区块链探索实验

1 概述

此实验的目的是让学生自己动手与区块链交互，从中得到实际操作的经验，我们使用目前流行的以太坊区块链系统。由于区块链是一个相当复杂的系统，很难在一个实验中覆盖各个方面。因此，我们设计开发了一系列专门针对这项新兴技术的实验。本实验是这一系列实验的基础，它的目的是帮助学生熟悉此实验平台。在本实验中，学生将使用现有工具，也会开发自己的工具来与以太坊区块链交互。本实验涵盖以下几个主题：

- MetaMask、钱包、账户
- 交易和区块、发送交易
- 以太坊节点

实验环境 本文档中的实例已在我们预先搭建的 Ubuntu 20.04 虚拟机上进行了测试，可以从 SEED 官网下载。

2 实验设置：启动区块链仿真器

这个实验将在 SEED 网络仿真器（本文档中简称仿真器）中展开，如果这是你首次接触 SEED 仿真器，我们强烈建议你仔细阅读本节内容，同时建议教师们组织专门的实验课程帮助学生了解和熟悉仿真器的操作。

下载仿真器文件。 请从网页上下载 `Labsetup.zip` 文件，解压缩后会得到仿真器文件。这些仿真器文件存储在 `Labsetup/emulator_*` 文件夹中。对于 AMD64 机器，文件夹名为 `emulator_NN`；对于 Apple 硅芯片机器，则为 `emulator_arm_NN`。数字 `NN` 表示区块链网络上的节点数量。如果虚拟机的内存小于 4GB，建议学生选择较小的版本。

要运行仿真器，我们只需要这些容器文件。这些文件是由 `Labsetup/emulator_code` 文件夹中的 Python 代码生成的。除非你想修改仿真器文件，否则无需运行该代码（需要从 GitHub 安装 SEED Emulator 库才能运行此代码）。希望修改仿真器的老师可以修改 Python 代码并生成自己的仿真器文件。

为了简化操作，在仿真器内部运行的区块链使用的是 Proof-of-Authority （PoA）共识协议，而不是在 MAINET 中使用的 Proof-of-Stake 协议。实验中的活动不依赖于任何特定共识协议。

启动仿真器。 进入 `emulator` 文件夹，并运行以下 docker 命令以构建和启动容器。下面列出的命令是在 SEED VM 上创建的别名。如果您不是使用 SEED VM，可以使用原始命令。

```
$ dcbuild       # 别名为: docker-compose build
$ dcup          # 别名为: docker-compose up
```

我们建议您在提供的 SEED Ubuntu 20.04 虚拟机中运行仿真器，但在安装了 docker 软件的普通 Ubuntu 20.04 操作系统上进行操作也不会有问题。对于较新的操作系统版本，`docker-compose` 命令已被集成到 `docker` 命令中，您可以使用 `"docker compose"` 替代 `docker-compose` 运行它。读者可以从此链接找到 docker 手册。如果您是首次使用容器设置 SEED 实验环境，阅读用户手册非常重要。

所有容器都在后台运行。要对容器执行命令，我们通常需要在该容器上获取一个 shell。首先，我们需要使用 `docker ps` 命令来找到容器的 ID，然后使用 `docker exec` 在该容器上启动一个 shell。我们在 `.bashrc` 文件中为此创建了别名。

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

// 以下示例说明如何在 hostC 上获取 shell
$ dockps
b1004832e275  hostA-10.9.0.5
0af4ea7a3e2e  hostB-10.9.0.6
9652715c8e0a  hostC-10.9.0.7

