Network Security

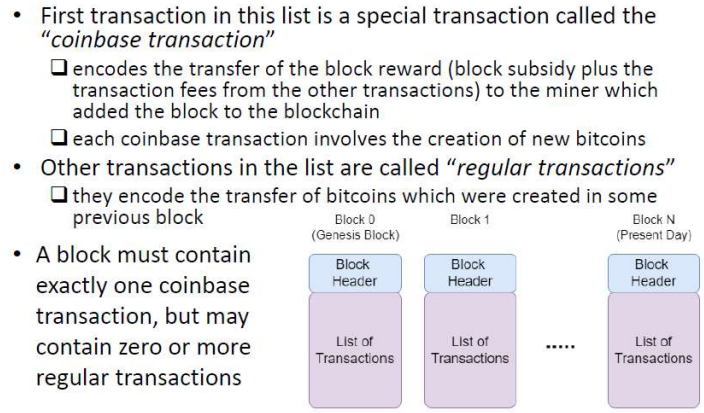
Professor Gaurav S. Kasbekar
Department of Electrical Engineering
Indian Institute of Technology, Bombay

Week - 11

Lecture - 63

The Bitcoin Cryptocurrency: Part 3

Hello, in this lecture, we will continue our discussion of the Bitcoin cryptocurrency. Recall that mining is the process by which new blocks are added to the blockchain. This figure shows the blockchain. This is the very first block, known as the Genesis block or block 0. Then, this is block one, and so on.



This is block N, which is the latest block in the blockchain. So, each block consists of a block header, which is shown here, followed by a list of transactions. These transactions are shown over here. These transactions are of two types. The first transaction in every block is a special transaction called the coinbase transaction.

And the other transactions are known as regular transactions. So, the first transaction—that is, the coinbase transaction—encodes the transfer of the block reward to the miner which added the block to the blockchain. Recall that we discussed that the block reward consists of the block subsidy plus the transaction fees from the other transactions. So, that's the block reward of a block. It consists of the block subsidy, which is a constant number of bitcoins (for example, 12.5 bitcoins), plus the transaction fees from the regular transactions in the block.

So, this coinbase transaction encodes the transfer of the block reward to the miner which added the block to the blockchain. So, each coinbase transaction involves the creation of new bitcoins. For example, 12.5 bitcoins may be created, and the coinbase transaction encodes the creation of these bitcoins. The other transactions in this list are called regular transactions. So, in summary, there is one coinbase transaction in every block, and there are zero or more other transactions known as regular transactions.

So, these regular transactions encode the transfer of bitcoins which were created in some previous block. For example, initially Alice may have created a block, and she earned some bitcoins, some new bitcoins. Later on, Alice paid some of these bitcoins to Bob. So, later on, in another block, a transaction was added which encoded the transfer of these bitcoins from Alice to Bob. So, such a transfer of Bitcoins which were created in some previous block are encoded in regular transactions.

A block must contain exactly one coinbase transaction, but it may contain zero or more regular transactions. Typically it contains several regular transactions, and these regular transactions are a source of revenue apart from the block subsidy for the miner who mines the block. So, nodes that want to record new regular transactions in the blockchain broadcast them on the Bitcoin network. So, these transmissions of new regular transactions are received by minor nodes, and when they create new blocks, they may include these transactions in the blocks that they add to the blockchain. So, when other nodes hear these new transactions they add them to a transaction memory pool.

It's called mempool, and it is stored in the local memory of the minor node. So, each minor node receives these transmissions of new regular transactions from other nodes in the network and they add them in this transaction memory pool called mempool. A minor node forms a candidate block by collecting some transactions from its mempool. Recall that the size of a block is 4 MB. The miner node collects enough transactions so that these transactions will roughly fit into 4 MB, leaving space for the coinbase transaction and the block header.

So, the miner node forms a candidate block by collecting some transactions from its mempool. And it includes a coinbase transaction in the candidate block, which transfers the block reward to its own Bitcoin address. And apart from this, the miner node adds the header to the candidate block. So, the block consists of these entities. One is the regular transactions from the mempool.

The other entity is the coinbase transaction and the third is the block header. At any time, there'll be several miner nodes competing to add the next block in the blockchain and claim the resulting block reward. Whichever miner node is the first one to generate a correct block, that miner node adds the block to the blockchain. The candidate blocks created by these different miner nodes will differ in the coinbase transaction. For example, consider candidate blocks created by three different miners, say A, B, and C.

