Blockchain Exploration Lab

1 Overview

The objective of this lab is to help students gain some hands-on experiences with blockchains, in particular, with the Ethereum blockchain. A blockchain system is quite complicated, and it is hard to cover every aspect of it in a single lab. We have developed a series of labs focusing on this new technology. This lab is the first one in this series, with a goal of getting students to become familiar with the platform. Students will use existing tools and implement their own tools to interact with the Ethereum blockchain. The following topics are covered in this lab:

- MetaMask, wallet, account
- Transactions and blocks, sending transactions
- Ethereum nodes

Lab environment. The activities in this document have been tested on our pre-built Ubuntu 20.04 VM, which can be downloaded from the SEED website.

2 Lab Setup: Starting the Blockchain Emulator

This lab will be performed inside the SEED Internet Emulator (simply called the emulator in this document). If this is the first time you use the emulator, it is important that you read this section. We recommend instructors to provide a lab session to help students get familiar with the emulator.

Download the emulator files. Please download the `Labsetup.zip` file from the web page, and unzip it. You will get the emulator files. The emulator consists of a number of container files, which are stored in the `Labsetup/emulator_*` folders. The `emulator_NN` folders are for AMD64 machines, while the `emulator_arm_NN` are for the Apple silicon machines. The number `NN` represents the number of nodes on the blockchain network: students can choose the smaller one if the RAM given for their virtual machine is less than 4GB.

To run the emulator, we only need these container files. These files are generated using the Python code stored in the `Labsetup/emulator_code` folder. Unless you want to modify the emulator files, you do not need to run the code in this folder (you need to install the SEED Emulator library from the GitHub to run the code). Instructors who would like to customize the emulator can modify the Python code and generate their own emulator files.

For the sake of simplicity, the blockchain running inside the emulator uses the Proof-of-Authority (PoA) consensus protocol, instead of the Proof-of-Stake protocol used in the MAINET. The activities conducted in this lab are not dependent on any specific consensus protocol.

Start the emulator. Go to the `emulator` folder, and run the following docker commands to build and start the containers. The commands listed below are aliases created on the SEED VM. If you are not using the SEED VM, you can use the original commands.

```
$ dcbuild       # Alias for: docker-compose build
$ dcup          # Alias for: docker-compose up
```

We recommend that you run the emulator inside the provided SEED Ubuntu 20.04 VM, but doing it in a generic Ubuntu 20.04 operating system should not have any problem, as long as the docker software is installed. For newer operating system version, the `docker-compose` command has already been phased out; it is integrated into the `docker` command, and you can run it using `"docker compose"`, instead of `docker-compose`. Readers can find the docker manual from this link. If this is the first time you set up a SEED lab environment using containers, it is very important that you read the user manual.

All the containers will be running in the background. To run commands on a container, we often need to get a shell on that container. We first need to use the `"docker ps"` command to find out the ID of the container, and then use `"docker exec"` to start a shell on that container. We have created aliases for them in the `.bashrc` file.

```
$ dockps        // Alias for: docker ps --format "{{.ID}}  {{.Names}}"
$ docksh <id>   // Alias for: docker exec -it <id> /bin/bash

// The following example shows how to get a shell inside hostC
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// Note: If a docker command requires a container ID, you do not need to
//       type the entire ID string. Typing the first few characters will
//       be sufficient, as long as they are unique among all the containers.
```

If you encounter problems when setting up the lab environment, please read the "Common Problems" section of the manual for potential solutions.

Stop the emulator. To stop the emulator, we just need to stop all the containers. We can go to the terminal where we run the `"docker-compose up"` command, type `Ctrl-C`. That will stop all the containers, but without removing them, i.e., all the data in the containers are still preserved, and they can be resumed by running `"docker-compose up"` again. If we want to remove them, we can run the `"docker-compose down"` command. Another way to do this is to go to a different terminal (but still in the same `emulator` folder) and directly run this command. That will stop and removing all the containers.

```
$ dcdown        # Alias for: docker-compose down
```

EtherView. We have implemented a simple web application called `EtherView` to display the activities on the Blockchain. To access the application, point your browser (within the VM) to `http://localhost:5000/`. From the `Blocks` page, you can see the newly created blocks and recent transactions. If nobody is sending transactions, the blocks are mostly empty, i.e., containing no transactions.