$ docksh 96
root@9652715c8e0a:/#

// 注意: 如果 docker 命令需要容器 ID, 不需要输入完整的 ID,
//       输入前面的一部分就可以了, 只要能唯一匹配一个 ID 就行。
```

如果您在设置实验环境时遇到问题，请阅读手册中的 “常见问题” 部分以获取解决方案。

停止仿真器。 要停止仿真器，我们只需停止所有容器。我们可以回到运行 `"docker-compose up"` 命令的终端中，输入 `Ctrl-C`。这将停止所有容器但不会删除它们，即容器中的数据仍然保留，并可以通过再次运行 `"docker-compose up"` 来恢复容器的运行。如果我们想要删除它们，则可以运行 `"docker-compose down"` 命令。另一种方法是打开一个不同的终端（但仍处在 `emulator` 文件夹中），直接运行该命令。这将停止并删除所有容器。

```
$ dcdown        # 别名为: docker-compose down
```

EtherView. 我们实现了一个名为 `EtherView` 的简单 Web 应用程序，以显示区块链上的活动。如果想使用这个应用程序，将浏览器（在 VM 中）指向 `http://localhost:5000/`。在 `Blocks` 页面中，您可以看到新创建的区块和最近的交易。如果没有人发送交易，则区块大多是空的，即不包含任何交易。一旦我们开始发送交易，我们应该能够看到它们。用户可以点击区块和交易来查看它们的详细信息。

3 任务 1：配置 MetaMask 钱包

目前有许多与区块链进行交互的方式，对于一些基础操作，我们可以使用钱包应用程序来管理密钥、查看账户余额并发送和接收交易。MetaMask 是一款非常流行的用于以太坊的钱包应用程序，它既可以作为浏览器插件使用，也提供了独立的移动应用版本。在本实验中，我们将使用其浏览器插件版本。

任务 1.a. 升级 Firefox。 SEED 虚拟机里的 Firefox 版本太低，安装 MetaMask 会有问题。我们先按照以下方法升级 Firefox.

```
$ firefox --version
Mozilla Firefox 83.0
$ sudo apt install firefox      # 升级 Firefox
$ firefox -version
Mozilla Firefox 133.0
```

任务 1.b. 安装 MetaMask 插件 在虚拟机中，进入 Firefox 的菜单页面，点击 " 附加组件和主题" ("Add-ons and themes")，搜索 metamask，找到由 danfinlay 开发的 MetaMask 插件，并按照安装指南进行操作。安装完成后，请截图证明你已成功安装了该插件。

任务 1.c. 连接到区块链 为了使 MetaMask 钱包与区块链建立连接，我们需要将 MetaMask 连接到区块链网络中的任意一个节点。通过执行 "docker ps" 命令，我们可以检索到所有以太坊节点的 IP 地址，这些地址已经被我们附加到容器名称上（你从仿真器中获得的实际 IP 地址可能与以下列出的不同）。

```
$ docker ps | 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
```

选择一个节点后，需要配置 MetaMask 钱包以便其能连接到该节点。首先进入 MetaMask 的 设置 (Settings) 菜单，并按照以下步骤操作。你需要将 <IP Address> 替换为你选定节点的实际 IP 地址。配置完成后，请截图证明你已成功将 MetaMask 连接至区块链仿真器。

