TABLE OF CONTENTS

OVERVIEW

1. The assignment **“Distributed Ledger”** has been made using **Java programming language** aimed to run in a simulated network environment such as **Mininet**.

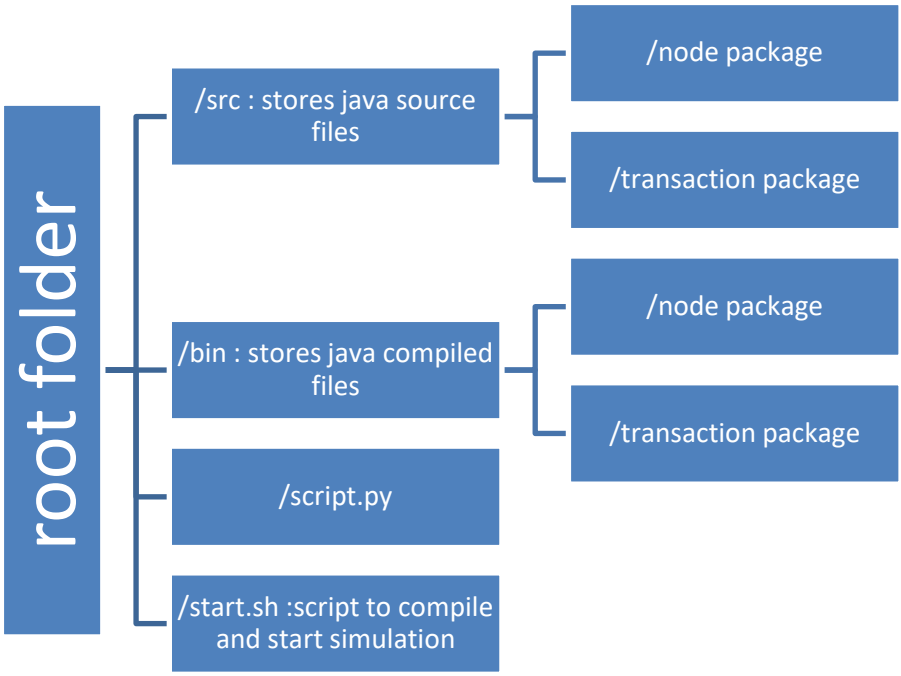
2. The communication among different instances of Java program has been achieved using **TCP and UDP socket through socket API**.

3. **Mininet API in python** has been utilized to simulate the network as required, allowing dynamic addition and deletion of terminals.

System Requirements:

- Linux Operating System
- Java SE 8 environment
- Mininet network simulator
- Python 2.7

PROJECT STRUCTURE



HOW TO RUN

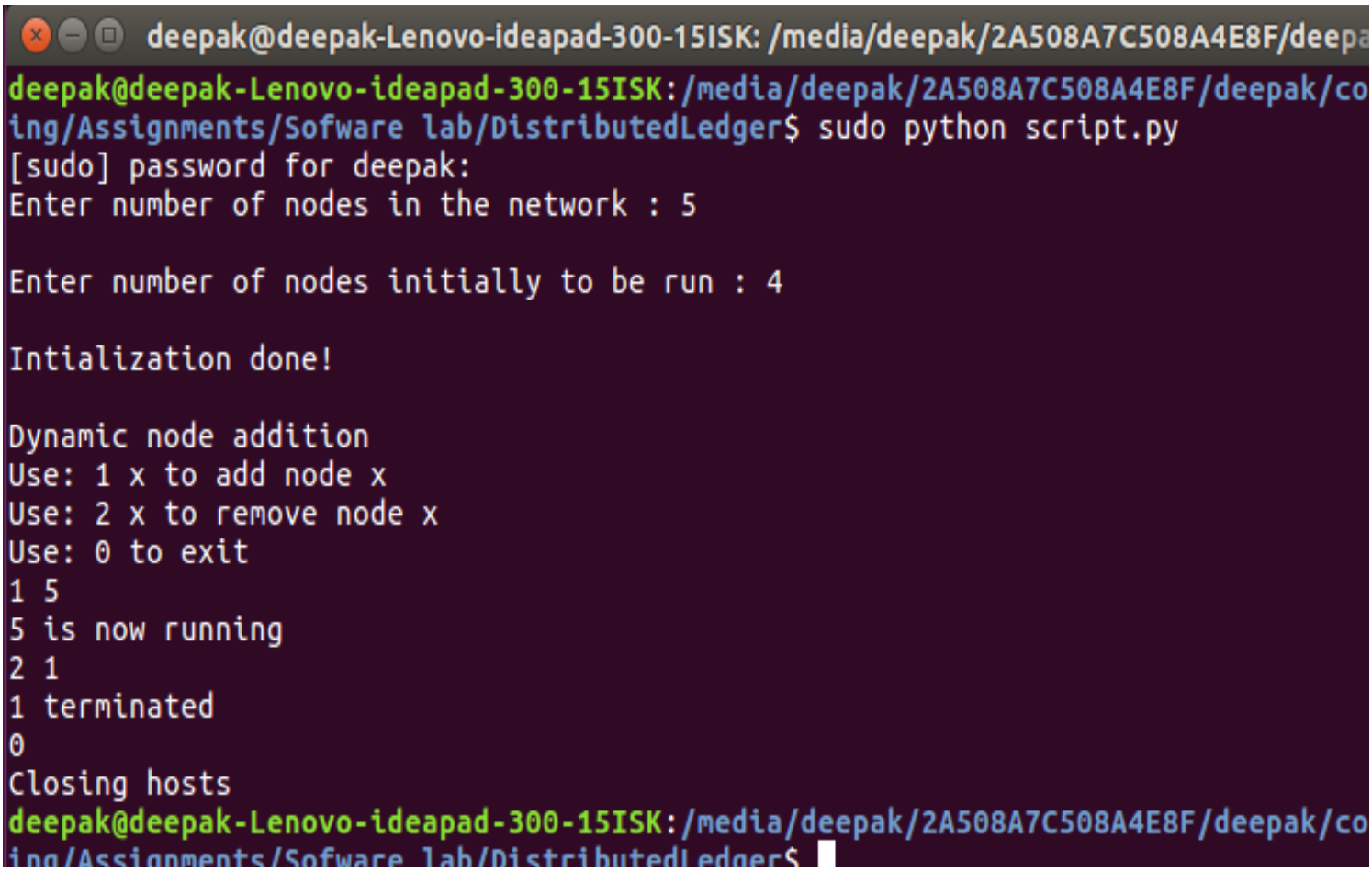
1. Compile java source files (in /src) (If not already present) using start.sh

