The Bitcoin Network

Implementation

Bitcoin network

The only source you really need:

https://en.bitcoin.it/wiki/Protocol_documentation

Simple explanation of the network protocol/messages

But let's go step by step

Bitcoin network

Bitcoin nodes:

- Run software implementing the logic of Bitcoin (transactions, blockchain, proof of work, addresses of other nodes)
- They create a p2p network for transmitting messages (gossip protocol)
- *Messages can also be interchanged outside of this network!!!*

Objective of this class:

- Implement (part of) the communication bewteen the nodes (be it full, SPV, or users that do not run a node)

A message on the Bitcoin network

What a message looks like?

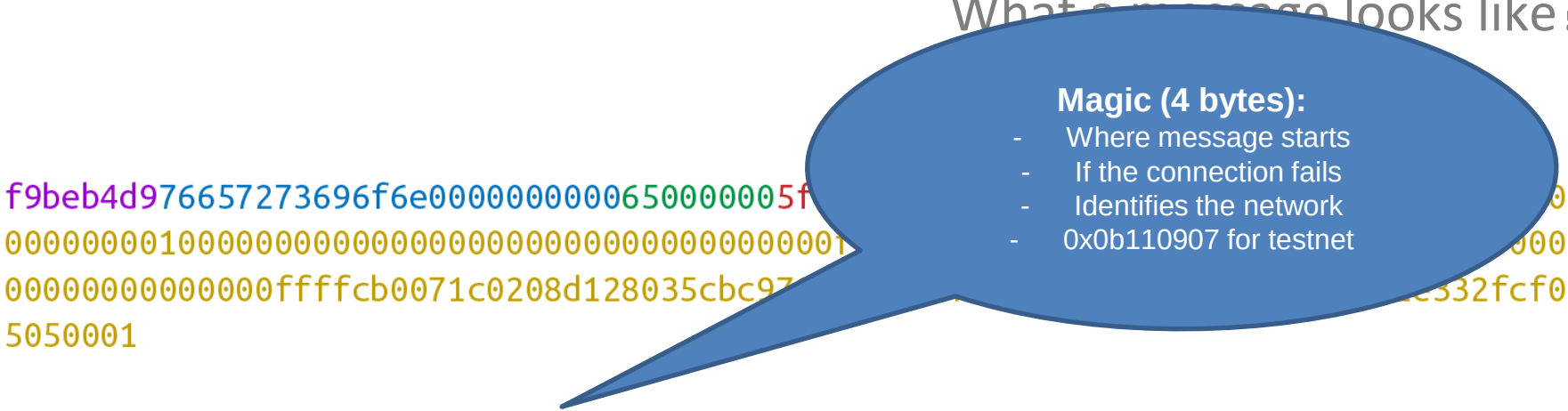
```
f9beb4d976657273696f6e0000000000650000005f1a69d2721101000100000000000000bc8f5e540
0000000010000000000000000000000000000000000ffff61b6409208d010000000000000000000000
000000000000ffffcb0071c0208d128035cbc97953f80f2f5361746f7368693a302e392e332fcf0
5050001
```

- f9beb4d9 - network magic (always 0xf9beb4d9 for mainnet)
- 76657273696f6e0000000000 - command, 12 bytes, human-readable
- 65000000 - payload length, 4 bytes, little-endian
- 5f1a69d2 - payload checksum, first 4 bytes of hash256 of the payload
- 7211...01 - payload

https://github.com/jimmysong/programmingbitcoin/blob/master/ch10.asciidoc

A message on the Bitcoin network

What a message looks like?



- f9beb4d9 - network magic (always 0xf9beb4d9 for mainnet)
- 76657273696f6e0000000000 - command, 12 bytes, human-readable
- 65000000 - payload length, 4 bytes, little-endian
- 5f1a69d2 - payload checksum, first 4 bytes of hash256 of the payload
- 7211...01 - payload

https://github.com/jimmysong/programmingbitcoin/blob/master/ch10.asciidoc

A message on the Bitcoin network

What a message looks like?

- f9beb4d9 - network magic (always 0xf9beb4d9 for mainnet)
- 76657273696f6e0000000000 - command, 12 bytes, human-readable
- 65000000 - payload length, 4 bytes, little-endian
- 5f1a69d2 - payload checksum, first 4 bytes of hash256 of the payload
- 7211...01 - payload

https://github.com/jimmysong/programmingbitcoin/blob/master/ch10.asciidoc

A message on the Bitcoin network

What a message looks like?

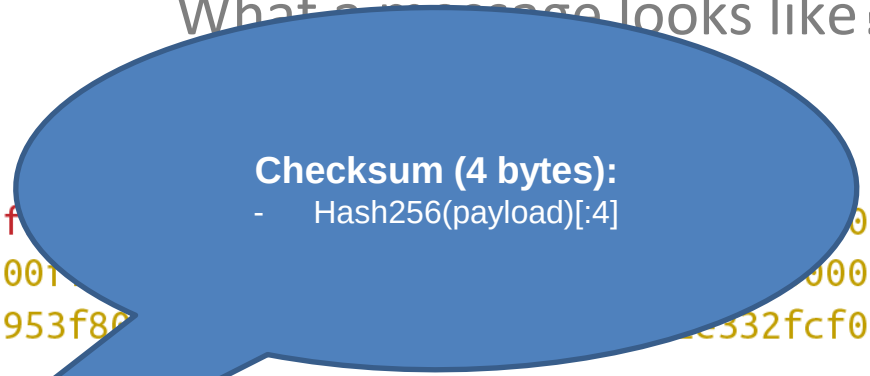

```
f9beb4d976657273696f6e0000000000650000005f
0000000010000000000000000000000000000000001
0000000000000ffffcb0071c0208d128035cbc97953f80
5050001
```

- f9beb4d9 - network magic (always 0xf9beb4d9 for mainnet)
- 76657273696f6e0000000000 - command, 12 bytes, human-readable
- 65000000 - payload length, 4 bytes, little-endian
- 5f1a69d2 - payload checksum, first 4 bytes of hash256 of the payload
- 7211...01 - payload

https://github.com/jimmysong/programmingbitcoin/blob/master/ch10.asciidoc

A message on the Bitcoin network

What a message looks like?



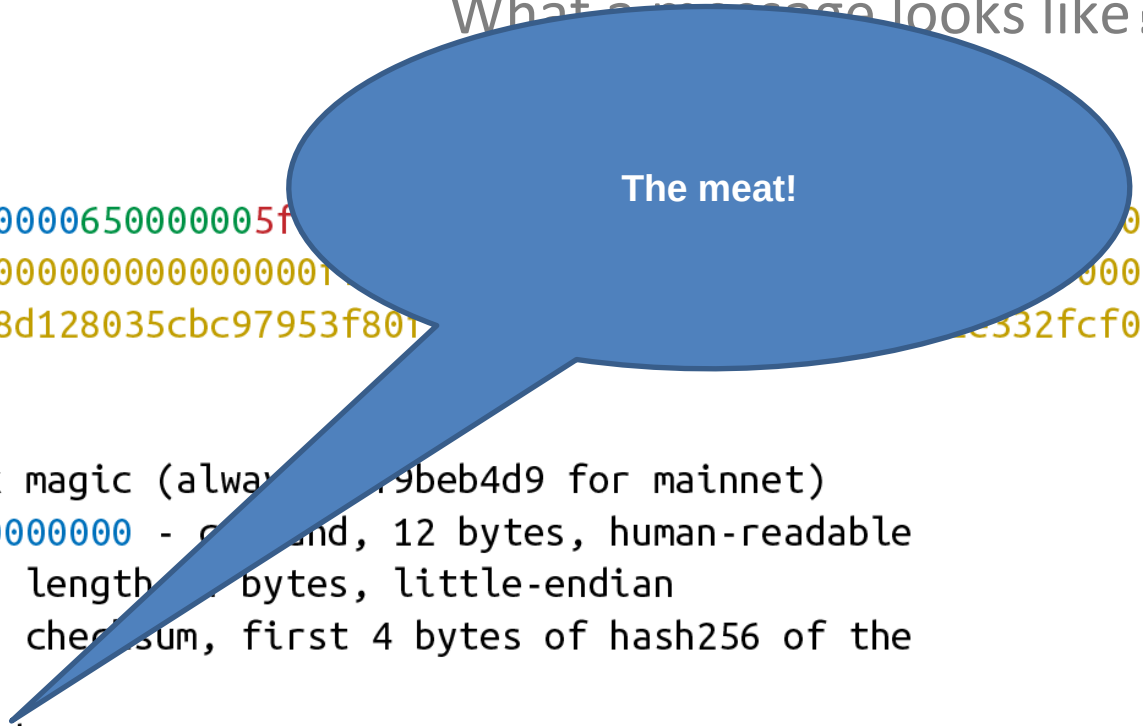

```
f9beb4d976657273696f6e0000000000650000005f
0000000010000000000000000000000000000000001
0000000000000ffffcb0071c0208d128035cbc97953f80
5050001
```

- f9beb4d9 - network magic (always 0xf9beb4d9 for mainnet)
- 76657273696f6e0000000000 - command, 12 bytes, human-readable
- 65000000 - payload length, 4 bytes, little-endian
- 5f1a69d2 - payload checksum, first 4 bytes of hash256 of the payload
- 7211...01 - payload

https://github.com/jimmysong/programmingbitcoin/blob/master/ch10.asciidoc

A message on the Bitcoin network

What a message looks like?




```
f9beb4d976657273696f6e0000000000650000005f
0000000010000000000000000000000000000000001
00000000000000ffffcb0071c0208d128035cbc97953f801
5050001
```

- f9beb4d9 - network magic (alwa… f9beb4d9 for mainnet)
- 76657273696f6e0000000000 - c… and, 12 bytes, human-readable
- 65000000 - payload length … bytes, little-endian
- 5f1a69d2 - payload che…sum, first 4 bytes of hash256 of the payload
- 7211...01 - payload

https://github.com/jimmysong/programmingbitcoin/blob/master/ch10.asciidoc

A message on the Bitcoin network

Implementation - class that stores the message

```
class NetworkEnvelope:

    def __init__(self, command, payload, testnet=False):
        self.command = command
        self.payload = payload
        if testnet:
            self.magic = TESTNET_NETWORK_MAGIC
        else:
            self.magic = NETWORK_MAGIC
```

