# 6
# Bitcoin Network and Payments

In this chapter, we will present the Bitcoin network, relevant network protocols, and wallets. We will explore different types of wallets that are available for bitcoin. Moreover, we will examine how the Bitcoin protocol works and the types of messages exchanged on the network between nodes, during various node and network operations. Also, we will explore various advanced and modern Bitcoin protocols that have been developed to address limitations in the original Bitcoin. Finally, we'll give you an introduction to bitcoin trading and investment.
We will start with the detailed introduction of the Bitcoin network.

## The Bitcoin network

The Bitcoin network is a peer-to-peer network where nodes exchange transactions and blocks. There are different types of nodes on the network. There are two main types of nodes, full nodes and SPV nodes. **Full nodes**, as the name implies, are implementations of Bitcoin core clients performing the wallet, miner, full blockchain storage, and network routing functions. However, it is not necessary to perform all these functions. **Simple Payment Verification** (**SPV**) nodes or lightweight clients perform only wallet and network routing functionality. The latest version of Bitcoin protocol is 70015 and was introduced with Bitcoin core client 0.13.2.

Some nodes prefer to be full blockchain nodes with complete blockchain as they are more secure and play a vital role in block propagation while some nodes perform network routing functions only but do not perform mining or store private keys (the wallet function). Another type is solo miner nodes that can perform mining, store full blockchain, and act as a Bitcoin network routing node.

There are a few nonstandard but heavily used nodes that are called **pool protocol servers**. These nodes make use of alternative protocols, such as the stratum protocol. These nodes are used in mining pools. Nodes that only compute hashes use the stratum protocol to submit their solutions to the mining pool. Some nodes perform only mining functions and are called mining nodes. It is possible to run an SPV client which runs a wallet and network routing function without a blockchain. SPV clients only download the headers of the blocks while syncing with the network and when required they can request transactions from full nodes. The verification of transactions is possible by using Merkle root in the block header with Merkle branch to prove that the transaction is present in a block in the blockchain.

Most protocols on the internet are line-based, which means that each line is delimited by a carriage return (`\r`) and newline (`\n`) character. Stratum is also a line-based protocol that makes use of plain TCP sockets and human-readable JSON-RPC to operate and communicate between nodes. Stratum is commonly used to connect to mining pools.

The Bitcoin network is identified by its different magic values. A list is shown as follows:

| Network | Magic value | Hex |
|---|---|---|
| main | 0xD9B4BEF9 | F9 BE B4 D9 |
| testnet3 | 0x0709110B | 0B 11 09 07 |

Bitcoin network magic values

Magic values are used to indicate the message origin network.

A full node performs four functions: wallet, miner, blockchain, and the network routing node.

Before we examine that how Bitcoin discovery protocol and block synchronization works, we need to understand that what are the different types of messages that Bitcoin protocol uses. The list of message types is provided here.

There are 27 types of protocol messages in total, but they're likely to increase over time as the protocol grows. The most commonly used protocol messages and their explanation are listed as follows:

- `version`: This is the first message that a node sends out to the network, advertising its version and block count. The remote node then replies with the same information and the connection is then established.

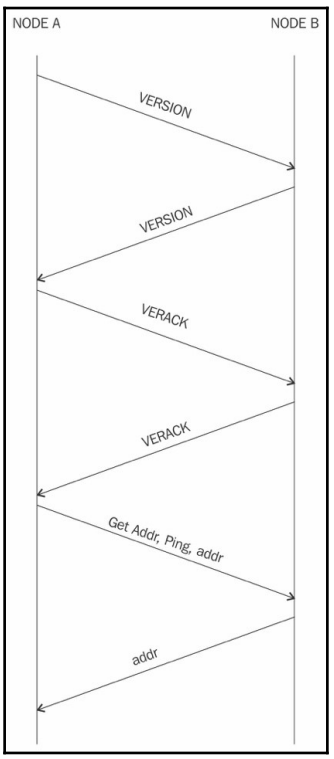
- `verack`: This is the response of the version message accepting the connection request.
- `inv`: This is used by nodes to advertise their knowledge of blocks and transactions.
- `getdata`: This is a response to `inv`, requesting a single block or transaction identified by its hash.
- `getblocks`: This returns an `inv` packet containing the list of all blocks starting after the last known hash or 500 blocks.
- `getheaders`: This is used to request block headers in a specified range.
- `tx`: This is used to send a transaction as a response to the `getdata` protocol message.
- `block`: This sends a block in response to the `getdata` protocol message.
- `headers`: This packet returns up to 2,000 block headers as a reply to the `getheaders` request.
- `getaddr`: This is sent as a request to get information about known peers.
- `addr`: This provides information about nodes on the network. It contains the number of addresses and address list in the form of IP address and port number.

When a Bitcoin core node starts up, first, it initiates the discovery of all peers. This is achieved by querying DNS seeds that are hardcoded into the Bitcoin core client and are maintained by Bitcoin community members. This lookup returns a number of DNS A records. The Bitcoin protocol works on TCP port `8333` by default for the main network and TCP `18333` for testnet. The following code shows an example of DNS seeds in `chainparams.cpp`:

```
// Pieter Wuille, only supports x1, x5, x9, and xd
vSeeds.emplace_back("seed.bitcoin.sipa.be");
// Matt Corallo, only supports x9
vSeeds.emplace_back("dnsseed.bluematt.me");
// Luke Dashjr
vSeeds.emplace_back("dnsseed.bitcoin.dashjr.org");
// Christian Decker, supports x1 - xf
vSeeds.emplace_back("seed.bitcoinstats.com");
// Jonas Schnelli, only supports x1, x5, x9, and xd
vSeeds.emplace_back("seed.bitcoin.jonasschnelli.ch");
// Peter Todd, only supports x1, x5, x9, and xd
vSeeds.emplace_back("seed.btc.petertodd.org");
```

First, the client sends a protocol message `version` that contains various fields, such as version, services, timestamp, network address, nonce, and some other fields. The remote node responds with its own `version` message followed by the `verack` message exchange between both nodes, indicating that the connection has been established.

