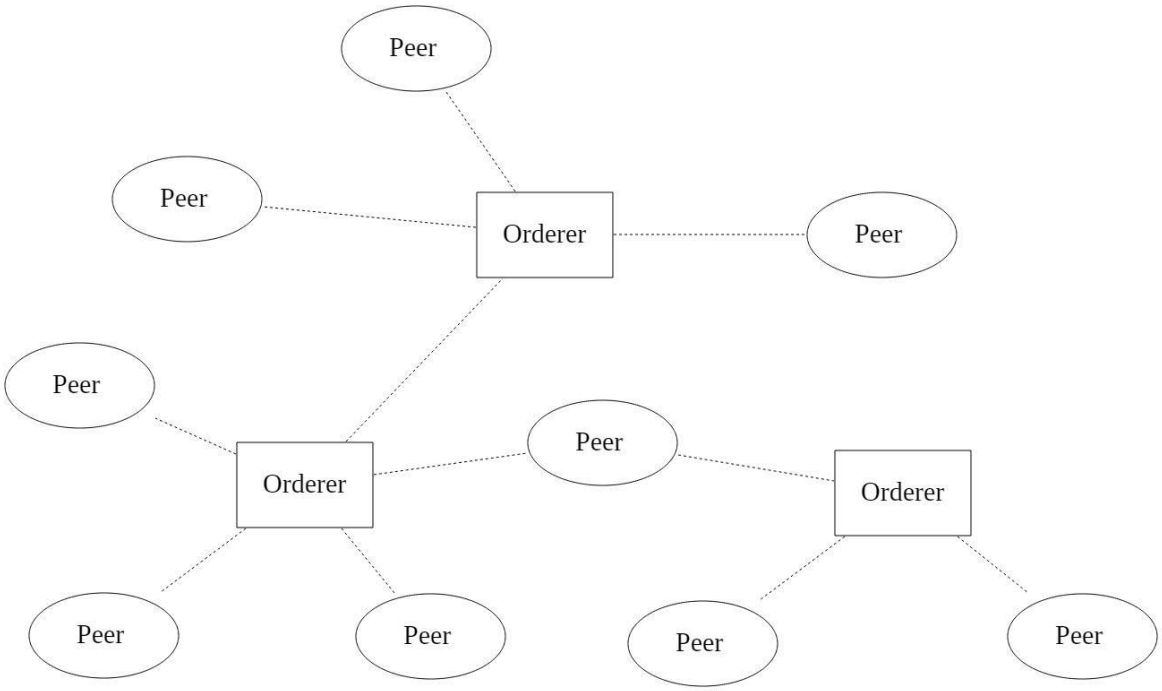
```
python bcb_server/peer.py --orderer 192.168.43.162 --ca 192.168.43.162
python vosy_app/vosy.py
```

this vosy will auto connect to local peer in address `0.0.0.0:5000`

1.3. Running Instruction

The following figures are some simple overview about the architecture of my network





Certificate Authority

It can validate connection when a node ask to join to network and Set permission for each node and validate transaction

Orderer

It can hold a list of peers and broadcast to all peer when receive a request broadcast new block or new transaction. It also have consensus method, which can return the longest blockchain in the network

Peer

It hold all data about blockchain, it have some method like mine, validate_transaction, return chain, open surveys, ...

Vosy

A blockchain-based application for voting system

How to use

- Mine : mine unconfirmed transaction
- Resync : Reload front-end
- Update Chaincode : Load smart contract from chaincode.py in vosy_app to blockchain transaction
- Pending Transaction : List unconfirmed transaction
- List Node : List node in the network

!!! Note. If you want to use chaincode, you need to click to Update Chaincode to load chaincode.py to blockchain transaction and mine it to confirm this chaincode. After that. you can create new survey (I writed a simple chaincode count_down_opening_time to auto close survey after a period of time). You can write your own contract and use it by create new execution transaction like this (You need to Update Chaincode and Mine it before use) :

```
[
  {
    "content": {
      "author": "192.168.1.38:5000",
      "contract": "count_down_opening_time",
      "argument": [opening_time, author, questionid, CONNECTED_NODE_ADDRESS],
      "timestamp": 1544369155.5413423
    },
    "timestamp": 1544369155.5453415,
    "type": "execute"
  }
```

```
]
```

