Network Security

Professor Gaurav S. Kasbekar
Department of Electrical Engineering
Indian Institute of Technology, Bombay

Week - 11
Lecture - 65
The Bitcoin Cryptocurrency: Part 5

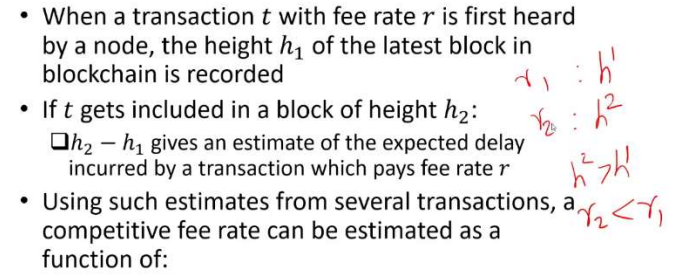
Hello, in this lecture, we will continue our discussion of the Bitcoin cryptocurrency. We now discuss how the transaction fee in a block is determined. Recall that the maximum size of a block is 4 MB. Each block must include the block header, a coinbase transaction, and several regular transactions. Miners seek to maximize the total transaction fees earned from a block.

Recall that the total revenue of a miner is the block subsidy plus the transaction fees from all the regular transactions in the block. Clearly, miners would like to maximize the total transaction fees since that will maximize the block reward. Miners prefer to include in candidate blocks transactions with high transaction fees as well as small sizes. So, if the transaction fees are high, then that will increase the revenue of the miner, and if the transactions are small, then the miner can fit a lot of transactions into the limit of 4 MB. Hence, again, the revenue of the miner will increase.

In particular, let the transaction fee per byte, or the fee rate of a transaction, be defined as the transaction fee divided by the size of the transaction. So, this is the transaction fee per byte. Miners prefer transactions with high fee rates. So, that is, per byte out of this 4 MB, they prefer transactions which maximize the transaction fees that they get. So, they prefer transactions with high fee rates.

Hence, nodes that want to add a regular transaction to the blockchain need to pay a fee rate that is competitive with other transactions being broadcast on the network. If the fee rate of a particular transaction is very low in comparison to the fee rates of other transactions that are being broadcast on the network, then this transaction will not be included by miners in the candidate blocks. So, there will be a long delay until the transaction is included in the blockchain. Hence, the fee rate of a transaction must be competitive with the fee rates of other transactions being broadcast on the network. How can a node that wants to add its regular transaction to the blockchain find out what fee rate is competitive?

It must have an idea of what fee rates other nodes are setting. It should set its own fee rate to be comparable with the fee rates of the other nodes. When a transaction t with fee rate r is first heard by a node, then the height h_1 of the latest block in the blockchain is recorded by the node. If t gets included subsequently in a block of height h_2, then h_2-h_1 gives an estimate of the expected delay incurred by a transaction, which pays fee rate r. So, as the fee rate r increases, typically this delay h_2-h_1 will reduce. That's because miners will prefer to include the transaction in their candidate blocks since it pays a high fee rate r.



So, the higher the fee rate, the lower the expected delay incurred by the transaction. Using such estimates from several transactions, a competitive fee rate can be estimated as a function of the amount of delay in blocks that the node creating the transaction is willing to tolerate. So, for example, the node sees several transactions and the corresponding delays of those transactions. Suppose it saw a particular transaction with fee rate, say r_1. The corresponding delay was, say h_1 blocks—or, to so as not to confuse with this h_1, suppose it is h^1.

So, this is the delay for the transaction with fee rate r_1, and another transaction with fee rate r_2 has delay h^2. Suppose that $h^2 > h^1$, and this will typically be the case if $r_2 < r_1$. Since, out of these transactions, this second transaction has a lower fee rate than the first transaction, naturally it incurs a higher delay in being included in the blockchain. A particular node that wants to decide how to set its fee rate will look at such fee rates of other transactions and the corresponding delays they incurred in being included in the blockchain. So, it gets an idea of the, for a particular delay, what fee rate should it set. So, it can get such an idea by observing the past history of fee rates and the corresponding delays incurred in the inclusion of the transaction in one of the valid blocks.

Now, we introduce some terminology: Unspent Transaction Output (UTXO). Recall that when an output of a previous transaction is unlocked by an input of a later transaction, then all the Bitcoins in the output are spent by the input. Hence, a transaction output can be in

one of only two states. One is spent, and another is unspent. So, initially, the output is in the unspent state.

Later on, when the bitcoins in the output are spent by some subsequent regular transaction, then the transaction output transitions to the spent state. UTXOs refer to outputs in transactions recorded on the blockchain which have not been unlocked by inputs of later transactions. What is the total amount of Bitcoins owned by an address that is a particular public key? So, the total amount of Bitcoins is clearly the sum of the amounts in all the UTXOs that it can unlock. When a blockchain fork occurs, the sets of UTXOs seen by different nodes in the Bitcoin network differ.