```
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
```

任务 1.d. 添加账户 在本任务中，我们将向钱包中添加若干账户，MetaMask 支持创建新的账户或者导入现有账户。在搭建仿真器的过程中，我们已经创建了几个预存资金的账户，这些账户都是基于下述助记词创建的，因此可以通过这些词语恢复。

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

在本任务中，我们将把这些已有账户添加至 MetaMask 钱包中，这样我们就能使用这些账户来发送交易。为此，我们需要先从 MetaMask 中退出（或锁定账户），这会把我们带回到登录界面。然后我们点击登录界面的 `"Forgot password"` 链接，对于 MetaMask 来说，如果你忘记了钱包账户的密码，除非你之前在其他地方备份了密钥，否则就只能通过使用助记词来恢复密钥，输入之前提供的助记词后，MetaMask 将帮助我们恢复密钥。

MetaMask 会显示区块链上所有余额非零的账户，请在你的实验报告中详细列出这些账户的余额情况。如果你在界面上没有看到 ETH 兑换成法定货币（例如美元），你可以通过访问 `Settings > Advanced` 菜单来打开 `"Show conversion on test networks"` 功能。这之后，MetaMask 会根据当前的转换汇率把 ETH 转换成法定货币。

任务 1.e. 发送交易 现在我们就可以使用账户来发送交易，请从一个账户向另一个账户转账资金，检查这些账户的余额变化并验证交易是否成功。请对整个过程进行截图记录，并展示 `EtherView` 工具提供的交易详细信息，检查这些信息是否与实际发送的交易内容相符。

4 任务 2：使用 Python 与区块链交互

我们之前已经探索了如何利用 MetaMask 这样的工具与区块链进行交互，在本任务中，我们将亲手编写自己的工具来与区块链进行交互，加深我们对区块链交互机制的理解。所有的操作都在主机虚拟机（VM）上执行，本任务中使用的代码存放在 `Labsetup/Files` 文件夹中。

任务 2.a：安装 Python 模块 在我们的 Python 程序中，我们将使用 `web3` 和 `docker` 这两个模块，安装命令如下（它们可能已经预装在你的 VM 中）。需要注意的是，我们的仿真器用的是 web3 的一个较早版本，这也是我们在安装命令中明确指定版本号的原因。

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

任务 2.b：检查账户余额 以下是一段示例代码，展示了如何从区块链中获取账户余额，请打开你的 MetaMask 钱包，查看前三个账户的地址，随后使用这段程序来查询它们的余额，并将程序显示的余额与 MetaMask 钱包显示的余额进行比较。

Listing 1: 获取账户余额 (`web3_balance.py`)

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

url  = 'http://10.150.0.71:8545'
```

```
web3 = Web3(Web3.HTTPProvider(url))  # 连接到区块链节点

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

任务 2.c：发送交易。 在本任务中，我们将编写一个 Python 程序来发送交易，该程序将构建一笔交易，并使用发送者的私钥对其签名，然后通过以太坊节点发送交易。该程序在发送交易后会阻塞，直到交易被确认（也就是交易已经被放在了区块链上）。运行该程序后，请检查 MetaMask 中发送者和接收者账户的余额是否发生了变化。在运行程序之前，请确保在 ① 、② 和 ③ 处填写所需的信息。你可以从 MetaMask 获取私钥，首先点击 "Account details" 菜单，然后点击 "Show private key" 按钮即可。

Listing 2: 发送交易 (web3_raw_tx.py)

```
from web3 import Web3
from eth_account import Account

web3 = Web3(Web3.HTTPProvider('http://ip-address:8545'))             (@\lineone@)

# 发送者的私钥
key = 'replace this with the actual private key'                     (@\linetwo@)
sender = Account.from_key(key)

recipient = Web3.toChecksumAddress('replace this with an account #') (@\linethree@)
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':     ''
}

# 给交易签名并发送
signed_tx  = web3.eth.account.sign_transaction(tx, sender.key)
tx_hash    = web3.eth.sendRawTransaction(signed_tx.rawTransaction)

# 等待交易出现在区块链上
tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
print("Transaction Receipt: {}".format(tx_receipt))
```

5 任务 3：使用 Geth 与区块链交互

我们可以直接通过一个区块链节点与区块链进行交互。在我们的仿真器中，每个以太坊节点都运行 Geth（go-ethereum）客户端，这是用 Go 语言实现的以太坊。与 Geth 客户端交互有多种方式，包括 websockets、HTTP 和本地 IPC。当我们使用 MetaMask 或 Python 程序与 Geth 节点交互时，我们采用的是 JSON-RPC 方法。此外，我们还可以登录到 Geth 节点，并使用本地 IPC 与其通信。以下是一个 `geth` 命令，用于在节点上获得一个交互式控制台。

