CST 428 BLOCK CHAIN TECHNOLOGIES

S8 CSE – ELECTIVE

MODULE – 3

Consensus Algorithms and Bitcoin

Consensus Problem

- The distributed consensus problem has been studied extensively in distributed systems research since the late 1970s
- Distributed systems are classified into two main categories, namely **message passing and shared memory**
- In the context of blockchain, we are concerned with the message passing type of distributed systems, where participants on the network communicate with each other via passing messages to each other.
- Blockchain is a distributed system that relies upon a consensus mechanism, which ensures the safety and liveness of the blockchain network.

Consensus Problem

- **Traditional consensus mechanisms**, which are also known as **fault tolerant distributed consensus, classical distributed consensus, or pre-Bitcoin distributed consensus.**
- The problem, however, is that any generals could potentially be disloyal and act maliciously to obstruct agreement upon a united plan.
- The requirement now becomes that every honest general must somehow agree on the same decision even in the presence of treacherous generals.

The Byzantine generals problem

- The problem of reaching agreement in the **presence of faults or Byzantine consensus** was first formulated by **M. Pease, R. Shostak, and L. Lamport**.
- In distributed systems, a common goal is to achieve consensus (agreement) among nodes on the network even in the presence of faults.
- In order to explain the problem, Lamport came up with an allegorical representation of the problem and named it the **Byzantine generals problem**.
- The Byzantine generals problem metaphorically depicts a situation where a Byzantine army,divided into different units, is spread around a city.

The Byzantine generals problem

- A general commands each unit, and they can only communicate with each other using a messenger.
- To be successful, the generals must coordinate their plan and decide whether to attack or retreat.
- The problem, however, is that any generals could potentially be disloyal and act maliciously to obstruct agreement upon a united plan.
- The requirement now becomes that every honest general must somehow agree on the same decision even in the presence of treacherous generals.

The Byzantine generals problem

- In order to address this issue, honest (loyal) generals must reach a majority agreement on their plan.
- In the digital world, generals are represented by computers (nodes) and communication links are messengers carrying messages. Disloyal generals are faulty nodes.

Fault tolerance

- A fundamental requirement in a consensus mechanism is that it must be fault-tolerant.
- In other words, it must be able to tolerate a number of failures in a network and should continue to work even in the presence of faults.
- This naturally means that there has to be some limit to the number of faults a network can handle, since no network can operate correctly if a large majority of its nodes are failing.
- Based on the requirement of fault tolerance, consensus algorithms are also called fault-tolerant algorithms, and there are two types of fault- tolerant algorithms.

Types of fault-tolerant consensus

- Fault-tolerant algorithms can be divided into two types of fault-tolerance.
- The first is **Crash fault-tolerance** (**CFT**) and the other is **Byzantine fault-tolerance** (**BFT**). CFT covers only crash faults or, in other words, benign faults. In contrast, BFT deals with the type of faults that are arbitrary and can even be malicious.
- **Replication** is a standard approach to make a system fault-tolerant.
- Replication results in a synchronized copy of data across all nodes in a network. This technique improves the fault tolerance and availability of the network.

Types of fault-tolerant consensus

This means that even if some of the nodes become faulty, the overall system/network remains available due to the data being available on multiple nodes.

There are two main types of replication techniques:

- **Active replication**, which is a type where each replica becomes a copy of the original state machine replica.
- **Passive replication**, which is a type where there is only a single copy of the state machine in the system kept by the primary node, and the rest of the nodes/replicas only maintain the state.

CFT algorithms

- A distributed system faces a number of threats. Processes may crash, devices at some location may fail or a network connection may stop working. For enterprise use, the consensus algorithm must have resilience against a variety of threats.
- **Crash fault tolerance (CFT)** builds a degree of resiliency in the protocol, so that the algorithm can correctly take the process forward and reach consensus, even if certain components fail.
- One of the most fundamental algorithms in this space is **Paxos**

Paxos

- Paxos - developed by Leslie Lamport
- It is the most fundamental distributed consensus algorithm, allowing consensus over a value under unreliable communications.
- In other words, Paxos is used to build a reliable system that works correctly, even in the presence of faults.
  - Paxos makes use of ***2F + 1*** processes to ensure fault tolerance in a network where processes can crash fault, that is, experience benign failures.
  - Benign failure means either the loss of a message or a process stops. In other words, Paxos can tolerate one crash failure in a three-node network.

Paxos

- Paxos is a **two-phase protocol**.
  - The first phase is called the ***prepare*** phase, and the
  - next phase is called the ***accept*** phase.
- Paxos has **proposer and acceptors** as participants,
  - where the proposer is the replicas or nodes that propose the values and
  - acceptors are the nodes that accept the value.

How Paxos works

- The Paxos protocol assumes an asynchronous message-passing network with less than 50% of crash faults.
- As usual, the critical properties of the Paxos consensus algorithm are safety and liveness.
- Under ***safety,*** we have:
  - **Agreement**, which specifies that no two different values are agreed on. In other words, no two different learners learn different values.
  - **Validity**, which means that only the proposed values are decided. In other words, the values chosen or learned must have been proposed by a processor.
- Under ***liveness***, we have:
  - **Termination**, which means that, eventually, the protocol is able to decide and terminate. In other words, if a value has been chosen, then eventually learners will learn it.

How Paxos works

- Processes can assume different roles, which are listed as follows:
  - **Proposers**, elected leader(s) that can propose a new value to be decided.
  - **Acceptors**, which participate in the protocol as a means to provide a majority decision.
  - **Learners**, which are nodes that just observe the decision process and value.

How Paxos works

- The key idea behind Paxos is that the proposer node proposes a value, which is considered final only if a majority of the acceptor nodes accept it.
- The learner nodes also learn this final decision.
- Paxos can be seen as a protocol that is quite similar to the two-phase commit protocol.
- **Two phase commit** (**2PC**) is a standard atomic commitment protocol to ensure that transactions are committed in distributed databases only if all participants agree to commit.
- Even if a single node cannot agree to commit the transaction, it is fully rolled back.

How Paxos works

- Similarly, in Paxos, in the first phase, the proposer sends a proposal to the acceptors, if and when they accept the proposal, the proposer broadcasts a request to commit to the acceptors.
- Once the acceptors commit and report back to the proposer, the proposal is considered final,and the protocol concludes.
- In contrast with the two-phase commit, Paxos introduced ordering (sequencing to achieve total order) of the proposals and majority-based acceptance of the proposals instead of expecting all nodes to agree (to allow progress even if some nodes fail).

How the Paxos protocol works step by step:

1. The proposer proposes a value by broadcasting a message, <*prepare(n)*>, to all acceptors.
2. Acceptors respond with an acknowledgment message if proposal *n* is the highest that the acceptor has responded to so far. The acknowledgment message <*ack(n,v, s)*> consists of three variables where *n* is the proposal number, *v* is the proposal value of the highest numbered proposal the acceptor has accepted so far, and *s* is the sequence number of the highest proposal accepted by the acceptor so far. This is where acceptors agree to commit the proposed value. The proposer now waits to receive acknowledgment messages from the majority of the acceptors indicating the **chosen** value.
3. If the majority is received, the proposer sends out the "accept" message <*accept(n, v)*> to the acceptors.
4. If the majority of the acceptors accept the proposed value (now the "accept" message), then it is decided: that is, agreement is achieved.
5. Finally, in the learning phase, acceptors broadcast the "accepted" message <*accepted(n,v)*> to the proposer. This phase is necessary to disseminate which proposal has been finally accepted. The proposer then informs all other learners of the decided value. Alternatively, learners can learn the decided value via a message that contains the accepted value (decision value) multicast by acceptors.

Paxos

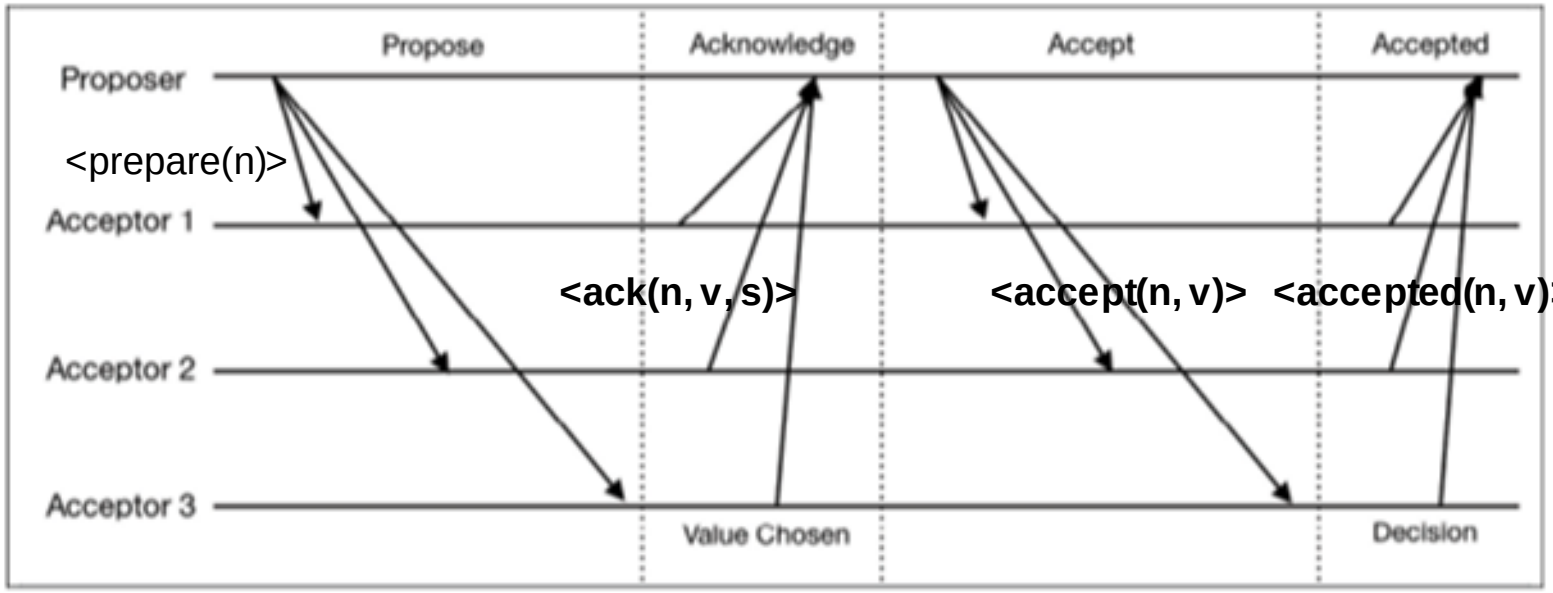


Figure 5.1: How Paxos works

How Paxos achieves safety and liveness

- **Agreement** is ensured by enforcing that only one proposal can win votes from a majority of the acceptors.
- • **Validity** is ensured by enforcing that only the genuine proposals are decided. In other words, no value is committed unless it is proposed in the proposal message first.
- • **Liveness** or termination is guaranteed by ensuring that at some point during the protocol execution, eventually there is a period during which there is only one fault-free proposer.