We saw in the example in the previous lecture that there was a fork in the blockchain. So, in that case, because of a fork, the sets of UTXOs seen by different nodes in the Bitcoin network will differ. Once the fork is resolved, the UTXO set seen by all the nodes in the Bitcoin network will be identical. Now, we discuss the concept of confirmations. Suppose Alice is a buyer who wants to use Bitcoins to pay for some goods that the seller Bob is selling.

- Alice creates a new regular transaction, say t_1, such that:
  - one or more of its inputs unlock UTXOs that Alice owns
  - one of its outputs contains the amount of bitcoins Alice wants to pay Bob and a challenge script such that:
    - to create a valid response to it, a private key owned by Bob is required
- Alice broadcasts the transaction t_1 on the Bitcoin network
- Miners include it in the candidate blocks they are mining
- When should Bob hand over the goods to Alice?
  - Bob keeps scanning the new blocks being added to the blockchain for t_1

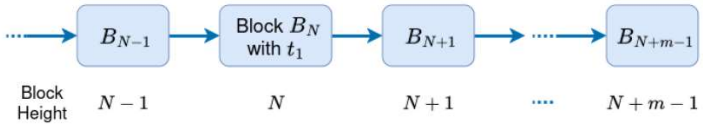
Alice creates a new regular transaction, let's call it t_1, such that one or more of its inputs unlock UTXOs that Alice owns, and one of its outputs contains the amount of Bitcoins that Alice wants to pay Bob, and a challenge script such that to create a valid response to it, a private key owned by Bob is required. So, these outputs will transfer the Bitcoins in the inputs that Alice owns. These Bitcoins will be transferred to Bob, since Bob has the private key required to respond to this challenge script. Alice broadcasts the transaction t_1 on the Bitcoin network. Miners include the transaction t_1 in the candidate blocks that they are mining. A question is when should Bob hand over the goods to Alice?

So, note that Bob must be confident that he will receive the payment from Alice. Bob should hand over the goods to Alice once he is confident that he will receive the payment

from Alice in Bitcoins. For this purpose, Bob keeps on scanning the new blocks that are being added to the blockchain for transaction t_1. So, he keeps on scanning the new blocks until he sees a block which includes the transaction t_1. Once Bob sees a new block with transaction t_1 included in it, then he will wait for the branch containing this block to grow further before handing over the goods to Alice.

So, once Bob sees a new block with t_1 in it, why does he not immediately hand over the goods to Alice? Why does he wait for the branch to grow further? We'll discuss the reason for that. In particular, consider an example where if the transaction t_1 is included in a block B_N at height N, then transaction t_1 is said to have received one confirmation. So, this is the transaction t_1.

- In particular:
  - ❑ if the transaction t_1 is included in a block B_N at height N, then t_1 is said to have received one confirmation
  - ❑ when a valid block B_{N+1} at height $N+1$ is added to the blockchain with B_N as its previous block, the transaction t_1 is said to have received two confirmations
  - ❑ more generally, when $m-1$ valid blocks have been appended to the block B_N, the transaction t_1 is said to have received m confirmations
- Bob waits until t_1 has received m confirmations before handing over the goods to Alice
  - ❑ m is chosen depending on the value of the goods
- Why does Bob wait for m confirmations, instead of handing over the goods to Alice after one confirmation has been received?
  - ❑ to safeguard against a "*double spending attack*" by Alice in which the transaction t_1 can be cancelled



Block Height $N-1$ N $N+1$ .... $N+m-1$

It is present in a block B_N at height N. So, when block B_N is added to the blockchain, then the transaction t_1 is said to have received one confirmation. When a valid block B_{N+1}, that is this one, at height N+1 is added to the blockchain with B_N as its previous block, then the transaction t_1 is said to have received two confirmations. So, the transaction t_1 is included in the block, and one block has been added subsequently. Hence, the transaction is said to have received two confirmations, and so on and so forth. More generally, when m-1 valid blocks have been appended to the block B_N, which contains the transaction t_1, then the transaction t_1 is said to have received m confirmations.

So, when this block has been appended to the blockchain, then the transaction t_1 is said to have received m confirmations. Bob waits until t_1 has received m confirmations before handing over the goods to Alice. A typical value of m is 6, so Bob waits until the transaction receives 6 confirmations before handing over the goods to Alice. m is chosen depending

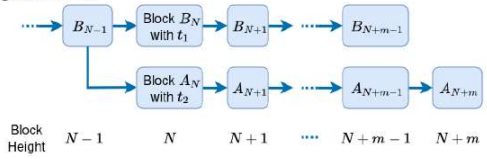
on the value of the goods. So, m=6 is a typical value, but m can vary depending on the value of the goods. So, for very expensive goods, Bob may wait until more confirmations, for example, 10 confirmations.