After this, `getaddr` and `addr` messages are exchanged to find the peers that the client does not know. Meanwhile, either of the nodes can send a `ping` message to see whether the connection is still active. The `getaddr` and `addr` messages are the types defined in the Bitcoin protocol. This process is shown in the following protocol diagram:




Visualization of node discovery protocol

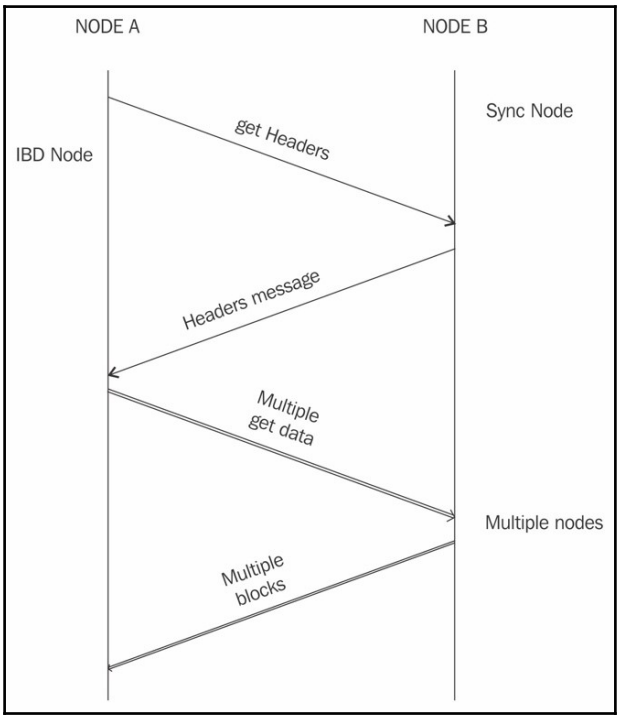
The preceding network protocol sequence diagram shows communication between two Bitcoin nodes during initial connectivity. **NODE A** is shown on the left side and **NODE B** on the right. First, **NODE A** starts the connection by sending the `version` message which contains version number and current time to the remote peer **NODE B**. **NODE B** then responds with its own `version` message containing the version number and current time. **NODE A** and **NODE B** then exchange a `verack` message indicating that the connection has been successfully established. After the connection is successful the peers can exchange `getaddr` and `addr` messages to discover other peers on the network.

Now the block download can begin. If the node already has all the blocks fully synchronized, then it listens for new blocks using the `inv` protocol message; otherwise, it first checks whether it has a response to `inv` messages and have inventories already. If yes, then it requests the blocks using the `getdata` protocol message; if not, then it requests inventories using the `getblocks` message. This method was used until version 0.9.3. This was a slower process known as **blocks-first approach** and was replaced with **headers-first approach** in 0.10.0.

Initial block download can use blocks-first or the headers-first method to synchronize blocks depending on the version of the Bitcoin core client. The blocks-first method is very slow and was discontinued since February 16, 2015 with the release of version 0.10.0.

Since version 0.10.0, the initial block download method named headers-first was introduced. This resulted in major performance improvement and the blockchain synchronization that used to take days to complete started taking only a few hours. The core idea is that the new node first asks peers for block headers and validates them. Once this is completed, blocks are requested in parallel from all available peers as the blueprint of the complete chain is already downloaded in the form of the block header chain.

In this method, when the client starts up, it checks whether the blockchain is fully synchronized already if the header chain is already synchronized; if not, which is the case the first time the client starts up, it requests headers from other peers using the `getheaders` message. If the blockchain is fully synchronized, it listens for new blocks via `inv` messages, and if it already has a fully synchronized header chain, then it requests blocks using the `getdata` protocol messages. The node also checks whether the header chain has more headers than blocks and then it requests blocks by issuing the `getdata` protocol message.