A message on the Bitcoin network

Implementation - class that stores the message

```
@classmethod
def parse(cls, s, testnet=False):
    '''Takes a stream and creates a NetworkEnvelope'''
    # check the network magic
    magic = s.read(4)
    if magic == b'':
        raise RuntimeError('Connection reset!')
    if testnet:
        expected_magic = TESTNET_NETWORK_MAGIC
    else:
        expected_magic = NETWORK_MAGIC
    if magic != expected_magic:
        raise RuntimeError('magic is not right {} vs {}'.format(magic.hex(), expected_magic.hex()))
    # command 12 bytes
    command = s.read(12)
    # strip the trailing 0's
    command = command.strip(b'\x00')
    # payload length 4 bytes, little endian
    payload_length = little_endian_to_int(s.read(4))
    # checksum 4 bytes, first four of hash256 of payload
    checksum = s.read(4)
    # payload is of length payload_length
    payload = s.read(payload_length)
    # verify checksum
    calculated_checksum = hash256(payload)[:4]
    if calculated_checksum != checksum:
        raise RuntimeError('checksum does not match')
    # return an instance of the class
    return cls(command, payload, testnet=testnet)
```

A message on the Bitcoin network

Implementation - class that stores the message

```
class NetworkEnvelope:
    def serialize(self):
        '''Returns the byte serialization of the entire network message'''
        # add the network magic
        result = self.magic
        # command 12 bytes
        # fill with 0's
        result += self.command + b'\x00' * (12 - len(self.command))
        # payload length 4 bytes, little endian
        result += int_to_little_endian(len(self.payload), 4)
        # checksum 4 bytes, first four of hash256 of payload
        result += hash256(self.payload)[:4]
        # payload
        result += self.payload
        return result
```

Bitcoin network

Some important commands

In this class we will implement:

- Version
- Verack
- GetHeaders
- Headers

In our implementation:

- To send a message: we need *serialize*
- To process a received message: we need *parse*

Bitcoin network

Version

```
7f11010000000000000000ad17835b00000000000000000000000000000000000000ffff0
0000000 8d20 0000000000000000000000000000000000000000ffff00000000 8d20 f6a8d7a440ec27a11b
2f70726f6772616d6d696e67626c6f636b636861696e3a302e312f 0000000001
```

7f110100 - Protocol version, 4 bytes, little-endian, 70015
0000000000000000 - Network services of sender, 8 bytes, little-endian
ad17835b00000000 - Timestamp, 8 bytes, little-endian
0000000000000000 - Network services of receiver, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of receiver, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of receiver, 2 bytes, 8333
0000000000000000 - Network services of sender, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of sender, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of sender, 2 bytes, 8333
f6a8d7a440ec27a1 - Nonce, 8 bytes, used for communicating responses
1b2f70726f6772616d6d696e67626c6f636b636861696e3a302e312f - User agent
        /programmingblockchain:0.1/
00000000 - Height, 0
01 - Optional flag for relay, based on BIP37

Bitcoin network

Version

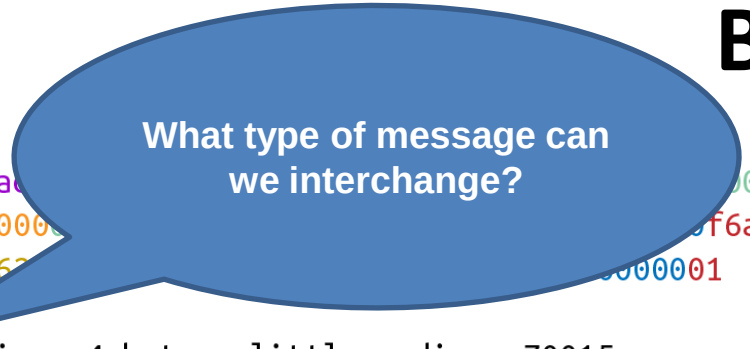
Version serves to establish a connection between two nodes!

```
7f11010000000000000000a                                    0000ffff0
0000000 8d20000000000000000                                f6a8d7a440ec27a11b
2f70726f6772616d6d696e67626c6f6                            000001
```

7f110100 - Protocol version, 4 bytes, little-endian, 70015
0000000000000000 - Network services of sender, 8 bytes, little-endian
ad17835b00000000 - Timestamp, 8 bytes, little-endian
0000000000000000 - Network services of receiver, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of receiver, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of receiver, 2 bytes, 8333
0000000000000000 - Network services of sender, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of sender, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of sender, 2 bytes, 8333
f6a8d7a440ec27a1 - Nonce, 8 bytes, used for communicating responses
1b2f70726f6772616d6d696e67626c6f636b636861696e3a302e312f - User agent
        /programmingblockchain:0.1/
00000000 - Height, 0
01 - Optional flag for relay, based on BIP37

Bitcoin network

Version

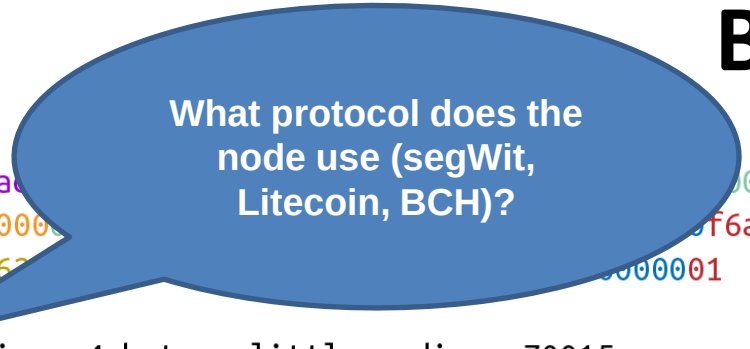


7f11010000000000000000ad 0000000000000ffff0
0000000 8d20000000000000000 f6a8d7a440ec27a11b
2f70726f6772616d6d696e676 0000001

7f110100 - Protocol version, 4 bytes, little-endian, 70015
0000000000000000 - Network services of sender, 8 bytes, little-endian
ad17835b00000000 - Timestamp, 8 bytes, little-endian
0000000000000000 - Network services of receiver, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of receiver, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of receiver, 2 bytes, 8333
0000000000000000 - Network services of sender, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of sender, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of sender, 2 bytes, 8333
f6a8d7a440ec27a1 - Nonce, 8 bytes, used for communicating responses
1b2f70726f6772616d6d696e67626c6f636b636861696e3a302e312f - User agent
        /programmingblockchain:0.1/
00000000 - Height, 0
01 - Optional flag for relay, based on BIP37

Bitcoin network

Version

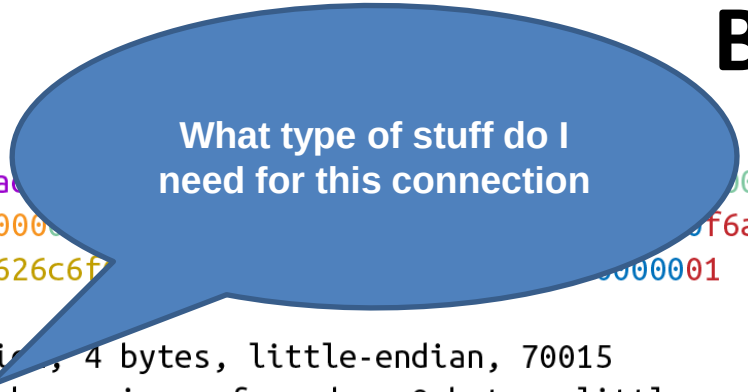


7f11010000000000000000a[illegible]000000000000ffff0
0000000[illegible]8d20000000000000000[illegible]f6a8d7a440ec27a11b
2f70726f6772616d6d696e676[illegible]000001

7f110100 - Protocol version, 4 bytes, little-endian, 70015
0000000000000000 - Network services of sender, 8 bytes, little-endian
ad17835b00000000 - Timestamp, 8 bytes, little-endian
0000000000000000 - Network services of receiver, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of receiver, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of receiver, 2 bytes, 8333
0000000000000000 - Network services of sender, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of sender, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of sender, 2 bytes, 8333
f6a8d7a440ec27a1 - Nonce, 8 bytes, used for communicating responses
1b2f70726f6772616d6d696e67626c6f636b636861696e3a302e312f - User agent
        /programmingblockchain:0.1/
00000000 - Height, 0
01 - Optional flag for relay, based on BIP37

Bitcoin network

Version



7f11010000000000000000ad [illegible] 000000000000ffff0
0000000 8d20 0000000000000000 [illegible] f6a8d7a440ec27a11b
2f70726f6772616d6d696e67626c6f [illegible] 000001

7f110100 - Protocol versi[illegible], 4 bytes, little-endian, 70015
0000000000000000 - Network services of sender, 8 bytes, little-endian
ad17835b00000000 - Timestamp, 8 bytes, little-endian
0000000000000000 - Network services of receiver, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of receiver, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of receiver, 2 bytes, 8333
0000000000000000 - Network services of sender, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of sender, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of sender, 2 bytes, 8333
f6a8d7a440ec27a1 - Nonce, 8 bytes, used for communicating responses
1b2f70726f6772616d6d696e67626c6f636b636861696e3a302e312f - User agent
        /programmingblockchain:0.1/
00000000 - Height, 0
01 - Optional flag for relay, based on BIP37