Chapter 2: The Voting System's architecture

My Voting System = Hyperledger fabric architecture + Proof of work
+ Transaction validation in Ethereum

2.1. The block's structure

A block consists of 5 components:

- **index**
- **transactions**
- **timestamp**
- **previous_hash**
- **nonce**

2.2. The blockchain's structure

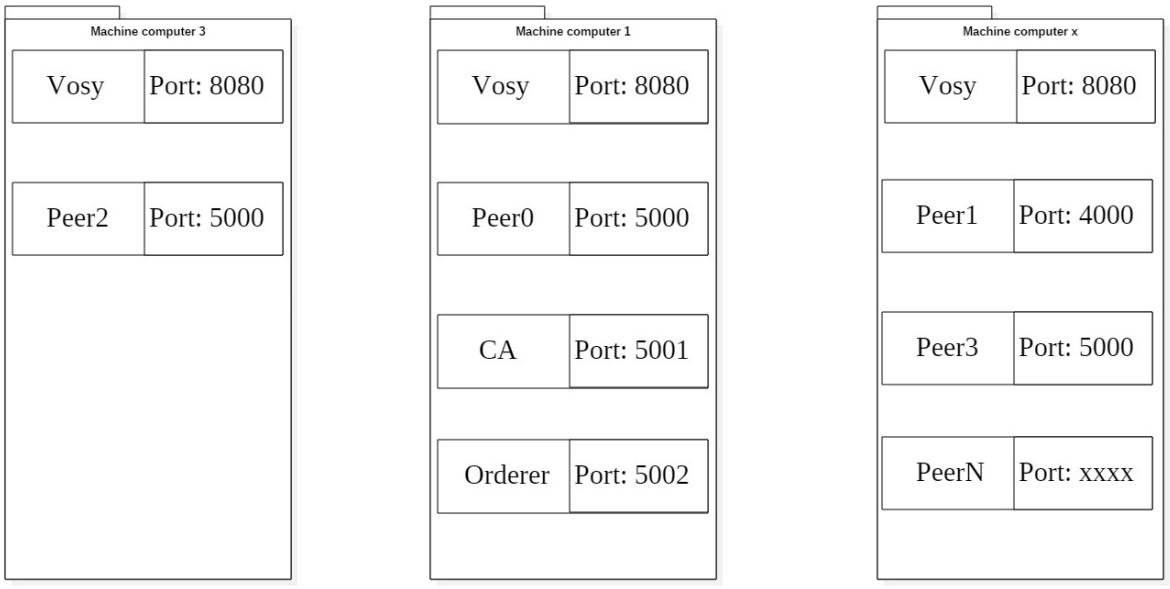
A blockchain consists of 3 components:

- **difficulty:** for solving Proof of Work
- **unconfirmed_transactions**
- **chain**

2.3. The Voting System Architecture

The Voting System's Architecture is based on Hyperledger Fabric's architecture.

