EE 374: Blockchain Foundations **Stanford, Spring 2022**

Lecture 6: Chains

Friday, April 15, 2022

Lecturer: Dr. Dionysis Zindros Scribes: Taher Poonawala and Scott Hickmann

1 Introduction

In this lecture, we will delve deeper into chains, resolve the last few problems with proof of work and the double-spending problem, explain how money is created, and describe how to update the UTXO. By the end of this lecture, while there will still be optimizations to be done, you will have all the necessary knowledge to implement a fully functional and secure blockchain.

2 Target Parameter (T)

In the previous lecture we spoke about the necessity of sending messages separated by time intervals, i.e. in the form of rare events. This is done to prevent adversaries from double spending by sending transactions from the same outpoint within a short period of time, thereby causing confusion about which transaction is valid.

The time interval between sending messages should be such that it is greater than the network delay so that transactions are allowed to propagate to every node in the network thus preventing adversaries from double spending.

In order to tie blocks to rare events, the protocol sets a target parameter T such that a block B is valid only if

$$H(B) < T \tag{1}$$

3 Safety and Liveness

Increasing T leads to a loss of safety. This means that there is an increased likelihood of honest nodes receiving conflicting transactions. Decreasing T leads to a loss of liveness, this is because it takes longer to generate a valid hash that is below the threshold. Safety can be thought of as the honest ledgers being in agreement with each other. Liveness can be thought as the ledgers making progress by adding new transactions.

4 Freshness

To produce the hash of the block we run our hashing algorithm on $B = (s, \overline{x}, \text{ctr})$ where s is the hash of the previous block, $\overline{x}$ is the ordered set of transactions that are included in the block and ctr is a randomly chosen nonce. We include the set of transactions in the proof-of-work so that the adversary cannot reuse the same proof-of-work multiple times.

The hash of the previous block is included in the new block we generate so as to maintain freshness. Freshness provides an arrow of time. Since each new block includes the hash of the previous block we can confidently say that the new block was produced after the previous block. This prevents adversaries from premining blocks and adding them when convenient because the adversary must include the hash of the previous block when producing the block that comes after that.

Linking blocks to each other with the help of hashes forms a *blockchain*. It is also known as a hash chain. This makes it impossible to go back and change a block without having to change all the blocks that came after that block as shown in Figure 1.

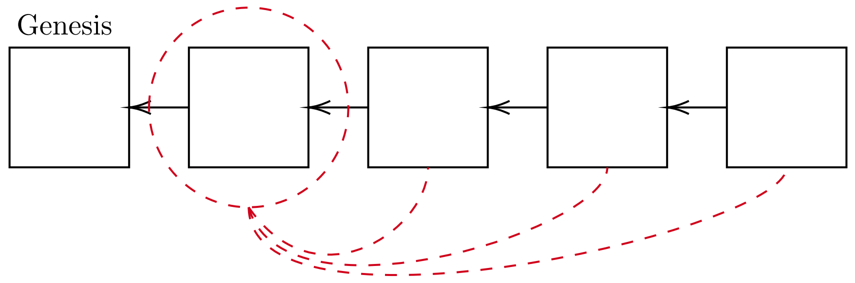

Figure 1: Chaining blocks prevents changing the old blocks as this would invalidate the proof-of-work equation for the blocks following it in the chain.

5 Determining the Target Parameter

For simplicity, consider that B is randomly chosen in $\{0,1\}^\kappa$. Then,

$$\Pr[H(B) \leq T] = \frac{T}{2^\kappa} = p \tag{2}$$

- p: Probability of successful query
- q: hash power of a single party
- n: number of parties participating in network

Here, we think of one party as one unit of computation. All parties in the network are simultaneously trying to create a block.

The target variable T should be chosen such that the time taken to generate a valid nonce (B) should be greater than the network delay.

$$\frac{1}{pnq} = \Delta \tag{3}$$

where Δ is the network delay.

$$T = \frac{2^\kappa}{\Delta nq} \tag{4}$$

Here, qn is the total hashing power of the network. We can see that as qn increases T should decrease. This means that it should become harder for each party to generate a valid nonce. Increasing the network delay will also similarly require T to be decreased.