Bitcoin network

Version

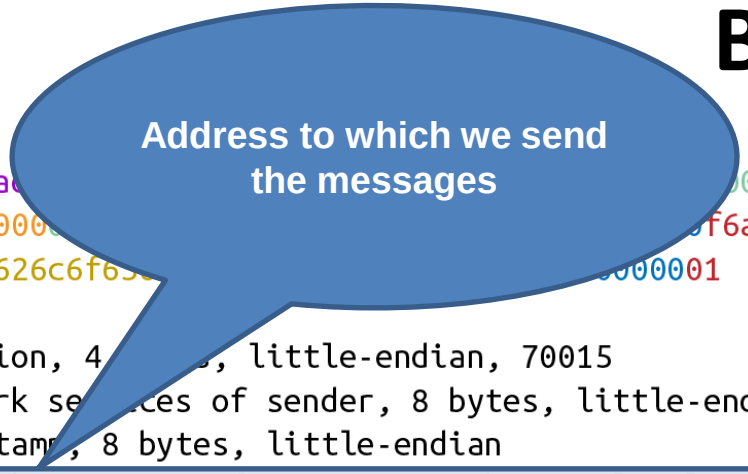
What is the time?
Serves to accept/reject
blocks (and for other things)

```
7f11010000000000000000000a                                  000000000000ffff0
0000000 8d20000000000000000000                              f6a8d7a440ec27a11b
2f70726f6772616d6d696e67626c6f6                              000001
```

7f110100 - Protocol version, 4 bytes, little-endian, 70015
0000000000000000 - Network services of sender, 8 bytes, little-endian
ad17835b00000000 - Timestamp, 8 bytes, little-endian
0000000000000000 - Network services of receiver, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of receiver, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of receiver, 2 bytes, 8333
0000000000000000 - Network services of sender, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of sender, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of sender, 2 bytes, 8333
f6a8d7a440ec27a1 - Nonce, 8 bytes, used for communicating responses
1b2f70726f6772616d6d696e67626c6f636b636861696e3a302e312f - User agent
        /programmingblockchain:0.1/
00000000 - Height, 0
01 - Optional flag for relay, based on BIP37

Bitcoin network

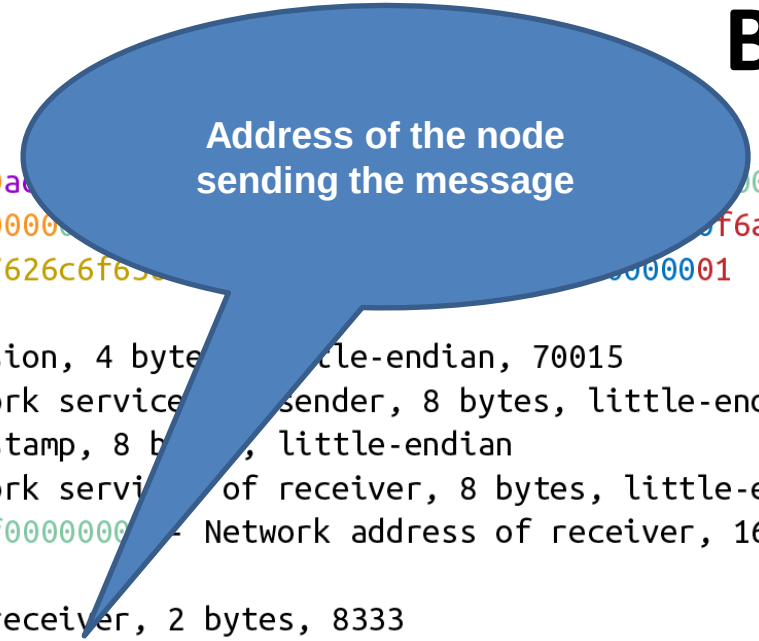
Version




```
7f11010000000000000000ad                                    000000000000ffff0
0000000 8d20000000000000000                                 f6a8d7a440ec27a11b
2f70726f6772616d6d696e67626c6f6                             0000001

7f110100 - Protocol version, 4 , little-endian, 70015
0000000000000000 - Network se ces of sender, 8 bytes, little-endian
ad17835b00000000 - Timestam , 8 bytes, little-endian
0000000000000000 - Network services of receiver, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of receiver, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of receiver, 2 bytes, 8333
0000000000000000 - Network services of sender, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of sender, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of sender, 2 bytes, 8333
f6a8d7a440ec27a1 - Nonce, 8 bytes, used for communicating responses
1b2f70726f6772616d6d696e67626c6f636b636861696e3a302e312f - User agent
        /programmingblockchain:0.1/
00000000 - Height, 0
01 - Optional flag for relay, based on BIP37
```

Bitcoin network

Version

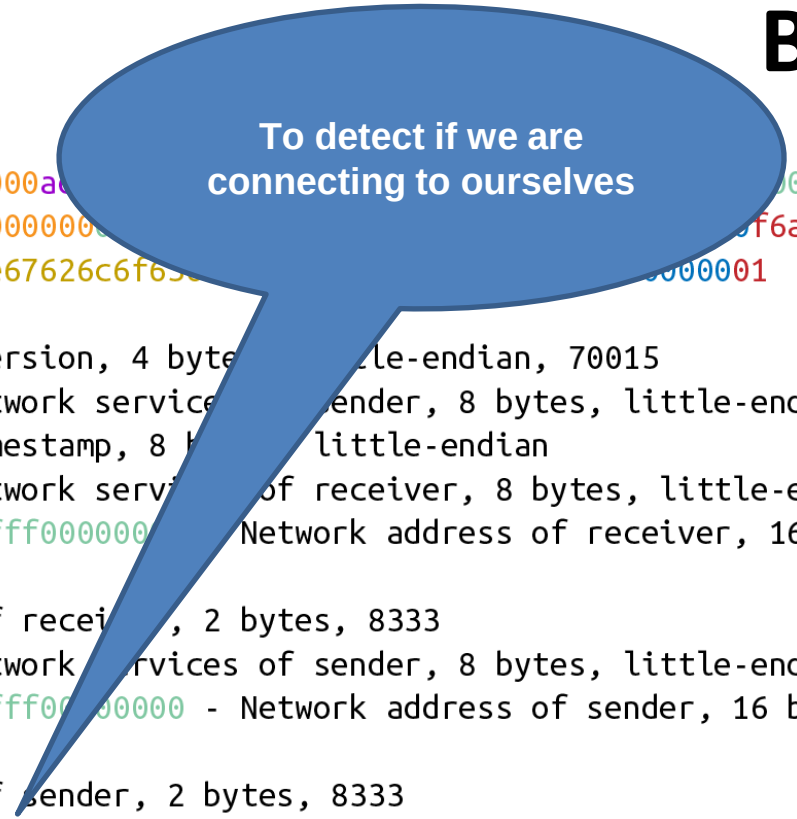


7f11010000000000000000a[illegible]000000000000ffff0
0000000[illegible]f6a8d7a440ec27a11b
2f70726f6772616d6d696e67626c6f[illegible]000001

7f110100 - Protocol version, 4 byte[illegible]tle-endian, 70015
0000000000000000 - Network service[illegible]sender, 8 bytes, little-endian
ad17835b00000000 - Timestamp, 8 b[illegible], little-endian
0000000000000000 - Network servi[illegible] of receiver, 8 bytes, little-endian
00000000000000000000ffff0000000[illegible] - Network address of receiver, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of receiver, 2 bytes, 8333
0000000000000000 - Network services of sender, 8 bytes, little-endian
00000000000000000000ffff00000000 - Network address of sender, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of sender, 2 bytes, 8333
f6a8d7a440ec27a1 - Nonce, 8 bytes, used for communicating responses
1b2f70726f6772616d6d696e67626c6f636b636861696e3a302e312f - User agent
        /programmingblockchain:0.1/
00000000 - Height, 0
01 - Optional flag for relay, based on BIP37

Bitcoin network

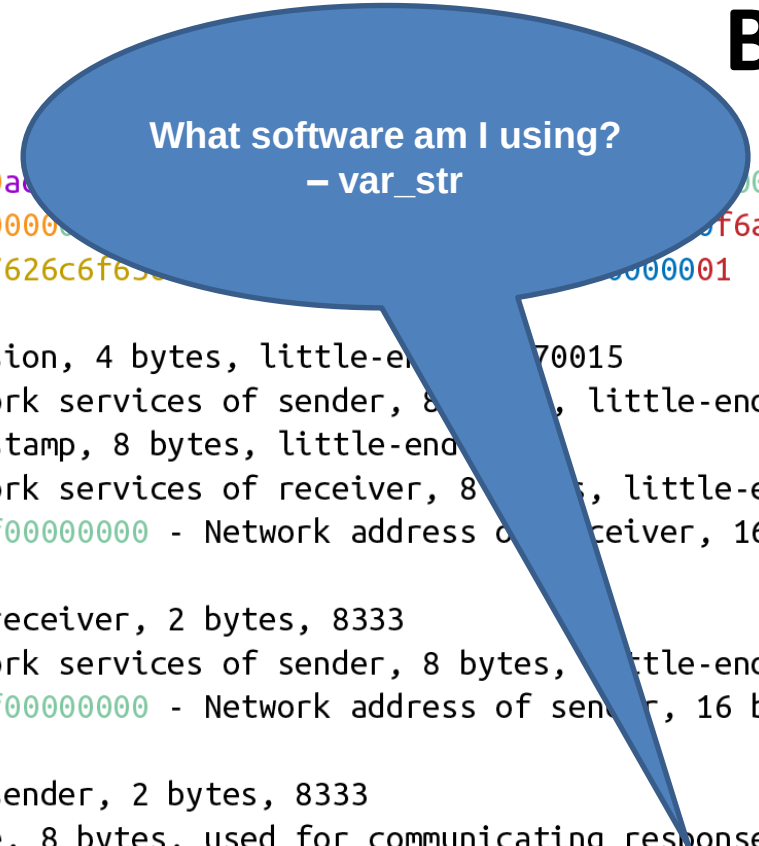
Version




```
7f11010000000000000000000a                                  000000000000ffff0
0000000 8d20000000000000000000                              f6a8d7a440ec27a11b
2f70726f6772616d6d696e67626c6f6                             000001

7f110100 - Protocol version, 4 byte                         le-endian, 70015
0000000000000000 - Network service                          ender, 8 bytes, little-endian
ad17835b00000000 - Timestamp, 8 b                           little-endian
0000000000000000 - Network servi                            of receiver, 8 bytes, little-endian
00000000000000000000ffff000000                              Network address of receiver, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of recei                                , 2 bytes, 8333
0000000000000000 - Network                                  rvices of sender, 8 bytes, little-endian
00000000000000000000ffff0000000 - Network address of sender, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of sender, 2 bytes, 8333
f6a8d7a440ec27a1 - Nonce, 8 bytes, used for communicating responses
1b2f70726f6772616d6d696e67626c6f636b636861696e3a302e312f - User agent
        /programmingblockchain:0.1/
00000000 - Height, 0
01 - Optional flag for relay, based on BIP37
```