2.3.1. The node structure

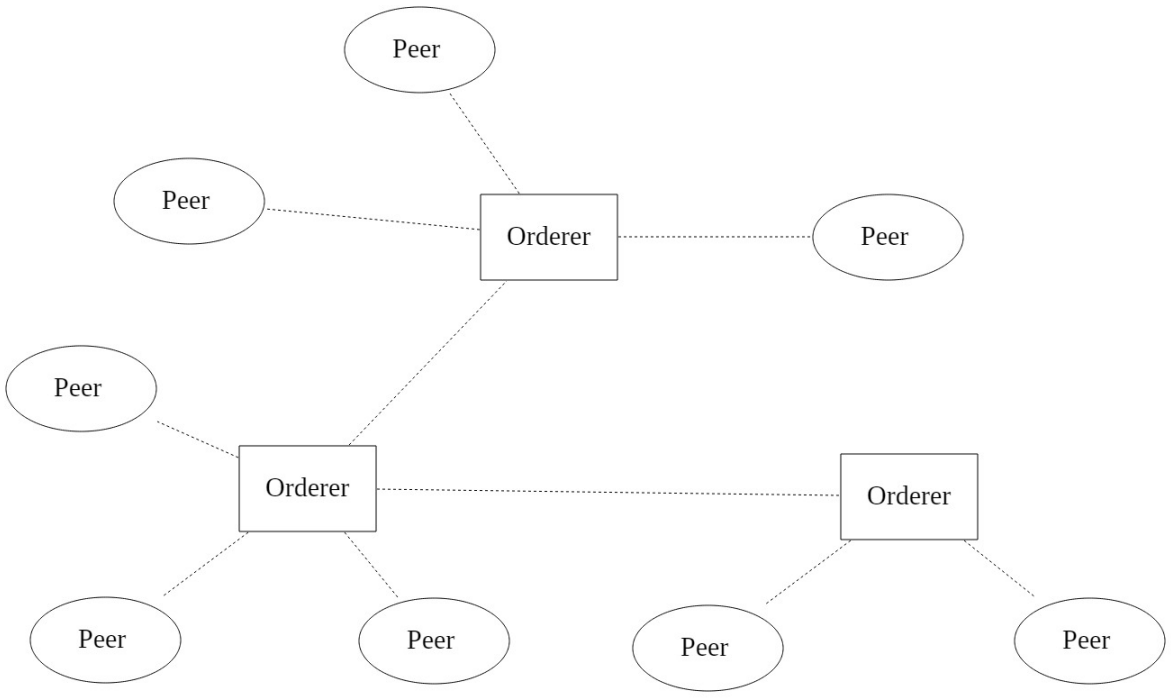


Each machine computer is considered as a node in the network.

A node can have the following components:

- **Certificate Authority (CA):** authenticates connection when a new node requests to join the network and authenticates permission of each node through the methods: “validate_connection”, “validate_permission”.
- **Orderer:** contains a list of peers, has “register_new_peers” method to register new peers, “announce_new_block” method to broadcast to every peer when receiving a new broadcast request, the “announce_new_transaction” method to broadcast to every peer when receiving a new broadcast-transaction request,... In addition, it also has a consensus mechanism that returns the longest blockchain in the network.
- **Peer:** contains all data about the blockchain. It contains some methods: “validate_and_add_block” to validate and add blocks; “validate_transaction” to confirm transactions; “compute_open_surveys” to open a survey, ...
- **Vosy:** An blockchain-based application for the Voting System.

2.3.2. The network’s architecture



The network architecture of the Voting System is based on the decentralized network architecture (Figure B), the "orderer" nodes is intermediate nodes and the "peer" nodes act as terminal nodes.

