INDIAN INSTITUTE OF TECHNOLOGY. DELHI

Design Document
Assignment - 1

Distributed Ledger

Instructor
Prof. Huzur Saran

TAs
Indu Joshi
Arpit Aggarwal
Bhavesh Sethi
Kartik Mittal

Team-2
Mahima Manik(mcs172093@iitd.ac.in)
Mansi Sharma(mcs172073@iitd.ac.in)
Sruti Goyal(mcs172078@iitd.ac.in)

Index

List of figures

List of tables

Hardware and Software Specifications

Software:

Language used:	Python
Version:	2.7.12
Operating System:	Ubuntu 16.04 LTS
Network Emulator:	Mininet
IDE:	gedit 3.18.3

Hardware:

Memory:	8 GB
Processor:	Intel® Core™ i7-3537U CPU @ 2.00GHz × 4
OS Type:	64-bit
Disk:	225 GB

Prerequisites

1. Mininet should be installed on the Ubuntu system
2. Python 2.7.13 must be installed on the target system

Mininet

We started by creating Single Switch Topology on Mininet. Number of nodes in the topology is taken as input from the user.

Terminal is present by default in Ubuntu, which is also known as gnome-terminal.

For each host in the topology (net.hosts), we open separate xterm window and run the python file named "nodes.py" on each host separately. Xterm in minimalistic but has more advanced features.

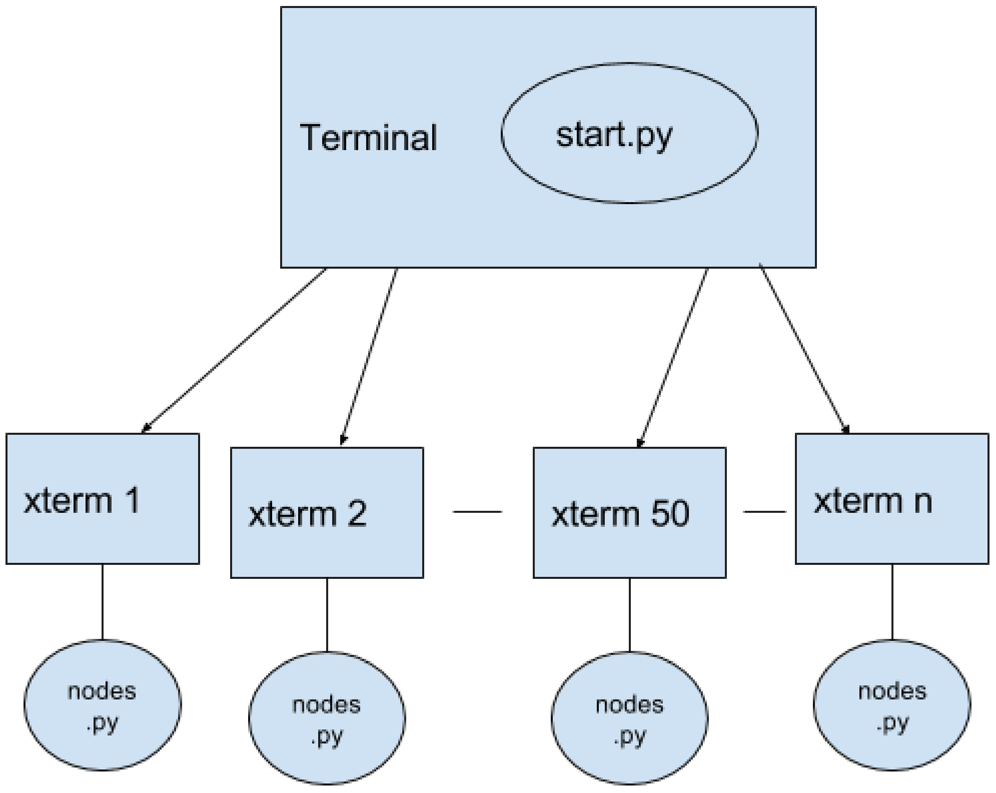


Fig. 1 Running files on Mininet

Each xterm terminal acts as separate host in bitcoin which has its own copy of data members and member functions.