Bitcoin network

Version

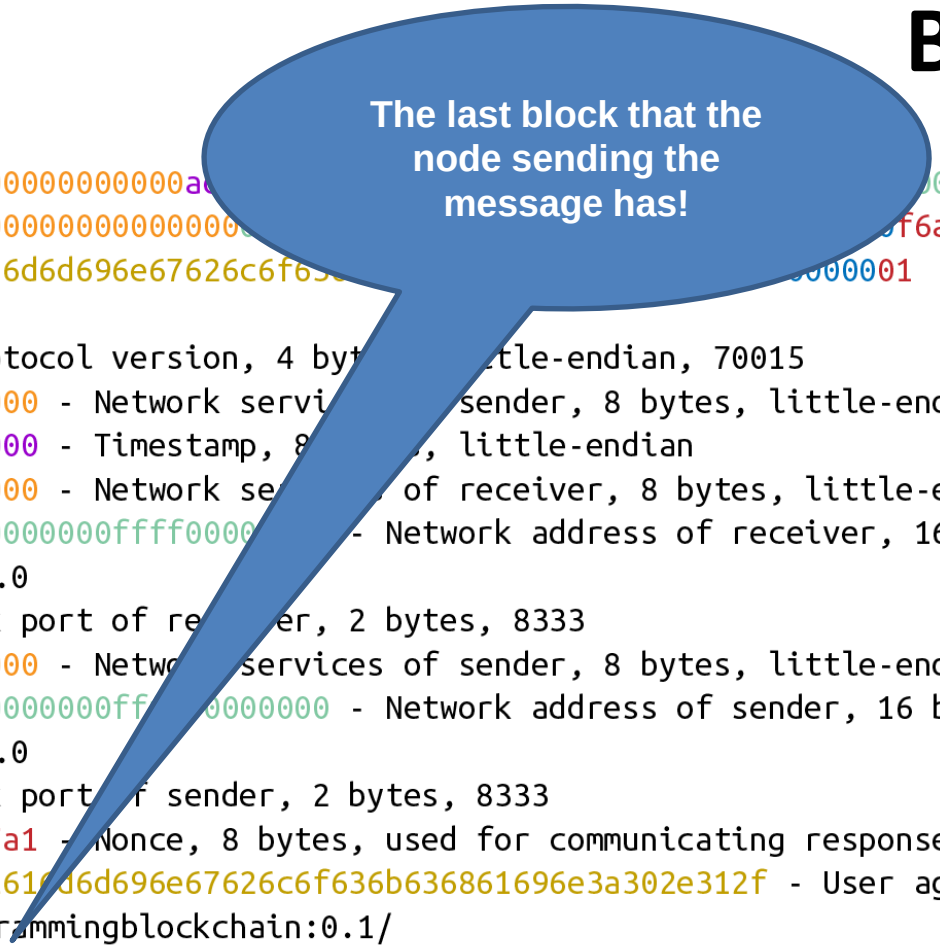


7f11010000000000000000a
0000000000000ffff0
0000000 8d20 0000000000000000
f6a8d7a440ec27a11b
2f70726f6772616d6d696e67626c6f
0000001

7f110100 - Protocol version, 4 bytes, little-e 70015
0000000000000000 - Network services of sender, 8 , little-endian
ad17835b00000000 - Timestamp, 8 bytes, little-end
0000000000000000 - Network services of receiver, 8 , little-endian
00000000000000000000ffff00000000 - Network address o ceiver, 16 bytes, IPv4
0.0.0.0
8d20 - Network port of receiver, 2 bytes, 8333
0000000000000000 - Network services of sender, 8 bytes, ttle-endian
00000000000000000000ffff00000000 - Network address of sen , 16 bytes, IPv4
0.0.0.0
8d20 - Network port of sender, 2 bytes, 8333
f6a8d7a440ec27a1 - Nonce, 8 bytes, used for communicating responses
1b2f70726f6772616d6d696e67626c6f636b636861696e3a302e312f - User agent
/programmingblockchain:0.1/
00000000 - Height, 0
01 - Optional flag for relay, based on BIP37

Bitcoin network

Version

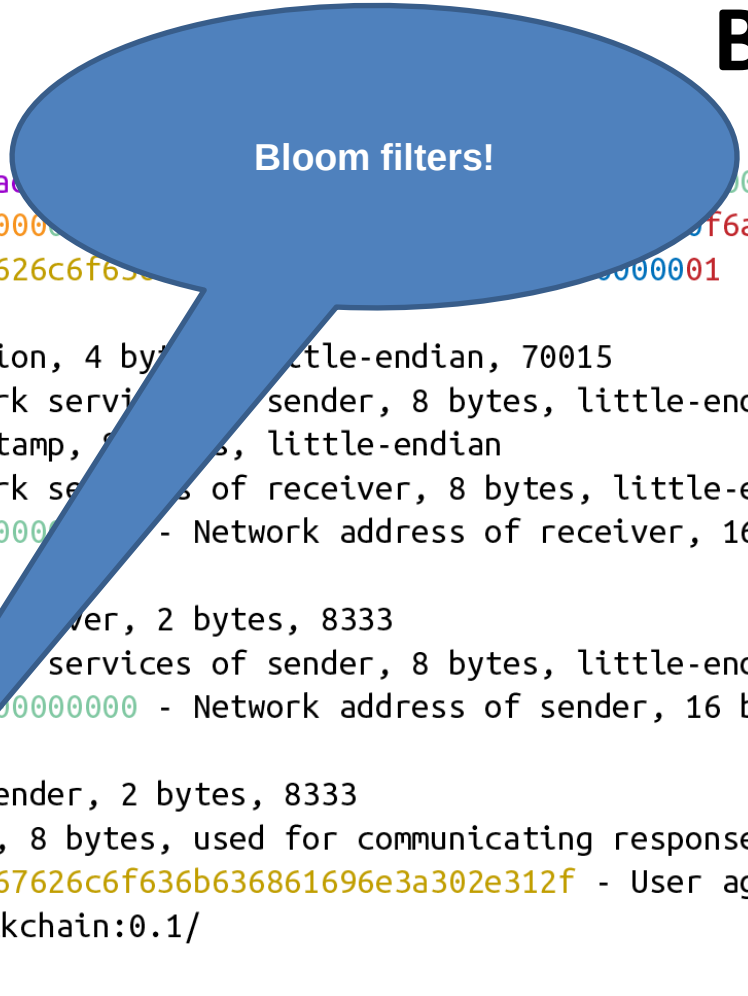


7f11010000000000000000a[illegible]000000000000ffff0
0000000[illegible]f6a8d7a440ec27a11b
2f70726f6772616d6d696e67626c6f[illegible]000001

7f110100 - Protocol version, 4 byt[illegible]ttle-endian, 70015
0000000000000000 - Network servi[illegible]sender, 8 bytes, little-endian
ad17835b00000000 - Timestamp, 8[illegible], little-endian
0000000000000000 - Network se[illegible] of receiver, 8 bytes, little-endian
00000000000000000000ffff0000[illegible] - Network address of receiver, 16 bytes, IPv4
0.0.0.0
8d20 - Network port of re[illegible]er, 2 bytes, 8333
0000000000000000 - Netwo[illegible]services of sender, 8 bytes, little-endian
00000000000000000000ff[illegible]0000000 - Network address of sender, 16 bytes, IPv4
0.0.0.0
8d20 - Network port[illegible] sender, 2 bytes, 8333
f6a8d7a440ec27a1 - Nonce, 8 bytes, used for communicating responses
1b2f70726f6772616[illegible]d6d696e67626c6f636b636861696e3a302e312f - User agent
/programmingblockchain:0.1/
00000000 - Height, 0
01 - Optional flag for relay, based on BIP37

Bitcoin network

Version




```
7f11010000000000000000000a                                  000000000000ffff0
0000000 8d20000000000000000000                              f6a8d7a440ec27a11b
2f70726f6772616d6d696e67626c6f6                             0000001

7f110100 - Protocol version, 4 by      ttle-endian, 70015
0000000000000000 - Network servi       sender, 8 bytes, little-endian
ad17835b00000000 - Timestamp,          , little-endian
0000000000000000 - Network se          of receiver, 8 bytes, little-endian
00000000000000000000ffff000            - Network address of receiver, 16 bytes, IPv4
        0.0.0.0
8d20 - Network port of r      ver, 2 bytes, 8333
0000000000000000 - Netw      services of sender, 8 bytes, little-endian
00000000000000000000f        00000000 - Network address of sender, 16 bytes, IPv4
        0.0.0.0
8d20 - Network por    of sender, 2 bytes, 8333
f6a8d7a440ec27a1      Nonce, 8 bytes, used for communicating responses
1b2f70726f67726      6d6d696e67626c6f636b636861696e3a302e312f - User agent
        /prog ammingblockchain:0.1/
00000000 - Height, 0
01 - Optional flag for relay, based on BIP37
```

Bitcoin network

Version

```
class VersionMessage:
    command = b'version'

    def __init__(self, version=70015, services=0, timestamp=None,
                 receiver_services=0,
                 receiver_ip=b'\x00\x00\x00\x00', receiver_port=8333,
                 sender_services=0,
                 sender_ip=b'\x00\x00\x00\x00', sender_port=8333,
                 nonce=None, user_agent=b'/programmingbitcoin:0.1/',
                 latest_block=0, relay=False):
        self.version = version
        self.services = services
        if timestamp is None:
            self.timestamp = int(time.time())
        else:
            self.timestamp = timestamp
        self.receiver_services = receiver_services
        self.receiver_ip = receiver_ip
        self.receiver_port = receiver_port
        self.sender_services = sender_services
        self.sender_ip = sender_ip
        self.sender_port = sender_port
        if nonce is None:
            self.nonce = int_to_little_endian(randint(0, 2**64), 8)
        else:
            self.nonce = nonce
        self.user_agent = user_agent
        self.latest_block = latest_block
        self.relay = relay
```