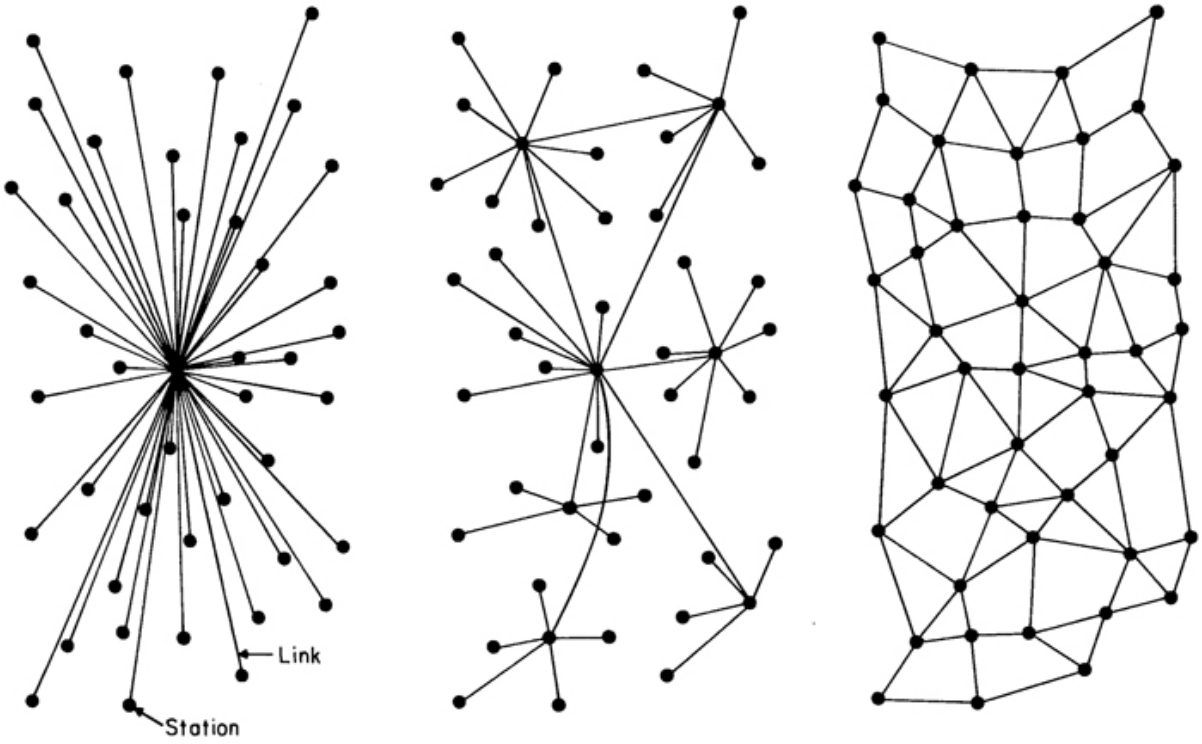


FIG. 1 — Centralized, Decentralized and Distributed Networks

2.4. Consensus mechanism of the Voting System

Currently, there are many consensus mechanisms in a blockchain system such as Byzantine Fault Tolerance, Proof of Work, Proof of Stake, Proof of Elapsed Time, Kafka, etc. Each mechanism has its own advantages and disadvantages.

The Voting System uses the Proof of Work consensus mechanism for some following reasons:

- Proof of Work consensus mechanism makes the system easy to expand in the future.
- Small systems require other security mechanisms, while large systems can be assured of resolving security issues with Proof of Work.

The system uses SHA256 hash algorithm to hash the data in the block. The difficulty for PoW in the system is 2 (The acceptable value after hashing the data in the block is a string that begins with 2 bits 0).

2.5. Types of transactions in the Voting System

There are 5 types of transactions in the system:

- Transaction open

```
[
  {
```

```
    "content": {
      "answers": {
        "Ha Long Bay": [],
        "Tam Dao": [],
        "Viet Tri": []
      },
      "author": "192.168.1.38:5000",
      "question": "Where will we visit this X-mas?",
      "questionid": "1",
      "timestamp": 1544368929.376783
    },
    "timestamp": 1544368929.3807764,
    "type": "open"
  }
]
```


Transaction open example

- Transaction vote


```
[
  {
    "content": {
      "author": "192.168.1.38:5000",
      "questionid": "1",
      "timestamp": 1544369424.89219,
      "vote": "Viet Tri"
    },
    "timestamp": 1544369424.8971915,
    "type": "vote"
  }
]
```


Transaction vote example

- Transaction close


```
[
  {
    "content": {
      "author": "192.168.1.38:5000",
      "questionid": "1",
      "timestamp": 1544369155.5413423
    },
    "timestamp": 1544369155.5453415,
    "type": "close"
  }
]
```