CFT Algorithm-RAFT

- The Raft protocol is a CFT consensus mechanism developed by Diego Ongaro and John Ousterhout at Stanford University.)
- In Raft, the leader is always assumed to be honest.
- At a conceptual level, it is a replicated log for a **replicated state machine** (**RSM**) where a unique leader is elected every "term" (time division) whose log is replicated to all follower nodes.
- Raft is composed of three sub-problems:
  - **Leader election** (a new leader election in case the existing one fails)
  - **Log replication** (leader to follower log synch)
  - **Safety** (no conflicting log entries (index) between servers)
- The Raft protocol ensures election safety, leader append only, log matching, leader completeness, and state machine safety.
- Each server in Raft can have either a **follower**, **leader**, or **candidate** state.
- The protocol ensures election **safety** (that is, only one winner each election term) and **liveness** (that is, some candidate must eventually win).

How Raft works

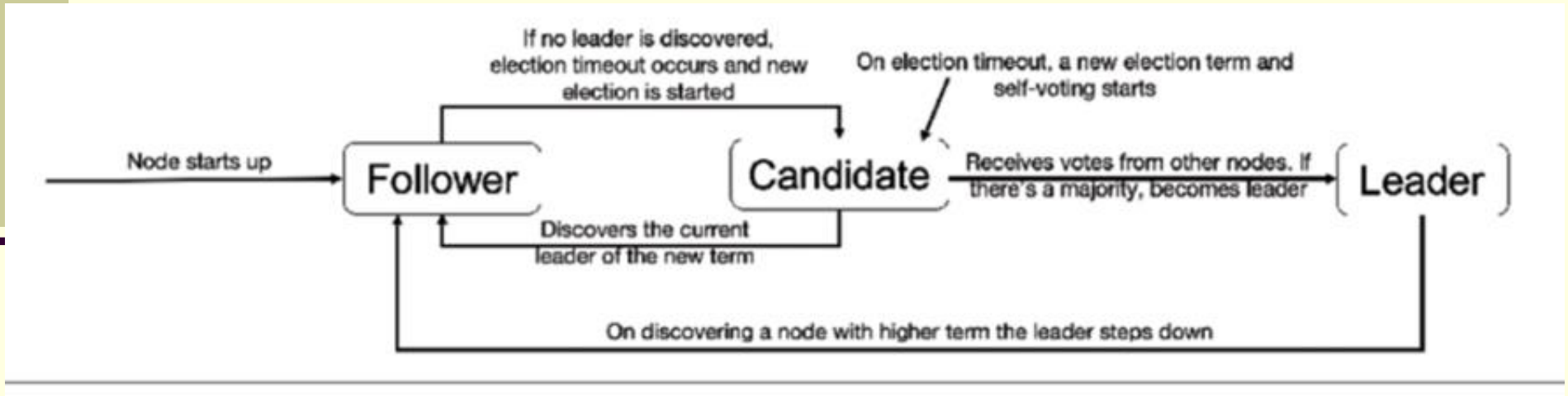
- At a fundamental level, the protocol is quite simple and can be described simply by the following sequence:

Node starts up –> Leader election –> Log replication

1. First, the node starts up.
2. After this, the leader election process starts. Once a node is elected as leader, all changes go through that leader.
3. Each change is entered into the node's log.
4. Log entry remains uncommitted until the entry is replicated to follower nodes and the leader receives write confirmation votes from a majority of the nodes, then it is committed locally.
5. The leader notifies the followers regarding the committed entry.
6. Once this process ends, agreement is achieved.

RAFT

- Each server in Raft can have either a follower, leader, or candidate state



Log replication

Log replication logic can be visualized in the following diagram. The aim of log replication is to synchronize nodes with each other.

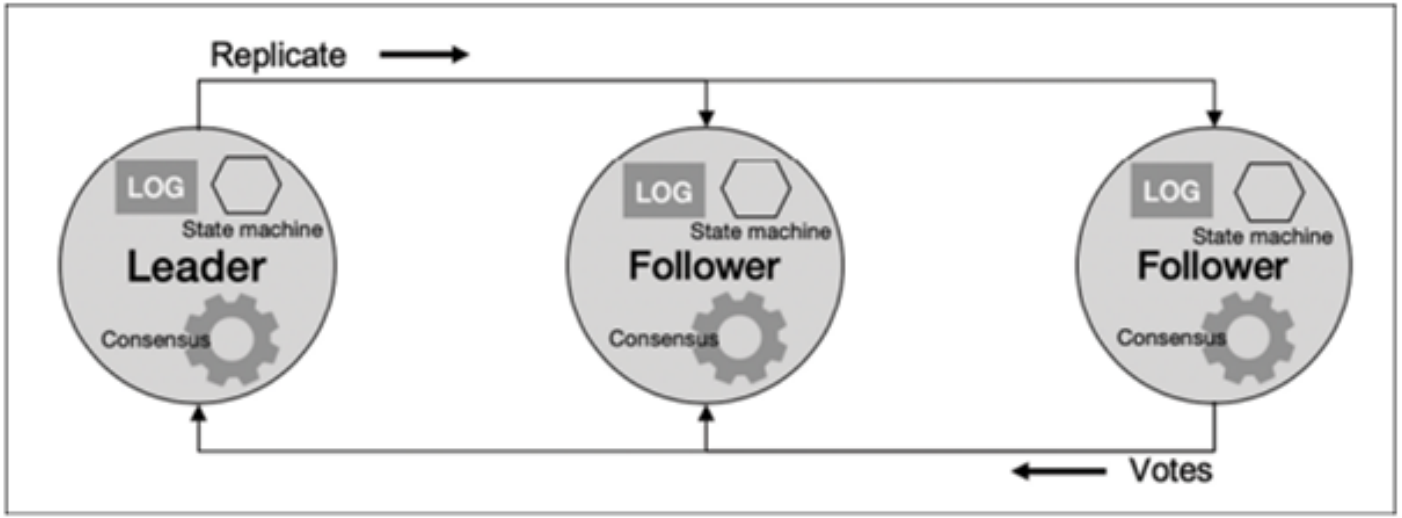


Figure 5.3: Log replication mechanism

RAFT

- Log replication is a simple mechanism. As shown in the preceding diagram, the leader is responsible for log replication.
- Once the leader has a new entry in its log, it sends out the
- requests to replicate to the follower nodes.
- When the leader receives enough confirmation votes back from the follower nodes indicating that the replicate request has been accepted and processed by the followers, the leader commits that entry to its local state machine.
- At this stage, the entry is considered committed.

Byzantine fault- tolerance (BFT) algorithms

1. Practical Byzantine Fault Tolerance (PBFT)

- **Practical Byzantine Fault Tolerance** (**PBFT**) was developed in 1999 by Miguel Castro and Barbara Liskov.
- PBFT, as the name suggests, is a protocol developed to provide consensus in the presence of Byzantine faults.
- PBFT comprises three sub-protocols called **normal operation**, **view change**, and **checkpointing**.

Normal operation sub-protocol refers to a scheme that is executed when everything is running normally and no errors are in the system.

View change is a sub-protocol that runs when a faulty leader node is detected in the system.

Checkpointing is another sub-protocol, which is used to discard the old data from the system.

1. Practical Byzantine Fault Tolerance (PBFT)

- The PBFT protocol comprises **three phases or steps**. These phases run in a sequence to achieve consensus.

pre-prepare, prepare, and commit,

- The protocol runs in rounds where, in each round, an elected leader node, called the primary node, handles the communication with the client.
- In each round, the protocol progresses through the three previously mentioned phases.
- The participants in the PBFT protocol are called **replicas**, where one of the replicas becomes **primary as a leader** in each round, and the rest of the nodes acts as **backups.**
- **E**ach node maintains a local log, and the logs are kept in sync with each other via the consensus protocol: that is, PBFT.

1. Practical Byzantine Fault Tolerance (PBFT)

- In order to tolerate Byzantine faults, the minimum number of nodes required is $N = 3F + 1$, where N is the number of nodes and F is the number of faulty nodes.
- PBFT ensures Byzantine fault tolerance as long as the number of nodes in a system stays $N >= 3F + 1$.

How the PBFT protocol works.

- In summary, when a client sends a request to a primary, the protocol initiates a sequence of operations between replicas, which eventually leads to consensus and a reply back to the client.
- These sequences of operations are divided into different phases:

  Pre-prepare

  Prepare

  Commit

- In addition, each replica maintains a local state comprising three main elements:

  Service state

  A message log

  A number representing that replica's current view

How the PBFT protocol works

- **Pre-prepare**:
- This is the first phase in the protocol, where the primary node, or **primary**, receives a request from the client.
- The primary node assigns a sequence number to the request. It then sends the pre-prepare message with the request to all backup replicas.
- When the pre-prepare message is received by the backup replicas, it checks a number of things to ensure the validity of the message:

  First, whether the digital signature is valid.

  After this, whether the current view number is valid.

  Then, that the sequence number of the operation's request message is valid.

  Finally, if the digest/hash of the operation's request message is valid.
- If all of these elements are valid, then the backup replica accepts the message. After accepting the message, it updates its local state and progresses toward the prepare phase.

How the PBFT protocol works

Prepare:

- A prepare message is sent by each backup to all other replicas in the system.
- Each backup waits for at least *2F + 1* prepare messages to be received from other replicas.
- They also check whether the prepare message contains the same view number, sequence number, and message digest values.
- If all these checks pass, then the replica updates its local state and progresses toward the commit phase.

How the PBFT protocol works

Commit:

- In the commit phase, each replica sends a commit message to all other replicas in the network.
- The same as the prepare phase, replicas wait for *2F + 1* commit messages to arrive from other replicas. The replicas also check the view number, sequence number, and message digest values. If they are valid for *2F+ 1* commit messages received from other replicas, then the replica executes the request, produces a result, and finally, updates its state to reflect a commit. If there are already some messages queued up, the replica will execute those requests first before processing the latest sequence numbers. Finally, the replica sends the result to the client in a reply message.
- The client accepts the result only after receiving *2F+ 1* reply messages containing the same result.

