CS 5433 HW2 Solution

Disheng Zheng	Zhu Xizi	Trishala Neeraj
dz336	xz573	tn338

Problem 1.2

I tried to run 6 nodes and also 3 nodes. During and after height 100. All nodes appear to be synchronized.
I then ran a 3 nodes blockchain. And I have interrupted node 3 and resume after a while. It appears that node 3 is no longer synchronized after. It suggests that our gossip protocol need to add an update mechanism to maintain a blockchain, meaning nodes on our gossip protocol should be updated and synchronized with the whole chain history when they join back into the chain. Perhaps gossip every message instead of only the current one.

Problem 1.3

It's not a permissionless blockchain, Because the size of the user poor is fixed and all users have a designated id. Nodes can not join and being removed freely.
Bitcoin Gossip Protocol suggests addition of addr message type, where providing information on known nodes of the network. Non-advertised nodes should be forgotten after typical 3 hours.

Problem 2

I have tested 2, and 3 nodes network where node 1 is always included. I have observed that nodes' logging output are all synchronized.

Problem 3

I ran 2 nodes (node 1 and 6). Then I ran start_ba.py and observed that they are in Byzantine Agreement:

```
disheng cs5433_hw2 $ python3 start_ba.py
Yay!
Yay!
disheng cs5433_hw2 $
```

Problem 4

(BFT to consensus)

(a) P*rovide two appropriate functions for choosing the sender/leader of the above BA protocol* Random

1) Sender = median(H(e, i)) (compute the hash for the players and get the median) where i is the player number
Reasons it achieves consensus we want:

It helps to find a honest sender with random chance because once a honest sender is selected (and it will be selected eventually), the BA above will be able to output the right message, assuming more than 50% of the players are honest. In addition, faulty player cannot just find the honest player and corrupt it.

2) Sender = H(e,i) < D
Reasons it achieves consensus we want:

It helps to find a honest sender at random because once a honest sender is selected (and it will be selected eventually), the BA above will be able to output the right message, assuming more than 50% of the players are honest. In addition, faulty player cannot just find the honest player and corrupt it.

(b) *What are the qualitative differences between these functions?*

There is no randomness in the first case: while the first one is guaranteed to find a player as the sender, the second has only certain probability (mentioned in the notes) to get a sender (sometimes it does not have one).

Problem 5

1.(Dolev-Strong)

(a) *Show that this protocol still satisfies Validity.*

Similar to proving that of BA-PKI, if the sender is honest, it only signs 1 message *M*. Then, since honest players only add a message *M* to their set *S* with sender's sig. Using this protocol, it has unforgeability of *M*. So m is the only message that can be added to other voters' set. Since after round 1, every honest player has added *M* to its set *S.* It will output *M* (even with two faulty players). Thus this proved validity.

(b) *Show that this protocol does not satisfy Consistency w.r.t. 2 faulty players.*

If there are 2 honest player: 1,2; 2 faulty players: 3,4
Round 1:
Say player 3 is the sender and send player 1 ,2 *m1* and player 3 *m2*
1 *m1_pks s1:m1*
2 *m1_pks s2:m1*
3
4 *m1_pks_pk4, m2_pks_pk4 s4:m1, m2*
Round 2:
Player 4 sends m1 to player 1, and m2 to player 2
1 *m1_pks_pk1_pk4* s1: m1
2 *m1_pks_pk1_pk4 s2: m1, m2*
3
4 *s4: m1 m2*
END
Output:
1: m1
2: 0
They output differently.
One more round will allow player 1 will receive multicast from player 2, thus output 0 and output the same result as that of player 2

2. Consensus Protocol

Protocol (the format of this protocol is borrowed from fig 5.1 textbook by R.Pass):

For every player i:
1. When protocol begins, let **MEMPOOL**i = ∅.
2. Whenever i receives a transaction *x* from *Z* (as an input), or from some other player j, if *x* isn't already in **MEMPOOL**i :
   – Add *x* to **MEMPOOL**i
   – Add *x_bar (x + arbitrary string)* from **LOG**i
   – broadcast *x* to all other players.
3. Whenever a ***new_trans*** is appended, remove all the *x_bar* from LOGj where *x_bar* is x + arbitrary string), where x ∈ ***new_trans***

For (epoch) e ≥ 1:
1. Let leader = (e mod n) + 1;
2. Let *m* = **MEMPOOL**i at the beginning of the epoch;
3. leader acts as the sender in a multi-value BA protocol sending out m. All other players act as receivers.
4. At the end of the BA protocol, each player j (including the leader) lets ***new_trans*** be their output of the BA protocol, and append ***new_trans*** to LOGj

Consistency (Satisfied):

All the honest players start the same and only add same MEMPOOL (that is equal to *new_trans)* to their logs according to consistency of the BA protocol. Thus, it satisfies consistency here.

Liveness (Satisfied):

Similar to the proof in the textbook, the only way to remove x from MEMPOOLi is to add it to the LOGi, thus it will satisfy the liveness principle. And once its added to LOGi, it is not removed from the LOGi.

Future Self Consistency (Not Satisfied):
Basic Idea: It will append an arbitrary X_bar everytime it has a new input X to LOGi at t1. Then if BA fails to deliver output at that round(which is true sometimes when the sender is faulty), LOGi at t2(when BA is finished and did not deliver) will have less things than LOGi at t1.
let t_2 be t_2 = t_1 + 1
In a 4 players (1 faulty player) case, F can let BA output 0 by being the leader, and then they have 25% chance of doing that, then they can hold this transaction in MEMPOOLi (until others are the leader). So in this case if t_2 = t_1 + 1, then P_execu(Log(r1) !≺ Log(r2)) = 1/3 . That is because (log(r1) includedx_bar) and log(r2) does not have that.

3. (Permissionless BA Premises)

Reasons PoW is needed:

Consensus is impossible to reach in permissionless setting (without consistency and liveness): If only using voter mechanism, then the protocol cannot avoid Sybil attack. (Faculty players have multiple identities, then those who are honest will be punished indirectly) PoW can help solving by changing the rules of protocols.

Reasons Computation power fraction restriction is used instead of player number fraction restriction:

Because when providing Proof of work (i.e. computing hash functions) in permissionless BA protocols, a lot of attack is based on the congregated computational power (it is not effectective to divide this into several players since they might have overlap in work).
In addition, computational power is a better measure of maximal threat as for the same reason.