Problem 1

1. Please see provided code (all tests pass).

2. Yes, when I examine the nodes after I run generate_example_pow_chain.py, all the nodes appear synchronized! However, when conducting the experiment of stopping and restarting one node, I have a very different result. I ran all nodes 1-6 but stopped node 1 halfway through and then restarted the node as the chain was almost generated. While nodes 2-6 all had a height of 100, node 1 only had a height of 50. At the end of the script's execution, node 1 was NOT synchronized with its fellow nodes. Node 1 appeared to have been left behind, and it hadn't received any additional POWs since going (temporarily) offline. In official Bitcoin protocol, a node is only "forgotten" if is it dormant for over 3 hours. In our system, node 1 was forgotten if dormant for just a few minutes. I would suggest that our gossip protocol be updated to match the Bitcoin protocol of giving the node a window of 3 hours to come back onto the scene and be updated. In order to ensure consistent gossip spread, a node needs to have some flexibility in going and coming offline, as temporary outages are inevitable in any system.

3. Yes, this does suggest that we are missing the flexibility to accept any actor to the network, thereby not achieving permissionless status. With the list of peers hard-coded, it is impossible for a new person to join without having to change the code in our config file. Our system is missing the ability to look for and add any new actors who come onto the scene. The closest analogous message type in the Bitcoin protocol documentation would be getaddr. According to the documentation, getaddr is a message type that searches for potential additional active nodes in the network. The message asks a node they already know to see if it knows of any other active peers and the node does, it responds with their addresses[1]. This ensures that the system is open to adding new nodes and actors without being constrained to what is hard-coded in the config file.

Problem 4

We want two main properties from a blockchain/consensus protocol: consistency and liveness. Consistency means that if we take any 2 nodes, at any given point in the execution, their LOGs need to be the same. Liveness means that if someone wants to add a transaction, it "eventually" (within some bounded time) gets added to everyone's LOG[2]. BFT protocol in turn presents different statuses of functioning / not functioning to different components in the system. This creates difficulty for the components to reach an agreement about who is functioning or not, as they are all seeing different statuses for the same other components. However, they **must** reach agreement to decide who to trust and

[1] https://en.bitcoin.it/wiki/Protocol_documentation

[2] Pass, Rafael, *Consensus for Blockchains Preliminary Draft Notes*, 3.12.2018

not trust as a sender/leader. A system is BFT if it can still function under such ambiguity and confusion and ensure consistency and liveness, despite the difficulties.

Two functions that can achieve both consistency and liveness of a BA protocol and ensure BFT are proof-of-work and proof-of-stake. POW works by requiring large amounts of computational power to conduct a complex calculation to verify a transaction, and if you are first to solve this problem, you can verify transaction and are awarded a prize in form of currency. POW ensures liveness and consistency while remaining BFT by disincentivizing bad actors due to the large amounts of computational power required to solve a problem and therefore win the right to approve a transaction.

Another function with both blockchain properties and is BFT would be a Proof-of-stake (POS) function. POS works by somewhat randomly (plus taking into account wealth in the system) assigning actors the right to propose blocks, but agreeing on which proposed block is accepted depends on a multi-round vote. In order to vote, an actor must put as collateral their wealth on the line, and if they validate a bad transaction, they lose their wealth. POS ensures liveness and consistency while remained BFT by disincentivizing bad actors by making actors place their own wealth on the line before letting them vote for a transaction.

The main difference between these two protocols is that in POW, senders/leaders are chosen by whoever first solves the hash to validate transactions and create new blocks, while in POS the leaders/senders are voted upon by the group. There is also the big difference of ownership in this case – in POW, the actors rarely own the currency they are mining, while in POS, the actors must own the currency (or have a stake) in order to vote. Another main difference between POW and POS is that there are different barriers to breaking consistency and validity. In POW, in order to add any information to the blockchain, the bad actor would need to do invest a lot of time and effort to create a POW. This massive amount of work required to achieve a (meaningful) dishonest move is therefore not particularly attractive. In POS, if an actor votes for an invalid transaction, he is putting his own wealth at stake and is therefore strongly disincentivized to do so.