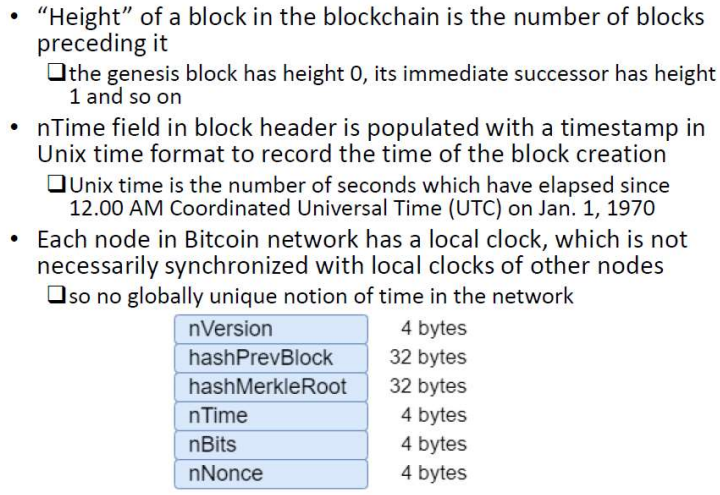
So, the candidate block that is created by A, in that candidate block, the Coinbase transaction will say that the block reward is transferred to A. Similarly, in the candidate block that is created by B, the coinbase transaction will say that the block reward is transferred to miner B. And in the candidate block of miner C, the coinbase transaction will say that the block reward is transferred to miner C. So, these candidate blocks created by these different miner nodes A, B, and C, they definitely differ in the coinbase transaction. These candidate blocks may also differ in the regular transactions included in them. Because different miner nodes may have different sets of transactions in their respective mempools. How can it happen that different miner nodes have different sets of transactions in the mempools?

Recall that nodes that want to record new regular transactions broadcast them on the Bitcoin network. So, all the miner nodes eventually receive all the regular transactions that are broadcasted on the Bitcoin network. But this can still happen—that is, different candidate blocks can still differ in the regular transactions included in them. And that happens because the minor nodes receiving transactions broadcast on the Bitcoin network at different times due to network latencies. Consider a particular node, say D.

This node D broadcasts a regular transaction which it wants to include in the blockchain. A is nearest to node D, so A receives this regular transaction first. B and C receive this regular transaction after some latency. So, in the candidate block of A, this regular transaction sent by node D is included, but in the candidate blocks of B and C, this regular transaction is not included because it was received by B and C after some latency. So, that's how it can happen that the set of regular transactions included in the candidate block of one node differs from the set of regular transactions in the candidate blocks of the other nodes.

Now we will discuss the different fields in the block header. We will discuss the nTime field. The height of a block in the blockchain is the number of blocks preceding it in the chain. So, in particular, the genesis block has a height of zero because there are zero blocks

preceding it. The immediate successor of the genesis block has height one, and so on and so forth.



Recall the block header structure that we discussed earlier. It contains all these fields, and recall that the fields, the 4-byte fields which have n at the beginning of the 4-byte fields, so these are integers. And these two fields, 'hashPrevBlock' and 'hashMerkleRoot,' are cryptographic hash function values. They are 32 bytes in length. This nTime field, which is 4 bytes in length in the block header, is populated with a timestamp in Unix time format to record the time of the block creation.

So, at the time when the block is created, the corresponding timestamp is put into the nTime field in the block header. It is in Unix time format. What is Unix time format? Unix time is the number of seconds which have elapsed since 12 a.m. UTC on January 1st, 1970. So, UTC stands for Coordinated Universal Time.

So, Unix time is the number of seconds which have elapsed since this time instant. Hence, it is an integer. And this integer is recorded in the nTime field in the block header. So, each node in the Bitcoin network has a local clock which is not necessarily synchronized with the local clocks of other nodes. Recall that the Bitcoin network is a peer-to-peer network, and in a network in general, the local clocks of different nodes are not synchronized.

That's because of, typically because of, clock drifts. For example, consider two nodes, A and B. Initially, their local clocks may be synchronized, but the crystals at the two nodes which maintain time, they may drift with time, or they may have slightly different frequencies. For example, one node may have a clock which runs at the speed of 10 MHz.