Bitcoin core client >= 0.10.0 header and block synchronization

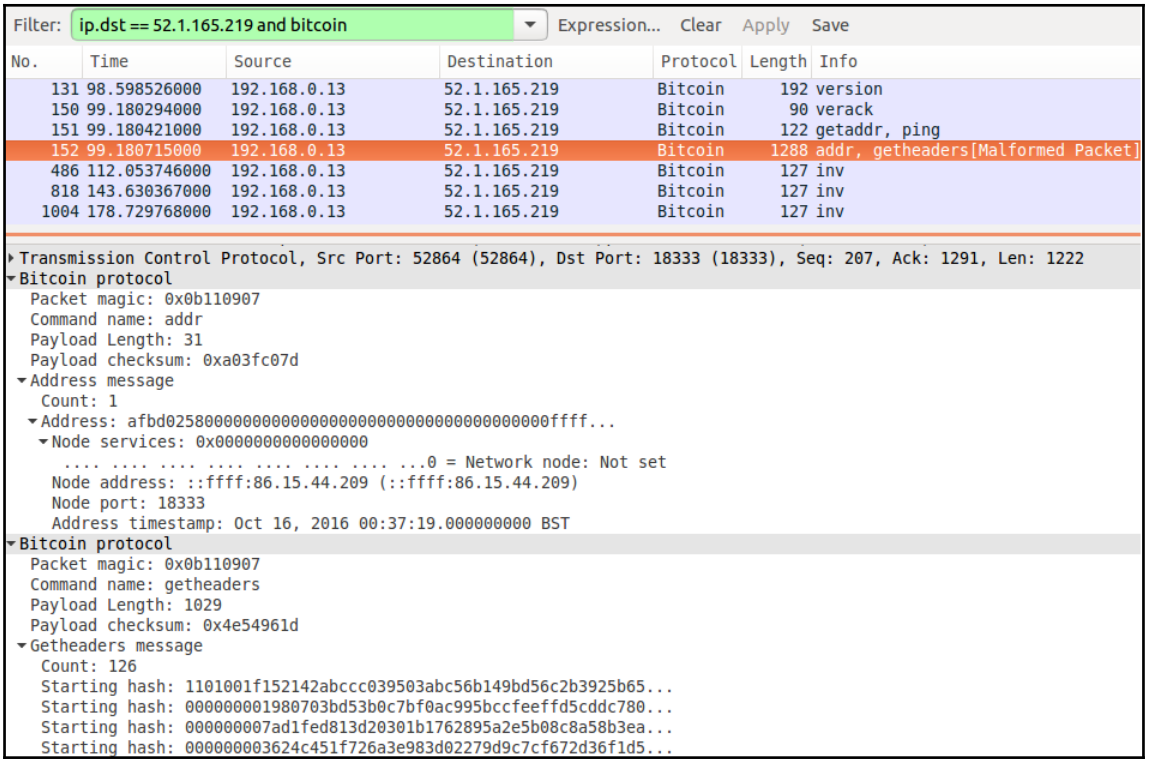
The preceding diagram shows the Bitcoin block synchronization process between two nodes on the Bitcoin network. **NODE A**, shown on the left side is called **Initial Block Download Node** (**IBD Node**) and **NODE B**, shown on the right is called **Sync Node**.

**IBD Node** means that this is the node that is requesting the blocks and **Sync Node** means the node from where the blocks are being requested. The process starts by **NODE A** first sending the `getheaders` message which is responded with the `headers` message from the **Sync Node**. The payload of the `getheaders` message is one or more header hashes. If it's a new node then there is only the first genesis block's header hash. The **Sync Node** replies with sending up to 2,000 block headers to the **IBD Node**. After this, the **IBD Node** starts to download more headers from the **Sync Node** and in parallel, downloads blocks from multiple nodes. In other words, **IBD Node** makes requests to multiple nodes and as a result multiple blocks are sent to **IBD Node** from **Sync Node** and other nodes. If the **Sync Node** does not have more headers than 2,000, when **IBD Node** makes a `getheaders` request then **IBD Node** sends `getheaders` message to other nodes. This process continues in parallel until the blockchain synchronization is complete.

The `getblockchaininfo` and `getpeerinfo` RPCs were updated with a new functionality to cater for this change. A **Remote Procedure Call** (**RPC**), `getchaintips`, is used to list all known branches of the blockchain. This also includes headers only blocks. The `getblockchaininfo` RPC is used to provide the information about the current state of the blockchain. The `getpeerinfo` RPC is used to list both the number of blocks and the headers that are in common between peers.

Wireshark can also be used to visualize message exchange between peers and can serve as an invaluable tool to learn about the Bitcoin protocol. A sample is shown here. This is a basic example showing the `version`, `verack`, `getaddr`, `ping`, `addr`, and `inv` messages.

In the details, valuable information such as the packet type, command name, and results of the protocol messages can be seen:




A sample block message in Wireshark

A protocol graph showing the flow of data between the two peers is shown in the preceding diagram. This can help you understand when a node starts up and what type of messages are used.

In the following example, the Bitcoin dissector is used to analyze the traffic and identify the Bitcoin protocol commands.

Exchange of messages such as `version`, `getaddr`, and `getdata` can be seen in the following example along with the appropriate comment describing the message name.

This exercise can be very useful in order to learn Bitcoin protocol and it is recommended that the experiments be carried out on the Bitcoin testnet (`https://en.bitcoin.it/wiki/Testnet`), where various messages and transactions can be sent over the network and then be analyzed by Wireshark.

Wireshark is a network analysis tool available at `https://www.wireshark.org`.

The analysis performed by Wireshark in the following screenshot shows the exchange of messages between two nodes. If you look closely, you'll notice that top three messages show the node discovery protocol that we have discussed before:

| Time | 192.168.0.13 (57868) → 136.243.139.96 (18333) | Comment |
|---|---|---|
| 97.734135000 | version | Bitcoin: version |
| 98.025045000 | verack | Bitcoin: verack |
| 98.025177000 | getaddr, pin.. | Bitcoin: getaddr, ping, addr |
| 98.025468000 | getheaders, .. | Bitcoin: getheaders, [unknown command], [unknown command], [unknown command], headers |
| 98.160419000 | [TCP Retran.. | Bitcoin: [TCP Retransmission] , getheaders, [unknown command], [unknown command], [unknown command] |
| 98.598399000 | getdata | Bitcoin: getdata |
| 144.343544000 | inv | Bitcoin: inv |
| 176.152240000 | getdata | Bitcoin: getdata |
| 179.493755000 | getdata | Bitcoin: getdata |
| 218.101646000 | ping | Bitcoin: ping |
| 218.192004000 | [unknown co.. | Bitcoin: [unknown command] |
| 218.444431000 | [TCP Retran.. | Bitcoin: [TCP Retransmission] , [unknown command] |
| 336.234936000 | getdata | Bitcoin: getdata |
| 337.843423000 | [unknown co.. | Bitcoin: [unknown command] |
| 338.143885000 | ping | Bitcoin: ping |
| 448.764093000 | getdata | Bitcoin: getdata |
| 457.894823000 | [unknown co.. | Bitcoin: [unknown command] |
| 458.195265000 | ping | Bitcoin: ping |
| 578.011774000 | [unknown co.. | Bitcoin: [unknown command] |
| 578.212044000 | ping | Bitcoin: ping |
| 585.587671000 | inv | Bitcoin: inv |
| 647.169633000 | inv | Bitcoin: inv |
| 671.962545000 | getdata | Bitcoin: getdata |
| 698.037067000 | [unknown co.. | Bitcoin: [unknown command] |
| 698.237350000 | ping | Bitcoin: ping |
| 701.563581000 | inv | Bitcoin: inv |
| 701.986269000 | inv | Bitcoin: inv |
| 705.022173000 | inv | Bitcoin: inv |
| 812.115878000 | inv | Bitcoin: inv |
| 818.198570000 | [unknown co.. | Bitcoin: [unknown command] |
| 818.298733000 | ping | Bitcoin: ping |

Node discovery protocol in Wireshark

Full clients are thick clients or full nodes that download the entire blockchain; this is the most secure method of validating the blockchain as a client. Bitcoin network nodes can operate in two fundamental modes: full client or lightweight SPV client. SPV clients are used to verify payments without requiring the download of a full blockchain. SPV nodes only keep a copy of block headers of the current valid longest blockchain. Verification is performed by looking at the Merkle branch that links the transactions to the original block the transaction was accepted in. This is not very practical and requires a more practical approach, which was implemented with BIP 37 (`https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki`), where bloom filters were used to filter out relevant transactions only.

Bloom filter is basically a data structure (a bit vector with indexes) that is used to test the membership of an element in a probabilistic manner. It basically provides probabilistic lookup with false positives but no false negatives. It means that this filter can produce an output where an element that is not a member of the set being tested is wrongly considered to be in the set, but it can never produce an output where an element does exist in the set, but it asserts that it does not.

Elements are added to the bloom filter after hashing them several times and then set the corresponding bits in the bit vector to `1` via the corresponding index. In order to check the presence of the element in the bloom filter, the same hash functions are applied and compared with the bits in the bit vector to see whether the same bits are set to `1`. Not every hash function (such as SHA-1) is suitable for bloom filters as they need to be fast, independent, and uniformly distributed. The most commonly used hash functions for bloom filters are FNV, Murmur, and Jenkins.

These filters are mainly used by SPV clients to request transactions and the Merkle blocks they are interested in. A Merkle block is a lightweight version of the block, which includes a block header, some hashes, a list of 1-bit flags, and a transaction count. This information can then be used to build a Merkle tree. This is achieved by creating a filter that matches only those transaction and blocks that have been requested by the SPV client. Once the `version` messages have been exchanged and connection has been established between peers, the nodes can set filters according to their requirements.

These probabilistic filters offer a varying degree of privacy or precision depending upon how accurately or loosely they have been set. A strict bloom filter will only filter transactions that have been requested by the node but at the expense of the possibility of revealing the user addresses to adversaries who can correlate transactions with their IP addresses, thus compromising privacy. On the other hand, a loosely set filter can result in retrieving more unrelated transactions but will offer more privacy. Also, for SPV clients, bloom filters allow them to use low bandwidth as opposed to downloading all transactions for verification.

BIP 37 proposed the Bitcoin implementation of bloom filters and introduced three new messages to the Bitcoin protocol:

- `filterload`: This is used to set the bloom filter on the connection
- `filteradd`: This adds a new data element to the current filter
- `filterclear`: This deletes the currently loaded filter

More details can be found in the BIP 37 specification. This is available at `https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki`.

# Wallets

The wallet software is used to store private or public keys and Bitcoin address. It performs various functions, such as receiving and sending bitcoins. Nowadays, software usually offers both functionalities: Bitcoin client and wallet. On the disk, the Bitcoin core client wallets are stored as the Berkeley DB file:

```
$ file wallet.dat
wallet.dat: Berkeley DB (B-tree, version 9, native byte-order)
```

Private keys are generated by randomly choosing a 256-bit number by wallet software. The rules of generation are predefined and were discussed in `Chapter 4`, *Public Key Cryptography*. Private keys are used by wallets to sign the outgoing transactions. Wallets do not store any coins, and there is no concept of wallets storing balance or coins for a user. In fact, in the Bitcoin network, coins do not exist; instead, only transaction information is stored on the blockchain (more precisely, UTXO, unspent outputs), which are then used to calculate the number of bitcoins.

In Bitcoin, there are different types of wallets that can be used to store private keys. As a software program, they also provide some functions to the users to manage and carry out transactions on the Bitcoin network.