Table 1. Description of data members of Peer

Name of Data members of Peer class	**Type/Data Structure**	**Value**	**Description**
nodeid	int		Unique identity number of the node
MyIP	string	10.0.0.x	IP address assigned to the host in topology
state	int	0: offline/ 1: online	Indicates whether the node is online or offline
ledger	list of lists	[]	Contains all the transaction in the bitcoin system upto now
inlist	list of lists	[]	List of all transactions where host is itself involved
hash_bit	int		Number of hash bits used in DHT which is log(number of nodes)
key	tuple		Pair of public and private key of the host
dht	dictionary	{}	{key:value} is {node id: public key}
ft	dictionary	{}	Routing table to store IP addresses of other hosts
nodes	list	[]	List of all online hosts
serverport	int	49152 - 65535	Port number assigned to host's TCP server
udpserverport	int	49152 - 65535	Port number assigned to host's UDP server
rollbacks	list	[]	List of failed transactions that the host has to roll back

Transaction Module

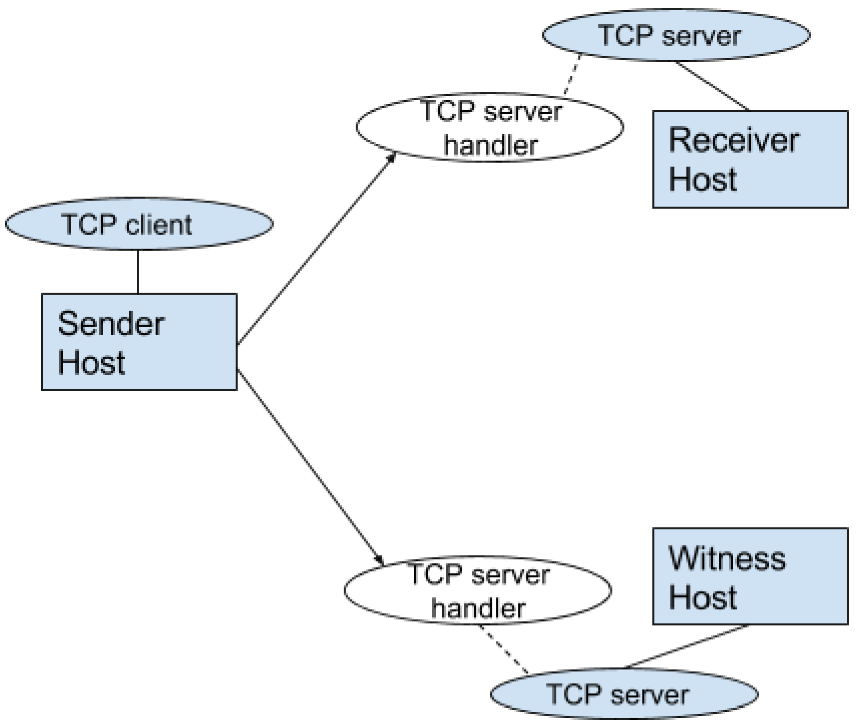


Figure 2. Handling Transactions

TCP and UDP servers are running on each host of the bitcoin system so that it is able to accept messages/requests from its peer hosts.
The sequence of function calls made are as follows:

1. **sender_trans:** It takes receiver's nodeid and number of bitcoins to be send as the input. It is called when a client wants to make transaction. It first verifies the transaction from its own ledger. Witness is chosen at random from list of online nodes. Transaction is then sent for verification at receiver and witness. If both sends back commit message to the sender then the transaction is said to be committed (2-phase commit protocol). If the transaction is not the valid transaction then it is surely aborted. Also, with a random probability either sender, receiver or witness can decide to abort the transaction
2. **check_ledger:** On receiving a transaction, each node will check their ledger to verify the transaction using this function.

3. **update_ledger:** Once all the 3 parties decide to commit the transaction, it is then broadcasted by UDP socket to all the online nodes. On receiving the broadcast, each host verifies the digital signature of the sender and then adds it to its ledger.

4. **abort_transaction:** If the bitcoins of the sender goes below zero for a transaction, then that transaction needs to be aborted at the sender. After aborting, this is broadcasted to all other nodes.
5. **rollbacktrans:** Once the broadcast of an aborted transaction is received at other nodes, they rollback that transaction in their ledgers.
6. **select_function(self, temp):** On receiving a UDP broadcast, the server needs to decide what type of message does the broadcast contains. This is distinguished by the use of flags. Based on the flags, we call different functions to handle it.