2. Run python script script.py or start.sh shell script to start the simulation

3. Enter the network details as follows:

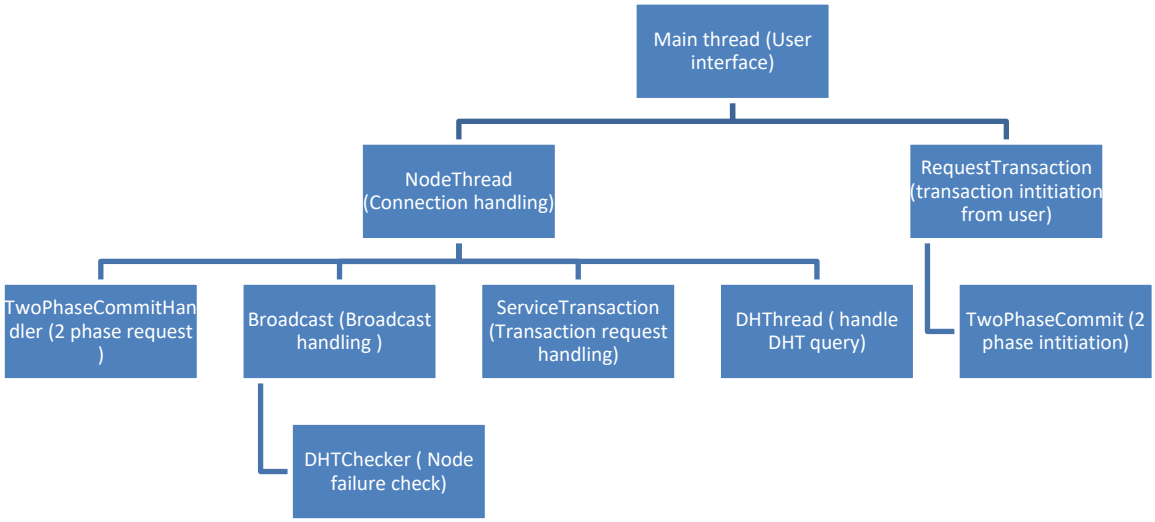
    a. Enter total number of nodes in network
    b. Enter number of nodes to be initially run in the network
    c. Dynamically add/delete nodes using the commands as displayed in user interface.

Example:



THREAD HIERARCHY

1. The below diagram displays the connections and interactions that happen among different threads that are created by the process.



DESIGN STRUCTURE

CLASS NAME	MAIN DATA MEMBERS/METHODS	DESCRIPTION
node.Broadcast	Handles broadcast functionalities	
	mailBox	Datagram Socket for UDP communication
	queue	Queue to hold transaction messages that cannot be yet added.
	run()	Receive messages and perform appropriate actions for intialisation and ABCAST multicast mechanism
	putMessage()	Send message as UDP datagram to required node
	broadcastMessage()	Broadcast message as UDP datagram
	broadcastTransaction()	Broadcast transaction encapsulated as message
node.DHT	Handles DHT functionalities	
	dHTableHashMap	Hash table that stores local keys
	buildDHT()	Build chord DHT ring from current nodes in network
	findNode()	Find node in DHT ring responsible for the given key
	putValue()	Put key,value pair in DHT
	getValue()	Get value corresponding to key in DHT
node.DHTChecker	Periodically checks successor of a node . If successor down it broadcasts node failure in network	
	run()	Implements above functionality

node.DHThread	Returns DHT query result through a socket connection	
	run()	Implements above functionality
node.DriverProgram	Starting point of execution (main thread) and provides user interface at a node	
	main()	Implements above functionality
node.Message	Structure that represents messages exchanged in ABCAST broadcast mechanism	
	TYPE	Type of message
	deliverable	Boolean flag to denote transaction in message is ready to add to transaction list.
	timestamp	Global timestamp for message
node.NodeThread	Thread that handles connections for DHT and Transaction processing and stores node specific information	
	threadMap	A hash map of (node,IP of node) pairs for all current nodes.
	pubKey,priKey	Public and private key for node
	run()	Handles connections and takes appropriate actions.
node.Request	Structure to represent DHT,Transaction and intialisation requests	
	requestCode	Type of Request
node.RequestTransaction	Intiates connection to receiver and witness nodes for a transaction in Two Phase commit protocol	
	run()	Implements above functionality
node.Security	Provides SHA-1 hash, public/private key generation and digital signature creation and verification	
	getHash()	Return SHA-1 hash of given string
	getPrivate/PublicKey()	Returns private/public key

