CS 39006: Computer Network Lab

Final Exam 5th April 2018 (2:00 PM)
Time: 2 Hours
Full Marks: 150

Instructions (Read very carefully):

1. The exam is for 2 hours. **At the end of the exam, you need to zip your code, and upload it to Moodle. If you fail to do it by the time Moodle timeout expires, we will not accept your submission under any circumstances. No justification will be entertained. Also ensure that you have uploaded the correct file. You'll not be given any second chance.**

2. The problem statement may look little lengthy ☺, but that is to give you sufficient details. The actual implementation should not be very complicated if you proceed methodically.

3. There are two parts of the question. PART A describes the basic idea about the problem. PART B gives the implementation details. Read both the parts very carefully.

4. **The function details, that you have to implement, is given in PART B. You need to write those functions correctly fulfilling the objectives to get the partial marks allotted for those functions.**

5. **The code should be properly commented**, so that a third person can understand your code.

6. You have to implement the entire application **using UDP sockets only**. We are assuming a scenario of a reliable and synchronous communication platform, so you should not face problems for flow control, congestion control etc. (more details are inside).

7. The marking distribution for different parts of the problem is also given at the end. We'll strictly follow this distribution. You may not get partial marks for partial implementation of a function.

8. The marks for correct compilation and correct execution will be given only if there is NO ERRROR. No partial marks will be awarded here.

PART A: Basic Concepts:

The objective of this problem is to develop a public ledger application (a precursor of Bitcoin!) over a peer-to-peer overlay network. A public ledger is a list of transactions which is available to all the peer nodes in the network. An example public ledger can look like as follows.

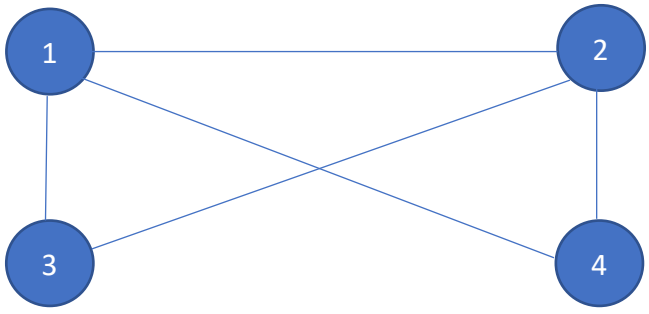
```
020418125532    A    B    50
020418125824    C    A    30
030418133235    D    B    90
030418145612    A    D    100
```

The first field is a timestamp (in DDMMYYHHNNSS format, D: Day, M: Month, Y: Year, H: Hours, N: Minutes, S: Seconds) This indicates that A has made a transaction of ₹50 to B on 02/04/2018 at 12:55:32 hrs and so on. The copy of the same ledger is available to all the peers.

A peer to peer overlay network looks like as follows. Assume that there are a set of nodes that form an overlay peer-to-peer network architecture among themselves. The overlay connections are logical peer connections among the nodes, indicating that two nodes can directly communicate with each other if they have a peer relationship. Consider a peer relationship A--B--C of three nodes, where (A, B) and (B, C) have peer relationships; therefore, A can directly send a message to B or vice versa, and similarly, B can directly send a message to C or vice versa. However, if A wants to send some message to C, the message needs to be transferred via B. Consider that you have a topology file `topo.conf` which contains the network topology for an overlay network. The structure of the topology file is as follows.

```
1    2    3    4
2    1    3    4
3    1    2
4    1    2
```

This indicates that Node 1 has a peer relationship with Nodes 2, 3 and 4; Node 2 has a peer relationship with Nodes 3 and 4, and so on. The structure of this overlay topology is as follows.



Say, we want to develop a distributed commitment protocol over this architecture, so that a new transaction gets committed to the public ledger available to all the peers. Any node in the network can initiate a transaction. Once a node initiates a transaction, the transaction information is broadcast to the peers (over UDP messages) through TRANSACTION messages that looks like as follows.

```
020418125532     A     B     50
```

TRANSACTION messages are propagated across the network so that every peer in the system can hear every transaction message. This is done as follows.

(a) Once a node hears a TRANSACTION message, it resends the TRANSACTION message to all its peers except the one from which it has received the message.
(b) If a node has seen a TRANSACTION message already (verify by the timestamp, transaction originator and transaction destination), it does to resend the TRANSACTION message again. This prevents looping of the TRANSACTION messages in the network.

Every peer validates the transaction against the existing public ledger and applies a voting-based consensus mechanism to accept the transaction and include it in the public ledger. The transaction validation and consensus mechanism are as follows.

We have three assumptions here,

1. There can be some nodes in the overlay network who are malicious. They may give a negative vote for a valid transaction and a positive vote for an invalid transaction.
2. The network is synchronous and reliable, meaning that messages are received always without error, and also within a predefined time. ***This indicates that the maximum delay for a message is finite and bounded.***
3. During the initiation, every node has ₹100 with it.