Once we start sending transactions, we should be able to see them. Users can click on the blocks and transactions to see their details.

3 Task 1: Setting Up MetaMask Wallet

There are many ways to interact with the blockchain. For the basic operations, we can use a wallet application, which manages our keys, displays the balances of our accounts, and sends and receives transactions. MetaMask is a very popular wallet application for the Ethereum blockchain. It is available as a browser extension or a standalone mobile application. In this lab, we will use the browser extension.

Task 1.a. Installing the MetaMask extension. Inside the virtual machine, go to Firefox's menu, click the "`Add-ons and themes`" item. Search for `metamask`, and you will find an extension developed by `danfinlay`. Follow the instruction to install it. Take a screenshot to show that you have successfully installed the extension.

Task 1.b. Connecting to the Blockchain. We need to connect the MetaMask wallet to our blockchain. This is done via connecting MetaMask to one of the nodes on the blockchain. We can find the IP addresses of all the Ethereum nodes using the "`docker ps`" command. We have appended the IP address to the container name (the actual IP address you get from your emulator may be different from those listed in the following).

```
$ dockps | grep Eth
e372096bb926  as150h-Ethereum-POA-00-Signer-BootNode-10.150.0.71
f0ef91ef9e22  as150h-Ethereum-POA-01-10.150.0.72
3b8c1d191058  as151h-Ethereum-POA-02-Signer-10.151.0.71
...
aea1106d932d  as164h-Ethereum-POA-18-Signer-BootNode-10.164.0.71
7cd6fa6888b2  as164h-Ethereum-POA-19-10.164.0.72
```

Pick one of the nodes, and then configure MetaMask to connect to this node. Go to the `Settings` menu inside MetaMask and follow the instructions below. Replace the `<IP Address>` with the actual IP address of the node that you have selected. Take a screenshot to show that you have successfully connected to the emulator.

```
Settings > Networks > Add a network > Add a network manually

Network name:    pick any name (e.g., SEED emulator)
New RPC URL:     http://<IP Address>:8545
Chain ID:        1337
Currency symbol: ETH
```

Task 1.c. Adding accounts. In this task, we will add a few accounts to our wallet. We can ask MetaMask to create new accounts for us or import existing accounts. We have already created several accounts when creating the emulator, and these accounts already have funds in them. These accounts are generated from the following mnemonic phrase, so they can be recovered using the same phrase.

```
gentle always fun glass foster produce north tail security list example gain
```

In this task, we will add these existing accounts to our MetaMask wallet, so we can use them to send transactions. To do that, we need to log out from our MetaMask account (or lock our account). That will lead us to the login window. We will pretend that we have forgotten the password, so we will click the "Forgot password" link. For MetaMask, if you forget the password to your wallet account, unless you have already stored your keys somewhere, the only way to get your account keys back is through a secret recovery phrase. By typing the above recovery phrase, we can ask MetaMask to recover our keys.

MetaMask will generate all the existing accounts that have a non-zero balance on the blockchain. Please show the balance of these accounts in your lab report. If you do not see the conversion of ETH to fiat currency (such as dollars), you can go to Settings > Advanced and enable the "Show conversion on test networks". MetaMask will now show the conversion.

Task 1.d. Sending transactions. Now we can send transactions using our accounts. Please send money from one account to another, and check the balance of these accounts to verify that the transaction is successful. Please provide the screenshots for the entire process. Please also show the details of transactions using EtherView, and verify that the transaction details are consistent with what you have sent.