```
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
>
```

这是一个交互式的 JavaScript 控制台，我们可以在其中执行 JavaScript 代码。`eth` 类提供了丰富的 API 接口，使得我们可以方便地与区块链进行交互。以下是一个示例，展示了如何利用这些 API 来查询账户余额。

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

任务 3.a：获取余额。 请从你的 MetaMask 钱包中获取前三个账户的余额，并查看结果是否与 MetaMask 上显示的一致。

任务 3.b：发送交易。 每个节点都维护着一租账户，这些账户信息存储在 `/root/.ethereum/keystore` 下，它们的地址被加载到 `eth.accounts[]` 数组中。例如，如果我们想要获取数组中第一个账户地址，可以通过 `eth.accounts[0]` 来获得。这些账户默认是锁定状态的（即使用密码加密），因此在使用这些账户进行交易之前，我们需要先对它们解锁。在我们的仿真器环境中，所有账户都可以使用固定密码 admin 来解锁。

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

现在我们可以从这些账户向我们 MetaMask 钱包中的账户转账资金了，请按照下面的示例操作，向你的 MetaMask 钱包中的一个账户转账，并查看这笔交易的结果是否显示在 MetaMask 上。

```
> sender = eth.accounts[0]
> target = "0xF5406927254d2dA7F7c28A61191e3Ff1f2400fe9"
> amount =  web3.toWei(0.2, "ether")
```

```
> eth.sendTransaction ({from: sender, to: target, value: amount})
"0x8c6c57d5a32de...7304"
```

任务 3.c：使用不同账户发送交易。 我们现在不使用 `eth.accounts[0]`，而是使用 MetaMask 钱包中的任意一个账户发送交易。即设置 `sender` 为我们钱包中的某个账户，请尝试进行交易，并分享或者解释你的结果。

6 任务 4：添加一个全节点

在本任务中，我们将学习如何将一个新的节点加入到现有的区块链网络中。我们已经准备了一个名为 `new_eth_node` 的空容器，你的任务是对这个容器进行配置，使其成为一个完备的以太坊节点。首先，我们需要使用区块链的初始信息初始化我们的节点，这些信息被保存在创世区块中（genesis block），创世区块是区块链的第一个区块，可以从仿真器中以太坊节点的 `/tmp/eth-genesis.json` 文件中找到创世区块的内容。

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

然后，我们执行 `geth` 命令将新节点加入到现有的区块链网络中。为此，我们需要提供一份引导节点列表，你可以在任何现有的以太坊节点（非引导节点）的`/tmp` 目录下找到一个名为 `eth-node-urls` 的文件。该文件包含了区块链网络上的所有引导节点的信息。你需要将这个文件的内容复制并粘贴到 `new_eth_node` 容器中的 `/tmp/eth-node-urls` 文件中。在执行以下的 geth 命令时，我们将使用/tmp/eth-node-urls 文件中的内容作为 bootnodes 选项的参数，以便新节点能够连接到网络中的其他节点。

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
```

任务. 请根据上述步骤将 `new_eth_node` 容器配置成以太坊节点。配置完成后，请执行以下任务：

- 在此节点上执行 `"geth attach"` 命令以获得 JavaScript 控制台，接着使用 `admin.peers` 命令查看当前节点连接的对等节点列表。
- 在同一控制台中，执行 `personal.newAccount()` 命令创建一个新账户，我们将这个新创建的账户称之为账户 Z。
- 修改任务 2 中的 Python 代码 (`web3_raw_tx.py`)，连接到这个新节点，并通过这个新节点向账户 Z 发送交易，转一些以太币给账户 Z。
- 在 JavaScript 控制台中，从账户 Z 发送交易到另一个账户。

7 提交

你需要提交一个详细的，带有截图的实验报告来描述你做了什么以及你观察到了什么。你还需要对有趣的或是令人惊讶的观察结果进行解释。请同时列出重要的代码段并附上解释。简单地附上代码而不作任何解释将不会得到学分。