<u>Transaction Validation</u>

Validation is done against the existing public ledger, and the algorithm is simple. Consider that the existing public ledger is as follows.

```
020418125824     C     A     30
030418133235     D     B     90
030418145612     A     D     100
```

Say, A wants to make a transaction of ₹50 to B. This transaction is invalid. Because, A had ₹100 with her initially, and then she had received ₹30 from C, so the total money that A has is ₹130. After that, she has transferred ₹100 to D, so only ₹30 is left now. Therefore, she cannot transfer ₹50 B, and the proposed transaction is invalid.

Consensus

The consensus mechanism is based on voting. Every node creates a vote message that looks like as follows.

```
C     VOTE  030418145612 A   B     50    N
```

The first gives the ID of the node which is giving the vote. The second column signifies it is a VOTE message. The third, fourth, fifth and sixth columns represent the transaction, and the last column represents the vote – N for NO and Y for YES. The above example indicates that C is giving a NO vote for the transaction A->B: ₹50.

After receiving a transaction, a node waits for 100 seconds to receive the vote messages for that transaction. The vote messages are also sent via UDP messages. A transaction is included in the public ledger only if more than 50% of the received votes are YES votes.

(c) Once a node receives a VOTE message, it resends the VOTE message to all its peers except the one from which it has received the message.
(d) If a node has seen a VOTE message already (verify by the timestamp, transaction originator and transaction destination), it does to resend the VOTE message again. This prevents looping of the VOTE message in the network.

PART B: Implementation Idea:

Consider a node as a process running in your machine. Every process has an ID which is the node ID as well. Node ID can be an integer number. The **`topo.conf`** file is available to every process so that they can know their peers. Every process should run a UDP server socket to listen for the incoming messages. There are two types of messages – TRANSACTION messages and VOTE messages.

You may also include the IP address and the Port number through which every node is listening for the incoming messages. So a final `topo.conf` file looks like as follows.

```
1    10.0.0.1   8111  2    3    4
2    10.0.0.1   8112  1    3    4
3    10.0.0.1   8113  1    2
4    10.0.0.1   8114  1    2
```

So, the final format is <node ID> <server IP> <server port> <Peer List>.

The system initiates with every node having ₹100 in their wallet. Write a function **`sys_init()`** to initialize the system with a local public ledger.

Therefore, you should have four open terminals if you have four peers. Initiate the transactions from the terminals through a command TRAN, as follows.

```
@A> TRAN B 20
```

So, here A initiates a transaction of ₹20 to node B. The process constructs the transaction message and broadcast in the peer-to-peer network through a function **`initiate_transaction()`**. You can use system timestamp to timestamp a transaction.

Every process, once it receives a TRANSACTION message, does the following tasks.

1. Implement a function **`receive_transaction()`** to receive transactions from other peers. Start the consensus timer for 100 secs once you receive a new transaction. Consensus timers are specific to every transaction.
2. Broadcast the message again following the loop free principle as discussed earlier. Implement a function **`broadcast_transaction()`** for this purpose.
3. Validate the transaction against the already existing public ledger. Implement a function named **`validate_transaction()`** for this purpose.
4. Vote against or for a received transaction. Implement a function **`vote_transaction()`** for this purpose.
5. Wait and receive voted for an already broadcast transaction. Implement a function **`receive_vote()`** for this.
6. Store the vote count in a local variable and rebroadcast the VOTE message to the peer to peer network following the loop free principle, through a function **`broadcast_vote().`**
7. Finally, once the 100 sec consensus timeout occurs, execute a function **`reach_consensus()`** to calculate the received votes, and based on 50% rule,

accept or reject a transaction. If a process accepts a transaction, then it is included in its locally available public ledger.

Malicious node implementation: Some of the nodes in the system work like a malicious node. The malicious nodes change their own view of the transaction validity with a probability 0.3. This indicates that they mark a valid transaction and invalid, or vice versa with a probability 0.3. Note that they do this only on their own VOTE messages, but do not make any change in the VOTE messages from others. Once a malicious node receives a VOTE message from a peer, it processes the message and rebroadcast it to other peers without making any change (or drop it if it has already rebroadcast the same message).

At any moment, you can print the public ledger available at a node by following command.

```
@A> VIEWLEDGER
```

Marks Distribution:

Operation	**Marks**
`sys_init()`	5
Correct implementation of peering	10
`initiate_transaction()`	10
`receive_transaction()`	5
Consensus Timer Implementation	10
`broadcast_transaction()`	10 (deduct 5 if it is not loop free)
`validate_transaction()`	5
Malicious node implementation	10
`vote_transaction()`	10
`receive_vote()`	10
`broadcast_vote()`	10 (deduct 5 if is not loop free)
`reach_consensus()`	20
Print ledger	5
Proper commenting of the code	10
Compilation without error	10
Correct execution without error	10