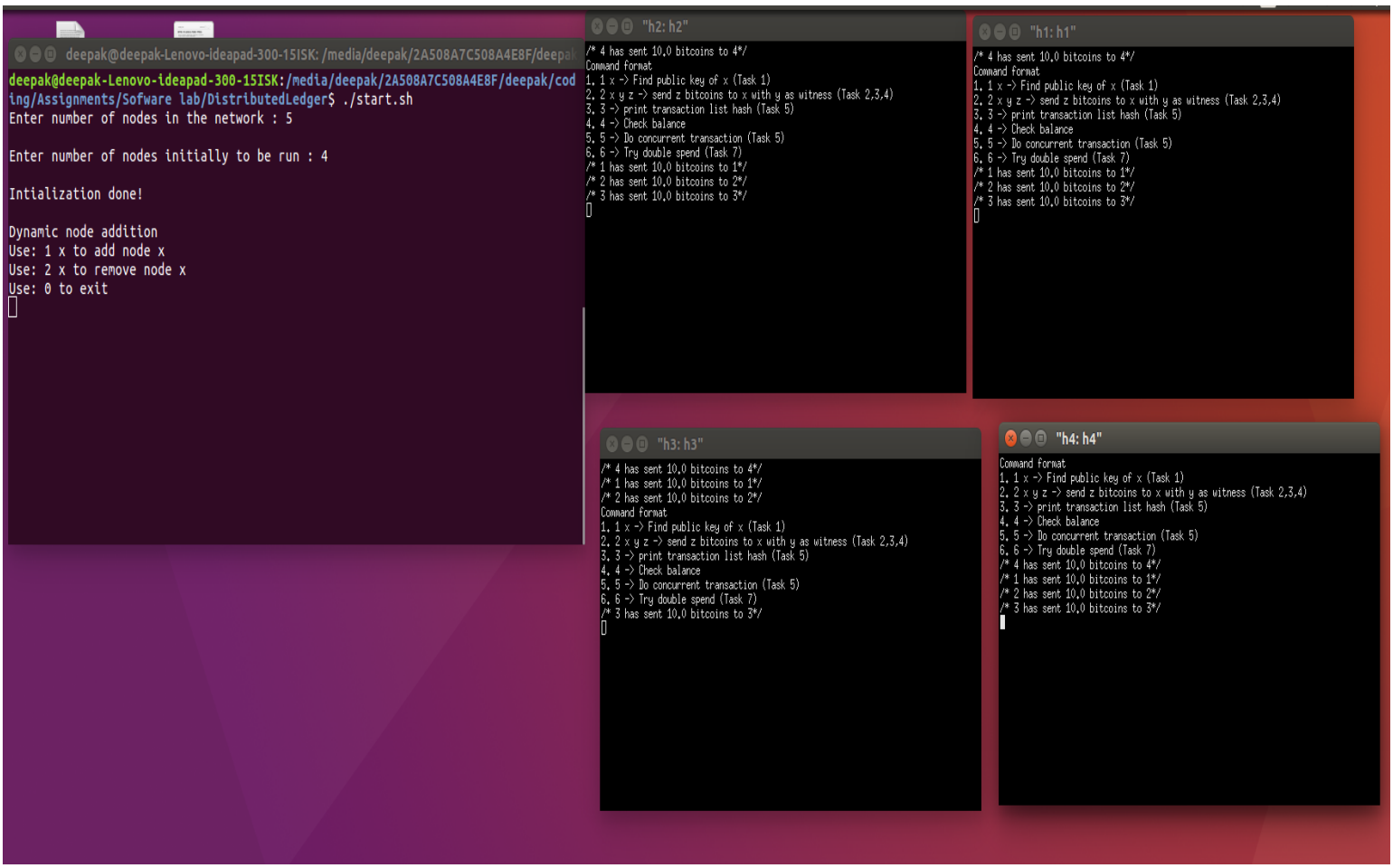
	create/verifySignature()	Create and verify signature
node.ServiceTransaction	Verify and add transaction	
	run()	Implements above functionality
node.TwoPhaseCommit	Simulates two-phase commit protocol (sender side)	
	run()	Implements above functionality
node.TwoPhaseCommitHandler	Simulates two-phase commit protocol (receiver/witness side)	
	run()	Implements above functionality
node.TwoPhaseProtocol	Structure for two-phase commit protocol messages	
	MessageCodes	Enum class for type of two-phase commit message
transaction.Input	Structure to represent an input for a transaction	
transaction.Output	Structure to represent an output for a transaction	
transaction.Transaction	Structure to represent transaction.	
	txnId	Transaction Id (globally ordered)
	inputList	List of inputs
	outputList	List of outputs
	digitalSignature	Signature of sender
transaction.TransactionManager	Class to create and store transactions	
	transactionList	List of transactions done so far.
	addTransaction()	Add transaction to list
	createTransaction()	Create a transaction with parameters
	verifyTransaction()	Verify transaction before accepting
	getHashCode()	Return hash of transaction list
transaction.UTXO	Structure to keep track of unspent transactions	

	Output List	Unspent outputs of transactions
	Add To Unspent Txn List ()	Add a transactions' output to unspent
	Check If Unspent ()	Checks if a transaction output unspent