Distributed Hash Table

Distributed Hash Table(DHT) is being used for storing the key-value pairs at nodes in a distributed manner. There are various implementations of DHTs available, some of them being Chord, Pastry and Kademlia. We are using Chord implementation. We are using DHT to store public key of each node that needs to be accessed by all other nodes while verifying a transaction made by the node. In our assignment, key is the node id of each node and value is its public key.

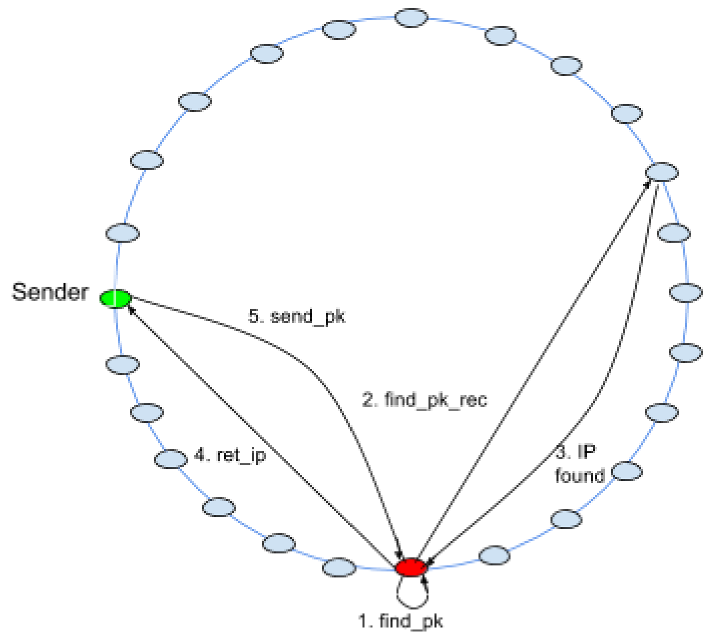


Fig 3 . Obtaining public key of the sender

The following are the modules that are being used for the implementation of DHT:

1. **create_ft(self):** When a host is created, this module creates a finger table(routing table) for it.
2. **update_ft_ledger(self):** Whenever a node comes online or goes offline, this function updates the finger table of all the nodes and the ledger of the node that came online.
3. **find_pk(self, nid):** When a host has to verify a transaction, it requires the public key of the sender(nid). It starts to search the IP address of the sender in its finger table. If it gets the IP address in its table, it sends a udp unicast message to the sender to send the public key stored in the sender's DHT. If it doesn't get the IP address of the sender, it sends a request to the node preceding the sender whose IP address is in its routing table(def find_pk_rec).

4. **find_pk_rec(self, xip, nid):** When a node gets a request from another node(xip) to search the IP address of the sender(nid), it searches its IP address in its finger table. If it gets the IP address in its table, it sends a udp unicast message(def ret_ip) to the requesting node(xip) with the IP address of the sender(nid). If it doesn't get the IP address of the sender, it recursively sends a request to the node preceding the sender whose IP address is in its routing table(def find_pk_rec).
5. **ret_ip(self, nip):** When a node receives the IP address of the sender from another node to which it has requested, it sends a udp unicast message to the sender to send the public key stored in the sender's DHT.
6. **send_pk(self, temp):** When a node receives a request for its public key, it sends as reply its public key as a udp unicast message.

States of hosts

1. **randomoffon(self):** Each host decides to go offline with a random probability. To ensure this we run a thread that will run on background and generate a random number based on which the host decided to go offline or remain online.

2. **node_went_offline(self, t):** When a node (t) goes offline, it inform all other nodes by a UDP broadcast message. This function updates the finger table and list of online nodes accordingly once it receives the broadcast.

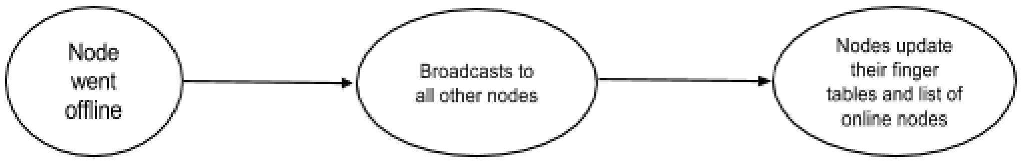

Node went offline
Broadcasts to all other nodes
Nodes update their finger tables and list of online nodes

Fig 4. Node Went Offline

3. **node_went_online(self, t):** When a node (t) comes online, it inform all other nodes by a UDP broadcast message. This function updates the finger table and list of online nodes accordingly once it receives the broadcast. It also calls the update_ft_ledger() function for the node that came online.