6 Accounting for Stochastic Nature of Proof of Work

Ideally we would want that all blocks are produced with the same delay as decided by the equation above. However, the Proof-of-Work algorithm is stochastic. Thus based on our above calculation, the expected time to produce a block will be Δ but in practice there may be instances where blocks are produced in quick succession without the required time delay. This takes us back to square one in terms of thinking about how to prevent double spends. Figure 2 shows convergence opportunities (i.e. periods of time where an event happened spaced out by more than Δ from both the previous and next event) and conflicting events (i.e. periods of time where two events happened in a time interval shorter than Δ) produced during the mining process.

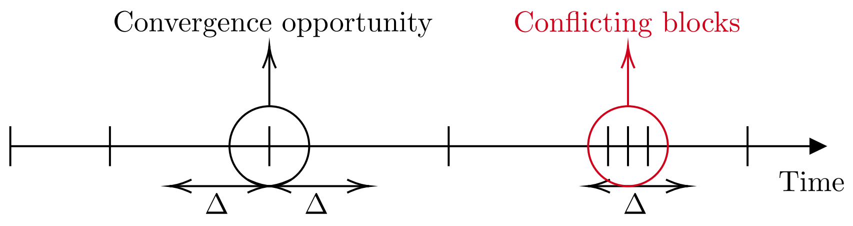

Figure 2: Block production during operation is a stochastic process

7 Honest Majority Assumption

Thus when we have a convergence opportunity, i.e. block produced with the stipulated time delay we want all the honest nodes to agree that a particular block is the freshest. We need some sort of voting scheme for the honest blocks to accept the latest block as the freshest one. We cannot have each node vote once on which block is the freshest as the adversary could carry out a Sybil attack.

Instead, we will have the honest node add blocks to the longest chain. The assumption here is that a majority of the computational power is controlled by honest parties. Due to this, the length of the chain mined by honest blocks will always be greater than the length of the chain mined by the adversary. As a result, new honest blocks will always be added to the longest chain. Figure 3 shows the block production of honest nodes and the adversary. In order to start building off of the longest chain, we still require convergence events, i.e. blocks separated by a specified time delay because otherwise we would not know which chain to build upon (Figure 4). Thus we now have a blockchain that all the honest nodes can agree upon.

In order to have a mathematical guarantee that the blockchain converges, the honest majority assumption stated below has to be upheld.

$$t < n - t \tag{5}$$

where t is the computational power of the adversary and n.