Transaction close example

- Transaction smartcontract


```
[
  {
    "content": {
      "author": "192.168.1.38:5000",
      "code": "print('hello world')"
      "timestamp": 1544369155.5413423
    },
    "timestamp": 1544369155.5453415,
    "type": "smartcontract"
```

```
  }
]
```


- Transaction execute


```
[
  {
    "content": {
      "author": "192.168.1.38:5000",
      "contract": "count_down_opening_time",
      "argument": [opening_time, author, questionid,
CONNECTED_NODE_ADDRESS],
      "timestamp": 1544369155.5413423
    },
    "timestamp": 1544369155.5453415,
    "type": "execute"
  }
]
```


2.6. Transaction authentication mechanism of the Voting System

The transaction authentication mechanism of the Voting System is based on the transaction authentication mechanism in Ethereum - Account Based Transaction Models.

This Account Based Transaction Models is like a global state stores a list of accounts. Each account in Ethereum has balance, storage, and code-space separately. A transaction is valid if an account has a sufficient balance. If the account has code, the code will run and change in the internal repository to create additional messages that may affect the debits and the credits of other accounts. Therefore, all newly created blocks may potentially affect the status of all other accounts.

Surveys and votes in the Voting System is similar with accounts and balances of Account Based Transaction Models.

Before a transaction creates, node's permission must be checked.

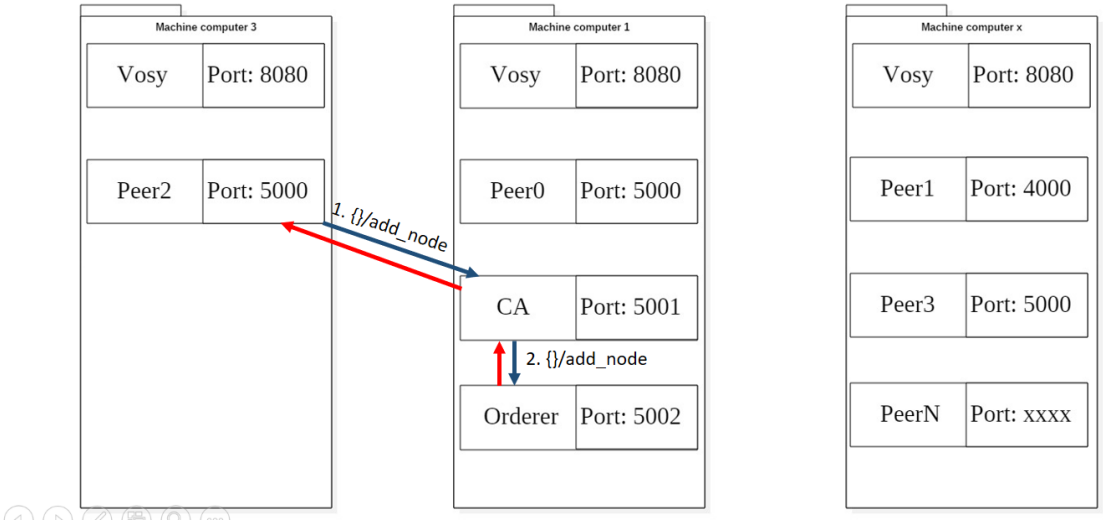
In the Voting System, a transaction is valid if:

- For open transaction: questionId must not match the previous questionId.
- For vote transaction: Voting account of that option cannot vote again (equivalent to each account can only have one vote for a survey)
- For close transaction: questionId must be in the opened survey and only the account that opened that survey can cloes it.
- For smart contract transaction: smart contract will check the validity of the added code, if it is valid, save the read contracts into chain_code, otherwise cancel the transaction

- For execute transaction: try to create a new thread and run contract with the added parameters. If the contract cannot be run, cancel the transaction