# 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 generates keys as and when required. Managing a large number of keys is very difficult and an error-prone process can lead to 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 out of 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 BIP 39, a Bitcoin improvement proposal for mnemonic code for generating deterministic keys. This BIP is available at `https://github.com/bitcoin/bips/blob/master/bip-0039.mediawiki`. This phrase can be used to recover all keys and makes private key management comparatively easier.

# Hierarchical Deterministic wallets

Defined in BIP32 and BIP44, **Hierarchical Deterministic** (**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. There are many free and commercially available HD wallets available. For example, Trezor (`https://trezor.io`), Jaxx (`https://jaxx.io/`) and Electrum (`https://electrum.org/`).

# Brain wallets

The master private key can also be derived from the hash 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.

Paper wallets can be generated online from various service providers, such as `https://bitcoinpaperwallet.com/` or `https://www.bitaddress.org/`.

# Hardware wallets

Another method is to use a tamper-resistant device to store keys. This tamper-resistant device can be custom-built or 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. The following is the photo of a Trezor wallet:

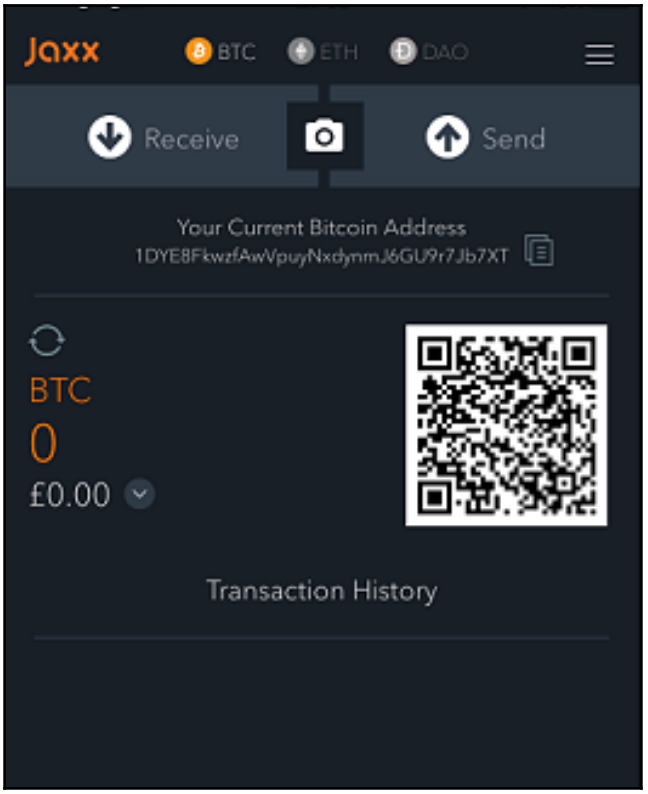

Trezor wallet

# 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 online wallet is GreenAddress, which is available at `https://greenaddress.it/en/`.

# Mobile wallets

Mobile wallets, as the name suggests, are installed on mobile devices. They can provide various methods to make payments, most notably the ability to use smartphone cameras to scan QR codes quickly and make payments. Mobile wallets are available for the Android platform and iOS, for example, Blockchain, breadwallet, Copay, and Jaxx.




Jaxx mobile wallet

The choice of Bitcoin wallet depends on several factors such as security, ease of use, and available features. Out of all these attributes, security of course comes first and when making a decision about which wallet to use, security should be of paramount importance. Hardware wallets tend to be more secure as compared to web wallets because of their tamper resistant design. Web wallets by nature are hosted on websites, which may not be as secure as a tamper resistant hardware device. Generally, mobile wallets for smartphone devices are quite popular due to a balanced combination of features and security. There are many companies offering these wallets on the iOS App Store and Android Play. It is however quite difficult to suggest that which type of wallet should be used, it also depends on personal preferences and features available in a wallet. It is advisable that security should be kept in mind while making decision on which wallet to choose.

# Bitcoin payments

Bitcoins can be accepted as payments using various techniques. Bitcoin is not recognized as a legal currency in many jurisdictions, but it is increasingly being accepted as a payment method by many online merchants and e-commerce websites. There are a number of ways in which buyers can pay the business that accepts bitcoins. For example, in an online shop, Bitcoin merchant solutions can be used, whereas in traditional, physical shops, point of sale terminals and other specialized hardware can be used. Customers can simply scan the QR code with the seller's payment URI in it and pay using their mobile devices. Bitcoin URIs allow users to make payments by simply clicking on links or scanning QR codes. **Uniform Resource Identifier** (**URI**) is basically a string that represents the transaction information. It is defined in BIP 21. The QR code can be displayed near the point of the sale terminal. Nearly all Bitcoin wallets support this feature.

Businesses can use the following logo to advertise that they accept bitcoins as payment from customers.


bitcoin accepted here logo

Various payment solutions, such as XBTerminal and 34 Bytes bitcoin **Point of Sale** (**POS**) terminal are available commercially.

Generally, these solutions work by following these steps:

1. The sales person enters the amount of money to be charged in Fiat currency, for example, US Dollars
2. Once the value is entered in the system the terminal prints a receipt with QR code on it and other relevant information such as amount
3. The customer can then scan this QR code using their mobile Bitcoin wallet to send the payment to the Bitcoin address of the seller embedded within the QR code
4. Once the payment is received on the designated Bitcoin address, a receipt is printed out as a physical evidence of sale

A Bitcoin POS device from 34 Bytes is shown here:

34 Bytes POS solution

The bitcoin payment processor, offered by many online service providers, allows integration with e-commerce websites. There are many options available. These payment processors can be used to accept bitcoins as payments. Some service providers also allow secure storage of bitcoins. For example, bitpay, `https://bitpay.com`. Another example is Bitcoin Merchant Solutions available at `https://www.bitcoin.com/merchant-solutions`.