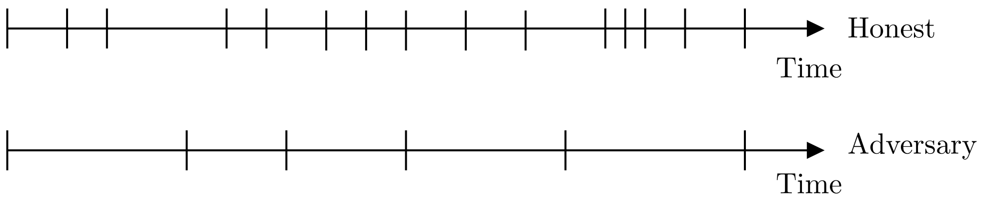

Figure 3: Block production of honest nodes vs. the adversary

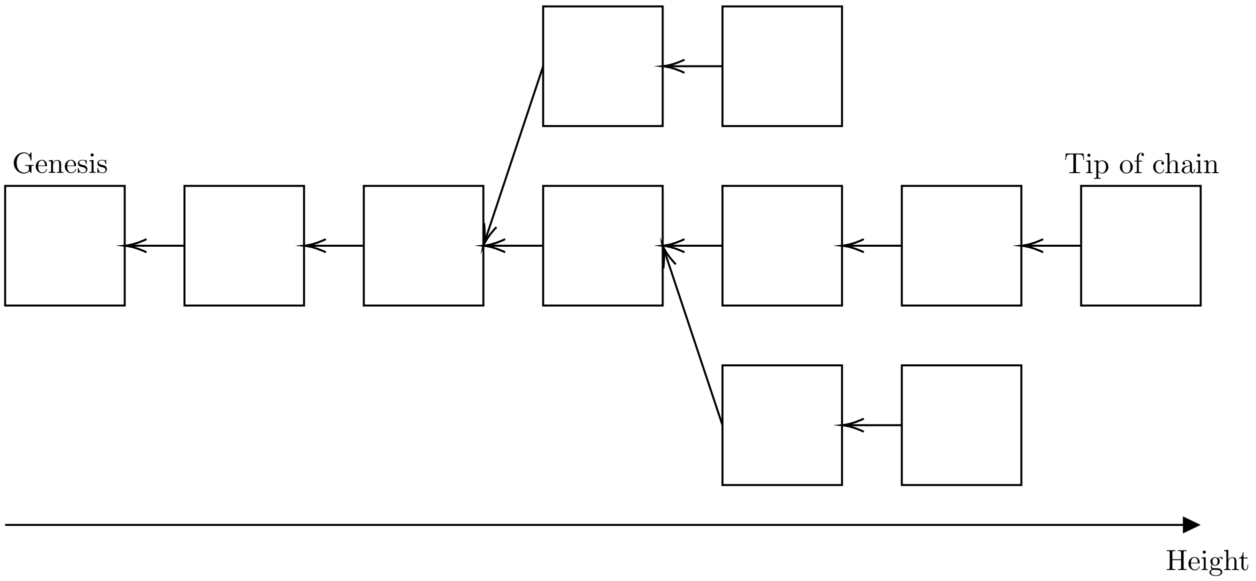

Figure 4: Block trees and the longest chain rule. There may be temporary forks because of conflicting blocks. But, the occurrence of convergence opportunities ensures that eventually, there will be a unique tip of the longest chain.

8 Coinbase Transactions

While we have successfully implemented a way to validate ownership of money in a decentralized manner, we have yet to see how to tackle the creation of money.

To do so, we introduce the notion of a "coinbase transaction". This transaction is uniquely present in every block. The miner creates this transaction, and can reward any public key they want with an amount less than or equal to a fixed amount f (the block reward) determined by the network plus transaction fees (the fee reward). These fees correspond to the difference between inputs and outputs in each transaction confirmed in the block. Due to the weak law of conservation, recall that there can be a difference between the sum of the inputs and the sum of the outputs in each transaction. This difference is paid out as fees. In other words, the miner can create a transaction with no input and one output with an amount a respecting the following equation:

$$a \leq f + \sum_{i \in \text{input}} i.v - \sum_{o \in \text{ouput}} o.v \tag{6}$$

where the sum is over all transactions in the block. In our Marabu protocol, we use $f = 50$ bu.

Finally, to avoid multiple identical transactions that would have the same hash as a result of having the same amount and same public key output, an additional *height* parameter is included in the coinbase transaction. This parameter corresponds to the number of blocks that this block is away from genesis.

9 Verifying Objects

9.1 Verifying a Block

Upon receiving a block, a honest node should:

1. Verify the proof of work (and do this first to avoid spam attacks).
2. Verify the parent block recursively (and if it does not exist locally, request it from the network).
3. Verify the transactions inside the block, including the coinbase transaction (and if it does not have them, request them from the network).

9.2 Verifying a Chain

Upon receiving a chain, a honest node should:

1. Verify each block in the chain.
2. Check that it starts with genesis (or is connected to a block known to be connected to genesis).
3. Adopt the longest chain or one of the longest chains.

9.3 Verifying a Regular Transaction

See lecture 4

9.4 Verifying a Coinbase Transaction

A valid coinbase transaction must:

1. be the only one in that block,
2. be the first transaction in the block,
3. have no input and exactly one output,
4. have a value which is less than or equal to the sum of the fixed block reward and the difference between the input and output amounts of all other transactions in the block,
5. not be spent in the same block.

10 Comparing Transactions and Blocks

Now that we understand how blocks and transactions work, it is worth drawing parallels between the two.

	Transaction	**Block**
Inductive base	Coinbase	Genesis
Inductive hypothesis	Outpoint UTXO	Previous ID (s)
Inductive step	Consuming produced UTXO Signatures Conservation laws	Proof of Work Causality Transactions

11 Updating the UTXO

For each received transaction, if the transaction is valid, apply the transaction to the previous UTXO to get the new UTXO.