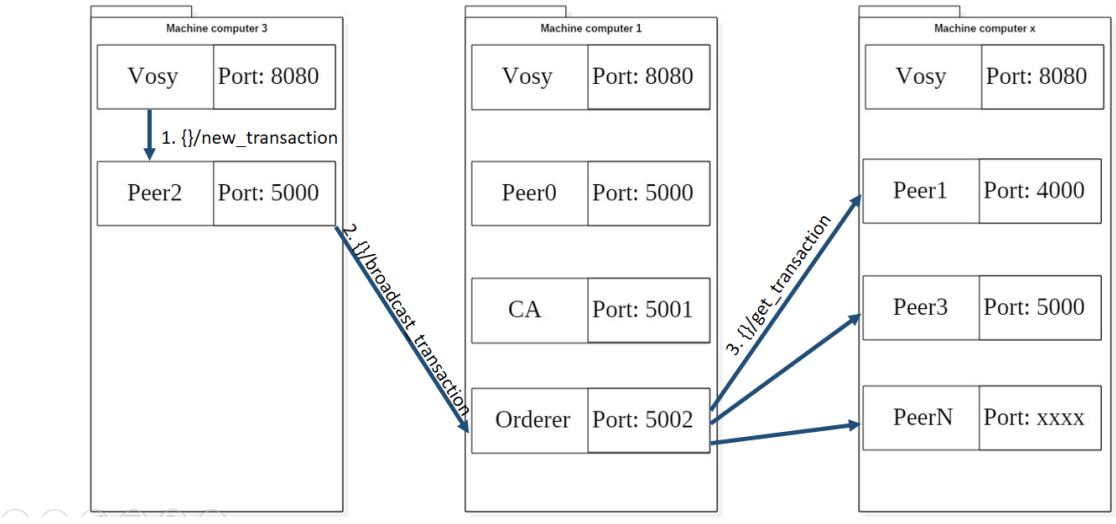
2.7. Main event flows

2.7.1. A node wants to join the network of the system



When a peer wants to join the network: that peer must know a CA in the network and send a join request containing its IP and Port to that CA. The CA will authenticates the connection. If the CA refuses to connect, the message returned to the peer is connection failed. In contrast, the CA will send a request to the orderer to add the peer to the list of peers that it keeps and returns a message of success or failure to the CA. The CA will return a message to the peer.

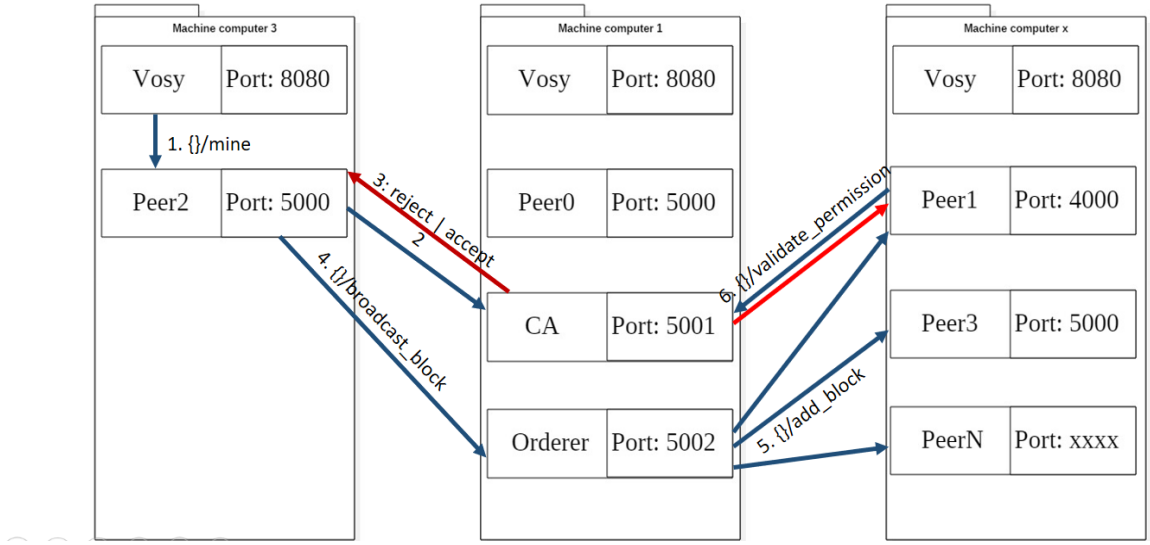
2.7.2. A new transaction is requested



When the vosy requests a new transaction to the peer which is connecting to that vosy (vosy now acts as the author). The Peer that receives the request will