Another may have a crystal which runs at 10.00001 MHz. So, because of such differences in the frequencies of the crystals which maintain time and the drifts in these frequencies of these crystals, so hence, different nodes don't have always synchronized clocks.

Hence, the local clock of a node in the Bitcoin network may not be synchronized with the local clocks of the other nodes. So, there is no globally unique notion of time in the network. As an example, if the local clock in one Bitcoin network is 4 p.m. on a certain day, then the local clock in another node may be 4.01 p.m. on the same day, and the local clock of another node may be 3.59 p.m. on the same day, and so on and so forth. The Bitcoin system does not specify an explicit algorithm for calculating the nTime field in a candidate block. But it specifies a lower bound and an upper bound on the nTime.

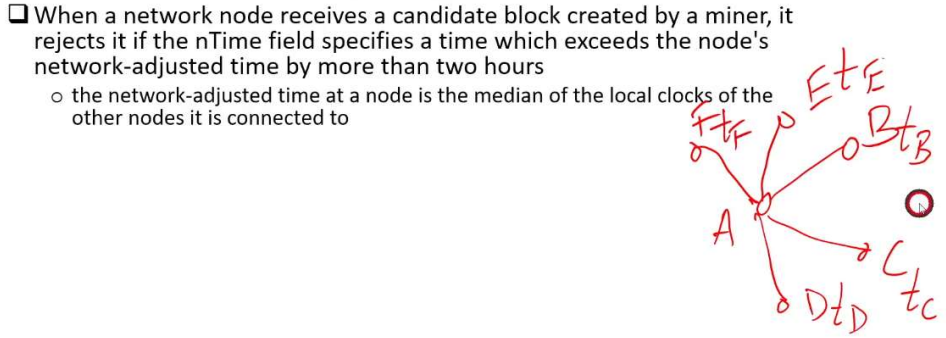
So, we'll specify what these lower bound and upper bound are. The Bitcoin system imposes the following two constraints to ensure that the timestamp in the nTime field is approximately correct. So, out of these constraints, one is a lower bound on nTime, and the other is an upper bound on nTime. So, the first constraint is this: in a candidate block at height N, the nTime field should be strictly greater than the median of the nTime values in the 11 blocks in the blockchain at heights N-1, N-2 up to N-11. Consider the previous 11 blocks in the blockchain and find the median of their nTime values.

- However, it imposes two constraints to ensure that the timestamp in the nTime field is approximately correct:
  - In a candidate block at height N, the nTime field is required to be strictly greater than the median of the nTime values in the 11 blocks in the blockchain at heights N-1, N-2, …, N-11

The nTime field in the current block should be strictly greater than this median. So, this says that the timestamps in different blocks are roughly in an increasing order, but they may not be strictly in an increasing order because this nTime field in the current block is only required to be greater than the median of the nTime values in the 11 blocks at these heights. Hence, the nTime values are not strictly in increasing order. So, this is the other constraint. When a network node receives a candidate block created by a miner, it rejects the block if the nTime field specifies the time which exceeds the node's network-adjusted time by more than two hours.

What is the network-adjusted time of a node? The network-adjusted time at a node is the median of the local clocks of the other nodes it is connected to. For example, consider a node A in the blockchain network. Suppose, it is connected to these other nodes, B, C, D, E and F. We can find the median of the local clocks of these other nodes. For example, B

has a local clock of t_B, then C has a local clock of t_C, D has a local clock of t_D, and so on and so forth.



F has a local clock of t_F. So, the neighbors of node A are these: B, C, D, E, F. And their corresponding local clock values are t_B, t_C, t_D, t_E and t_F. We find the median of these five values: t_B, t_C, t_D, t_E, and t_F. And that is a network-adjusted time of node A. A miner node is free to set the nTime field to any value which satisfies these above constraints. So, this is a constraint which specifies the lower bound on the nTime field, and this constraint specifies an upper bound on the nTime field.

Between this lower bound and upper bound, any value is valid. So, the miner node can set the nTime field to any value which satisfies these constraints. So, this first constraint specifies a lower bound on nTime, and it can be calculated from the current blocks in the blockchain. So, the miner node has to just look at the previous 11 blocks in the blockchain and read their nTime values and find the median. And then the minor node has to just ensure that nTime field in the current block has to be some value that is greater than that median value which it has just calculated.