Bitcoin network

Version

```
class VersionMessage:
    command = b'ver
    def __init__(se
                 re
                 re
                 se
                 se
                 no
                 la
        self.versio
        self.servic
        if timestam
            self.ti
        else:
            self.ti
        self.receiv
        self.receiv
        self.receiv
        self.sender
        self.sender
        self.sender
        if nonce is
            self.no
        else:
            self.no
        self.user_a
        self.latest
        self.relay
```

```
def serialize(self):
    '''Serialize this message to send over the network'''
    # version is 4 bytes little endian
    result = int_to_little_endian(self.version, 4)
    # services is 8 bytes little endian
    result += int_to_little_endian(self.services, 8)
    # timestamp is 8 bytes little endian
    result += int_to_little_endian(self.timestamp, 8)
    # receiver services is 8 bytes little endian
    result += int_to_little_endian(self.receiver_services, 8)
    # IPV4 is 10 00 bytes and 2 ff bytes then receiver ip
    result += b'\x00' * 10 + b'\xff\xff' + self.receiver_ip
    # receiver port is 2 bytes, big endian
    result += self.receiver_port.to_bytes(2, 'big')
    # sender services is 8 bytes little endian
    result += int_to_little_endian(self.sender_services, 8)
    # IPV4 is 10 00 bytes and 2 ff bytes then sender ip
    result += b'\x00' * 10 + b'\xff\xff' + self.sender_ip
    # sender port is 2 bytes, big endian
    result += self.sender_port.to_bytes(2, 'big')
    # nonce should be 8 bytes
    result += self.nonce
    # useragent is a variable string, so varint first
    result += encode_varint(len(self.user_agent))
    result += self.user_agent
    # latest block is 4 bytes little endian
    result += int_to_little_endian(self.latest_block, 4)
    # relay is 00 if false, 01 if true
    if self.relay:
        result += b'\x01'
    else:
        result += b'\x00'
    return result
```

Bitcoin network

Verack

verack

The *verack* message is sent in reply to *version*. This message consists of only a message header with the command string "verack".

Hexdump of the verack message:

```
0000   F9 BE B4 D9 76 65 72 61  63 6B 00 00 00 00 00 00   ....verack......
0010   00 00 00 00 5D F6 E0 E2                           ........

Message header:
 F9 BE B4 D9                          - Main network magic bytes
 76 65 72 61  63 6B 00 00 00 00 00 00 - "verack" command
 00 00 00 00                          - Payload is 0 bytes long
 5D F6 E0 E2                          - Checksum (little endian)
```

https://en.bitcoin.it/wiki/Protocol_documentation

Bitcoin network

Verack

verack

I.e. "OK"

The *verack* message is sent in reply to *version*. This message consist ng "verack".

Hexdump of the verack message:

```
0000   F9 BE B4 D9 76 65 72 61  63 6B 00 00 00 00 00 00   ....verack......
0010   00 00 00 00 5D F6 E0 E2                           ........

Message header:
 F9 BE B4 D9                          - Main network magic bytes
 76 65 72 61  63 6B 00 00 00 00 00 00 - "verack" command
 00 00 00 00                          - Payload is 0 bytes long
 5D F6 E0 E2                          - Checksum (little endian)
```

https://en.bitcoin.it/wiki/Protocol_documentation

Bitcoin network

Verack

```
class VerAckMessage:
    command = b'verack'

    def __init__(self):
        pass

    @classmethod
    def parse(cls, s):
        return cls()

    def serialize(self):
        return b''
```

A simple node

```
class SimpleNode:

    def __init__(self, host, port=None, testnet=False, logging=False):
        if port is None:
            if testnet:
                port = 18333
            else:
                port = 8333
        self.testnet = testnet
        self.logging = logging
        # connect to socket
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.connect((host, port))
        # create a stream that we can use with the rest of the library
        self.stream = self.socket.makefile('rb', None)
```

A simple node

```
class SimpleNode:

    def send(self, message):
        '''Send a message to the connected node'''
        # create a network envelope
        envelope = NetworkEnvelope(
            message.command, message.serialize(), testnet=self.testnet)
        if self.logging:
            print('sending: {}'.format(envelope))
        # send the serialized envelope over the socket using sendall
        self.socket.sendall(envelope.serialize())
```

A simple node

```
class SimpleNode:

    def __init__(self, host, port=None, testnet=False, logging=False):
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket.connect((host, port))
        # create a stream that we can use with the rest of the library
        self.stream = self.socket.makefile('rb', None)
```

```
def read(self):
    '''Read a message from the socket'''
    envelope = NetworkEnvelope.parse(self.stream, testnet=self.testnet)
    if self.logging:
        print('receiving: {}'.format(envelope))
    return envelope
```

A simple node

```
class SimpleNode:
    def wait_for(self, *message_classes):
        '''Wait for one of the messages in the list'''
        # initialize the command we have, which should be None
        command = None
        command_to_class = {m.command: m for m in message_classes}
        # loop until the command is in the commands we want
        while command not in command_to_class.keys():
            # get the next network message
            envelope = self.read()
            # set the command to be evaluated
            command = envelope.command
            # we know how to respond to version and ping, handle that here
            if command == VersionMessage.command:
                # send verack
                self.send(VerAckMessage())
            elif command == PingMessage.command:
                # send pong
                self.send(PongMessage(envelope.payload))
        # return the envelope parsed as a member of the right message class
        return command_to_class[command].parse(envelope.stream())
```

A simple node

Ping pong

```
class PingMessage:
    command = b'ping'

    def __init__(self, nonce):
        self.nonce = nonce

    @classmethod
    def parse(cls, s):
        nonce = s.read(8)
        return cls(nonce)

    def serialize(self):
        return self.nonce


class PongMessage:
    command = b'pong'

    def __init__(self, nonce):
        self.nonce = nonce

    def parse(cls, s):
        nonce = s.read(8)
        return cls(nonce)

    def serialize(self):
        return self.nonce
```

A simple node

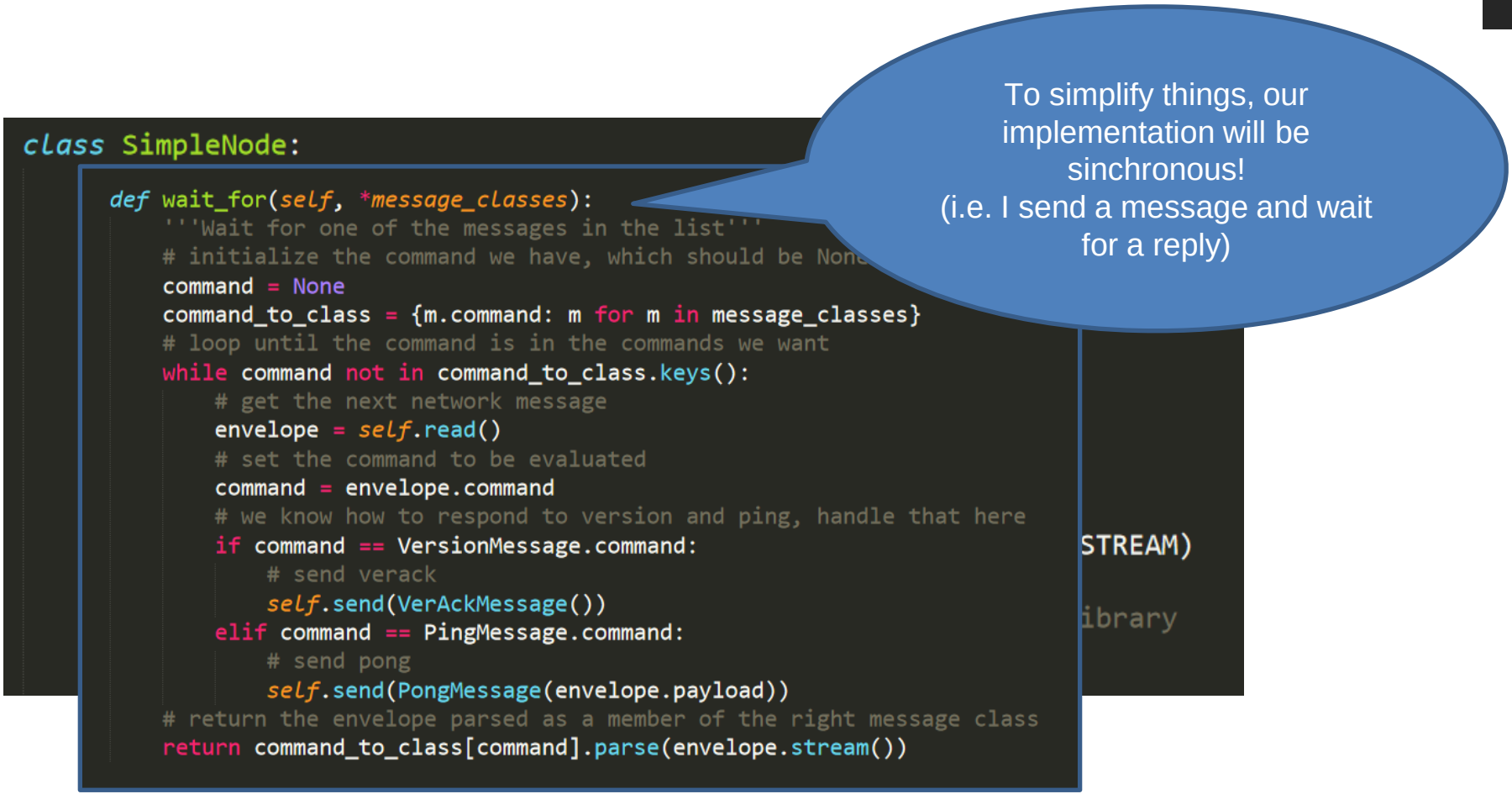
```
class SimpleNode:
    def wait_for(self, *message_classes):
        '''Wait for one of the messages in the list'''
        # initialize the command we have, which should be None
        command = None
        command_to_class = {m.command: m for m in message_classes}
        # loop until the command is in the commands we want
        while command not in command_to_class.keys():
            # get the next network message
            envelope = self.read()
            # set the command to be evaluated
            command = envelope.command
            # we know how to respond to version and ping, handle that here
            if command == VersionMessage.command:
                # send verack
                self.send(VerAckMessage())
            elif command == PingMessage.command:
                # send pong
                self.send(PongMessage(envelope.payload))
        # return the envelope parsed as a member of the right message class
        return command_to_class[command].parse(envelope.stream())
```