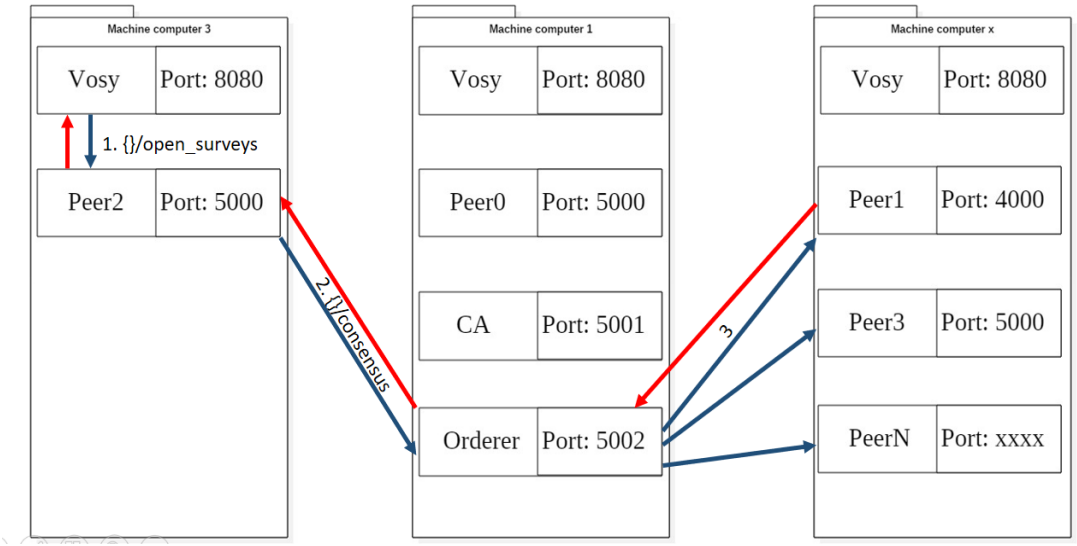
transmit that request to the orderer to boardcast the request to all the peers in the list of the peers that the orderer contains. Requested peers receive new transaction requests will add the transaction to the queue of unconfirmed transactions.

2.7.3. Mine unconfirmed transaction



The Vosy sends mine request to the peer. “Peer2” creates a new block and checks validate_transaction by sending identity and transaction type to CA. If the CA confirms that the author does not have permission to create the transaction, the CA will return a reject message to the peer. If CA accept the author's transaction creating permission, it wil return the accept message to the peer. After receiving the accept message, the peer will take some transactions from the list of unconfirmed transactions to add to newly created block, check the validity of those transactions and start the process of finding nonce number that satisfies the difficulty of Proof of Work. Any peer which finds the first nonce number has the right to create a new block and send the block request to the orderer connecting to it to borardcast the block request to the peers connected to it, the process is similar to the process propagate the transaction request to the entire network.

2.7.4. Consensus flows (to display on vosy)



The vosy sends requests to the peer to stored the peer's blockchain on the local machine. The peer sends requests to the orderer for consensus checking. The orderer boardcasts request to get the longest blockchain of the peers connecting to it. The local chain is returned to the orderer and the orderer finds the longest chain and returns to the peer. The peer then returns the blockchain to the vosy. The vosy compares the length of the blockchain in its peer with the length of the new blockchain it receives and chooses the longer blockchain to become its peer's blockchain (according to the mechanism of Proof of Work). So that all the peers in the system have the same blockchain.

2.8. Smart contract

The system can handle smart contracts defined in the chaincode.py file. To be able to handle smart contracts.