How PBFT works in some more detail.

- At a high level, the protocol works as follows:

1**. A client sends a request to invoke a service operation in the primary.**

2. The primary multicasts the request to the backups.

3. Replicas execute the request and send a reply to the client.

4. The client waits for replies from different replicas with the same result; this is the result of the operation.

- **The pre-prepare sub-protocol algorithm**:

1. Accepts a request from the client.
2. Assigns the next sequence number.
3. Sends the pre-prepare message to all backup replicas.

How PBFT works in some more detail

- **The prepare sub-protocol algorithm**:

1. Accepts the pre-prepare message. If the backup has not accepted any pre-prepare messages for the same view or sequence number, then it accepts the message.

2. Sends the prepare message to all replicas.

- **The commit sub-protocol algorithm**:

1. The replica waits for *2F* prepare messages with the same view, sequence, and request.

2. Sends a commit message to all replicas.

3. Waits until a *2F + 1* valid commit message arrives and is accepted.

4. Executes the received request.

5. Sends a reply containing the execution result to the client.

1. Practical Byzantine Fault Tolerance (PBFT)

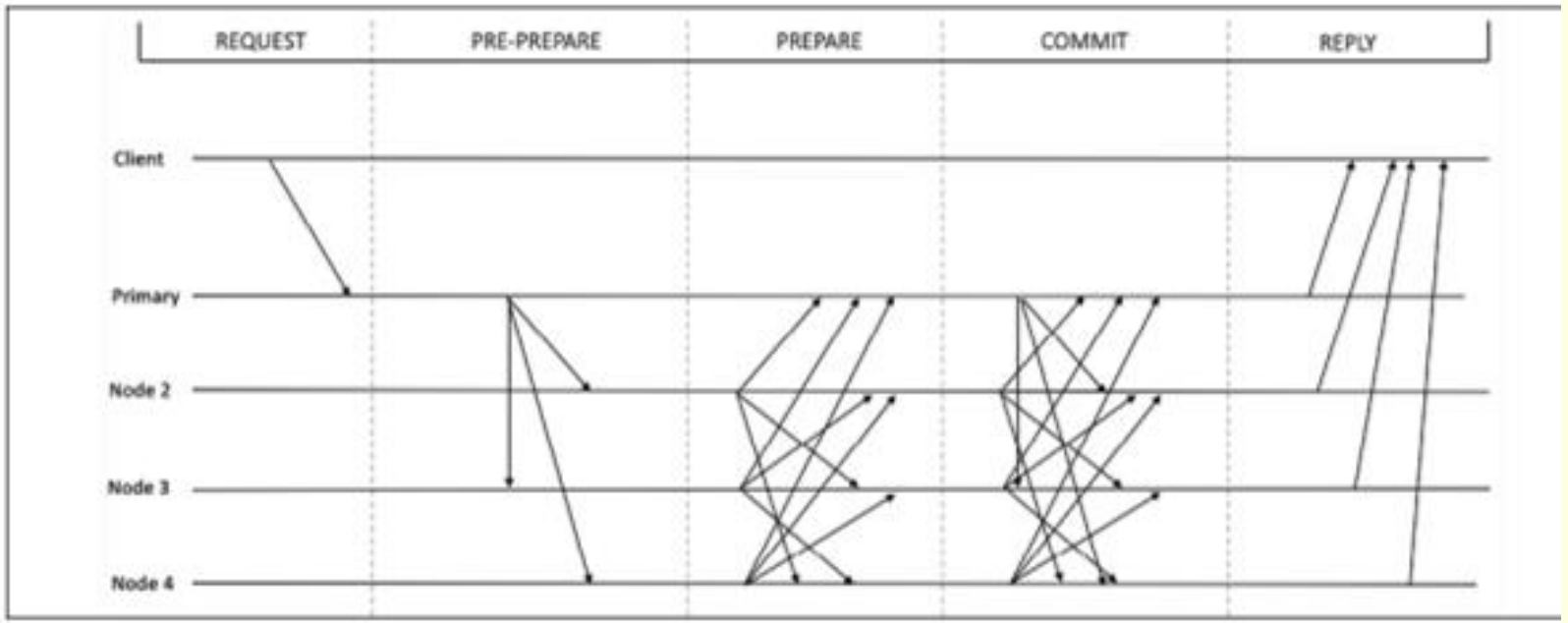


Figure 5.4: PBFT protocol

1. Practical Byzantine Fault Tolerance (PBFT)

- The algorithm for the view-change protocol is shown as follows:

  1. Stop accepting pre-prepare, prepare, and commit messages for the current view

  2. Create a set of all the certificates prepared so far

  3. Broadcast a view-change message with the next view number and a set of all the prepared certificates to all replicas

the collection of 2F + 1 messages of a particular type is a certificate

Certificates in PBFT:

- Certificates in PBFT protocols are used to demonstrate that at least *2F + 1* nodes have stored the required information.
- In other words, the collection of *2F + 1* messages of a particular type is considered a certificate.
- For example, if a node has collected *2F + 1* messages of type prepare, then combining it with the corresponding pre-prepare message with the same view, sequence, and request represents a certificate, called a prepared certificate. Similarly, a collection of *2F +1* commit messages is called a commit certificate.

- There are also a number of variables that the PBFT protocol maintains in order to execute the algorithm.

State variable	Explanation
v	View number
m	Latest request message
n	Sequence number of the message
h	Hash of the message
i	Index number
C	Set of all checkpoints
P	Set of all pre-prepare and corresponding prepare messages
O	Set of pre-prepare messages without corresponding request messages

Types of messages:

- The PBFT protocol works by exchanging several messages. A list of these messages is presented as follows with their format and direction.

Message	From	To	Format	Signed by
Request	Client	Primary	<REQUEST, m>	Client
Pre-Prepare	Primary	Backups	<PRE-PREPARE, v, n, h>	Client
Prepare	Replica	Replicas	<PREPARE, v, n, h, i>	Replica
Commit	Replica	Replicas	<COMMIT, v, n, h, i>	Replica
Reply	Replicas	Client	<REPLY, r, i>	Replica
View change	Replica	Replicas	<VIEWCHANGE, v+1, n, C, P, i>	Replica
New view	Primary replica	Replicas	<NEWVIEW, v + 1, v, O>	Replica
Checkpoint	Replica	Replicas	<CHECKPOINT, n, h, i>	Replica

Some specific message types that are exchanged during the PBFT protocol.-**View Change**

- View-change occurs when a primary is suspected faulty. This phase is required to ensure protocol progress.
- With the view change sub-protocol, a new primary is selected, which then starts normal mode operation again. The new primary is selected in a round-robin fashion.
- When a backup replica receives a request, it tries to execute it after validating the message, but for some reason, if it does not execute it for a while, the replica times out and initiates the view change sub-protocol.
- In the view change protocol, the replica stops accepting messages related to the current view and updates its state to view-change.
- The only messages it can receive in this state are checkpoint messages, view-change messages, and new-view messages. After that, it sends a view-change message with the next view number to all replicas.

View-change

- When this message arrives at the new primary, the primary waits for at least *2F* view-change messages for the next view. If at least *2F* view-change messages are received it broadcasts a new view message to all replicas and progresses toward running normal operation mode once again.
- When other replicas receive a new-view message, they update their local state accordingly and start normal operation mode.
- The algorithm for the view-change protocol is shown as follows:

1. Stop accepting pre-prepare, prepare, and commit messages for the current view.

2. Create a set of all the certificates prepared so far.

3. Broadcast a view-change message with the next view number and a set of all the prepared certificates to all replicas.

View-change protocol

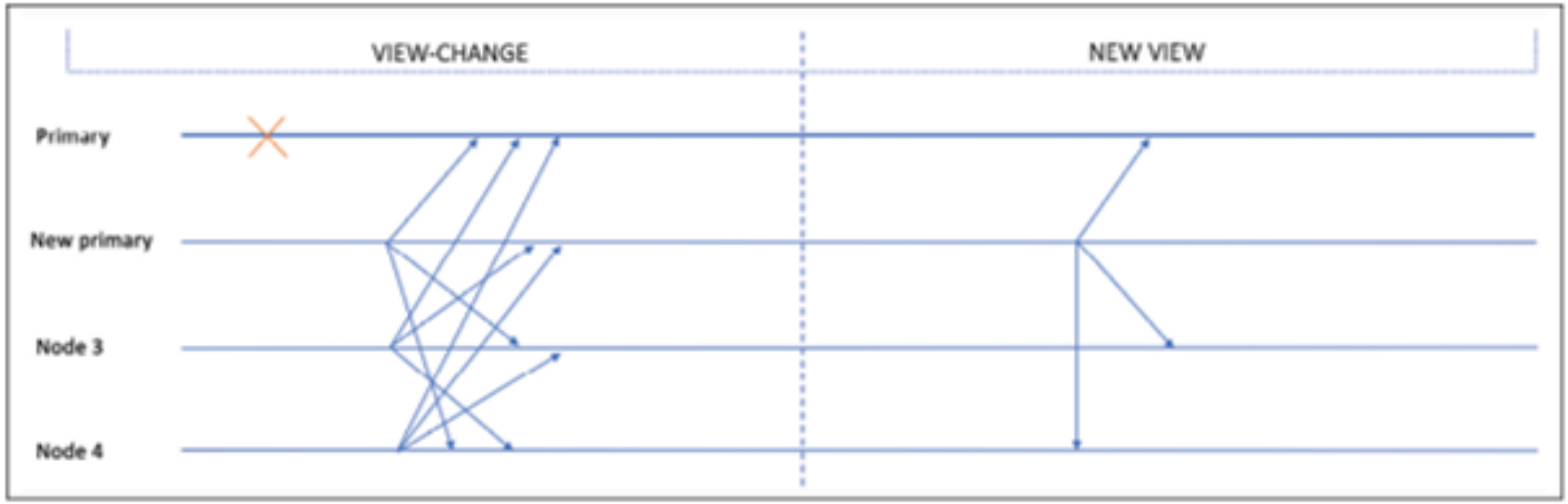


Figure 5.5: View-change sub-protocol

The view-change sub-protocol is a mechanism to achieve liveness. Three smart techniques are used in this sub-protocol to ensure that, eventually, there is a time when the requested operation executes:

1. **A replica that has broadcast the view-change message waits for *2F+1* view-change messages and then starts its timer. If the timer expires before the node receives a new-view message for the next view, the node will start the view change for the next sequence but will increase its timeout value. This will also occur if the replica times out before executing the new unique request in the new view.**

2. **As soon as the replica receives *F+1* view-change messages for a view number greater than its current view, the replica will send the view-change message for the smallest view it knows of in the set so that the next view change does not occur too late. This is also the case even if the timer has not expired; it will still send the view change for the smallest view.**

3. **As the view change will only occur if at least *F+1* replicas have sent the view-change message, this mechanism ensures that a faulty primary cannot indefinitely stop progress by successively requesting view changes.**

The checkpoint sub-protocol:

- Checkpointing is a crucial sub-protocol. It is used to discard old messages in the log of all replicas.
- With this, the replicas agree on a stable checkpoint that provides a snapshot of the global state at a certain point in time.
- This is a periodic process carried out by each replica after executing the request and marking that as a checkpoint in its log.
- A variable called **low watermark** (in PBFT terminology) is used to record the sequence number of the last stable checkpoint.
- This checkpoint is then broadcast to other nodes. As soon as a replica has at least *2F+1* checkpoint messages, it saves these messages as proof of a stable checkpoint. It discards all previous **pre-prepare, prepare**, and **commit** messages from its logs.