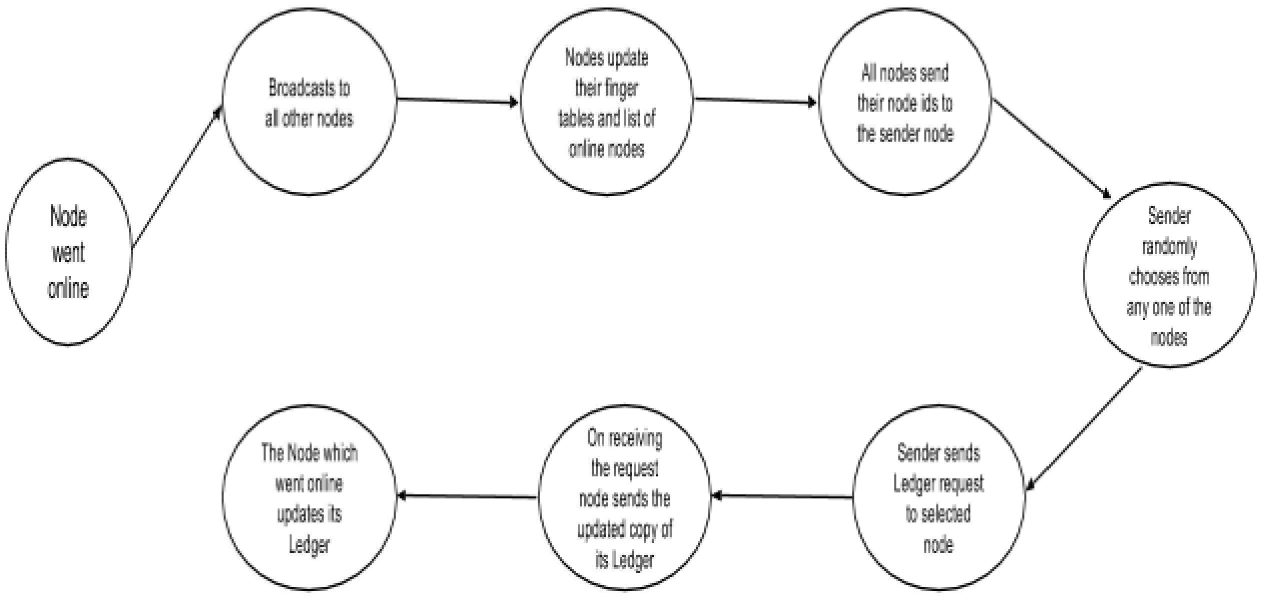

Node went online
Broadcasts to all other nodes
Nodes update their finger tables and list of online nodes
All nodes send their node ids to the sender node
Sender randomly chooses from any one of the nodes
Sender sends Ledger request to selected node
On receiving the request node sends the updated copy of its Ledger
The Node which went online updates its Ledger

Fig 5. Node Went Online

Double Spend

We have successfully taken care of the fact that the same amount of bitcoins is not spent in more than one way. To demonstrate the behaviour of system under any such attempt, we have created similar situation using threading.
One host simultaneously want to send all its bitcoins to some other host and itself too. In such situation, one of the two transactions is rolled back.

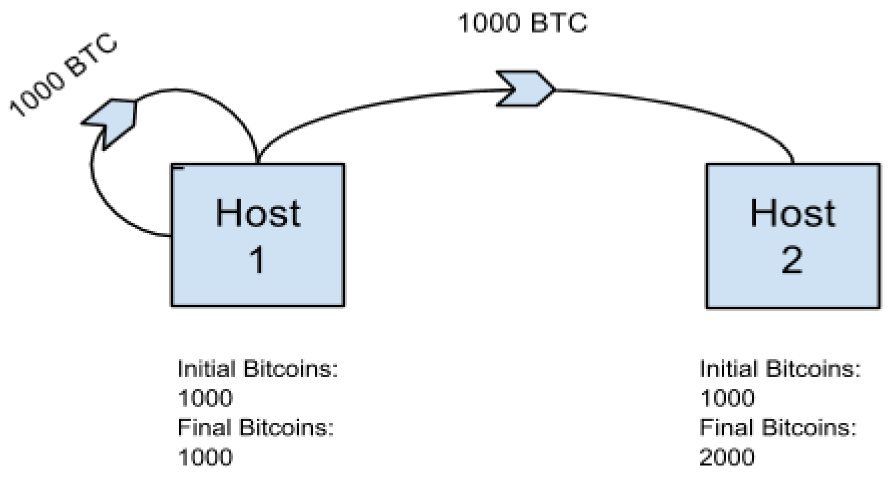


Fig 6. Double Spend Problem

abort_transaction: If, while updating its own ledger, sender finds its number of bitcoins is getting lower than 0, this function is called. It broadcasts the transaction to be aborted to all the other online nodes in the system.