But this upper bound specified by the second constraint cannot be explicitly calculated by the miner because it does not know the network-adjusted times at all the other nodes in the network. For example, consider this node A. It does not know the network-adjusted time at node E because it does not know the set of neighbors of node E, their local clock values, and so on and so forth. Hence, this upper bound cannot be explicitly calculated by a miner node. But the miner node can hope to satisfy this upper bound with high quality by using nTime values that are equal or close to its own network at the set time. So, typically in practice, the local clocks of different nodes are at least loosely synchronized.

For example, if the local clock value at a particular node is 4 p.m., then the local clocks of other nodes are close to 4 p.m. Hence, a miner node can use nTime values that are equal or

close to its own network-adjusted time, and this calculated network-adjusted time, that is, its own network-adjusted time, will be likely close to the network-adjusted times of the other nodes, that is, its neighboring nodes. Hence, by using this process, the miner node can hope to satisfy this second constraint. And also, we should notice that there is an interval of two hours, which is specified here. So, when a network node receives a candidate block, it rejects it only if it specifies a time which exceeds the node's network address time by more than two hours.

So, the possibility of the local clocks of different nodes being away from each other by more than two hours is, it's quite unlikely. Hence, typically this nTime field satisfies these constraints. We have discussed the nTime field in the block header. Now let's go back to the block header. We now discuss this nBits field in the block header.

- "nBits" field in block header encodes a 256-bit unsigned integer, called "target threshold" using base 256 scientific notation
- Let $b_1b_2b_3b_4$ be the four bytes of "nBits"
  - b_1: exponent
  - $b_2b_3b_4$: mantissa
- Target threshold is given by:
  - $T = b_2b_3b_4 \times 256^{b_1-3}$
  - where b_1, b_2, b_3, b_4 are interpreted as unsigned integers
- Examples are given in table
- Max. value of b_1 is 32 to ensure that T can be represented by a 256-bit string
- Reason:
  - T is compared with double SHA-256 hash of a block header
- Target threshold T is a network-wide setting which is adjusted by all the network nodes every 2016 blocks

nBits	Target Threshold	$b_1 - 3$
0x03123456	0x123456	0
0x02123456	0x1234	-1
0x05123456	0x1234560000	2
0x08123456	0x1234560000000000	5

It specifies a target which is related to the hard problem that a miner node must solve in order to successfully add a block. So, this nBits field in the block header, it encodes a 256-bit unsigned integer called the target threshold using base 256 scientific notation. Recall that the nBits field is 4 bytes in length. Let b_1, b_2, b_3, and b_4 be the 4 bytes of nBits. So, in this scientific notation, b_1 is the exponent, and b_2, b_3, and b_4 are the mantissa.

So, that is, 256-bit unsigned integer that is represented by these 4 bytes is given by the following. The target threshold corresponding to the nBits value is $T = b_1b_2b_3 \times 256^{b_1-3}$. We can see that b_1 appears in the exponent and b_2, b_3, b_4 are the mantissa. So, the target threshold is mantissa multiplied by 256 to the power exponent, that is b_1-3. Here b_1, b_2, b_3, b_4 are interpreted as unsigned integers.

So, some examples are given in this table. For example, suppose the nBits field is this: b_1 is 03; b_2 is 12; b_3 is 34; and b_4 is 56. Then the mantissa is b_2 b_3 b_4 which is this: 123456 and b_1-3 is 0 because b_1 is 03. So, b_1-3 is 0. Hence 256 power 0, that means no shift. Hence the target threshold is simply this: 0x123456. Now, consider another example where b_1 b_2 b_3 b_4 are as in this field.

So, b_1 is 2. So, b_1-3 is -1. So, this 256^{-1}, we have the exponent is -1. Hence we have $b_2 b_3 b_4 \times 256^{-1}$, which corresponds to a shift to the left by 1 byte. Hence, we take this value 123456 and shift it to the left by 1 byte; that gives us 1234. Hence, the mantissa is 1234. So, the target threshold is 1234 in hex.