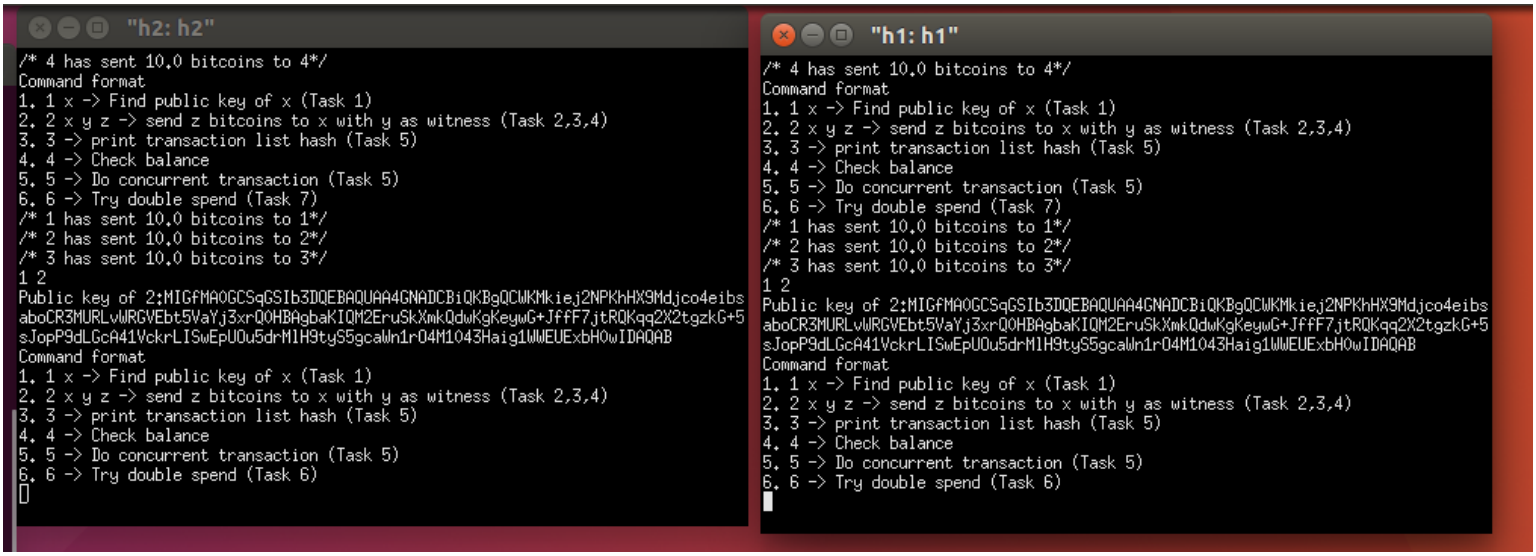
SAMPLE DEMONSTRATION

1) ***Network Creation***: A network consisting of 5 nodes is created, where initially 4 nodes are running. After a few seconds, all nodes learn the network and initial transactions.

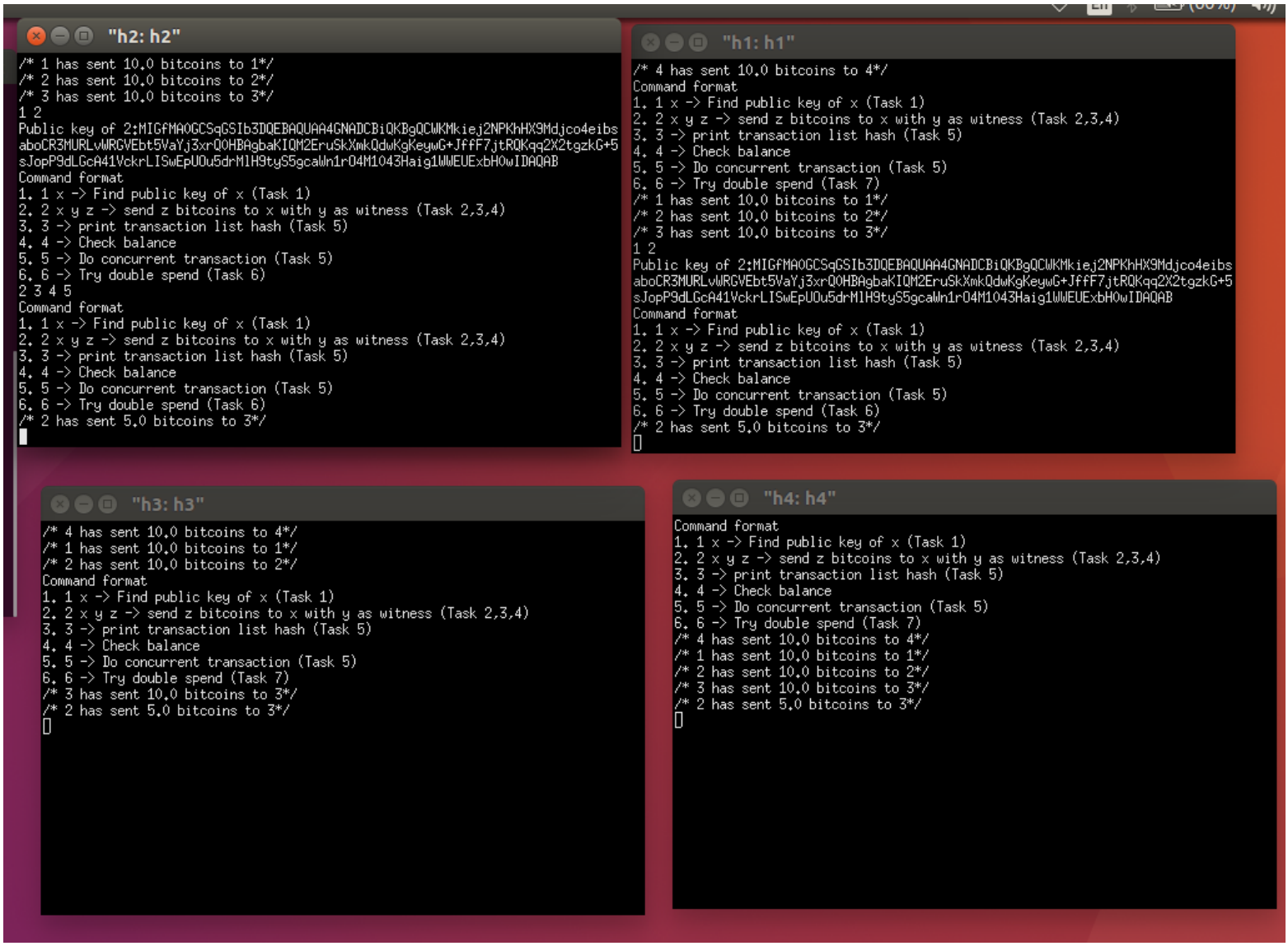
All nodes display a command menu.