For each reveived block, apply each transaction of the block to the previous UTXO. If at any point a transaction is invalid, reject the entire block and revert to the previous UTXO.

For all transactions not yet in a block, add them to a mempool with a temporary UTXO which starts as the same UTXO as the current longest chain tip. These transactions are added in the order in which they are received. If the transaction is invalid with respect to the current UTXO, reject it. When mining a block, use all of the transactions in the mempool to create this block.

Upon receiving a block at the tip of the current longest chain, validate the block and update the UTXO. Also update the mempool and temporary UTXO by applying transactions in the block or removing any transaction that is now either invalid or already in the newly received block.

Upon receiving a block which changes the longest chain to another chain, set the UTXO to the fork between the current longest chain and this new chain (by undoing all transactions after the fork in the current chain). Then, go through each block until the tip in the new longest chain and update the UTXO. If at any point, a block is invalid, reject the new chain and revert to the previous chain and previous UTXO. If all blocks after the fork in the new longest chain are valid, update the mempool by starting with the UTXO at the tip of the new longest chain, applying every valid transaction in the previous longest chain that is not present in the new longest chain, then applying all the still-valid transactions that were previously in the mempool. This process of adopting the longest chain is shown in Figure 5.

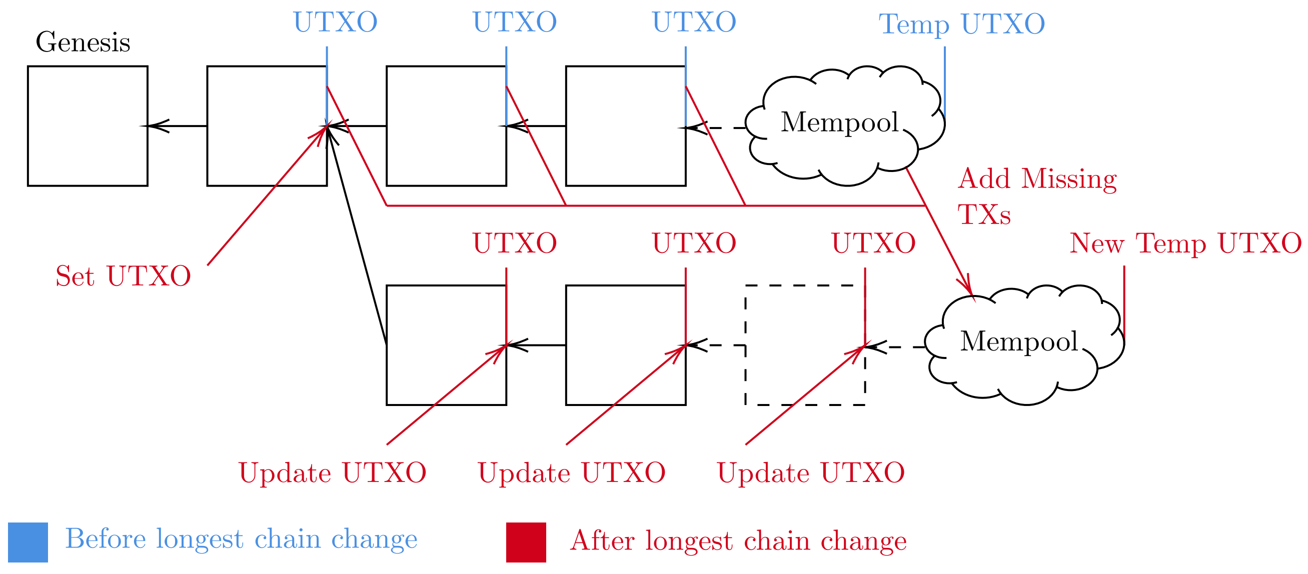

Figure 5: Updating the UTXO upon chain change: 1. Set UTXO to the fork. 2. Update UTXO by applying blocks in the new chain. 3. Create a new temporary UTXO starting at the tip of the new chain. 4. Add transactions from the previous chain and previous mempool that are missing in the new chain to the new mempool.