Similarly, in this case, b_1 is 5. So, b_1-3 is 2. So, that corresponds to 256^2; that is a shift to the right by 2 bytes. Hence, the corresponding value of the target threshold is this: in hex, 1234560000, 256^2, which means we have shifted to the right by 2 bytes, and so on and so forth. So, these are some example values of nBits and the corresponding target threshold that is encoded by them. The maximum value of b_1 is 32.

This ensures that the target threshold T can be represented by a 256-bit string. So, the maximum value of b_1 is 32, so the maximum value of b_1-3 is 29. So, we have some values, b_2 b_3 b_4, and the exponent is 29, which means a shift to the right by 29 bytes. So, that corresponds to a 32-byte value. So, 32 bytes is 256 bits; hence, T can be represented by a 256-bit string.

So, the maximum value of b_1 is 32. So, what is the reason for ensuring that T can be represented by a 256-bit string? The reason is that this target threshold T is compared with the double SHA-256 hash of a block header. And recall that the output of an SHA-256 hash, as the name suggests, that is a 256-bit string. Since T is compared with the double SHA-256 hash of a certain block header, T is restricted to be a 256-bit string.

Now the target threshold T is a network-wide setting which is adjusted by all the network nodes every 2016 blocks. So, we'll discuss how the target threshold is adjusted by the network nodes. We now discuss how to find a valid block. So, the goal of a miner is to find a candidate block such that the double SHA-256 hash of its block header is $\leq T$. So, recall that in order to mine a new block, a miner has to solve a difficult problem. And this is that difficult problem that the miner has to solve.

The miner has to find the candidate block such that its double SHA-256 hash of the block header is $\leq T$. Now, in the block header, there is the nNonce field, which we discussed

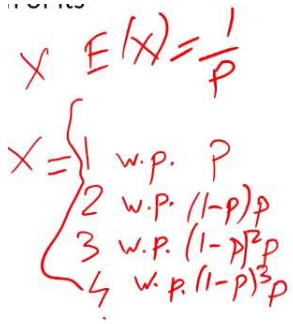
earlier. So, if we go back to the header of the block, then recall that one of the fields is the nNonce field, which is 4 bytes in length. So, the miner is free to set this nNonce field in the header, which is 4 bytes in length, to any value to achieve this goal. Since the nNonce field is 4 bytes in length, 4 bytes is 32 bits, so there are 2^{32} possibilities for setting the nNonce field. The miner has to try out different possibilities of nNonce values until it gets a candidate block such that the double SHA-256 hash of the block header is $\leq T$. So, how does the miner find such a block?

- Goal of miner is to find a candidate block such that double SHA-256 hash of its block header is $\leq T$
  - ❑ miner free to set "nNonce" field in header (which is 4 bytes long) to any value to achieve above
- Since computationally hard to find preimage of a given hash value miner needs to:
  - ❑ try out different "nNonce" values until double SHA-256 hash of its block header comes out to be $\leq T$
- Probability that for a given "nNonce" value, double SHA-256 hash of its block header comes out to be $\leq T$:
  - ❑ $p = \frac{T+1}{2^{256}}$
- Average number of trials required until success:
  - ❑ $\frac{1}{p}$
- E.g.: value of "nBits" field on Jan. 1, 2017 was 0x180375FF
  - ❑ average number of trials until success $\approx 2^{70} \approx 10^{21}$
- Such a large number of trials required until success is the reason why mining is a computationally hard problem
- A miner which successfully finds a block such that double SHA-256 hash of its block header comes out to be $\leq T$ is said to have:
  - ❑ found or mined a valid block

Recall that it is computationally hard to find the pre-image of a given hash value, so hence the miner has to try out different nNonce values until the double SHA-256 hash of its block header comes out to be $\leq T$. So, for example, recall that the miner initially creates a candidate block by putting some regular transactions, and the coinbase transaction, and a block header. So, first it tries by setting nNonce = 0, for example, and finds out the SHA-256 hash of the block header. If it is $\leq T$, then the miner is done. It has found a block whose double SHA-256 hash of the block header is $\leq T$. If not, then it tries with the next value, that is, nNonce = 1, for example, and then checks whether the double SHA-256 hash of its block header is $\leq T$, and so on and so forth. Then it tries nNonce=3 and then nNonce=4, and so on and so forth.