To simplify things, our implementation will be sinchronous!
(i.e. I send a message and wait for a reply)

A simple node

```
class SimpleNode:
    def wait_for(self, *message_classes):
        '''Wait for one of the messages in the list'''
        # initialize the command we have, which should be None
        command = None
        command_to_class = {m.command: m for m in message_classes}
        # loop until the command is in the commands we want
        while command not in command_to_class.keys():
            # get the next network message
            envelope = self.read()
            # set the command to be evaluated
            command = envelope.command
            # we know how to respond to version and ping, handle that here
            if command == VersionMessage.command:
                # send verack
                self.send(VerAckMessage())
            elif command == PingMessage.command:
                # send pong
                self.send(PongMessage(envelope.payload))
        # return the envelope parsed as a member of the right message class
        return command_to_class[command].parse(envelope.stream())
```

A simple node



A simple node

```
class SimpleNode:
    def wait_for(self, *message_classes):
        '''Wait for one of the messages in the list'''
        # initialize the command we have, which should be None
        command = None
        command_to_class = {m.command: m for m in message_classes}
        # loop until the command is in the commands we want
        while command not in command_to_class.keys():
            # get the next network message
            envelope = self.read()
            # set the command to be evaluated
            command = envelope.command
            # we know how to respond to version and p
            if command == VersionMessage.command:
                # send verack
                self.send(VerAckMessage())
            elif command == PingMessage.command:
                # send pong
                self.send(PongMessage(envelope.payload))
        # return the envelope parsed as a member of the right message class
        return command_to_class[command].parse(envelope.stream())
```

To simplify things, our implementation will be sinchronous!
(i.e. I send a message and wait for a reply)

Real nodes communicate in asynchronous manner!

A simple node

```
class SimpleNode:

    def handshake(self):
        '''Do a handshake with the other node.
        Handshake is sending a version message and getting a verack back.'''
        # create a version message
        version = VersionMessage()
        # send the command
        self.send(version)
        # wait for a verack message
        self.wait_for(VerAckMessage)
```

A simple node

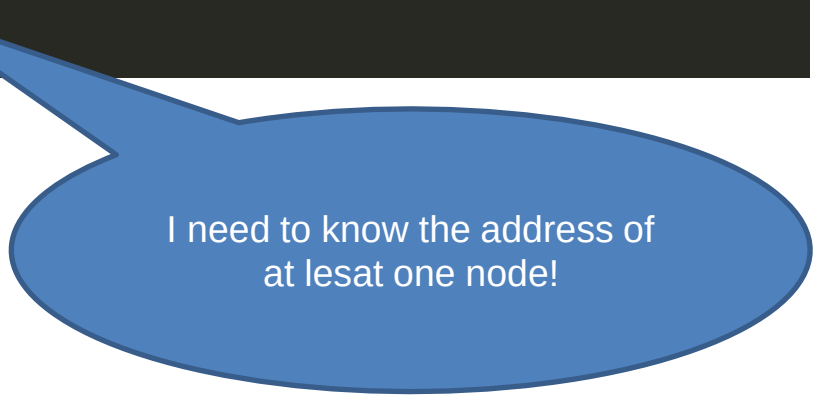
```
class SimpleNode:

    def handshake(self):
        '''Do a handshake with the other node.
        Handshake is sending a version message and getting a verack back.'''
        # create a version message
        version = VersionMessage()
        # send the command
        self.send(version)
        # wait for a verack message
        self.wait_for(VerAckMessage)
```

This is how we start

A simple node

Initial connection

```
# Establish a connection to a testnet node
node = SimpleNode('testnet.programmingbitcoin.com', testnet=True)
node.handshake()
```



Bitcoin network

SPV nodes

If I am an SPV node, I need the blockchain of headers!

- ~~Version~~
- ~~Verack~~
- GetHeaders
- Headers

Bitcoin network

GetHeaders

With **getheaders** I can obtain headers (max 2000!)

Payload:

```
7f11010001a35bd0ca2f4a88c4eda6d213e2378a5758dfcd6af437120000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

- 7f110100 - Protocol version, 4 bytes, little-endian, 70015
- 01 - Number of hashes, varint
- a35b...00 - Starting block, little-endian
- 0000...00 - Ending block, little-endian

Bitcoin network

GetHeaders

With **getheaders** I can obtain headers (max

Payload:

```
7f11010001a35bd0ca2f4a88c4eda6d213e2378a575
0000000000000000000000000000000000000000000000000000000000000000
```

- 7f110100 - Protocol version, 4 bytes, little-endian, 70015
- 01 - Number of hashes, varint
- a35b...00 - Starting block, little-endian
- 0000...00 - Ending block, little-endian

In reality, I can start from multiple blocks (e.g. for forks)

Bitcoin network

GetHeaders

With **getheaders** I can obtain headers (ma

Payload:

```
7f11010001a35bd0ca2f4a88c4eda6d213e2378a5758dfcd6af                    0000000000000000
0000000000000000000000000000000000000000000000000
```

- 7f110100 - Protocol version, 4 by , litt
- 01 - Number of hashes, varint
- a35b...00 - Starting block, little-endian
- 0000...00 - Ending block, little-endian

If this is 0, we should receive **up to** the next 2000 blocks

Depends on the node we connect to!

Bitcoin network

GetHeaders

```
class GetHeadersMessage:
    command = b'getheaders'

    def __init__(self, version=70015, num_hashes=1, start_block=None, end_block=None):
        self.version = version
        self.num_hashes = num_hashes
        if start_block is None:
            raise RuntimeError('a start block is required')
        self.start_block = start_block
        if end_block is None:
            self.end_block = b'\x00' * 32
        else:
            self.end_block = end_block

    def serialize(self):
        '''Serialize this message to send over the network'''
        # protocol version is 4 bytes little-endian
        result = int_to_little_endian(self.version, 4)
        # number of hashes is a varint
        result += encode_varint(self.num_hashes)
        # start block is in little-endian
        result += self.start_block[::-1]
        # end block is also in little-endian
        result += self.end_block[::-1]
        return result
```

Bitcoin network

headers

Reply to **getheaders** is **headers**

Payload:

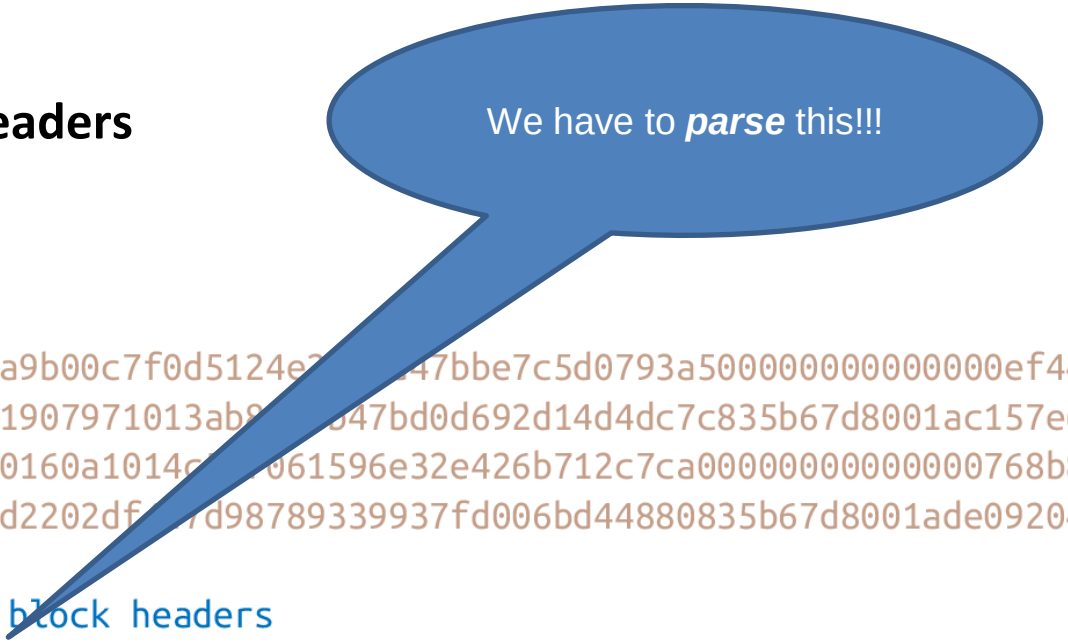
```
02000000020df3b053dc46f162a9b00c7f0d5124e2676d47bbe7c5d0793a500000000000000ef445fef
2ed495c275892206ca533e7411907971013ab83e3b47bd0d692d14d4dc7c835b67d8001ac157e6700
0000000203 0eb2540c41025690160a1014c577061596e32e426b712c7ca00000000000000768b89f07
044e6130ead292a3f51951adbd2202df447d98789339937fd006bd44880835b67d8001ade0920460 0
```

- 02 - Number of block headers
- 00...67 - Block header
- 00 - Number of transactions (always 0)

Bitcoin network

headers

Reply to **getheaders** is **headers**



Payload:

```
02000000020df3b053dc46f162a9b00c7f0d5124e2[illegible]47bbe7c5d0793a500000000000000ef445fef
2ed495c275892206ca533e7411907971013ab8[illegible]547bd0d692d14d4dc7c835b67d8001ac157e6700
00000002030eb2540c41025690160a1014c[illegible]061596e32e426b712c7ca00000000000000768b89f07
044e6130ead292a3f51951adbd2202df[illegible]7d98789339937fd006bd44880835b67d8001ade0920460 0
```

- 02 - Number of block headers
- 00...67 - Block header
- 00 - Number of transactions (always 0)

Block headers

Block Headers

Block headers are sent in a headers packet in response to a getheaders message.

Field Size	Description	Data type	Comments
4	version	int32_t	Block version information (note, this is signed)
32	prev_block	char[32]	The hash value of the previous block this particular block references
32	merkle_root	char[32]	The reference to a Merkle tree collection which is a hash of all transactions related to this block
4	timestamp	uint32_t	A timestamp recording when this block was created (Will overflow in 2106[2])
4	bits	uint32_t	The calculated difficulty target being used for this block
4	nonce	uint32_t	The nonce used to generate this block… to allow variations of the header and compute different hashes
1+	txn_count	var_int	Number of transaction entries, this value is always 0

https://en.bitcoin.it/wiki/Protocol_documentation

Block headers

Our implementation checks this in the payload of the header message!

Block Headers

Block headers are sent in a headers packet in response to a getheaders message.

Field Size	Description	Data type	Comments
4	version	int32_t	Block version information (p… s signed)
32	prev_block	char[32]	The hash value of th… ous block this particular block references
32	merkle_root	char[32]	The referen… d Merkle tree collection which is a hash of all transactions related to this block
4	timestamp	uint32_t	A ti… stamp recording when this block was created (Will overflow in 2106[2])
4	bits	uint32_t	The calculated difficulty target being used for this block
4	nonce	uint32_t	The nonce used to generate this block… to allow variations of the header and compute different hashes
1+	txn_count	var_int	Number of transaction entries, this value is always 0

https://en.bitcoin.it/wiki/Protocol_documentation

Block headers

In bytes

```
020000208ec39428b17323fa0ddec8e887b4a7c53b8c0a0
a220cfd0000000000000000005b0750fce0a889502d4050
8d39576821155e9c9e3f5c3157f961db38fd8b25be1e77a
759e93c0118a4ffd71d
```

- 02000020 - version, 4 bytes, LE
- 8ec3...00 - previous block, 32 bytes, LE
- 5b07...be - merkle root, 32 bytes, LE
- 1e77a759 - timestamp, 4 bytes, LE
- e93c0118 - bits, 4 bytes
- a4ffd71d - nonce, 4 bytes

https://github.com/jimmysong/programmingbitcoin/blob/master/ch09.asciidoc

Block headers

In bytes

```
020000208ec39428b17323fa0ddec8e887b4a7c53b8c0a0
a220cfd0000000000000000005b0750fce0a889502d4050
8d39576821155e9c9e3f5c3157f961db38fd8b25be1e77a
759e93c0118a4ffd71d
```

- 02000020 - version, 4 bytes, LE
- 8ec3...00 - previous block, 32 bytes, LE
- 5b07...be - merkle root, 32 bytes, LE
- 1e77a759 - timestamp, 4 bytes, LE
- e93c0118 - bits, 4 bytes
- a4ffd71d - nonce, 4 bytes

https://github.com/jimmysong/programmingbitcoin/blob/master/ch09.asciidoc

Block headers

In Python

```
class Block:

    def __init__(self, version, prev_block, merkle_root,
                 timestamp, bits, nonce):
        self.version = version
        self.prev_block = prev_block
        self.merkle_root = merkle_root
        self.timestamp = timestamp
        self.bits = bits
        self.nonce = nonce
```

https://github.com/jimmysong/programmingbitcoin/blob/master/ch09.asciidoc

Block headers

In Python

```
class Block:

    @classmethod
    def parse(cls, s):
        '''Takes a byte stream and parses a block. Returns a Block object'''
        # s.read(n) will read n bytes from the stream
        # version - 4 bytes, little endian, interpret as int
        version = little_endian_to_int(s.read(4))
        # prev_block - 32 bytes, little endian (use [::-1] to reverse)
        prev_block = s.read(32)[::-1]
        # merkle_root - 32 bytes, little endian (use [::-1] to reverse)
        merkle_root = s.read(32)[::-1]
        # timestamp - 4 bytes, little endian, interpret as int
        timestamp = little_endian_to_int(s.read(4))
        # bits - 4 bytes
        bits = s.read(4)
        # nonce - 4 bytes
        nonce = s.read(4)
        # initialize class
        return cls(version, prev_block, merkle_root, timestamp, bits, nonce)
```

https://github.com/jimmysong/programmingbitcoin/blob/master/ch09.asciidoc

Block headers

In Python

```
class Block:

    def serialize(self):
        '''Returns the 80 byte block header'''
        # version - 4 bytes, little endian
        result = int_to_little_endian(self.version, 4)
        # prev_block - 32 bytes, little endian
        result += self.prev_block[::-1]
        # merkle_root - 32 bytes, little endian
        result += self.merkle_root[::-1]
        # timestamp - 4 bytes, little endian
        result += int_to_little_endian(self.timestamp, 4)
        # bits - 4 bytes
        result += self.bits
        # nonce - 4 bytes
        result += self.nonce
        return result
```

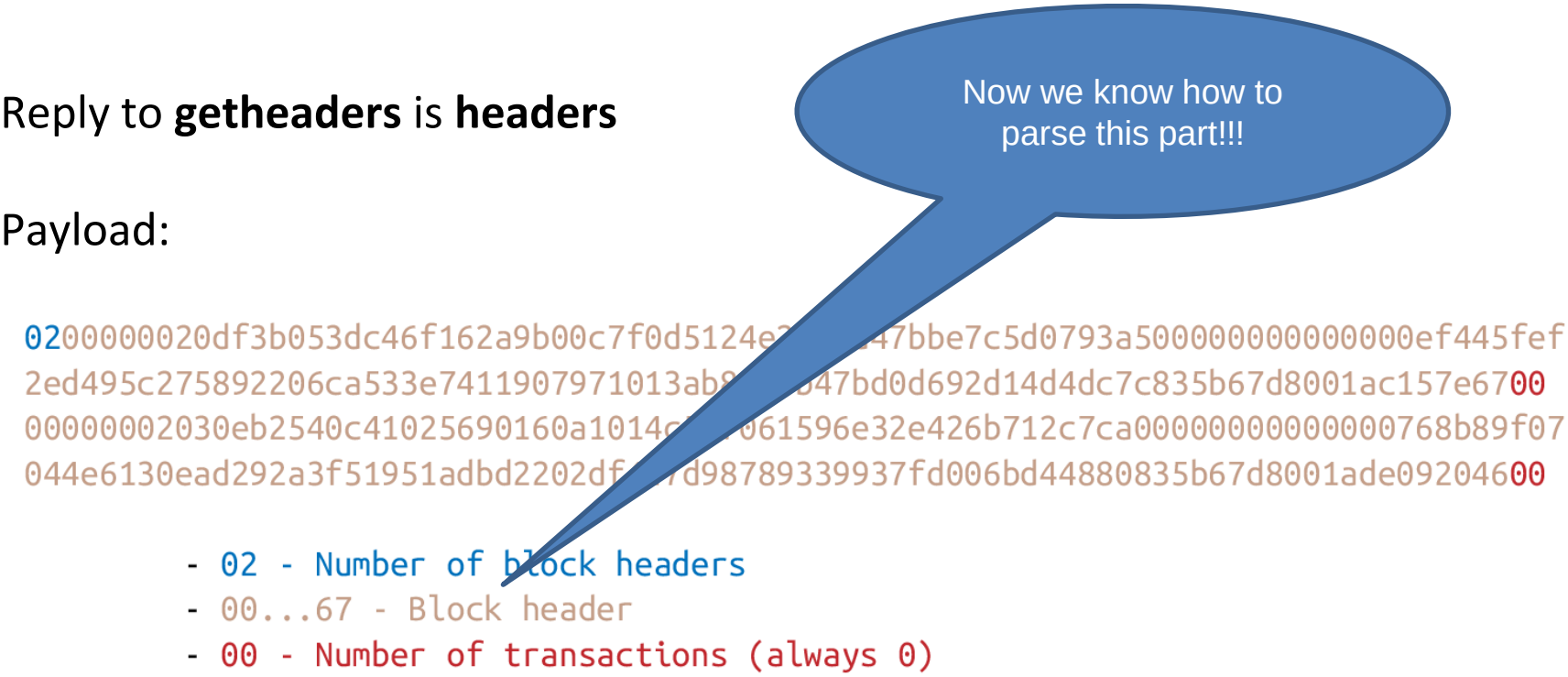
https://github.com/jimmysong/programmingbitcoin/blob/master/ch09.asciidoc

Bitcoin network

headers

Reply to **getheaders** is **headers**

Payload:



Block headers

We will get headers message