PBFT-Advantages & Disadvantages

- Advantages
  - Provides immediate and deterministic transaction finality
  - Energy efficient as compared to PoW
- Disadvantages
  - Not very scalable - more suitable for consortium networks, instead of public blockchains
  - Faster than PoW protocols
  - Sybil attacks can be carried out

Nakamoto consensus or PoW

- **Nakamoto consensus**, or **PoW**, was first introduced with Bitcoin in 2009. Since then, it has stood the test of time and is the longest-running blockchain network.
- This test of time is a testament to the efficacy of the PoW consensus mechanism.
- At a fundamental level, the PoW mechanism is designed to mitigate Sybil attacks, which facilitates consensus and the security of the network.
- It is quite easy to obtain multiple identities and try to influence the network. However, in Bitcoin, due to the hashing power requirements, this attack is mitigated.

How PoW works

- PoW makes use of hash puzzles.
- A node proposes a block has to find a nonce such that *H (nonce || previous hash || Tx || Tx|| . . . ||Tx) < Threshold value*.
- **The process can be summarized as follows:**

• New transactions are broadcast to all nodes on the network.

• Each node collects the transactions into a candidate block.

• Miners propose new blocks.

• Miners concatenate and hash with the header of the previous block.

• The resultant hash is checked against the target value, that is, the network difficulty target value.

• If the resultant hash is less than the threshold value, then PoW is solved, otherwise, the nonce is incremented and the node tries again. This process continues until a resultant hash is found that is less than the threshold value.

Nakamoto consensus or PoW

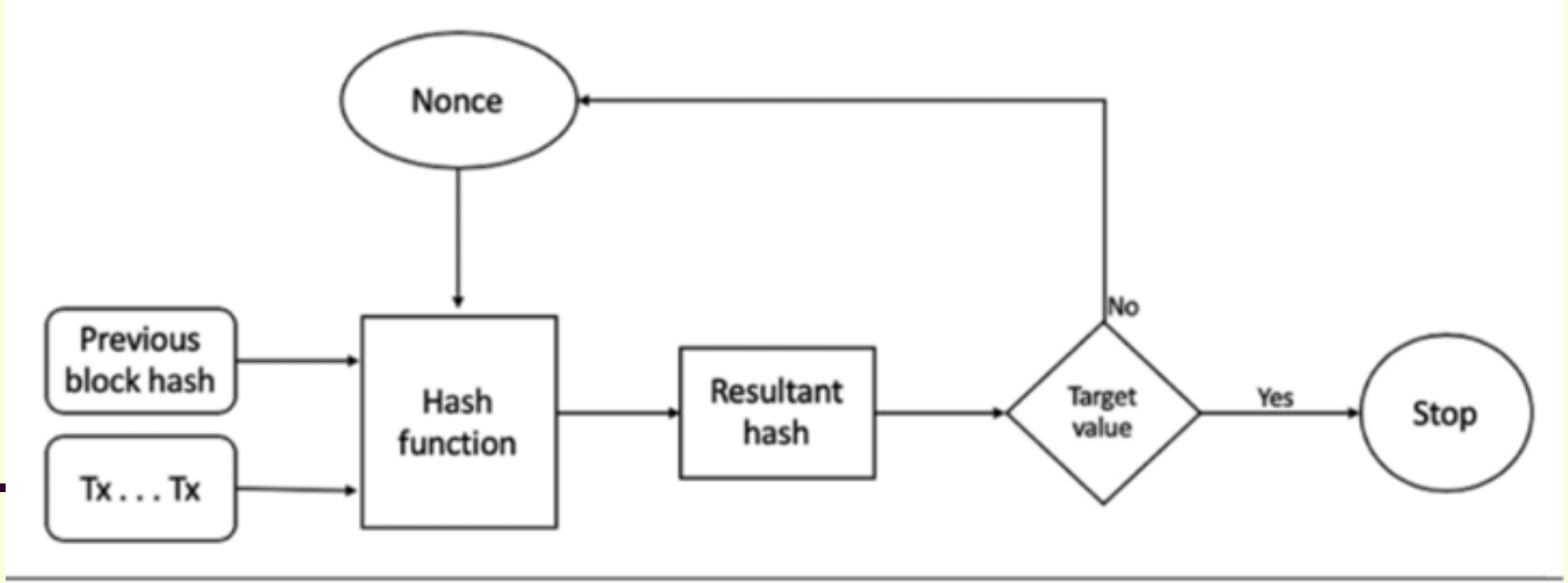


Figure 5.10: PoW diagram

Nakamoto consensus or PoW

- The key idea behind PoW as a solution to the Byzantine generals problem is that all honest generals (miners in the Bitcoin world) achieve agreement on the same state (decision value).
- As long as honest participants control the majority of the network, PoW solves the Byzantine generals problem. Note that this is a probabilistic solution and not deterministic.

Nakamoto versus traditional consensus

- We can map the properties of PoW consensus to traditional Byzantine consensus. This mapping is useful to understand what properties of traditional consensus can be applied to the Nakamoto world and vice versa.
- In traditional consensus algorithms, we have **agreement**, **validity**, and **liveness** properties, which can be mapped to Nakamoto-specific properties of the **common prefix**, **chain quality**, and **chain growth** properties respectively.
- The **common prefix** property means that the blockchain hosted by honest nodes will share the same large common prefix. If that is not the case, then the **agreement** property of the protocol cannot be guaranteed, meaning that the processors will not be able to decide and agree on the same value.

Nakamoto versus traditional consensus

- The **chain quality** property means that the blockchain contains a certain required level of correct blocks created by honest nodes (miners). If chain quality is compromised, then the **validity** property of the protocol cannot be guaranteed. This means that there is a possibility that a value will be decided that is not proposed by a correct process, resulting in safety violation.
- The **chain growth** property simply means that new correct blocks are continuously added to the blockchain. If chain growth is impacted, then the **liveness** property of the protocol cannot be guaranteed. This means that the system can deadlock or fail to decide on a value.

Variants of PoW

- Two main variants of PoW algorithms, based on the type of hardware used for processing
  - CPU-bound PoW - processing required to find the solution to the cryptographic hash puzzle is directly proportional to the calculation speed of the CPU or hardware such as ASICs
  - Memory-bound PoW - performance is bound by the access speed of the memory or the size of the memory

Proof of stake (PoS)

- Energy-efficient alternative to the PoW
- First used in Peercoin, and now, prominent cryptocurrency blockchains such as EOS, NxT, Steem, and Tezos are using PoS algorithms
- Stake represents the number of coins (money) in the consensus protocol staked by a blockchain participant.
- Key idea is that if someone has a stake in the system, then they will not try to sabotage the system
- The chance of proposing the next block is directly proportional to the value staked by the participant
- Variations of PoS - chain-based PoS, committee-based PoS, BFT-based PoS, delegated PoS, leased PoS, and master node PoS

Proof of stake (PoS)

- PoS miner is called either a validator, minter, or stakeholder
- Right to win the next proposer role is usually assigned randomly.
- Proposers are rewarded either with transaction fees or block rewards.
- Control over the majority of the network in the form of the control of a large portion of the stake is required to attack and control the network.

Proof of stake (PoS)

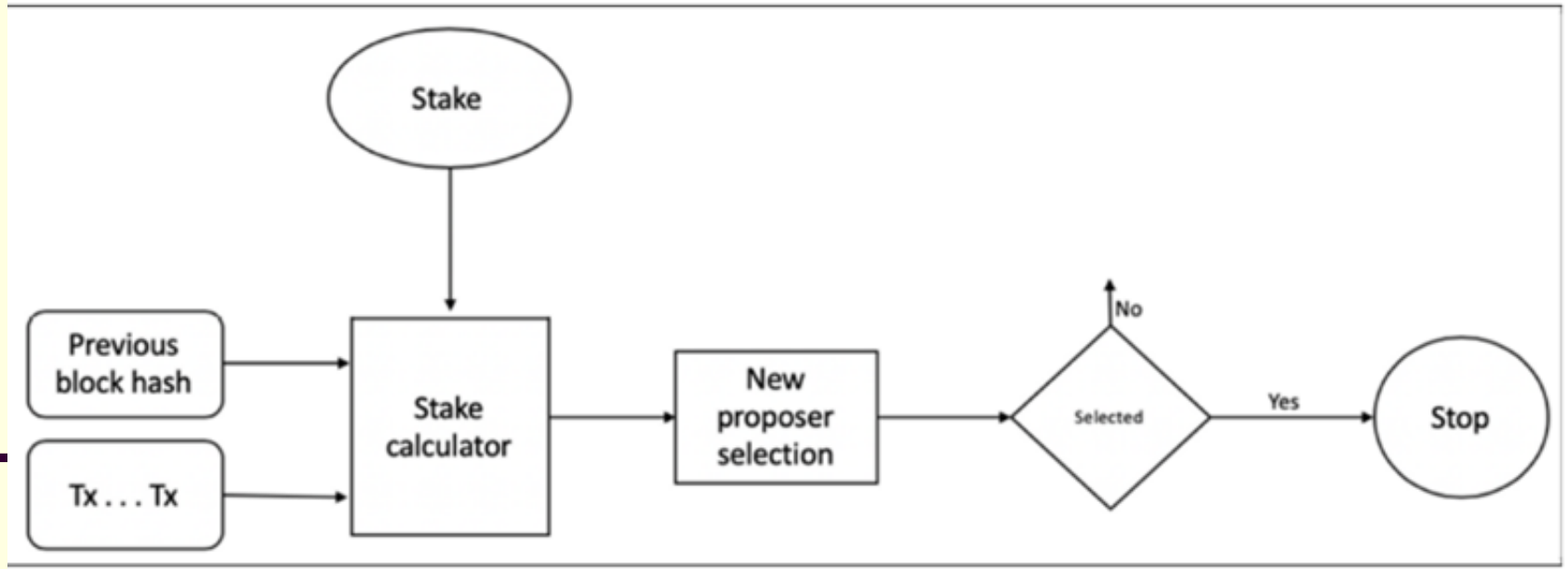


Figure 5.11: PoS diagram

Proof of stake (PoS) types

1	Chain-based PoS	• very similar to PoW • only change is the block generation method • Transactions are picked up from the memory pool and a candidate block is created • Hash (B ‖ clock time) < target x stake value • hashing puzzle in PoS is solved at regular intervals based on the clock tick
2	Committee-based PoS	• group of stakeholders is chosen randomly, using a verifiable random function (VRF) • VRF produces a random set of stakeholders based on their stake and the current state of the blockchain • The chosen group of stakeholders becomes responsible for proposing blocks in sequential order