So, it tries different values of nNonce until it gets a block for which the double SHA-256 hash of its block header is $\leq T$. Now, the probability that for a given nNonce value, the double SHA-256 hash of its block header comes out to be $\leq T$ is this: $T + 1/2^{256}$, so this

is the required probability, and that is because recall that the output of a cryptographic hash function can be assumed to take any value equally uniformly at random among the 2^{256} possible values. So, there are 2^{256} possible output values of the double SHA-256 hash, and out of these, the hash is $\leq T$ in T+1 of the values, that is 0, 1, up to T. Hence, the probability is $T+1/2^{256}$. The probability that for a given nNonce value, the double SHA-256 hash is $T+1/2^{256}$. Now, what is the average number of trials required until success?



The claim is that the average number of trials is required until success is 1/p. So, let x be the number of trials required until success. So, x is an integer; it is a positive integer, which is the number of trials required until success. So, in each trial, there is a success with probability p. So, x=1, if the first trial results in success, so x=1 with probability p. And x=2 if the first trial results in failure, which happens with probability 1-p, and the second trial results in success, which happens with probability p. Similarly, x is 3 if the first two trials result in failure and the third trial results in success, which happens with probability $(1-p)^2 p$. x is 4 with probability $(1-p)^3 p$, and so on and so forth.

So, what is this distribution? This is a geometric distribution. So, x is a geometric random variable. So, the success probability of x is p. So, there is a standard result that if you have a geometric random variable with success probability p, then its expectation is 1/p. So, in this case, we have that expectation of x is 1/p. So, from this fact it follows that the average number of trials required until success is 1/p. Here's an example: the value of the nBits field on Jan 1st, 2017, was this, in hex: 180375FF.

The average number of trials until success was $\approx 2^{70}$. If you compute this, it comes out to be 10^{21}. Now, since a large number of trials are required until success, that is the reason why mining is a computationally hard problem. So, on average one has to try, at this time, that is Jan 1st, 2017, one had to try approximately 10^{21} different values until one got a value which corresponded to success. A miner which successfully finds a block such that double

SHA-256 hash of its block header comes out to be $\leq T$ is said to have found or mined a valid block.

Now, what are the actions taken after finding a valid block? So, what does a miner do after it finds a valid block at height T? So, it immediately broadcasts the block on the Bitcoin network. And then the miner appends the block to its local copy of the blockchain, and it begins mining for the next block at height N+1. Since the miner has successfully found a valid block at height N, it has received the block reward for this particular block. Then, it broadcasts the block on the Bitcoin network so that the other nodes know that a block has been successfully mined, and they add that block to their local copy of the blockchain.

And then this node appends the block to its local copy and begins mining for the next block at height N+1. Initially, assume that all the other nodes have the same copy of the blockchain, consisting of blocks from the Genesis block to a block of height N-1. When the new block at height N arrives at one of the other nodes, the recipient node—which was still mining for a block at height N—stops mining and appends the new block which it just received to its local copy of the blockchain. And then it starts mining for the next block at height N+1. So, these are actions taken by a miner which receives a new block at height N. It stops mining and appends a new block to its local copy of the blockchain and starts mining for the next block, which is at height N+1.

Now, let's discuss typical double SHA-256 hash computation rates. The rate at which a computing device can calculate the double SHA-256 hashes of block headers is measured in these units: megahashes per second, gigahashes per second, or terahashes per second. Now, the rate of a typical PC is less than 100 megahashes per second. So, in that example that we discussed earlier, one would need to calculate 2^{70} double SHA-256 hashes on average for success. So, to calculate 2^{70} double SHA-256 hashes, a PC operating at 100 megahashes per second will require more than 300,000 years.

- Rate of a typical personal computer:
  - ❑ less than 100 MH/s
- To calculate 2^{70} double SHA-256 hashes, a PC operating at 100 MH/s will require:
  - ❑ more than 300,000 years
- Nowadays, mining is done using ASICs specifically designed to compute several instances of the double SHA-256 function in parallel
  - ❑ mining rigs, which combine several such ASICs, are available in the market and can deliver hash rates of the order of a few TH/s
- A single mining rig operating at 1 TH/s will still require more than 30 years to calculate 2^{70} double SHA-256 hashes of block headers

Thus, it is practically impossible for a PC to have a high probability of success. Nowadays, mining is done using application-specific integrated circuits that are specifically designed to compute several instances of the double SHA-256 function in parallel. So, it's likely that out of these parallelly computed hash values, at least one of them will be $\leq T$ after several instances of computation. Mining rigs, which combine several such ASICs, are available in the market and can deliver hash rates of the order of a few terahashes per second. A single mining rig that operates at 1 terahash per second will still require more than 30 years to calculate 2^{70} double SHA-256 hashes of block headers.

