CS 5433 Blockchains, Cryptocurrencies, and Smart Contracts
Homework 2

Group: Sophie Fang (yf338), Rory Connolly (rlc367),
Ryan Pencak (rvp32), Skyler Erickson (mse53)

Problem 1 – P2P Warm Up

2. **First clear your databases for nodes 1-6 and the parent/master node (see the instructions on the web documentation under "Running Nodes"). Run 3-6 nodes in separate command line windows, and rerun the generate_example_pow_chain.py file. This file has been modified to broadcast its generated blockchain to all nodes in the gossip network. Open the block explorers of each of your nodes at the end of that script's execution; do they appear synchronized?**

Case 1:

First, we were running nodes 1,2,3,4. Then we execute the generate_example_pow_chain.py file. After it was completed, we opened the browser for nodes 1,2,3,4. We see the pages contained exactly the same content. Pages appeared synchronized.

Now, repeat the experiment, but this time stop one of the nodes around halfway through the execution, and restart it at the end. What do you observe, and does this suggest any additions required to our gossip protocol (if so, what changes, otherwise, why not)?

Case 2:

After cleaning up the databases we were running nodes 1,2,3,4. Then we executed the generate_example_pow_chain.py file. Halfway through, we stopped node 1. After the script was completed, we re-started node1. This time, we saw all other pages were synchronized except node 1. So in this example, the node that went offline does not contain the blocks it missed when it is restarted after the generation script completes.

Such a case is quite common in real life, and no matter what happened, all nodes should be synchronized. This suggests that for our gossip protocol, even when one node is shut down for a while, when it goes online, it should be synchronized. In order to accomplish this goal we would need to add functionality to catch up nodes who have been offline. When a node comes back online it should be updated to the most recent transactions so that it is synchronized with the rest of the nodes.

3. **Notice that our node's list of peers is hard-coded in config.py. Does this suggest another missing component required to achieve a permissionless blockchain? Why not, or if so, what is the closest analogous message type in the Bitcoin p2p protocol documentation linked above?**

Hard-coded peers suggests that our protocol is missing a method to discover peers without having a predefined list of them. The closest analogous message type in the protocol documentation is addr message. From the protocol documentation, the description for addr can be seen below. This will replace the hard-coded list of peers and allow nodes to get information on other known nodes on the network.

addr

Provide information on known nodes of the network. Non-advertised nodes should be forgotten after typically 3 hours.

Problem 4 – Theory of Consensus

1. (**Dolev-Strong) Consider the Dolev-Strong BA protocol (which you implemented above) in Figure 4.1 of the lecture notes, and consider a setting with 4 players. Use the abstract protocol provided in the notes for your solutions, not your above implementation. Recall that we have shown that this protocol is secure w.r.t. 2 faulty players as long as we run it up to round 3. What happens if we consider a variant of the Dolev-Strong protocol that stops after round 2.**

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

Validity states that if the sender is honest, they will sign at most one message and therefore all honest receivers will only add that message to their set and output it.

We consider a Dolev-Strong protocol with 4 players, 2 of which are faulty, and stop the protocol after round 2. Validity holds after round 2 because honest players will only add a message to their set if the sender has signed it. Even though there are 2 faulty players who could pass a vote with threshold 2, the sender is honest and therefore does not sign an invalid message. Honest players will therefore output the sender's message and therefore validity holds even when stopping after round 2.

b) **Show that this protocol does not satisfy Consistency w.r.t. 2 faulty players. (Hint: Consider a situation where the sender and one of the receivers is faulty.) Explain what prevents your attack if we run the protocol for one more round.**

Consistency states that all honest players output the same message, or that if some honest player has a message m in its set at the end of some round then all honest players will also have m in their sets.

The protocol does not satisfy Consistency w.r.t. 2 faulty players. Consider a faulty sender and a faulty receiver. At round 2, the threshold is 2, and these two faulty nodes have enough signatures to pass a message. Because the sender is faulty, the sender will have signed the message and thus honest players will add the message to their set. The faulty sender and receiver both sign two different messages and send one to an honest player and the other to the other honest player. These honest players will now output different messages and the other honest player will not have that outputted message in its set. Thus, consistency

will not hold because a faulty sender and receiver can get two honest receivers to output different messages.

If we run the protocol for one more round, the threshold is now 3. There must now be at least three signatures on a message for it to be added to the set of an honest player. Because there are still only 2 faulty players, they will not have enough signatures on the message for the above attack to work. Thus, with a third round consistency will hold.

2. (**Future Self Consistency) Use any BA protocol (for sending strings) to construct a consensus protocol which satisfies Consistency and Liveness, but not Future Self-Consistency. Intuitively explain why your protocol does not implement a ”secure public ledger”.**

For Liveness to hold in the consensus protocol, a message posted by a player must be added to the log of every node in bounded time. In order for Consistency to hold, the logs of two nodes must be the same for any two nodes. Now, since Future Self-Consistency does not hold, logs should be able to be changed retroactively.

In order to satisfy Consistency and Liveness while Future Self-Consistency does not hold, there must be a way for LOGi and LOGj to change retroactively for any nodes i and j in a way that LOGi still equals LOGj. Thus, a sender must be able to get the majority of receivers to agree to a log that changes previous transactions to make sure every node makes these changes.

Consider a consensus protocol that uses Dolev-Strong, but instead of agreeing on a message to add to the log, nodes are agreeing on the contents of the entire log. We know that Consistency and Liveness hold in Dolev-Strong. However, because the message in the protocol is now the log that will be agreed upon, Future Self-Consistency does not hold. Using Dolev-Strong, all nodes will agree to a log that may or may not change previously agreed upon transactions. Because the new log can make changes retroactively, we do not have Future Self-Consistency. All nodes will still be able to have transactions posted in bounded time, retaining Liveness and also output the same log, retaining consistency.

The protocol does not implement a secure public ledger because Future Self-Consistency does not hold. If a malicious player is able to make changes to the log retroactively, they would be able to change or remove past transactions for their own benefit. Thus, the public ledger is no longer secure.

3. (**Permissionless BA) Explain informally why we need to use proofs of work (and bound the fraction of adversarial computing power, as opposed to simply giving a bound on the fraction of adversarial players) to get a BA protocol in the permissionless setting. (We are not expecting a formal proof, just an intuition.)**

In a permissionless setting, anyone can participate as a miner, and a single adversarial entity can run multiple nodes. We bound the computing power because running multiple nodes is cheap, but having the compute power to fulfill the proof of work is expensive. This gives a better bound on network power than the fraction of adversarial players. Tying proof of work to compute power then makes it difficult to alter the blockchain because it would require re-mining all prior blocks.

Evaluation

Did you find the homework easy, appropriately difficult, or too difficult?

We found this homework to be appropriately difficult.

How many hours total were spent on the completion of the assignment?

We spent approximately 6 hours on this assignment.

Did you feel there was too much coding, the appropriate amount of coding, or not enough coding?

We felt that there was not enough coding in proportion to the length of the written problems.