Proof of stake (PoS) types

3	Delegated PoS	● Instead of using a random function to derive the group of stakeholders, the group is chosen by stake delegation ● group selected is a fixed number of minters that create blocks in a round-robin fashion ● Delegates are chosen via voting by network users ● Votes are proportional to the amount of the stake that participants have on the network ● used in Lisk, Cosmos, and EOS

Bitcoin definition

- Defined in various ways; it's a protocol, a digital currency, and a platform
- It is a combination of a peer-to-peer network, protocols, and software that facilitates the creation and usage of the digital currency
- Nodes in this peer-to-peer network talk to each other using the Bitcoin protocol

Bitcoin definition

- The double-spending problem arises when, a user sends coins to two different users at the same time and they are verified independently as valid transactions
- Resolved in Bitcoin by using a distributed ledger (the blockchain) where every transaction is recorded permanently, and by implementing a transaction validation and confirmation mechanism

Bitcoin Payment

Payment request is initiated through mail or SMS with address

Figure 6.2: Bitcoin payment request (using the Blockchain wallet)

Bitcoin Payment

2. The sender either enters the receiver's address or scans the QR code that has the Bitcoin address, amount, and an optional description encoded in it. The wallet application recognizes this QR code and decodes it into something like:

```
"Please send <amount> BTC to address <receiver's Bitcoin ad
```

3. With actual values, this will look like the following:

```
"Please send 0.00033324 BTC to address 1JzouJCVmMQBmTcd8K4Y
```

4. This is also shown in the following screenshot:

Figure 6.3: Bitcoin payment QR code

Bitcoin Payment

- fee is calculated based on size of the transaction and a fee rate, which is a value that depends on the volume of the transactions in the network at that time
- This is measured in Satoshis per byte Bitcoin network fees ensure that your transaction will be included by miners in the block

Figure 6.4: Sending BTC using the Blockchain wallet

Bitcoin Payment

- Transaction has been constructed, signed, and sent out to the Bitcoin network.
- This transaction will be picked up by miners to be verified and included in the block confirmation is pending for this transaction.
- These confirmations will start to appear as soon as the transaction is verified, included in the block, and mined the appropriate fee will be deducted from the original value to be transferred and will be paid to the miner who has included it in the block for mining.

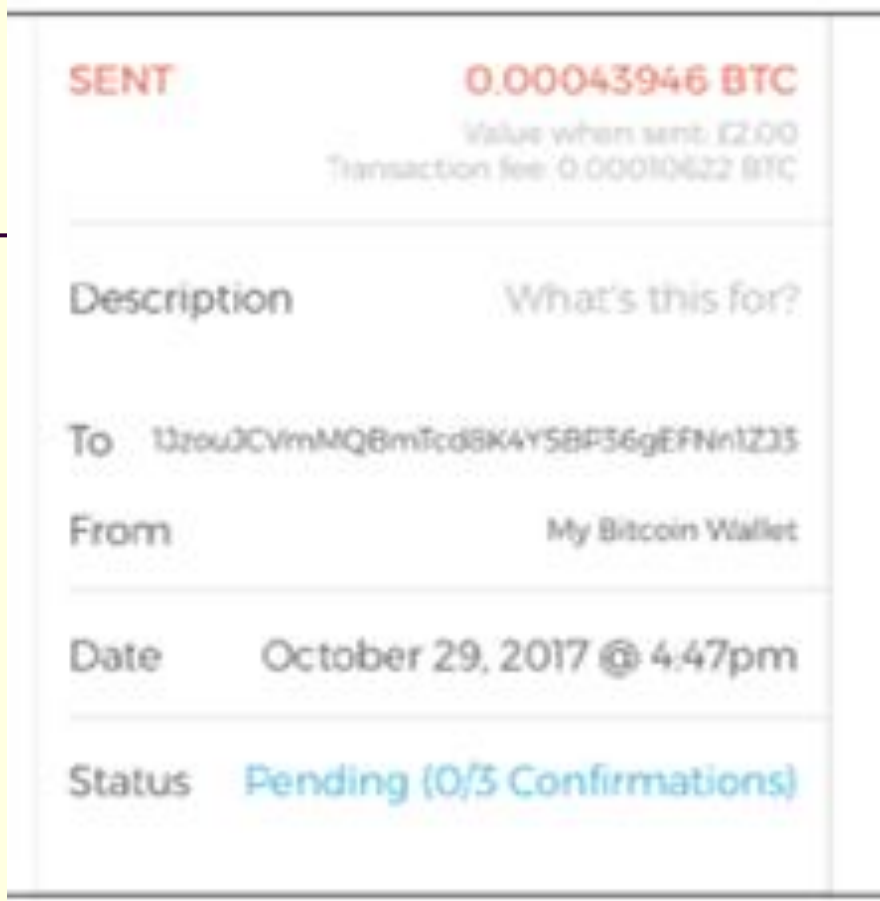


Figure 6.5: Transaction sent

Bitcoin Payment

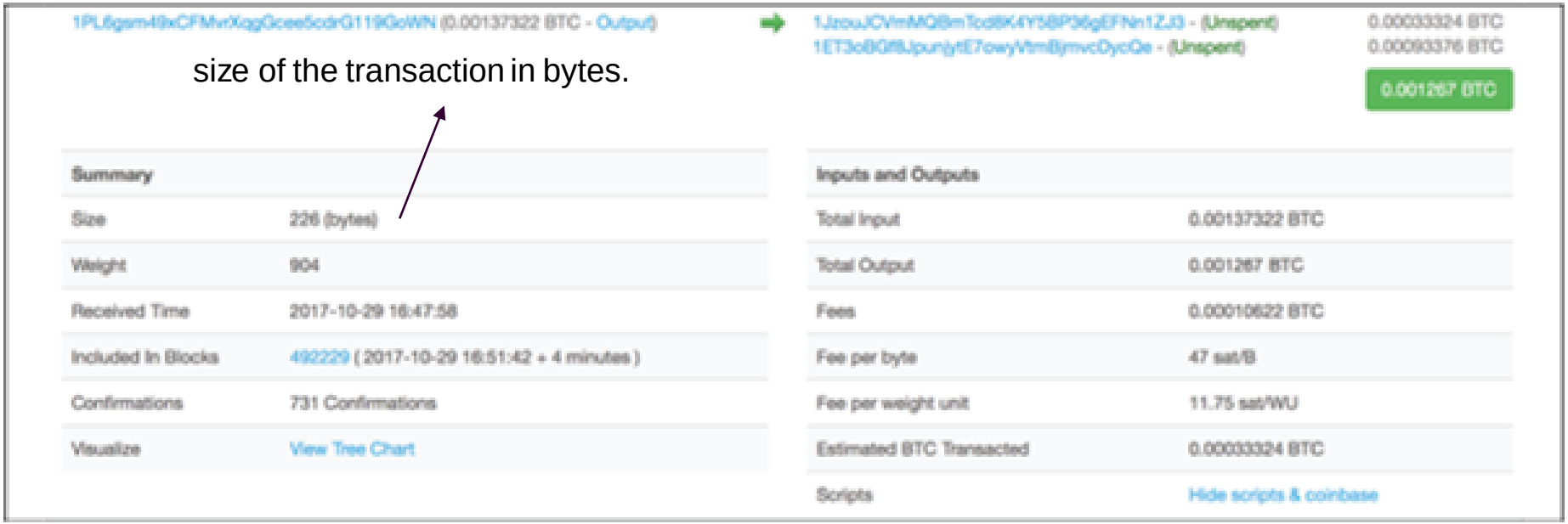


Figure 6.7: Snapshot of the transaction taken from blockchain.info

Bitcoin Payment

In summary, a payment transaction in the Bitcoin network can be divided into the following steps:

1. The transaction starts with a sender signing the transaction with their private key.
2. The transaction is serialized so that it can be transmitted over the network.
3. The transaction is broadcast to the network.
4. Miners listening for transactions pick up the transaction.
5. The transaction is verified for its legitimacy by the miners.
6. The transaction is added to the candidate/proposed block for mining.
7. Once mined, the result is broadcast to all nodes on the Bitcoin network.
8. Usually, at this point, users wait for up to six confirmations to be received before a transaction is considered final; however, a transaction can be considered final at the previous step. Confirmations serve as an additional mechanism to ensure that there is probabilistically a very low chance for a transaction to be reverted, but otherwise, once a mined block is finalized and announced, the transactions within that block are final at that point.

Bitcoin Payment

DENOMINATION	ABBREVIATION	FAMILIAR NAME	VALUE IN BTC
Satoshi	SAT	Satoshi	0.00000001 BTC
Microbit	µBTC (uBTC)	Microbitcoin or Bit	0.000001 BTC
Millibit	mBTC	Millibitcoin	0.001 BTC
Centibit	cBTC	Centibitcoin	0.01 BTC
Decibit	dBTC	Decibitcoin	0.1 BTC
Bitcoin	BTC	Bitcoin	1 BTC
DecaBit	daBTC	Decabitcoin	10 BTC
Hectobit	hBTC	Hectobitcoin	100 BTC
Kilobit	kBTC	Kilobitcoin	1000 BTC
Megabit	MBTC	Megabitcoin	1000000 BTC

Figure 6.8: Bitcoin denominations

Bitcoin-Cryptographic keys

- On the Bitcoin network, possession of Bitcoins and the transfer of value via transactions are reliant upon private keys, public keys, and addresses.
- **Elliptic Curve Cryptography** (**ECC**) is used to generate public and private key pairs in the Bitcoin network.