Various **Bitcoin Improvement Proposals** (**BIPs**) have been proposed and finalized in order to introduce and standardize bitcoin payments. Most notably, BIP 70 (*Payment Protocol*) describes the protocol for secure communication between a merchant and customers. This protocol uses X.509 certificates for authentication and runs over HTTP and HTTPS. There are three messages in this protocol: `PaymentRequest`, `Payment`, and `PaymentACK`. The key features of this proposal are defense against man-in-the-middle attacks and secure proof of payment. Man-in-the-middle attacks can result in a scenario where the attacker is sitting between the merchant and the buyer and it would seem to the buyer that they are talking to the merchant, but in fact, the man in the middle is interacting with the buyer instead of the merchant. This can result in manipulation of the merchant's Bitcoin address to defraud the buyer.

Several other BIPs, such as BIP 71 (*Payment Protocol MIME types*) and BIP 72 (*URI extensions for Payment Protocol*), have also been implemented to standardize payment scheme to support BIP 70 (*Payment Protocol*).

Bitcoin lightning network, is a solution for scalable off-chain instant payments. It was introduced in early 2016, which allows off-blockchain payments. This network makes use of payments channels that run off the blockchain which allows greater speed and scalability of Bitcoin.

This paper is available at `https://lightning.network` and interested readers are encouraged to read the paper in order to understand the theory and rationale behind this invention.

# Innovation in Bitcoin

Bitcoin has undergone many changes and still evolving into a more and more robust and better system by addressing various weaknesses in the system. Especially, performance has been a topic of hot debate among Bitcoin experts and enthusiasts for many years. As such, various proposals have been made in the last few years to improve Bitcoin performance resulting in greater transaction speed, increased security, payment standardization and overall performance improvement at the protocol level.

These improvement proposals are usually made in the form of BIPs or fundamentally new versions of Bitcoin protocols resulting in a new network altogether. Some of the changes proposed are implementable via a soft fork but few need a hard fork and as a result, give birth to a new currency.

In the following sections, we will see what are the various BIPs that can be proposed for improvement in Bitcoin and then we will discuss some advanced protocols that have been proposed and implemented to address various weaknesses in the Bitcoin.

# Bitcoin Improvement Proposals (BIPs)

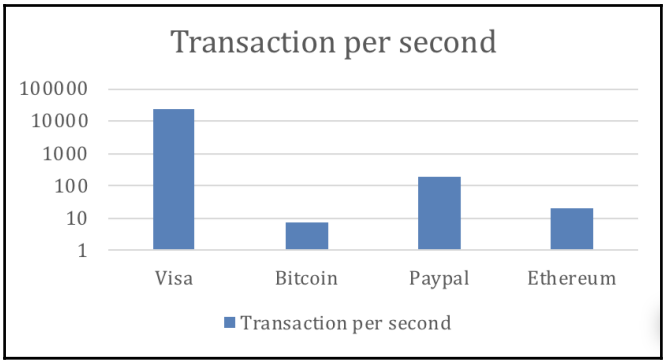
These documents are used to propose or inform the Bitcoin community about the improvements suggested, the design issues, or information about some aspects of the bitcoin ecosystem. There are three types of Bitcoin improvement proposals, abbreviated as BIPs:

- **Standard BIP**: Used to describe the major changes that have a major impact on the Bitcoin system, for example, block size changes, network protocol changes, or transaction verification changes.
- **Process BIP**: A major difference between standard and process BIPs is that standard BIPs cover protocol changes, whereas process BIPs usually deal with proposing a change in a process that is outside the core Bitcoin protocol. These are implemented only after a consensus among bitcoin users.
- **Informational BIP**: These are usually used to just advise or record some information about the Bitcoin ecosystem, such as design issues.

# Advanced protocols

In this section, we will see that what are the various advanced protocols that have been suggested or implemented for improving the Bitcoin protocol.

Transaction throughput is one of the critical issues that need to be addressed. Inherently, the Bitcoin network can only process from approximately 3 to 7 transactions per second which is a tiny number as compared to other financial networks, such as Visa which can process approximately, on average, 24,000 transactions per second. PayPal can process approximately 200 transactions per second whereas Ethereum can process up to on average 20. As Bitcoin Network grew exponentially over the last few years, these issues started to grow even further. The difference of processing speed is also shown below in a graph which shows the scale of difference between Bitcoin and other networks' transaction speeds.




Bitcoin transaction speed as compared to other networks (on logarithmic scale)

Also, security issues such as transaction malleability are of real concern which can result in denial of service. Various proposals have been made to improve the Bitcoin proposal to address various weaknesses. A selection of these proposals is presented in the following subsections.

# Segregated Witness (SegWit)

The SegWit or Segregated Witness is a soft fork based update to the Bitcoin protocol which addresses some weaknesses such as throughput and security in the Bitcoin protocol. SegWit offers a number of improvements as listed here:

- Fix for transaction malleability due to the separation of signature data from transactional data. In this case, it is no longer possible to modify transaction ID because it is no longer calculated based on the signature data present within the transaction.
- Reduction in transaction size results in cheaper transaction fees.
- Reduction in transaction signing and verification times, which results in faster transactions.
- Script versioning, which allows version number to be prefixed to locking scripts. This change can result in improvements in the scripting language without requiring a hard fork and by just increasing the version number of the script.
- Reduction in input verification time.