rollbacktrans: On receiving the broadcast, each host removes it from its ledger. If the transaction has not yet been updated in its ledger, then it keeps it in a list called rollbacks. Whenever a new transaction is added, it is checked against rollbacks, to make sure aborted transactions are not added in the ledger.

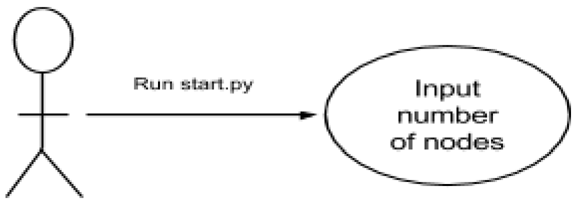
Main

Once the program runs, the user is asked for the number of nodes to be created. Based on that our topology is formed.

Then each host is given two options:
1. To make a transaction with any other host by giving the host name (like h1)
2. To check, at any point of time, if it's ledger is consistent with the ledger of all other nodes in the bitcoin system.

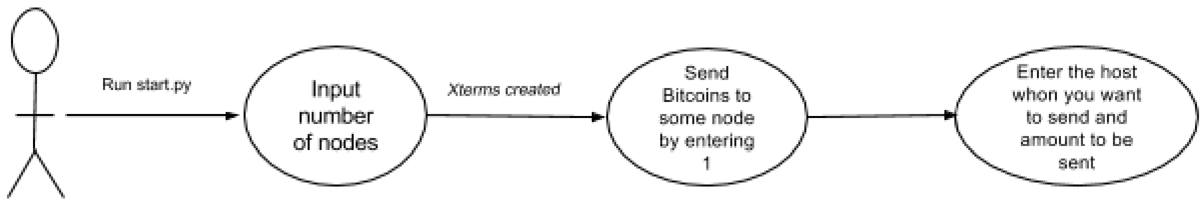
Use Cases

User runs start.py.



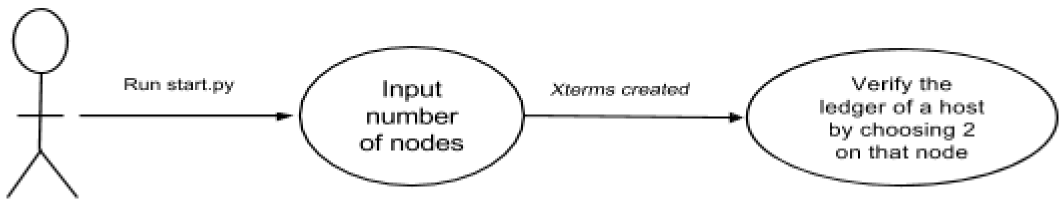
Option A:

If user wants to send bitcoins, he should choose option 1 and enter host whom user wants to send and the amount to be sent. We make sure that the host name is in the form h1, h2 etc. and the amount is greater than 0. This is done using regular expression.


Run start.py
Input number of nodes
Xterms created
Send Bitcoins to some node by entering 1
Enter the host whon you want to send and amount to be sent

Option B:

If user wants to verify ledger of some host, he should choose option 2 on the host whose ledger he wants to verify.



REFERENCES AND BIBLIOGRAPHY

1. Mininet Walkthrough: http://mininet.org/walkthrough/
2. Python 2.7 documentation
3. Distributed Systems: http://privateweb.iitd.ac.in/%7srsarangi/docs/dist/ft.pdf
4. Chord Implementation: http://www.cse.iitd.ac.in/~srsarangi/csl860/docs/chord-lec.pdf
5. Explanation of Bit Coins : https://www.youtube.com/watch?v=Lx9zgZCMqXE
6. UNIX® Network Programming Volume 1, Third Edition: The Sockets Networking API By W. Richard Stevens, Bill Fenner, Andrew M. Rudoff