4 Task 2: Interacting with Blockchain Using Python

We have already seen to interact with the blockchain using an existing tool like MetaMask. In this task, we will write our own tool to interact with the blockchain. This will give us more insight on how to interact with a blockchain. We will conduct this task from the host VM. The code used in this task can be found at the Labsetup/Files folder.

Task 2.a: Installing Python modules. We will use the web3 and docker modules in our Python program. We need to install these modules using the following command (they may have already been installed in your VM). It should be noted that our code uses an older version of web3. That is why we have specified the version number in our command.

```
pip3 install web3==5.31.1 docker
```

Task 2.b: Checking account balance. The code (provided) below shows how to get a balance from the blockchain. Please go to your MetaMask wallet, get the first three accounts, and then use the following program to check their balances. Compare the values with what you see in MetaMask.

Listing 1: Get balance (web3_balance.py)

```
#!/bin/env python3
from web3 import Web3

url  = 'http://10.150.0.71:8545'
web3 = Web3(Web3.HTTPProvider(url))  # Connect to a blockchain node

addr = Web3.toChecksumAddress('0xF5406927254d2dA7F7c28A61191e3Ff1f2400fe9')
balance = web3.eth.get_balance(addr) # Get the balance
print(addr + ": " + str(Web3.fromWei(balance, 'ether')) + " ETH")
```

Task 2.c: Sending transactions. In this task, we will use a Python program to send out a transaction. This program constructs a transaction, signs it using the sender's private key, and then sends out the transaction via an Ethereum node. The program will block, waiting for the transactions to be confirmed (i.e., to be included in a block). Run this program, and then check from MetaMask whether the balance of the sender/receiver accounts have changed. You should fill in the needed information in Lines ①, ②, and ③. For the private key, you can get it from MetaMask: click the "`Account detaials`" menu, and then click the "`Show private key`" button.

Listing 2: Send transaction (`web3_raw_tx.py`)

```
from web3 import Web3
from eth_account import Account

web3 = Web3(Web3.HTTPProvider('http://ip-address:8545'))              ①

# Sender's private key
key = 'replace this with the actual private key'                      ②
sender = Account.from_key(key)

recipient = Web3.toChecksumAddress('replace this with an account #') ③
tx = {
  'chainId':  1337,
  'nonce':    web3.eth.getTransactionCount(sender.address),
  'from':     sender.address,
  'to':       recipient,
  'value':    Web3.toWei("11", 'ether'),
  'gas':      200000,
  'maxFeePerGas':         Web3.toWei('4', 'gwei'),
  'maxPriorityFeePerGas': Web3.toWei('3', 'gwei'),
  'data':     ''
}

# Sign the transaction and send it out
signed_tx  = web3.eth.account.sign_transaction(tx, sender.key)
tx_hash    = web3.eth.sendRawTransaction(signed_tx.rawTransaction)

# Wait for the transaction to appear on the blockchain
tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
print("Transaction Receipt: {}".format(tx_receipt))
```

5 Task 3: Interacting with Blockchain Using Geth

We can also interact with the blockchain directly from a blockchain node. In our emulator, each Ethereum node runs the Geth (go-ethereum) client, which is a Go implementation of Ethereum. There are multiple ways to interact with the Geth client, including websockets, HTTP, and local IPC. When we interact with the Geth nodes using MetaMask or Python programs, we are using the JSON-RPC method. We can also log into a Geth node, and communicate with it using local IPC. The following `geth` command gets an interactive console on the node.

```
root@f6fb88f9e09d / # geth attach
Welcome to the Geth JavaScript console!
```

```
instance: Geth/NODE_8/v1.10.26-stable-e5eb32ac/linux-amd64/go1.18.10
coinbase: 0xa888497f7938825f80f35867a1e707f42b9b347d
...
To exit, press ctrl-d
>
```

This is an interactive JavaScript console, so we can run JavaScript code in it. The `eth` class has many APIs that we can use to interact with the blockchain. The following example shows how to get an account balance.

```
> myaccount = "0xc20ab9a1ab88c9fae8305b302836ee7734c6afbe"
> eth.getBalance(myaccount)
100000000
```

Task 3.a: Getting balance. Please get the balances for the first 3 accounts in your MetaMask wallet, and see whether they are the same as the one shown on MetaMask.

