**A.1 Experimental Environment Setup**

The configuration details of the experimental environment used in this research are listed below:

|  |  |
| --- | --- |
| Component | Configuration/Version |
| Operating System | Ubuntu 20.04 LTS |
| Docker | v24.0.5 |
| Hyperledger Fabric | v2.4 |
| Programming Languages | Golang (chaincode), Python (scripts) |
| Hyperledger Caliper | v0.5.0 |
| Test Network Path | ~/Desktop/fabric-samples/test-network |

**Machine Specifications**:

* CPU: 4 cores
* RAM: 8 GB
* Storage: 30 GB SSD

**A.2 Detailed Experimental Procedures**

The experimental procedure is clearly structured into four main stages as follows:

**Step 1: Network Initialization and Cleanup**

Firstly, navigate to the Fabric test network directory:

cd ~/Desktop/fabric-samples/test-network

Then, shut down and clean up any existing Docker network, containers, and chaincode images:

bash

./network.sh down

docker system prune -a -f

docker rm -f $(docker ps -aq)

docker volume prune -f

docker network prune -f

docker rmi -f $(docker images dev-\* -q)

rm -rf organizations/peerOrganizations \

organizations/ordererOrganizations channel-artifacts log.txt \*.tar.gz

文本

AI 生成的内容可能不正确。

**Step 2: Hyperledger Fabric Network Setup**

Start the network, create a channel named mychannel, and launch the Fabric CA server for identity management (such as generating certificates):

./network.sh up createChannel -c mychannel -ca

文本

AI 生成的内容可能不正确。

Set environment variables required for subsequent chaincode installation:

export CORE\_PEER\_TLS\_ENABLED=true

export CORE\_PEER\_LOCALMSPID="Org1MSP"

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

export CORE\_PEER\_ADDRESS=localhost:7051

export PATH=${PWD}/../bin:$PATH

export FABRIC\_CFG\_PATH=${PWD}/../config/

Brief explanations:

* CORE\_PEER\_TLS\_ENABLED=true: Enables TLS-secured communication between Fabric nodes.
* CORE\_PEER\_LOCALMSPID="Org1MSP": Identifies the Membership Service Provider (MSP) of the current peer as Organization 1.
* CORE\_PEER\_TLS\_ROOTCERT\_FILE: Specifies the TLS certificate to authenticate peer communications securely.
* CORE\_PEER\_MSPCONFIGPATH: Points to the directory containing the admin user's certificates for identity management.
* CORE\_PEER\_ADDRESS=localhost:7051: Indicates the target peer node's address for transactions (peer0 of Org1).
* PATH=${PWD}/../bin:$PATH: Adds Fabric binaries (such as peer, fabric-ca-client) to the system’s executable path.
* FABRIC\_CFG\_PATH=${PWD}/../config/: Defines the path to Fabric configuration files (core.yaml, etc.).

**Step 3: Chaincode Packaging and Installation**

**3.1 Package Chaincode**

First, package the smart contract (chaincode) into a compressed file:

peer lifecycle chaincode package energyTrading.tar.gz \

--path ../energy-trading/chaincode-go/ \

--lang golang \

--label energyTrading\_1.1

Brief explanations:

* --path: Directory path of your smart contract source code.
* --lang: Programming language of the chaincode (Golang).
* --label: A unique label identifying the chaincode version.
  1. **Install Chaincode on Peer Nodes**

Next, install the packaged chaincode on each organization's peer nodes:

* Organization 1 (Org1):  
  Set environment variables for Org1 (already set in Step 2):

peer lifecycle chaincode install energyTrading.tar.gz

* Organization 2 (Org2):  
  Set environment variables specific for Org2:

export CORE\_PEER\_LOCALMSPID="Org2MSP"

export CORE\_PEER\_TLS\_ROOTCERT\_FILE=${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

export CORE\_PEER\_MSPCONFIGPATH=${PWD}/organizations/peerOrganizations/org2.example.com/users/Admin@org2.example.com/msp

export CORE\_PEER\_ADDRESS=localhost:9051

peer lifecycle chaincode install energyTrading.tar.gz

Brief explanations (for Org2 variables):

* CORE\_PEER\_LOCALMSPID="Org2MSP": Specifies Org2's MSP identity.
* CORE\_PEER\_TLS\_ROOTCERT\_FILE: TLS certificate file for Org2 peer authentication.
* CORE\_PEER\_MSPCONFIGPATH: Directory for Org2 admin certificates.
* CORE\_PEER\_ADDRESS=localhost:9051: Address of Org2’s peer node.