Private keys in Bitcoin

- **Private keys** are required to be kept safe and ormally reside only on the owner's side. Private keys are used to digitally sign the transactions, proving ownership of the bitcoins.
- Private keys are fundamentally 256-bit numbers randomly chosen in the range specified by the ***SECP256K1 ECDSA*** curve recommendation. Any randomly chosen 256-bit number from **0x1 to 0xFFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFE BAAE DCE6 AF48 A03B BFD2 5E8C D036 4140** is a valid private key.
- Private keys are usually encoded using **Wallet Import Format** (**WIF**) in order to make them easier to copy and use. It is a way to represent the full-size private key in a different format.
- WIF can be converted into a private key and vice versa.

The steps are described here.

For example, the consider the following private key:

```
A3ED7EC8A03667180D01FB4251A546C2B9F2FE33507C68B7D9D4E1FA5714195201
```

When converted into WIF format, it looks as shown here:

```
L2iN7umV7kbr6LuCmgM27rBnptGbDVc8g4ZBm6EbgTPQXnj1RCZP
```

- **Mini private key format** is sometimes used to create the private key with a maximum of 30 characters in order to allow storage where physical space is limited;
- for example, etching on physical coins or encoding in damage-resistant QR codes.
- The QR code is more damage resistant because more dots can be used for error correction and less for encoding the private key.

Bitcoin-Private key

- A private key encoded using mini private key format is also sometimes called a **minikey**.
- The first character of the mini private key is always the **uppercase letter S**. A mini private key can be converted into a normal-sized private key, but an existing normal-sized private key cannot be converted into a mini private key.
- This format was used in **Casascius** physical bitcoins.
- The Bitcoin core client also allows the encryption of the wallet that contains the private keys.

Public keys in Bitcoin

- **Public keys** exist on the blockchain and all network participants can see them.
- Public keys are derived from private keys due to their special mathematical relationship with those private keys.
- Once a transaction signed with the private key is broadcast on the Bitcoin network, public keys are used by the nodes to verify that the transaction has indeed been signed with the corresponding private key.
- This process of verification proves the ownership of the bitcoin.
- Bitcoin uses ECC based on the SECP256K1 standard.
- More specifically, it makes use of an **Elliptic Curve Digital Signature Algorithm** (**ECDSA**) to ensure that funds remain secure and can only be spent by the legitimate owner.

Public keys in Bitcoin

- Public keys can be represented in uncompressed or compressed format, and are fundamentally *x* and *y* coordinates on an elliptic curve.
- In uncompressed format, public keys are presented with a prefix of **0x4** in hexadecimal format.
- *x* and *y* coordinates are both 32 bytes in length.
- In total, a compressed public key is 33 bytes long, compared to the 65-byte uncompressed format.
- The compressed version of public keys include only the *x* part, since the *y* part can be derived from it.
- The reason why the compressed version of public keys works is that if the ECC graph is visualized, it reveals that the *y* coordinate can be either below the *x* axis or above the *x* axis, and as the curve is symmetric, only the location in the prime field is required to be stored.

Public keys in Bitcoin

- If *y* is even then its value lies above the *x* axis, and if it is odd then it is below the *x* axis. This means that instead of storing both *x* and *y* as the public key, only *x* needs to be stored with the information about whether *y* is even or odd.
- Initially, the Bitcoin client used uncompressed keys, but starting from Bitcoin Core client 0.6, compressed keys are used as standard. This resulted in an almost 50% reduction of space used to store public keys in the blockchain.

Keys are identified by various prefixes, described as follows:

- Uncompressed public keys use 0x04 as the prefix. Uncompressed public keys are 65 bytes long. They are encoded as 256-bit unsigned big-endian integers (32 bytes), which are concatenated together and finally prefixed with a byte 0x04. This means 1 byte for the 0x04 prefix, 32 bytes for the *x* integer, and 32 bytes for *y* integer, which makes it 65 bytes in total.
- Compressed public keys start with 0x03 if the *y* 32-byte (256-bit) part of the public key is odd. It is 33 bytes in length as 1 byte is used by the 0x03 prefix (depicting an odd *y*) and 32 bytes for storing the *x* coordinate.
- Compressed public keys start with 0x02 if the *y* 32-byte (256-bit) part of the public key is even. It is 33 bytes in length as 1 byte is used by the 0x02 prefix (depicting an even *y*) and 32 bytes for storing the *x* coordinate.

Addresses in Bitcoin

A Bitcoin address is created by taking the corresponding public key of a private key and hashing it twice, first with the SHA256 algorithm and then with RIPEMD160.

The resultant 160-bit hash is then prefixed with a version number and finally encoded with a Base58Check encoding scheme.

The Bitcoin addresses are 26-35 characters long and begin with digits 1 or 3.

Addresses in Bitcoin

A typical Bitcoin address looks like the string shown here:

```
1ANAguGG8bikEv2fYsTBnRUmx7QUcK58wt
```

Addresses are also commonly encoded in a QR code for easy distribution. The QR code of the preceding Bitcoin address is shown in the following image:

Figure 6.10: QR code of the Bitcoin address 1ANAguGG8bikEv2fYsTBnRUmx7QUcK58wt

Addresses in Bitcoin

- Currently, there are two types of addresses, the commonly used P2PKH and another P2SH type, starting with numbers 1 and 3, respectively.
- In the early days, Bitcoin used direct Pay-to- Pubkey, which is now superseded by P2PKH..
- However, direct Pay-to-Pubkey is still used in Bitcoin for coinbase addresses.
- Addresses should not be used more than once; otherwise, privacy and security issues can arise.
- Avoiding address reuse circumvents anonymity issues to an extent, but Bitcoin has some other security issues as well, such as transaction malleability, Sybil attacks, race attacks, and selfish mining, all of which require different approaches to resolve.

Address Generation in Bitcoin

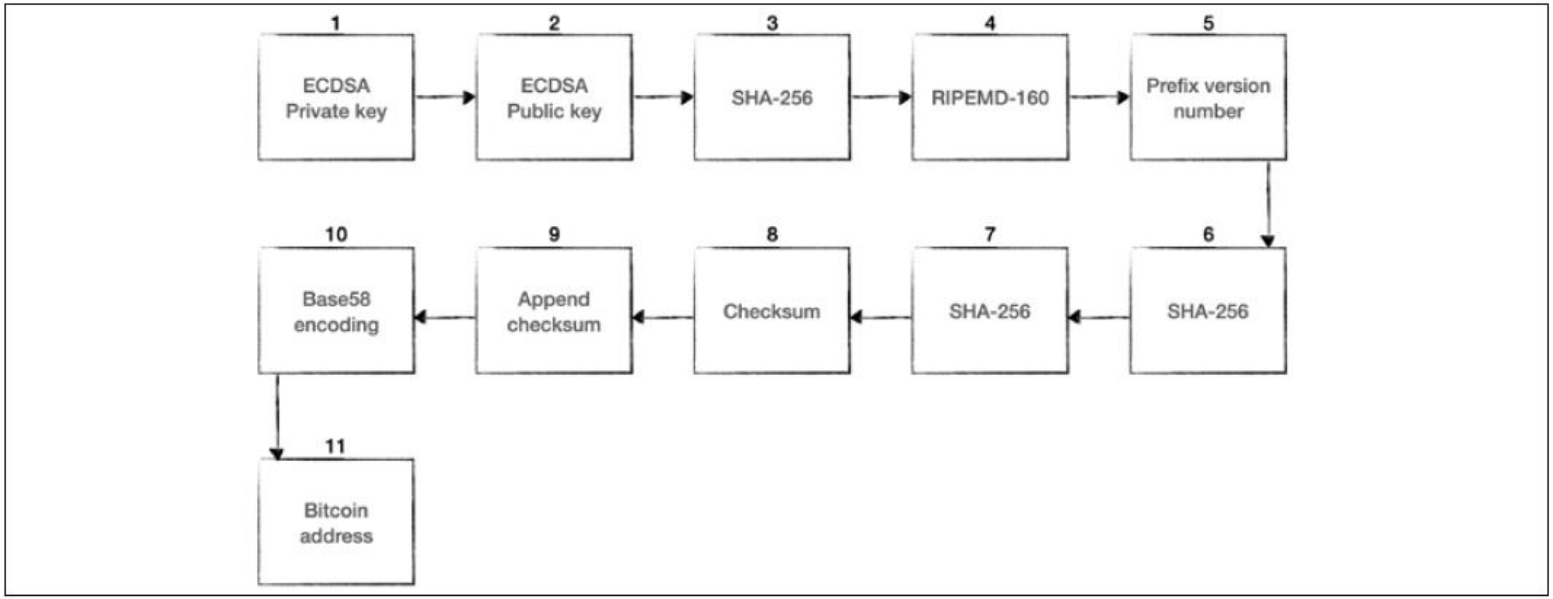


Figure 6.12: Address generation in Bitcoin

Address Generation in Bitcoin

In the preceding diagram, there are several steps that we will now explain:

1. In the first step, we have a randomly generated ECDSA private key.
2. The public key is derived from the ECDSA private key.
3. The public key is hashed using the SHA-256 cryptographic hash function.
4. The hash generated in *step 3* is hashed using the RIPEMD-160 hash function.
5. The version number is prefixed to the RIPEMD-160 hash generated in *step 4*.
6. The result produced in *step 5* is hashed using the SHA-256 cryptographic hash function.
7. SHA-256 is applied again.
8. The first 4 bytes of the result produced from *step 7* is the address `checksum`.
9. This `checksum` is appended to the RIPEMD-160 hash generated in *step 4*.
10. The resultant byte string is encoded into a Base58-encoded string by applying the Base58 encoding function.
11. Finally, the result is a typical Bitcoin address.

Advanced types of addresses available in Bitcoin

Vanity addresses

- As Bitcoin addresses are based on Base58 encoding, it is possible to generate addresses that contain human-readable messages.
- Vanity addresses are generated using a purely brute-force method

Multi-signature addresses

- As the name implies, these addresses require multiple private keys.
- In practical terms, this means that in order to release the coins, a certain set number of signatures is required. This is also known as *M of N multisig*.
- Here, *M* represents the threshold or minimum number of signatures required from *N* number of keys to release the Bitcoins.