SegWit was proposed in BIP 141, BIP 143, BIP 144 and BIP 145. It has been activated on Bitcoin main network on August 24, 2017. The key idea behind SegWit is the separation of signature data from transaction data, which results in reduced size of the transaction. This results in block size increase up to 4 MB. However, the practical limit is between 1.6 MB to 2 MB. Instead of hard size limit of 1 MB blocks, SegWit has introduced a new concept of block weight limit.

To spend an **Unspent Transaction Output** (**UTXO**) in Bitcoin, it needs a valid signature to be provided. In the pre-SegWit scenario, this signature is provided within the locking script whereas in SegWit this signature is not part of the transaction and is provided separately.

There are two types of transaction that can now be constructed using SegWit wallets but note that these are not new transaction types as such, these are just new ways by which UTXOs can be spent. These types are:

- **Pay to Witness public key hash** (**P2WPKH**)
- **Pay to Witness Script hash** (**P2WSH**)

# Bitcoin Cash

Bitcoin Cash increases the block limit to 8 MB. This immediately increases the number of transactions that can be processed in one block to a much larger number as compared to 1 MB limit in original Bitcoin protocol. It uses PoW as consensus algorithm, and mining hardware is still ASIC based. The block interval is changed from 10 minutes to 10 seconds and up to 2 hours. It also provides replay protection and wipe-out protection.

# Bitcoin Unlimited

In this proposal, the size of the block is increased but not set to a hard limit. Instead, miners come to a consensus on the block size cap over a period of time. Other concepts such as *parallel validation* and *extreme thin blocks* have also been proposed in Bitcoin Unlimited.

Its client is available for download at `https://www.bitcoinunlimited.info`.

Extreme thin blocks allow for a faster block propagation between Bitcoin nodes. In this scheme the node requesting blocks sends a `getdata` request along with a bloom filter to another node. Purpose of this bloom filter is to filter out the transactions that already exists in the **mempool** (short for **memory pool**) of the requesting node. The node then sends back a *thin block* only containing the missing transactions. This fixes an inefficiency in Bitcoin where by transaction are regularly received twice, once at the time of broadcast by the sender and then again when a mined block is broadcasted with the confirmed transaction.

Parallel validation allows nodes to validate more than one block along with new incoming transactions in parallel. This mechanism is in contrast to Bitcoin where a node during its validation period after receiving a new block cannot relay new transactions or validate any blocks until it has accepted or rejected the block.

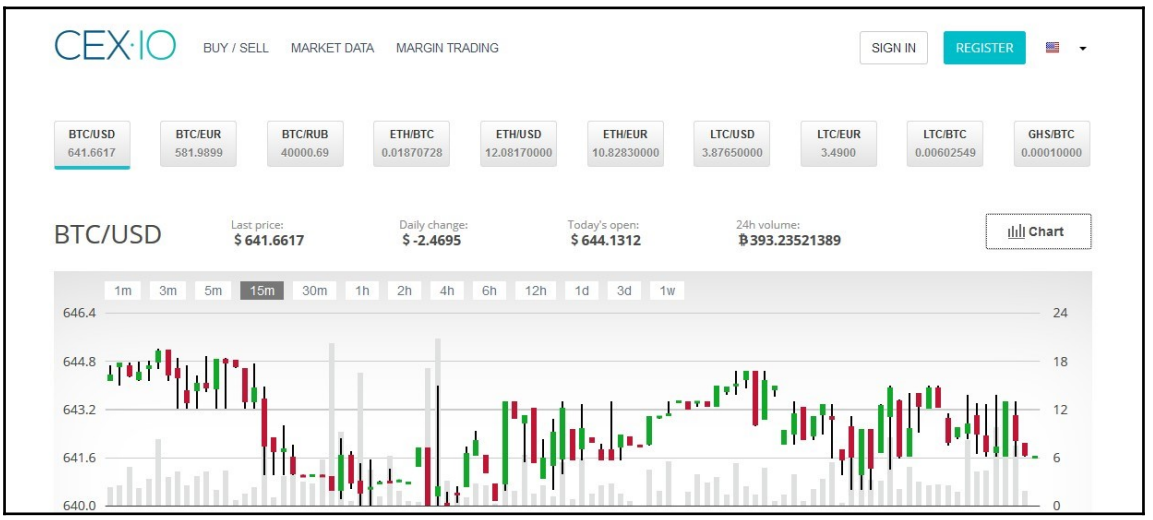
# Bitcoin Gold

This proposal has been implemented as a hard fork since block 49,1407 of the original Bitcoin blockchain. Being a hard fork, it resulted in a new blockchain, named **Bitcoin Gold** (**BTG**). The core idea behind this concept is to address the issue of mining centralization which has hurt the original Bitcoin idea of decentralized digital cash whereby more hash power has resulted in a power shift towards miners with more hashing power. BTG uses the Equihash algorithm as its mining algorithm instead of PoW; hence it is inherently ASIC resistant and uses GPUs for mining.

There are other proposals like Bitcoin Next Generation, Solidus, Spectre, and SegWit2x which will be discussed later in this book in `Chapter 18`, *Scalability and Other Challenges*, in the context of performance improvement in blockchain networks.