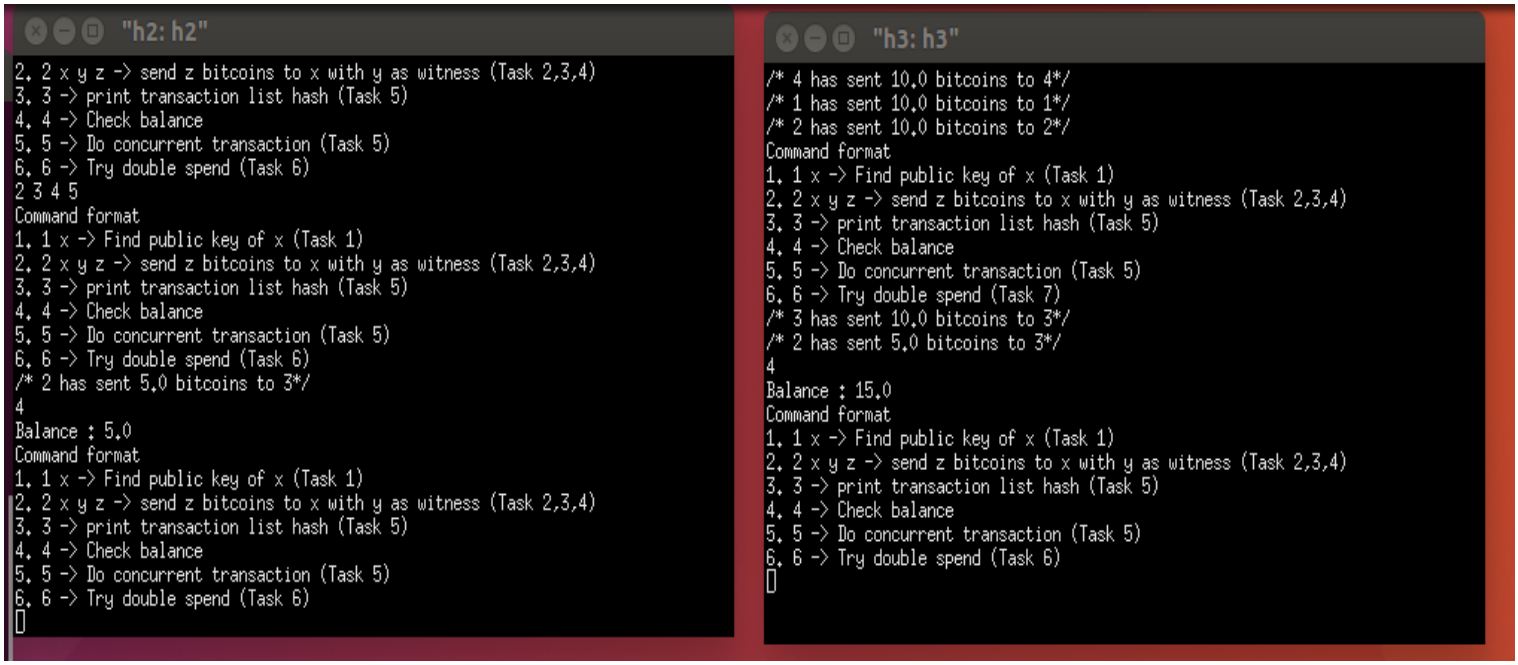
2) ***DHT query (Task 1)***: Public keys of nodes are queried using command ‘1 x’.



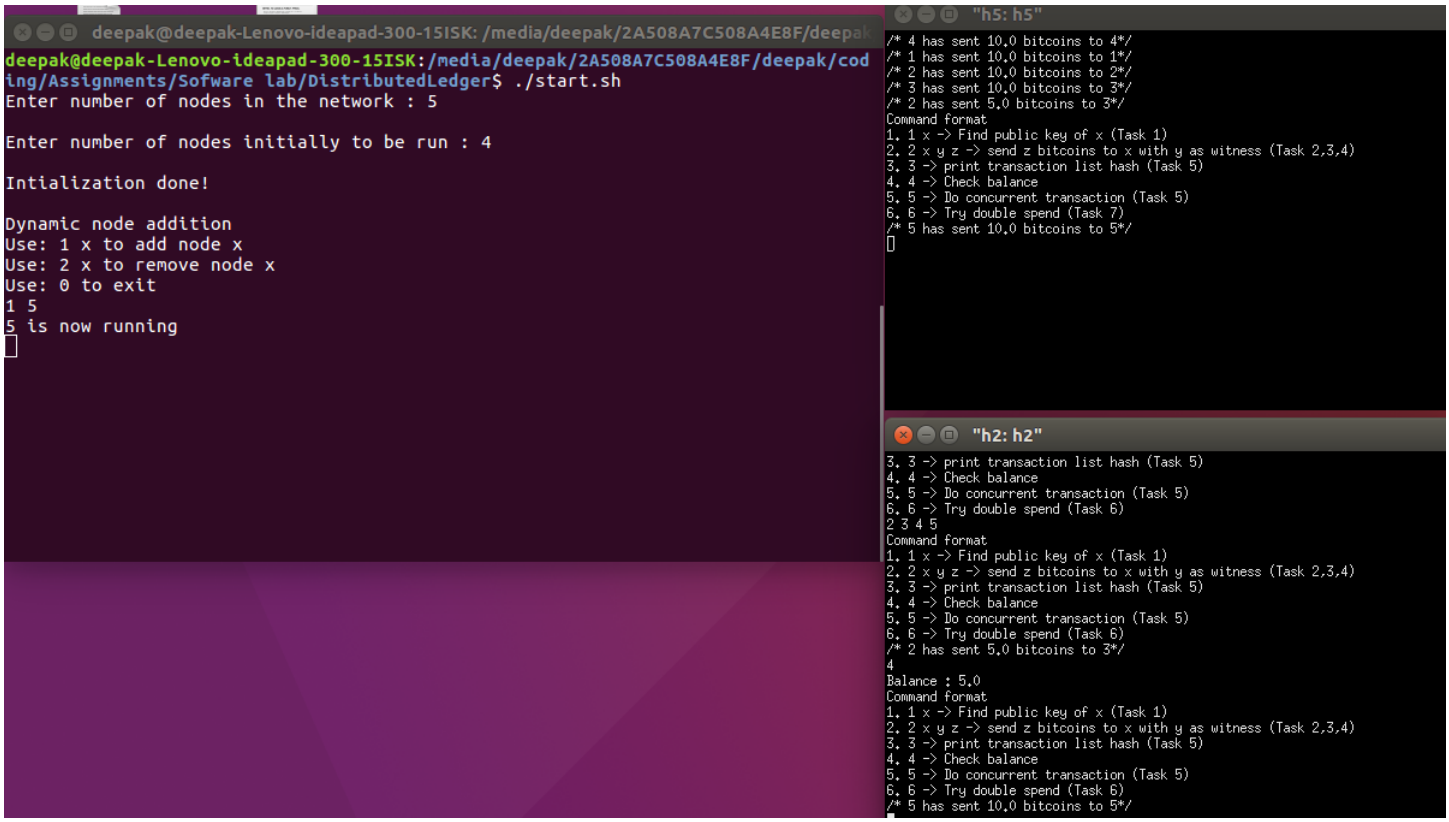
3) ***Executing bitcoin transaction (Task 2,3,4)***: A transaction is carried out on node 2, sending 5-bit coins to node 3 with node 4 as witness. All nodes then display a pop up regarding confirmation receipt of transaction.