文本

AI 生成的内容可能不正确。

**Step 4: Chaincode Approval and Commitment**

After installing the chaincode on peer nodes, we need each organization to approve and commit the chaincode definition to the network.

**4.1 Approve Chaincode Definition for Organization 1 (Org1)**

Ensure environment variables for Org1 are set as in previous steps. Approve the chaincode definition using:

peer lifecycle chaincode approveformyorg \

--orderer localhost:7050 \

--ordererTLSHostnameOverride orderer.example.com \

--channelID mychannel \

--name energyTrading \

--version 1.1 \

--package-id energyTrading\_1.1:<Your-Package-ID> \

--sequence 1 \

--tls \

--cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

Brief explanations of parameters:

* **--orderer**: Address of the orderer node for submitting the transaction.
* --ordererTLSHostnameOverride: Overrides the TLS certificate hostname verification.
* --channelID: Channel name to which the chaincode belongs (mychannel).
* --name: Name assigned to the chaincode (energyTrading).
* --version: Version of the chaincode definition (1.1).
* --package-id: Identifier from the chaincode installation step (replace with your actual ID).
* --sequence: Sequence number of the chaincode definition.
* --tls: Enables TLS security for the transaction.
* --cafile: TLS certificate file path for the orderer node.

**4.2 Approve Chaincode Definition for Organization 2 (Org2)**

Set environment variables for Org2 (as previously described in Step 3), and approve the chaincode:

peer lifecycle chaincode approveformyorg \

--orderer localhost:7050 \

--ordererTLSHostnameOverride orderer.example.com \

--channelID mychannel \

--name energyTrading \

--version 1.1 \

--package-id energyTrading\_1.1:< Package-ID> \

--sequence 1 \

--tls \

--cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

文本

AI 生成的内容可能不正确。

**4.3 Commit Chaincode Definition to the Channel**

Finally, commit the approved chaincode definition to the channel (mychannel):

peer lifecycle chaincode commit \

-o localhost:7050 \

--ordererTLSHostnameOverride orderer.example.com \

--channelID mychannel \

--name energyTrading \

--version 1.1 \

--sequence 1 \

--tls \

--cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \

--peerAddresses localhost:7051 \

--tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt \

--peerAddresses localhost:9051 \

--tlsRootCertFiles ${PWD}/organizations/peerOrganizations/org2.example.com/peers/peer0.org2.example.com/tls/ca.crt

Brief explanations of additional parameters:

* --peerAddresses: Addresses of peer nodes involved in the commit operation.
* --tlsRootCertFiles: TLS certificate paths for verifying peer node identities.

文本

AI 生成的内容可能不正确。

**Step 5: Chaincode Initialization**

After committing the chaincode definition, we initialize the ledger by invoking the chaincode. This sets up the initial state for transactions within the blockchain ledger.

Invoke the initialization function using:

peer chaincode invoke \

-o localhost:7050 \

--ordererTLSHostnameOverride orderer.example.com \

--tls true \

--cafile ${PWD}/organizations/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem \

-C mychannel \

-n energyTrading \

-c '{"function":"InitLedger","Args":[]}'

Brief explanations of parameters:

* -o: Orderer node's network address used for transaction ordering.
* --ordererTLSHostnameOverride: Overrides the TLS hostname for secure connection.
* --tls true: Enables TLS for secure peer-orderer communication.
* --cafile: TLS certificate file for authenticating the orderer.
* -C mychannel: Specifies the channel where the chaincode is invoked.
* -n energyTrading: Specifies the chaincode name.
* -c '{"function":"InitLedger","Args":[]}': Calls the InitLedger function defined in the chaincode, initializing data for transactions.

图片包含 图形用户界面

AI 生成的内容可能不正确。

**Step 6: User Data Initialization**

After chaincode initialization, we used a Python script (init\_users.py) to initialize user accounts with their respective reputation scores and account balances. This step provides initial test data necessary for experimental simulations.

Execute the user initialization script:

python3 init\_users.py

The script registers users into the ledger with initial values, as demonstrated by the following terminal output:

电脑萤幕画面

AI 生成的内容可能不正确。

**Step 7: Simulation and Testing**

After initializing the ledger and user accounts, we performed simulation tests by running a Python script (test.py) to simulate transactions under realistic conditions. This script initialized 100 users with randomly assigned reputation scores between 40 and 60.

Run the simulation test script:

python3 test.py

During simulation, each round of transactions included order matching between users and automated chaincode invocation to record transaction outcomes on the blockchain ledger.