Task 3.b: Sending transactions. Each node maintains a list of accounts, which are stored in `/root/.ethereum/keystore`. Their address are loaded into the `eth.accounts[]` array. For example, to get the first account address, we can use `eth.accounts[0]`. These accounts are locked (i.e., encrypted using a password), so to use them to send transactions, we need to unlock it using the password. In our emulator, all the accounts are locked using the hardcoded password `admin`.

```
> eth.accounts
["0x3e64b5b296ccb365eab980b094a4af7b1009825e"]
> personal.unlockAccount(eth.accounts[0], "admin")
true
```

Now, we can send fund from these accounts to an account in our MetaMask wallet. Please use the following example to send fund to one of your MetaMask wallet accounts, and see whether the results show up correctly on MetaMask.

```
> sender = eth.accounts[0]
> target = "0xF5406927254d2dA7F7c28A61191e3Ff1f2400fe9"
> amount =  web3.toWei(0.2, "ether")
> eth.sendTransaction ({from: sender, to: target, value: amount})
"0x8c6c57d5a32de...7304"
```

Task 3.c: Sending transactions from a different account. Instead of using `eth.accounts[0]`, let us use an account from our MetaMask wallet, i.e., we set `sender` to one of the accounts in our wallet. Try send a transaction and report/explain your observation.

6 Task 4: Adding a Full Node

In this task, we will learn how to join an existing blockchain network. We have already provided an empty container, with word `new_eth_node` encoded in its name. Your job is to configure this container, turning it into an Ethereum node. First, we initialize our node using the blockchain information, which is captured

in its genesis block. A genesis block is the first block of a block chain. You can get a copy of the genesis block from an existing Ethereum node (in `/tmp/eth-genesis.json`).

```
geth --datadir /root/.ethereum init /eth-genesis.json
```

Second, we run the `geth` command to join an existing blockchain network. To do that, we need to provide a list of bootnodes. Go to any of existing Ethereum nodes (non-bootnode), you should be able to see a file called `eth-node-urls` in the `/tmp` folder. This file contains a list of the bootnodes on the blockchain network. Copy and paste its content to the `/tmp/eth-node-urls` file in the `new_eth_node` container. In the following `geth` command, we feed the content of the `/tmp/eth-node-urls` file to the `bootnodes` option.

Listing 3: `start.sh`

```
geth --datadir /root/.ethereum --identity="NEW_NODE_01" --networkid=1337 \
     --syncmode full --snapshot=False --verbosity=2  --port 30303 \
     --bootnodes "$(cat /tmp/eth-node-urls)" --allow-insecure-unlock \
     --http --http.addr 0.0.0.0 --http.corsdomain "*" \
     --http.api web3,eth,debug,personal,net,clique,engine,admin,txpool
```

Task. Please follow the instructions above to turn the `new_eth_node` container into an Ethereum node. After setting it up, please do the following tasks:

- Run `"geth attach"` on this node to get the JavaScript console; use `admin.peers` to see the peers of the node.
- In the same console, use `personal.newAccount()` to create an new account on this node. We call it Account Z.
- Modify the Python code (`web3_raw_tx.py`) from Task 2, so it connects to this new node, and use this new node to send transactions. Send some amount of ethers to Account Z.
- In the JavaScript console, send a transaction from Account Z to another account.

7 Submission

You need to submit a detailed lab report, with screenshots, to describe what you have done and what you have observed. You also need to provide explanation to the observations that are interesting or surprising. Please also list the important code snippets followed by explanation. Simply attaching code without any explanation will not receive credits.