<Project Name>

IOT blockchain

<Functional Spec Title>

Functional Specification of Geth Nodes

Version 0.1

<Author Name>

Chia-yuan Kuo

(<email-address>)

chia-yuan.kuo@sjsu.edu

[Version History](#h.gjdgxs)

[Introduction](#h.30j0zll)

[References](#h.1fob9te)

[Requirements](#h.3znysh7)

[Functional Overview](#h.2et92p0)

[Configuration/ External Interfaces](#h.tyjcwt)

[Debug](#h.3dy6vkm)

[Logging](#h.1t3h5sf)

[Counters](#h.4d34og8)

Implementation

[Testing](#h.17dp8vu)

[General Approach](#h.3rdcrjn)

[Unit Tests](#h.26in1rg)

[Appendix](#h.lnxbz9)

# Version History

|  |  |
| --- | --- |
| **Version** | **Changes** |
| 0.1 | Initial setup for ethereum node setting using Go-ethereum(Geth) |

# Introduction

In this spec, we will demonstrate steps of how to get started with Go-ethereum(Geth) to create a private network. An private Ethereum network is a network where the nodes are not connected to the main network nodes. In other words, the private network refers to isolated network. The private network can be used for running smart contracts.

# References

https://github.com/ethereum/go-ethereum/wiki/Private-network

https://medium.com/@mikesmolenski/getting-started-with-ethereum-4a3841276b6e

# Requirements

First we will set up a private testnet by using a genesis file, which will be used to build our own private blockchain. Then, we start a bootstrap node that each node can use to find each other in our network. After that, we build subsequent Geth nodes that point to the bootnode for peer discovery. Finally, we create an Ethereum miner.

# Functional Overview

Private testnet setup; bootstrap node creation; Geth nodes creation; Ethereum miner creation

## Configuration/ External Interfaces

**Satisfy the computational needs required by the framework.**

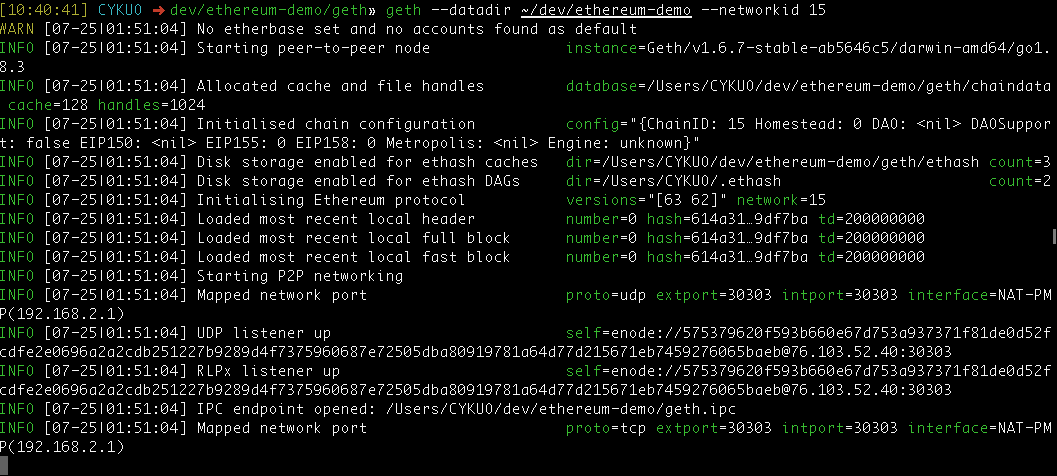
The simulated application may be able to run on a modern desktop/laptop computer, we still need to determine whether it is feasible for it to be run on a low-end device by looking at its CPU and memory consumption.

**Designing and implementing a user interface that is compatible with the framework**

Even though Ethereum framework provides a user interface for developing Dapps, it may not be very straightforward to implement our user interface with it. We need to design our user interface to be compatible with the existing framework while maintaining functionality.

## Debug

We can take advantage of go-ethereum console to debug as follows:



# Implementation

We will use go-ethereum to setup nodes containing smart contracts as follow:

**Installing the client tools**

$ brew tap ethereum/ethereum

$ brew install ethereum

**Creating The Genesis Block**

geth --datadir path/to/custom/data/folder init genesis.json

**Network Connectivity**

bootnode --genkey=boot.key

bootnode --nodekey=boot.key

**Starting Up Your Member Nodes**

geth --datadir path/to/custom/data/folder --networkid 15 --bootnodes <bootnode-enode-url-from-above>

**Running A Private Miner**

$ geth <usual-flags> --mine --minerthreads=1 --etherbase=0x0000000000000000000000000000000000000000

# Testing

## General Approach

After creating the genesis block, we can run a miner to start mining the first block by using the following commands: "miner.start(); admin.sleepBlocks(1); miner.stop();"

Then, we check the ethereum balance by using the following commands: balance = eth.getBalance(eth.coinbase). If the mining result is equal 10000000000000000000, we confirm the node creation successful.

## Unit Tests

Due to Ethereum's novelty, so far we do not find any tools to help on unit testing of the nodes. But, we will keep searching for available tools.

# Appendix

https://github.com/ethereum/go-ethereum

https://medium.com/taipei-ethereum-meetup/%E4%BD%BF%E7%94%A8-go-ethereum-1-6-clique-poa-consensus-%E5%BB%BA%E7%AB%8B-private-chain-1-4d359f28feff