And for less expensive goods, he may wait for fewer confirmations—for example, just 3 confirmations. Now the question is: why does Bob wait for m confirmations instead of handing over the goods to Alice after one confirmation has been received? So, this is to safeguard against an attack by Alice called "double spending attack" in which the transaction t_1 can be cancelled. So, what is this double spending attack? We will discuss this attack now.

Double Spending Attack

- Suppose:
  - ❑ Bob asks Alice for x bitcoins as payment for the goods
  - ❑ Alice can unlock a UTXO O_A which has at least x bitcoins
- A double spending attack proceeds as follows:

1) Alice creates two transactions t_1 and t_2:
   - ❑ in t_1, an input unlocks O_A and an output pays Bob x bitcoins
   - ❑ in t_2, an input unlocks O_A, but x bitcoins are paid back to Alice in an output
   - ❑ Note that the transactions t_1 and t_2 conflict with each other because they both spend the same UTXO O_A; only one of them can be included in blockchain
2) Alice broadcasts t_1 on the Bitcoin network for inclusion in the blockchain
   - ❑ she keeps t_2 a secret
3) Suppose a miner includes t_1 in a valid block B_N at height N
   - ❑ Bob waits until t_1 has received m confirmations before handing over the goods to Alice
4) Immediately after broadcasting t_1, Alice begins work on constructing a branch containing t_2
   - ❑ she does not announce the valid blocks found on this branch to the Bitcoin network before Bob transfers the goods to her



Suppose Bob asks Alice for x Bitcoins as payment for the goods that Bob is going to transfer to Alice. Alice can unlock the UTXO, say O_A, which has at least x Bitcoins. Now, a double-spending attack proceeds as follows. It has the following steps. Alice first creates two transactions, t_1 and t_2.

In t_1, an input unlocks O_A, and an output pays Bob x Bitcoins. So, this t_1 is the transaction that is of interest to Bob. This transaction t_1 will transfer the x Bitcoins to Bob. But this transaction t_2 has an input that unlocks O_A again, but the x Bitcoins are paid back to Alice in an output. So, this is a secret transaction, and the same output, which corresponds to the previous transaction t_1. So, the same output is unlocked and the x bitcoins are paid back to Alice.

Note that the transactions t_1 and t_2 conflict with each other because they both spend the same UTXO O_A. Hence, only one of them can be included in the blockchain. So, for example, if transaction t_1 is included in a block in the blockchain, then no miner will include t_2 in a subsequent candidate block because t_2 conflicts with t_1 and vice versa. Now, in the next step, Alice broadcasts t_1 on the Bitcoin network for inclusion in the blockchain. So, t_1 must be included in one of the blocks, and Bob must see that t_1 has been included. Once it receives the required number of confirmations, Bob will transfer the goods to Alice. Alice keeps t_2 a secret.

So, Bob does not know the existence of transaction t_2, which conflicts with transaction t_1 in this double spending attack. Suppose a miner includes t_1 in a valid block B_N at height N. So, this transaction t_1 is present in block B_N, which is at height N. Now, Bob waits until t_1 has received m confirmations before handing over the goods to Alice. So, when this block is added, the transaction t_1 has received m confirmations by definition. At this point, Bob hands over the goods to Alice.

But in parallel, immediately after broadcasting t_1, Alice begins work on constructing a branch containing t_2. So, t_2 is the secret transaction, and immediately after broadcasting transaction t_1, Alice works on constructing a branch containing t_2. That branch is shown here. So, Alice does not announce the valid blocks found on this branch to the Bitcoin network before Bob transfers the goods to her. The purpose of this transaction t_2 is to conflict with t_1 and to spend the same Bitcoins, which correspond to the output O_A.

These bitcoins are paid back to Alice so that Alice does not lose the x bitcoins, but Alice still gets the goods. This is the double spending attack. What is the next step in this double-spending attack? After receiving the goods from Bob, if Alice succeeds in creating a branch containing t_2, which is longer than the branch containing t_1—as shown in this example here—we can see that this branch containing t_2 is longer than the branch containing t_1. Then, Alice broadcasts all the blocks in the t_2 branch in the Bitcoin network. So, we have discussed the rule that every node must accept blocks belonging to the longest branch that they are aware of. Hence, all the nodes in the Bitcoin network will eventually switch to the t_2 branch, and the t_1 branch will be abandoned.