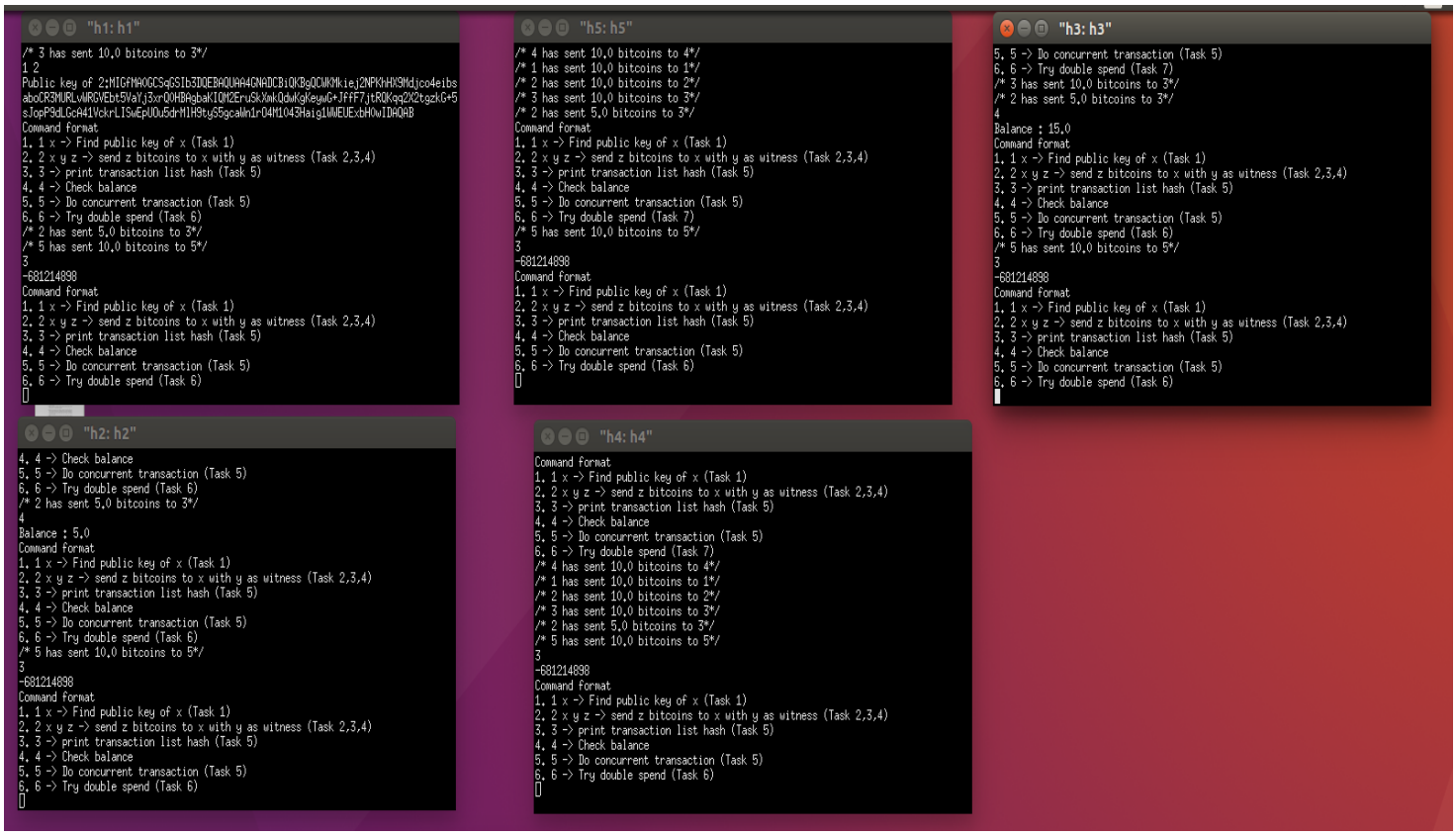
4) ***Balance check***: Reflects the change in bitcoins owned by 2 and 3 after above transaction.



5) ***Adding a new node***: The new node added learns the network and transactions done till now, and creates its own initial transaction.