Hence, it's unlikely that a single mining rig will succeed in finding a valid block in a reasonable time. Mining is typically performed by companies that have consolidated thousands of such mining rigs into data centers. So, if we add up the hash computing capabilities of all these mining rigs in parallel, then it comes out to be a large value. Now, let us discuss how the target threshold T, which is encoded in nBits, is chosen. The Bitcoin protocol specifies that the average time required to mine a valid block should be 10 minutes.

So, how can the target threshold T be chosen to achieve this target? So, one rule is that the target threshold T is updated once every 2016 blocks by all the nodes in the Bitcoin network. In particular, each time a miner node starts mining for a candidate block whose height is a multiple of 2016, it updates the value of the target threshold T. So, why is this value, 2016, chosen? This 2016 is the number of blocks which would be found in two weeks if a block was found every 10 minutes.

- ❑ Note: 2016 is the number of blocks which would be found in two weeks if a block was found every 10 minutes, i.e., $2016 = 14 \times 24 \times 6$
- Time which was spent in finding the previous 2016 blocks is estimated by:
  - ❑ taking the difference of the nTime fields of blocks whose heights differ by 2016
- Recall: average number of trials required to find a valid block is $\frac{2^{256}}{T+1}$
- The update formula for finding the new value T_{new} from the old value T_{old} is given by:
  - ❑ $T_{new} = T_{old} \times \frac{\text{Measured duration for finding 2016 blocks in seconds}}{2016 \times 600}$

That is, 2016 is this: 14 days in two weeks multiplied by 24 hours per day multiplied by 6 ($2016 = 14 \times 24 \times 6$). That is, since a block is found every 10 minutes, there are 6 blocks found in 1 hour. That is how we get this value, 2016. The time which was spent in finding the previous 2016 blocks is estimated by taking the difference of the nTime fields of the

blocks whose heights differ by 2016. So, we can consider the nTime field in the latest block and the block which is 2016 blocks previously.

So, we can take the difference of the nTime fields in these blocks, and thereby we can obtain the time which was spent in finding the previous 2016 blocks. Recall that the average number of trials required to find a valid block is $\frac{2^{256}}{T+1}$.

Hence, the amount of time required to find a valid block is inversely proportional to T. And that is quite consistent with intuition. The larger the T, it becomes easier to find a block whose block header hashes to some value $\leq T$. So, this fact is important that the time required to find a valid block is inversely proportional to T. The update formula for finding the new value T_{new} from the old value T_{old} is given by this expression, by this equation.

$$T_{new} = T_{old} \times \frac{measured\ duration\ for\ finding\ 2016\ blocks\ in\ seconds}{2016 \times 600}.$$

We now explain this formula. So, recall that we want that the next block should be mined in roughly 10 minutes. Now, 10 minutes corresponds to this. The duration for finding the next block is 10 minutes, which is 600 seconds.

So, for finding the next 2016 blocks, the duration we want is 2016×600. And from the previous blocks, we can look up the measured duration for finding 2016 blocks in seconds. Now we know that the duration is inversely proportional to the value T; from this formula, we can see that. So, we have a simple inverse proportion problem. So, T_{old} corresponds to this measured duration, which is in the numerator of this formula.

So, this T_{old} corresponds to this measured duration. And T_{new} corresponds to 2016×600, which is what we want it to be. Since T is inversely proportional to the duration, we get this formula. $T_{new} = T_{old} \times \frac{measured\ duration\ for\ finding\ 2016\ blocks\ in\ seconds}{2016 \times 600}$.

So, by setting the target to this value, we'll ensure that on average, 10 minutes will be required to find the next block.

We ensure that for finding the next 2016 blocks, approximately 2016 into 10 minutes will be required. So, this is how the threshold T is adjusted. So, in summary, we discussed more information about the Bitcoin network. So, we discussed several of the fields. We discussed how nTime is set in the Bitcoin network.

We discussed how the threshold is set. We discussed several aspects of mining. We will continue our discussion on Bitcoin in the next lecture. Thank you.