So, the blockchain will become this one. The blockchain does not contain the transaction t_1, which transfers the x Bitcoins to Bob. Hence, Bob does not gain the x Bitcoins, but nevertheless, Bob has transferred the goods to Alice. One important point to note is that usually, transactions present in blocks in abandoned branches are added back to the

transaction pool—that is, the mempool—if they have not already appeared in the surviving branch. Miners use this transaction pool for constructing new candidate blocks. But miners that have switched to the t_2 branch—that is, this one—will not add t_1 to their mempools because it conflicts with t_2.

So, consider what happens when all these blocks are announced on the blockchain network by Alice. Then, the blockchain copy at all the nodes becomes this. Now, what happens to the transaction t_1? Normally, transaction t_1 would have been put back into the mempool by miner nodes, and it would eventually have become part of the blockchain. But here, miners see that t_1 conflicts with t_2, and t_2 is already present in the blockchain.

Hence, miners will discard t_1 and will not include t_1 in the candidate blocks they mine. Hence, t_1 will never be included in the blockchain. So, in summary, Bob has already transferred the goods to Alice, but the x bitcoins that he thought he received from Alice in t_1 are back in Alice's possession. So, Alice can now spend these bitcoins again, and hence this attack is called a double-spending attack. Alice spends the same set of bitcoins twice or possibly more than twice if this attack is repeated.

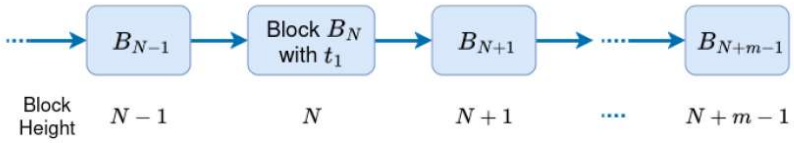
Hence, this attack is called a double spending attack. Now, one remark on the double spending attack is that to implement such an attack, a miner must clearly have a lot of resources. So, in this example, Alice must have a lot of mining resources because all the other miners except Alice are trying to extend the upper branch, but Alice is the only miner trying to extend the lower branch. So, Alice must be a miner with a lot of resources to be able to compete with all the other miners collectively and add blocks at a faster rate. So, that is typically difficult for most miners.

Only a miner with a huge amount of resources can hope to achieve the double spending attack with a high probability. Now, we discuss how we should set the number of confirmations m. So, in the above example, Bob waits until t_1 has received m confirmations before handing over the goods to Alice. How many confirmations m should Bob wait for? Assume that Alice controls less than 50% of the network hash rate.

So, this will typically be the case because there are a large number of miners in the Bitcoin network, and an individual miner will have less than half the network hash rate. Under this assumption, the success probability of a double spending attack decreases as m increases. So, if the block containing this transaction t_1 has already become long, then it will be unlikely that another miner with less than half the network hash rate will be able to find

another branch which is longer than this branch. But m cannot be very large, as each confirmation takes approximately 10 minutes to appear. So, that's a trade-off.

- Recall: in above example:
  - ❑ Bob waits until t_1 has received m confirmations before handing over the goods to Alice
- How many confirmations m should Bob wait for?
- Assuming that Alice controls less than 50% of the network hash rate:
  - ❑ the success probability of a double spending attack decreases as m increases
- But m cannot be very large as each confirmation takes approximately 10 minutes to appear
  - ❑ so a customer, irrespective of whether he/ she is honest or malicious, will experience a delay of about $10m$ minutes before the seller transfers the goods to him/ her
- Several merchants in the Bitcoin network wait for six confirmations $(m = 6)$, which corresponds to a delay of about an hour before goods are transferred from merchant to customer
  - ❑ smaller or larger values of m may be used by merchants depending on the value of the goods being sold



The success probability of a double spending attack decreases as m increases. So, we would like to have a large m. But m cannot be very large, as each confirmation takes approximately 10 minutes to appear. So, a customer, irrespective of whether he or she is honest or malicious, will experience a delay of around 10 minutes before the seller transfers the goods to him or her. So, as we mentioned earlier, several merchants in the Bitcoin network wait for 6 confirmations, that is, m=6. So, that corresponds to a delay of 10 times 6, that is, 60 minutes or 1 hour before the goods are transferred from the merchant to the customer.

But depending on the value of the goods being sold, smaller or larger values of m may be used by merchants. So, if the goods are very valuable, then typically a large value of m will be used, and if the goods are not so valuable, then a small value of m will be used. So, in summary, we discussed UTXOs and we discussed the concept of confirmations and also the double spending attack. And we discussed how the number of confirmations before the transfer of goods is set. We will continue our discussion of the Bitcoin cryptocurrency in the next lecture.

Thank you.