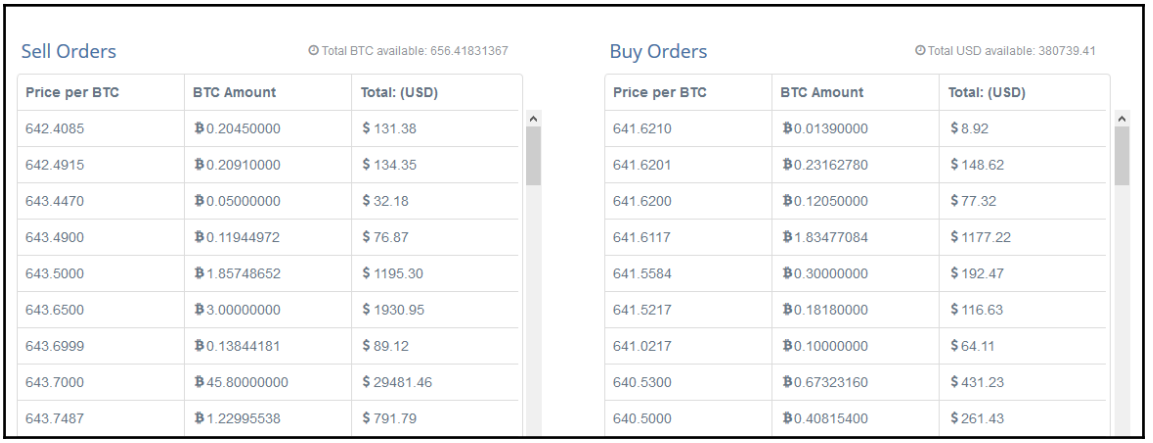
# Bitcoin investment and buying and selling bitcoins

There are many online exchanges where users can buy and sell bitcoins. This is a big business on the internet now and it offers bitcoin trading, CFDs, spread betting, margin trading, and various other choices. Traders can buy bitcoins or trade by opening long or short positions to make a profit when bitcoin's price goes up or down. Several other features, such as exchanging bitcoins for other virtual currencies, are also possible, and many online bitcoin exchanges provide this function. Advanced market data, trading strategies, charts, and relevant data to support traders is also available. An example is shown from CEX (`https://cex.io`) here. Other exchanges offer similar types of services.




Example of bitcoin exchange cex.io

The following screenshot shows the order book at the exchange where all buy and sell orders are listed:



Sell Orders — Total BTC available: 656.41831367

| Price per BTC | BTC Amount | Total: (USD) |
|---|---|---|
| 642.4085 | Ƀ0.20450000 | $ 131.38 |
| 642.4915 | Ƀ0.20910000 | $ 134.35 |
| 643.4470 | Ƀ0.05000000 | $ 32.18 |
| 643.4900 | Ƀ0.11944972 | $ 76.87 |
| 643.5000 | Ƀ1.85748652 | $ 1195.30 |
| 643.6500 | Ƀ3.00000000 | $ 1930.95 |
| 643.6999 | Ƀ0.13844181 | $ 89.12 |
| 643.7000 | Ƀ45.80000000 | $ 29481.46 |
| 643.7487 | Ƀ1.22995538 | $ 791.79 |

Buy Orders — Total USD available: 380739.41

| Price per BTC | BTC Amount | Total: (USD) |
|---|---|---|
| 641.6210 | Ƀ0.01390000 | $ 8.92 |
| 641.6201 | Ƀ0.23162780 | $ 148.62 |
| 641.6200 | Ƀ0.12050000 | $ 77.32 |
| 641.6117 | Ƀ1.83477084 | $ 1177.22 |
| 641.5584 | Ƀ0.30000000 | $ 192.47 |
| 641.5217 | Ƀ0.18180000 | $ 116.63 |
| 641.0217 | Ƀ0.10000000 | $ 64.11 |
| 640.5300 | Ƀ0.67323160 | $ 431.23 |
| 640.5000 | Ƀ0.40815400 | $ 261.43 |

Example of bitcoin order book at exchange cex.io

The order book shown in the preceding screenshot displays sell and buy orders. Sell orders are also called ask and buy orders are also called bid orders. This means that ask price is at what seller is willing to sell the bitcoin whereas bid price is what the buyer is willing to pay. If bid and ask prices match then a trade can occur. Most common order types are market orders and limit orders. Market orders mean that as soon as the prices match the order will be filled immediately. Limit orders allow buying and selling of set number of bitcoins at a specified price or better. Also, a period of time can be set during which the order can be left open, if not executed then it will be cancelled. We have introduced trading concepts in more detail in `Chapter 4`, *Public Key Cryptography*, under *Financial markets* section, interested readers can refer to this section for more details.

# Summary

We started this chapter with the introduction to Bitcoin network, following it with a discussion on Bitcoin node discovery and block synchronization protocols. Moreover, we presented different types of network messages. Then we examined different types of Bitcoin wallets and discussed various attributes and features of each type. Following this, we looked at Bitcoin payments and payment processors. In the last section, we discussed Bitcoin innovations, which included topics such as Bitcoin Improvement Proposals, and advanced Bitcoin protocols. Finally, we presented a basic introduction to Bitcoin buying and selling.

In the next chapter, we will discuss Bitcoin clients, such as Bitcoin Core client, which can be used to interact with the Bitcoin blockchain and also acts as a wallet. In addition, we will explore some of the APIs that are available for programming Bitcoin applications.