Tranactions

- Transactions are at the core of the Bitcoin ecosystem.
- Transactions can be as simple as just sending some bitcoins to a Bitcoin address, or can be quite complex, depending on the requirements.
- Each transaction is composed of at least one input and output.
- Inputs can be thought of as coins being spent that have been created in a previous transaction, and outputs as coins being created.
- If a transaction is minting new coins, then there is no input, and therefore no signature is needed.
- If a transaction should send coins to some other user (a Bitcoin address), then it needs to be signed by the sender with their private key.
- In this case, a reference is also required to the previous transaction to show the origin of the coins. Coins are unspent transaction outputs represented in Satoshis.
- Transactions are not encrypted and are publicly visible on the blockchain.
- Blocks are made up of transactions, and these can be viewed using any online blockchain explorer.

The transaction lifecycle

1. A user/sender sends a transaction using wallet software or some other interface.

2. The wallet software signs the transaction using the sender's private key.

3. The transaction is broadcasted to the Bitcoin network using a flooding algorithm.

4. Mining nodes (miners) who are listening for the transactions verify and include this transaction in the next block to be mined. Just before the transactions are placed in the block, they are placed in a special memory buffer called the transaction pool. The purpose of the transaction pool is explained in the next section.

The transaction lifecycle

5. Next, the mining starts, which is the process through which the blockchain is secured and new coins are generated as a reward for the miners who spend appropriate computational resources. Once a miner solves the PoW problem, it broadcasts the newly mined block to the network. The nodes verify the block and propagate the block further, and confirmations start to generate.

6. Finally, the confirmations start to appear in the receiver's wallet and after approximately three confirmations, the transaction is considered finalized and confirmed. However, three to six is just a recommended number; the transaction can be considered final even after the first confirmation. The key idea behind waiting for six confirmations is that the probability of double spending is virtually eliminated after six confirmations.

Transaction pool

- When a transaction is created by a user and sent to the network, it ends up in a special area on each Bitcoin software client.
- This special area is called the transaction pool or memory pool.
- Also known as memory pools, these pools are basically created in local memory (computer RAM) by nodes (Bitcoin clients) in order to maintain a temporary list of transactions that have not yet been added to a block.
- Miners pick up transactions from these memory pools to create candidate blocks. Miners select transactions from the pool after they pass the verification and validity checks.
- The selection of which transactions to choose is based on the fee and their place in the order of transactions in the pool.
- Miners prefer to pick up transactions with higher fees.
- To send transactions on the Bitcoin network, the sender needs to pay a fee to the miners. This fee is an incentive mechanism for the miners.

Transaction fees

- Transaction fees are charged by the miners. The fee charged is dependent upon the size and weight of the transaction. Transaction fees are calculated by subtracting the sum of the inputs from the sum of the outputs.
- A simple formula can be u

$$fee = sum(inputs) - sum(outputs)$$

- The fees are used as an incentive for miners to encourage them to include users' transactions in the block the miners are creating.
- There are different rules based on which fee is calculated for various types of actions, such as sending transactions, inclusion in blocks, and relaying by nodes.

The transaction data structure

- A transaction on the Bitcoin network is represented by a data structure that consists of several fields.
- A transaction at a high level contains metadata, inputs, and outputs. Transactions are combined to create a block's body.

Field	Size	Description
Version number	4 bytes	Specifies the rules to be used by the miners and nodes for transaction processing. There are two versions of transactions, that is, 1 and 2.
Input counter	1–9 bytes	The number (a positive integer) of inputs included in the transaction.

The transaction data structure

List of inputs	Variable	Each input is composed of several fields. These include: • The previous transaction hash • The index of the previous transaction • Transaction script length • Transaction script • Sequence number The first transaction in a block is also called a coinbase transaction. It specifies one or more transaction inputs. In summary, this field describes which Bitcoins are going to be spent.
Output counter	1–9 bytes	A positive integer representing the number of outputs.
List of outputs	Variable	Outputs included in the transaction. This field depicts the target recipient(s) of the Bitcoins.

The transaction data structure

Lock time	4 bytes	This field defines the earliest time when a transaction becomes valid. It is either a Unix timestamp or the block height.

Coinbase transactions

- A coinbase transaction or generation transaction is always created by a miner and is the first transaction in a block. It is used to create new coins.
- It includes a special field, also called the coinbase, which acts as an input to the coinbase transaction. This transaction also allows up to 100 bytes of arbitrary data storage.
- A coinbase transaction input has the same number of fields as a usual transaction input, but the structure contains the coinbase data size and fields instead of the unlocking script size and fields. Also, it does not have a reference pointer to the previous transaction

Coinbase transactions -Data structure

Field	Size	Description
Transaction hash	32 bytes	Set to all zeroes as no hash reference is used
Output index	4 bytes	Set to `0xFFFFFFFF`
Coinbase data length	1-9 bytes	2-100 bytes
Data	Variable	Any data
Sequence number	4 bytes	Set to `0xFFFFFFFF`

Transaction validation

This verification process is performed by Bitcoin nodes. There are three main things that nodes check when verifying a transaction:

1. **That transaction inputs are previously unspent. This validation step prevents double spending by verifying that the transaction inputs have not already been spent by someone else.**

2. That the sum of the transaction outputs is not more than the total sum of the transaction inputs. However, both input and output sums can be the same, or the sum of the input (total value) could be more than the total value of the outputs. This check ensures that no new bitcoins are created out of thin air.

3. That the digital signatures are valid, which ensures that the script is valid.

Blockchain

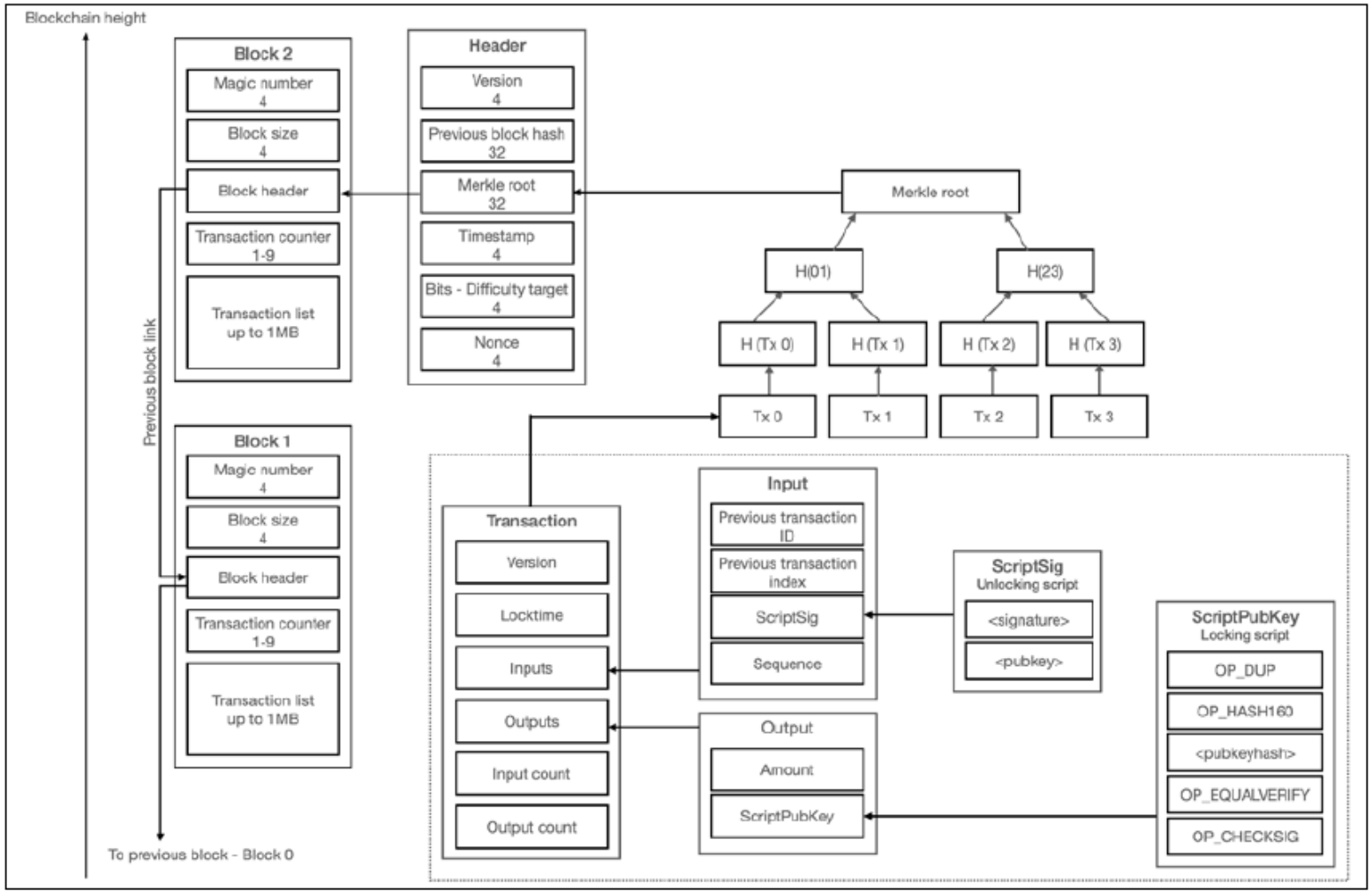
- A blockchain is a distributed ledger of transactions.
- Specifically, from Bitcoin's perspective, the blockchain can be defined as a public, distributed ledger holding a timestamped, ordered, and immutable record of all transactions on the Bitcoin network. Transactions are picked up by miners and bundled into blocks for mining.
- Each block is identified by a hash and is linked to its previous block by referencing the previous block's hash in its header.

The data structure of a Bitcoin block

Field	Size	Description
Block size	4 bytes	The size of the block.
Block header	80 bytes	This includes fields from the block header described in the next section.
Transaction counter	Variable	The field contains the total number of transactions in the block, including the coinbase transaction. Size ranges from 1-9 bytes.
Transactions	Variable	All transactions in the block.