References for Questions 1 and 4:

https://en.bitcoin.it/wiki/Protocol_documentation
https://medium.com/loom-network/understanding-blockchain-fundamentals-part-1-byzantine-fault-tolerance-245f46fe8419
https://en.wikipedia.org/wiki/Byzantine_fault_tolerance
https://www.nasdaq.com/article/byzantine-fault-tolerance-the-key-for-blockchains-cm810058
https://hackernoon.com/is-bft-consensus-effective-for-proof-of-stake-blockchain-implementations-dc01f429d225
https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ
https://en.wikipedia.org/wiki/Proof-of-stake

Problem 5

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.

If the sender is honest, it will sign at most 1 message. As such— since honest players only add a message m to their set if the sender has signed it—it follows by the unforgeability of the signature scheme that m is the only message that can be added to their set.

Additionally, all honest players will add m to their set in round 1. We thus conclude that they will all output m.

(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.

Let $r \leq t+1$ be the round when i added m to its set. Ifr<t+1, then the conclusion directly follows by Lemma 4.3. If $r = t+1$, then player i must have received m with $t + 1$ different signatures in round $t + 1$. These signatures must have originally been sent in some round $r' < r$. By the unforgeability of the digital signature scheme, except with negligible probability, one of these signatures must have come from some honest player j (as there are at most t faulty players), and the attacker cannot forge signatures (except with negligible probability).

But j would only sign m when it adds m to its own set; so there must exist some honest player j must have added m to its set at round r', and thus by Lemma 4.3, every honest player would have had to add m to its set by round $r' + 1 \leq r$.

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”.

The protocol can be constructed as follows:

For every player i:

- When protocol begins, let MEMPOOLi = empty.
- Whenever i receives a transaction x from Z as input, or from some other player j, if x isn't already in LOGi, or in MEMPOOLi:
  - Add x to MEMPOOLi;
  - Broadcast x to all other players.
- Whenever a player i's LOGi changes, remove the latest transaction in LOGi from MEMPOOLi

For epoch e >= 1:

- Let leader = (e mod n) + 1;
- Let m = MEMPOOLi at the beginning of the epoch;
- Leader acts as the sender in a multi-value BA protocol sending out m. All other players act as receivers.
- At the end of the BA protocol, each player j (including the leader), lets new_trans be their output of the BA protocol, and then appends new_trans to the end of LOGj.

Since the latest transaction added to any player i's MEMPOOL based on this protocol will always be the same, the protocol would simultaneously satisfy Consistency and Liveness. However, since the protocol would remove the latest transaction in LOGi from MEMPOOLi, future self-consistency would be violated when the latest transaction in LOGi is not the latest transaction in MEMPOOLi (a faulty player sends a faulty message and added to MEMPOOLi). However, liveness would still hold as all transactions would eventually be added to the LOG, just in different orders (so future consistency is not satisfied).

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 the permissionless setting, anyone can join or leave the protocol execution without any form of permission or authentication, and any participant may join or leave at will. This means there would be no known upper bound on both the total number of nodes in the system at a point in time and the amount of faulty participants. Since it's possible (and easy) for a malicious user to create multiple online identities, a certain type of barrier-to-entry needs to be established to prevent such malicious users from effortlessly introducing a large number of faulty participants and manipulate the consensus process in a Sybil attack. Proof-of-work (PoW) can be used to establish such a barrier-to-entry by requiring nodes to prove they have expended significant amount of energy towards solving a hard cryptographic puzzle. Such a barrier-to-entry would

make it become much more expensive for a malicious user to take control of a significant number of participants in the system and therefore greatly reduce the chance of success for a Sybil attack. However, expensive still does not mean impossible. Therefore we still need to assume the computational power that the adversary can gather is bounded and will not be sufficient to create a large enough group of faulty participants to manipulate the consensus process.