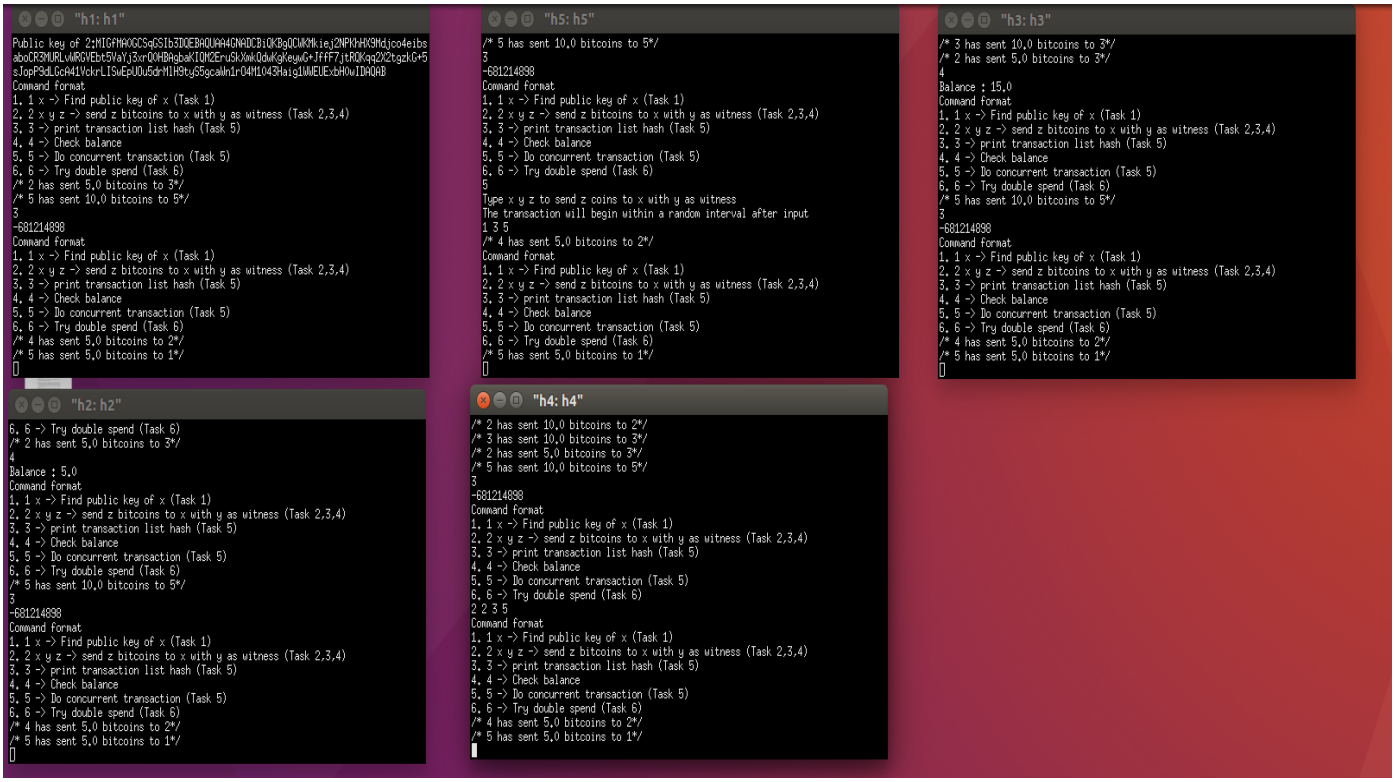


6) ***Display transaction list hashes (Task 5)*:**

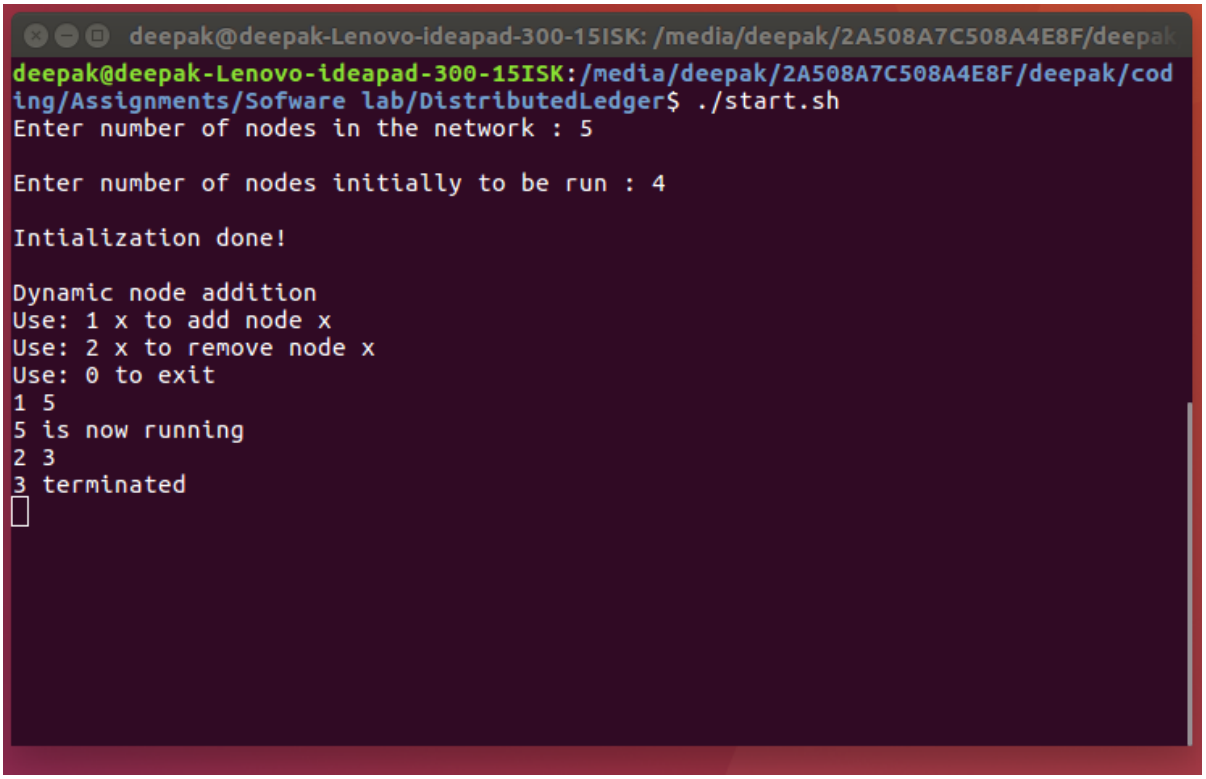


7) ***DO A CONCURRENT TRANSACTION (TASK 5)*:** To simulate a concurrent transaction, command 5 allows user to specify a transaction that will be initiated in a random interval before 5 secs. This allows user to move to different terminal and specify a transaction that can happen concurrently with the first one.

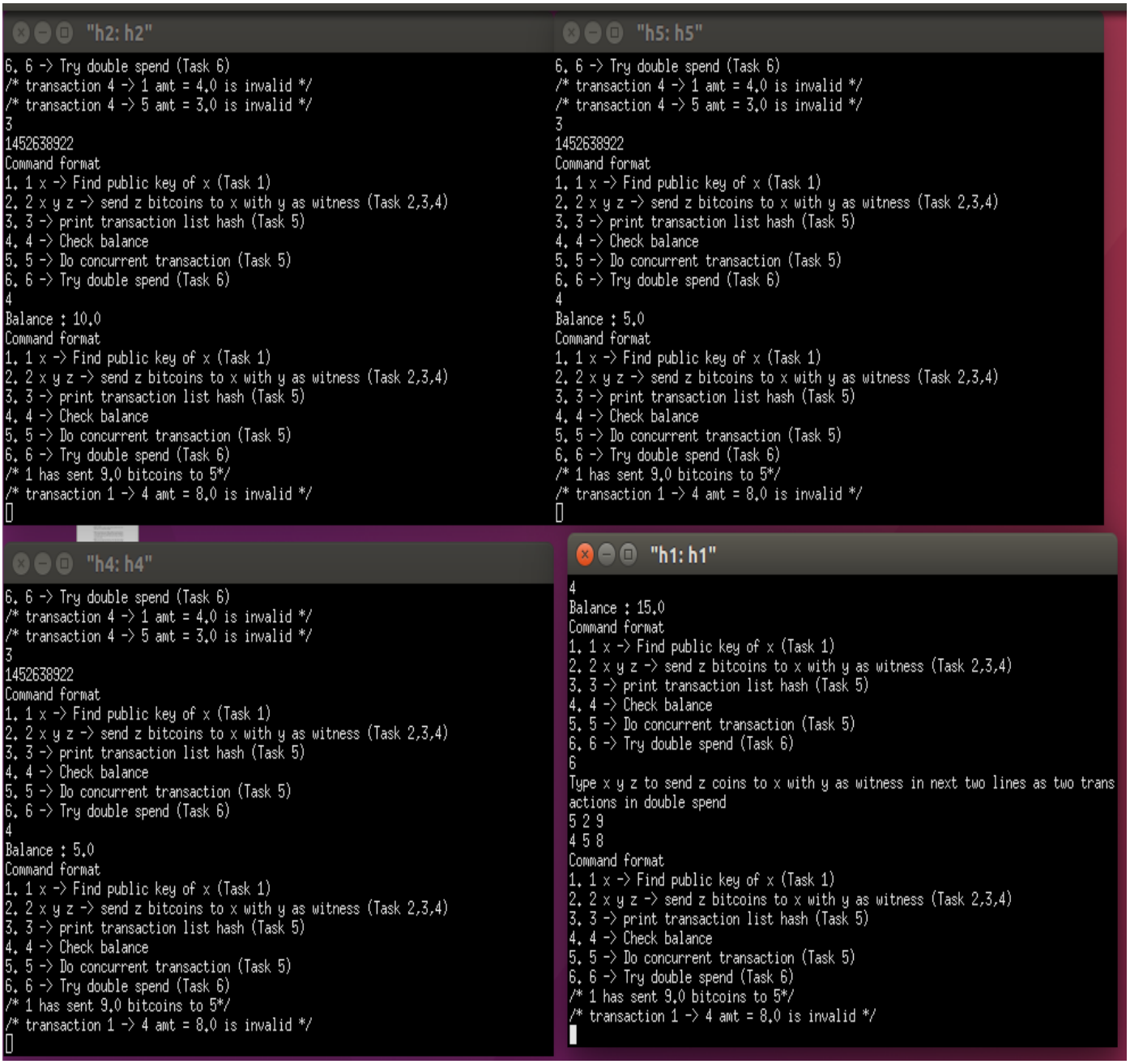
In following example, on terminal of node 5, a transaction “send 5 bitcoins to 1 with 3 as witness” is specified using command 5. At the same time, a transaction “send 5 bitcoins to 2 with 3 as witness” is specified at node 4. All nodes hence perceived the same order of transactions.



8) ***Removing a node***: Node 3 was removed.



9) ***Trying a double spend (Task 6)***: A double spend is tried at node 1 with two transactions “send 9 bitcoins to 5 with 2 witnesses” and “send 8 bitcoins to 4 with 5 as witness” using command 6, which bypasses local check for transaction validity. Since earlier balance was 15 coins, only one of the two transactions can happen and in this case first transaction happened first, thus second was not accepted.



10) ***Display transaction list hashes (Task 5)*:** Before simulation termination, transaction list hashes are displayed.