The block header-Data structure

Field	Size	Description
Version	4 bytes	The block version number that dictates the block validation rules to follow.
Previous block's header hash	32 bytes	This is a double SHA256 hash of the previous block's header.
Merkle root hash	32 bytes	This is a double SHA256 hash of the Merkle tree of all transactions included in the block.
Timestamp	4 bytes	This field contains the approximate creation time of the block in Unix-epoch time format. More precisely, this is the time when the miner started hashing the header (the time from the miner's location).
Difficulty target	4 bytes	This is the current difficulty target of the network/block.
Nonce	4 bytes	This is an arbitrary number that miners change repeatedly to produce a hash that is lower than the difficulty target.


Blockchain height
Block 2
Magic number
4
Block size
4
Block header
Transaction counter
1-9
Transaction list
up to 1MB
Header
Version
4
Previous block hash
32
Merkle root
32
Timestamp
4
Bits - Difficulty target
4
Nonce
4
Merkle root
H(01)
H(23)
H (Tx 0)
H (Tx 1)
H (Tx 2)
H (Tx 3)
Tx 0
Tx 1
Tx 2
Tx 3
Previous block link
Block 1
Magic number
4
Block size
4
Block header
Transaction counter
1-9
Transaction list
up to 1MB
Transaction
Version
Locktime
Inputs
Outputs
Input count
Output count
Input
Previous transaction
ID
Previous transaction
index
ScriptSig
Sequence
ScriptSig
Unlocking script
<signature>
<pubkey>
Output
Amount
ScriptPubKey
ScriptPubKey
Locking script
OP_DUP
OP_HASH160
<pubkeyhash>
OP_EQUALVERIFY
OP_CHECKSIG
To previous block - Block 0

The genesis block

This is the first block in the Bitcoin blockchain. The genesis block was hardcoded in the Bitcoin core software.

Mining

- Mining is a process by which new blocks are added to the blockchain. Blocks contain transactions that are validated via the mining process by mining nodes on the Bitcoin network.
- Blocks, once mined and verified, are added to the blockchain, which keeps the blockchain growing.
- This process is resource-intensive due to the requirements of PoW, where miners compete to find a number less than the difficulty target of the network.
- This difficulty in finding the correct value (also called sometimes the **mathematical puzzle**) is there to ensure that miners have spent the required resources before a new proposed block can be accepted.
- The miners mint new coins by solving the PoW problem, also known as the partial hash inversion problem.
- This process consumes a high amount of resources, including computing power and electricity.
- This process also secures the system against fraud and double-spending attacks while adding more virtual currency to the Bitcoin ecosystem

Tasks of the miners

Once a node connects to the Bitcoin network, there are several tasks that a Bitcoin miner performs:

1. **Synching up with the network**: Once a new node joins the Bitcoin network, it downloads the blockchain by requesting historical blocks from other nodes. This is mentioned here in the context of the Bitcoin miner; however, this not necessarily a task that only concerns miners.
2. **Transaction validation**: Transactions broadcast on the network are validated by full nodes by verifying and validating signatures and outputs.
3. **Block validation**: Miners and full nodes can start validating blocks received by them by evaluating them against certain rules. This includes the verification of each transaction in the block along with verification of the nonce value.
4. **Create a new block**: Miners propose a new block by combining transactions broadcast on the network after validating them.

Tasks of the miners

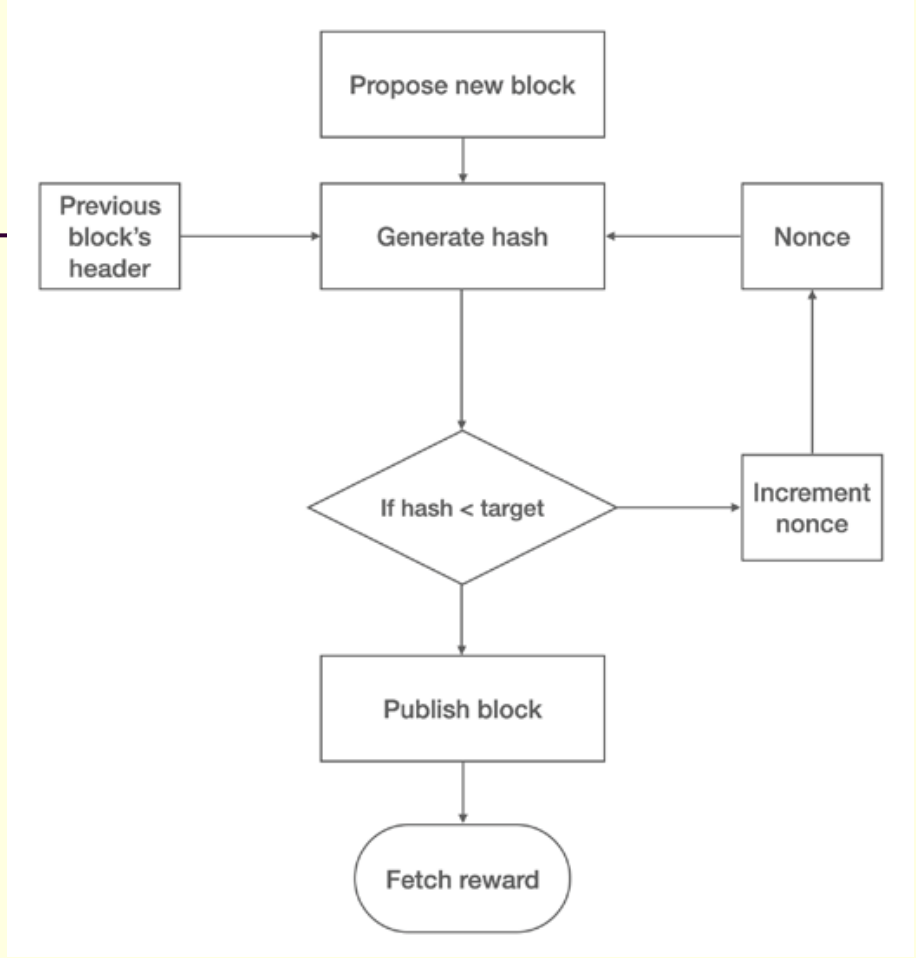
5. **Perform PoW**: This task is the core of the mining process and this is where miners find a valid block by solving a computational puzzle. The block header contains a 32-bit nonce field and miners are required to repeatedly vary the nonce until the resultant hash is less than a predetermined target.
6. **Fetch reward**: Once a node solves the hash puzzle (PoW), it immediately broadcasts the results, and other nodes verify it and accept the block. There is a slight chance that the newly minted block will not be accepted by other miners on the network due to a clash with another block found at roughly the same time, but once accepted, the miner is rewarded with 12.5 bitcoins and any associated transaction fees.

The mining algorithm

The mining algorithm consists of the following steps:

1. The previous block's header is retrieved from the Bitcoin network.
2. Assemble a set of transactions broadcast on the network into a block to be proposed.
3. Compute the double hash of the previous block's header, combined with a nonce and the newly proposed block, using the SHA256 algorithm.
4. Check if the resulting hash is lower than the current difficulty level (the target), then PoW is solved. As a result of successful PoW, the discovered block is broadcasted to the network and miners fetch the reward.
5. If the resultant hash is not less than the current difficulty level (target), then repeat the process after incrementing the nonce.

Mining Process



The hash rate

- The hash rate basically represents the rate of hash calculation per second. In other words, this is the speed at which miners in the Bitcoin network are calculating hashes to find a block.
- In the early days of Bitcoin, it used to be quite small, as CPUs were used, which are relatively weak in mining terms.
- However, with dedicated mining pools and **Application Specific Integrated Circuits** (**ASICs**) now, this has gone up exponentially in the last few years. This has resulted in increased difficulty in the Bitcoin network.

Wallets

The wallet software is used to generate and store cryptographic keys.
It performs various useful functions, such as receiving and sending Bitcoin, backing up keys, and keeping track of the balance available.
Bitcoin client software usually offers both functionalities: Bitcoin client and wallet.
On disk, the Bitcoin Core client wallets are stored as a Berkeley DB file:

Types of wallets

- **Non-deterministic wallets**
- **Deterministic wallets**
- **Hierarchical deterministic wallets**
- **Brain wallets**
- **Paper wallets**
- **Hardware wallets**
- **Online wallets**
- **Mobile wallets**

Non-deterministic wallets	• These wallets contain randomly generated private keys and are also called Just a Bunch of Key wallets. • The Bitcoin Core client generates some keys when first started and also generates keys as and when required. • Managing a large number of keys is very difficult and an error-prone process that can lead to the theft and loss of coins. • Moreover, there is a need to create regular backups of the keys and protect them appropriately, for example, by encrypting them in order to prevent theft or loss.
Deterministic wallets	• In this type of wallet, keys are derived from a seed value via hash functions. • This seed number is generated randomly and is commonly represented by human-readable mnemonic code words. • Mnemonic code words are defined in BIP39, a Bitcoin improvement proposal for Mnemonic code for generating deterministic keys.

Hierarchical deterministic wallets	• Defined in BIP32 and BIP44, HD wallets store keys in a tree structure derived from a seed. • The seed generates the parent key (master key), which is used to generate child keys and, subsequently, grandchild keys. • Key generation in HD wallets does not generate keys directly; instead, it produces some information (private key generation information) that can be used to generate a sequence of private keys. • The complete hierarchy of private keys in an HD wallet is easily recoverable if the master private key is known. • It is because of this property that HD wallets are very easy to maintain and are highly portable.
Brain wallets	• The master private key can also be derived from the hashes of passwords that are memorized. • The key idea is that this passphrase is used to derive the private key and if used in HD wallets, this can result in a full HD wallet that is derived from a single memorized password. • This is known as a brain wallet. • This method is prone to password guessing and brute-force attacks, but techniques such as key stretching can be used to slow down the progress made by the attacker.

Paper wallets	• As the name implies, this is a paper-based wallet with the required key material printed on it. • It requires physical security to be stored.
Hardware wallets	• Another method is to use a tamper-resistant device to store keys. • This tamper-resistant device can be custom-built. • With the advent of NFC-enabled phones, this can also be a secure element (SE) in NFC phones. • Trezor and Ledger wallets (various types) are the most commonly used Bitcoin hardware wallets:

Online wallets	Online wallets, as the name implies, are stored entirely online and are provided as a service usually via the cloud. They provide a web interface to the users to manage their wallets and perform various functions, such as making and receiving payments. They are easy to use but imply that the user trusts the online wallet service provider. An example of an online wallet is GreenAddress
Mobile wallets	Mobile wallets, as the name suggests, are installed on mobile devices. They can provide us with various methods to make payments, most notably the ability to use smartphone cameras to scan QR codes quickly and make payments.