```
class HeadersMessage:
    command = b'headers'

    def __init__(self, blocks):
        self.blocks = blocks

    @classmethod
    def parse(cls, stream):
        # number of headers is in a varint
        num_headers = read_varint(stream)
        # initialize the blocks array
        blocks = []
        # loop through number of headers times
        for _ in range(num_headers):
            # add a block to the blocks array by parsing the stream
            blocks.append(Block.parse(stream))
            # read the next varint (num_txs)
            num_txs = read_varint(stream)
            # num_txs should be 0 or raise a RuntimeError
            if num_txs != 0:
                raise RuntimeError('number of txs not 0')
        # return a class instance
        return cls(blocks)
```

Block headers

We will get headers message

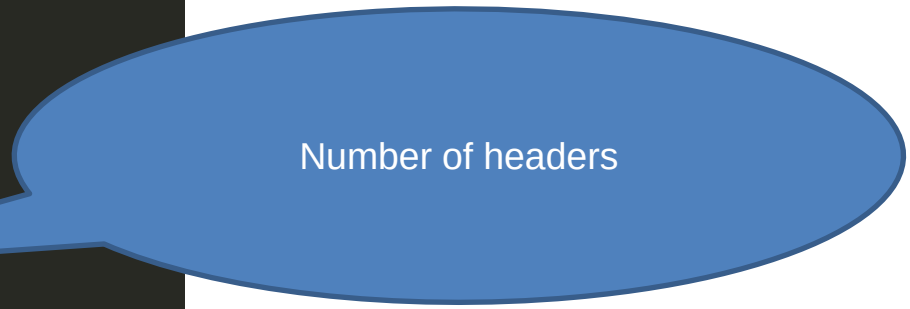
```
class HeadersMessage:
    command = b'headers'

    def __init__(self, blocks):
        self.blocks = blocks

    @classmethod
    def parse(cls, stream):
        # number of headers is in a varint
        num_headers = read_varint(stream)
        # initialize the blocks array
        blocks = []
        # loop through number of headers times
        for _ in range(num_headers):
            # add a block to the blocks array by parsing the stream
            blocks.append(Block.parse(stream))
            # read the next varint (num_txs)
            num_txs = read_varint(stream)
            # num_txs should be 0 or raise a RuntimeError
            if num_txs != 0:
                raise RuntimeError('number of txs not 0')
        # return a class instance
        return cls(blocks)
```

Blockchain of headers

Block headers

We will get headers message

```
class HeadersMessage:
    command = b'headers'

    def __init__(self, blocks):
        self.blocks = blocks

    @classmethod
    def parse(cls, stream):
        # number of headers is in a varint
        num_headers = read_varint(stream)
        # initialize the blocks array
        blocks = []
        # loop through number of headers times
        for _ in range(num_headers):
            # add a block to the blocks array by parsing the stream
            blocks.append(Block.parse(stream))
            # read the next varint (num_txs)
            num_txs = read_varint(stream)
            # num_txs should be 0 or raise a RuntimeError
            if num_txs != 0:
                raise RuntimeError('number of txs not 0')
        # return a class instance
        return cls(blocks)
```



Block headers

We will get headers message

```
class HeadersMessage:
    command = b'headers'

    def __init__(self, blocks):
        self.blocks = blocks

    @classmethod
    def parse(cls, stream):
        # number of headers is in a varint
        num_headers = read_varint(stream)
        # initialize the blocks array
        blocks = []
        # loop through number of headers times
        for _ in range(num_headers):
            # add a block to the blocks array by parsing the stream
            blocks.append(Block.parse(stream))
            # read the next varint (num_txs)
            num_txs = read_varint(stream)
            # num_txs should be 0 or raise a RuntimeError
            if num_txs != 0:
                raise RuntimeError('number of txs not 0')
        # return a class instance
        return cls(blocks)
```

Headers (we will store this as objects of the class block)

Block headers

We will get headers message

```
class HeadersMessage:
    command = b'headers'

    def __init__(self, blocks):
        self.blocks = blocks

    @classmethod
    def parse(cls, stream):
        # number of headers is in a varint
        num_headers = read_varint(stream)
        # initialize the blocks array
        blocks = []
        # loop through number of headers times
        for _ in range(num_headers):
            # add a block to the blocks array by parsing the stream
            blocks.append(Block.parse(stream))
            # read the next varint (num_txs)
            num_txs = read_varint(stream)
            # num_txs should be 0 or raise a RuntimeError
            if num_txs != 0:
                raise RuntimeError('number of txs not 0')
        # return a class instance
        return cls(blocks)
```

Number of transactions needs to be 0 (for a header)!

Block headers

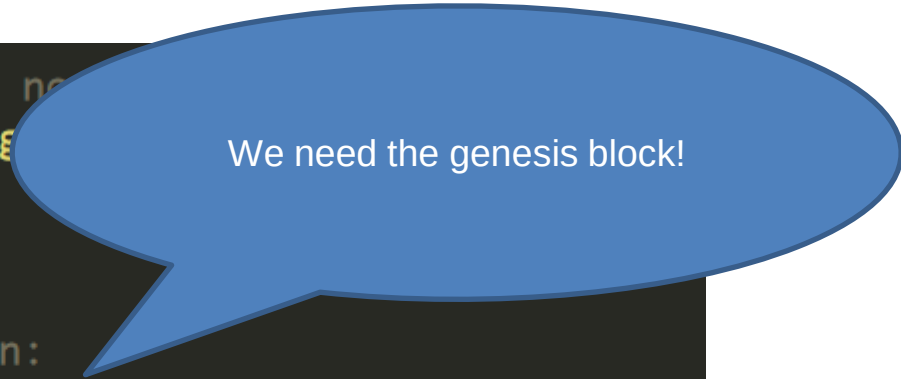
How to get the first 2000 blocks?

```
# Establish a connection to a testnet node
node = SimpleNode('testnet.programmingbitcoin.com', testnet=True)
node.handshake()


# Get the first 2000 blocks of Bitcoin:
previous = Block.parse(BytesIO(GENESIS_BLOCK))
getheaders = GetHeadersMessage(start_block = previous.hash())
node.send(getheaders)
headers = node.wait_for(HeadersMessage)

for x in headers.blocks:
    print(x.hash().hex(),x.prev_block.hex())
```

Block headers

How to get the first 2000 blocks?

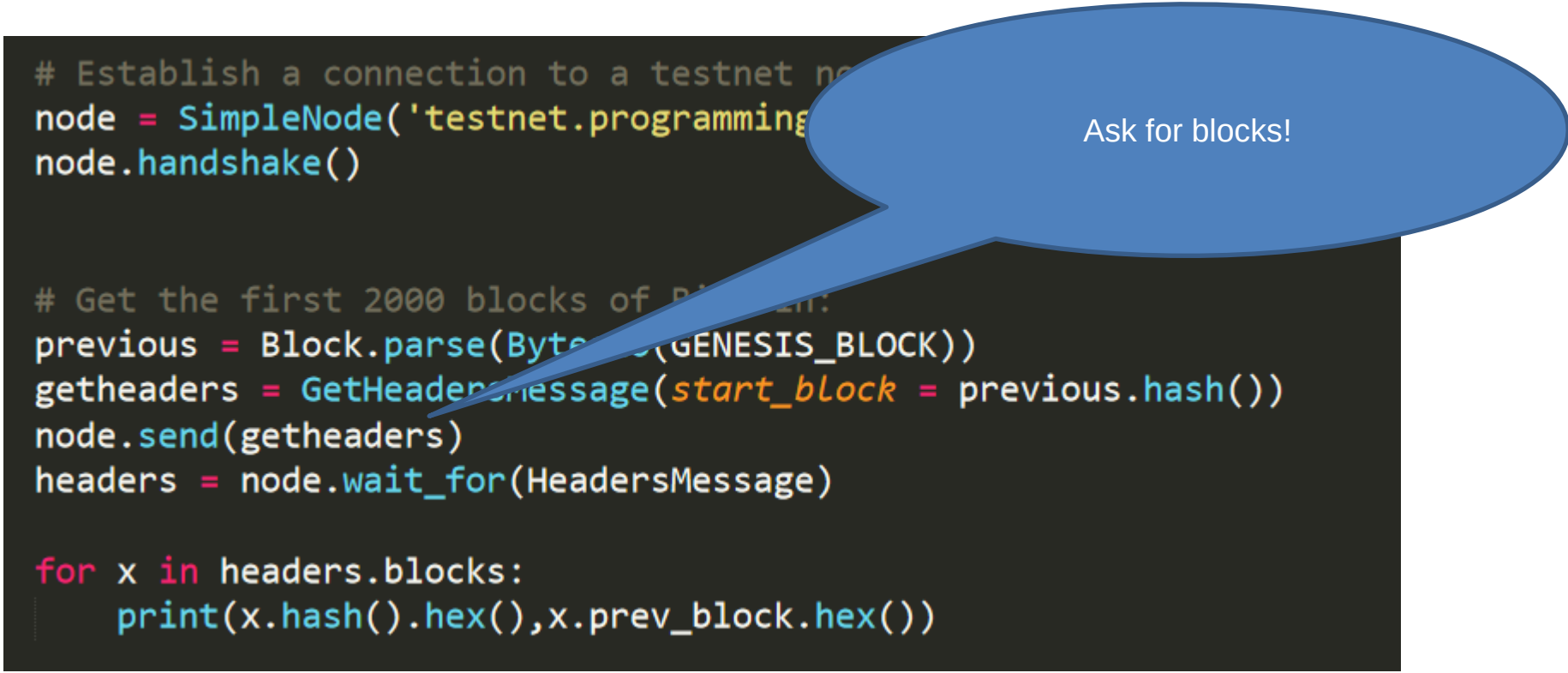
```
# Establish a connection to a testnet no
node = SimpleNode('testnet.programming
node.handshake()


# Get the first 2000 blocks of Bitcoin:
previous = Block.parse(BytesIO(GENESIS_BLOCK))
getheaders = GetHeadersMessage(start_block = previous.hash())
node.send(getheaders)
headers = node.wait_for(HeadersMessage)

for x in headers.blocks:
    print(x.hash().hex(),x.prev_block.hex())
```



Block headers

How to get the first 2000 blocks?

```
# Establish a connection to a testnet no
node = SimpleNode('testnet.programming
node.handshake()


# Get the first 2000 blocks of Bi      in:
previous = Block.parse(Byte      (GENESIS_BLOCK))
getheaders = GetHeade      essage(start_block = previous.hash())
node.send(getheaders)
headers = node.wait_for(HeadersMessage)

for x in headers.blocks:
    print(x.hash().hex(),x.prev_block.hex())
```



Block headers

How to get the first 2000 blocks?

```
# Establish a connection to a testnet no
node = SimpleNode('testnet.programming
node.handshake()


# Get the first 2000 blocks of Bitco
previous = Block.parse(BytesIO(GE      IS_BLOCK))
getheaders = GetHeadersMessa      (start_block = previous.hash())
node.send(getheaders)
headers = node.wait_for(HeadersMessage)

for x in headers.blocks:
    print(x.hash().hex(),x.prev_block.hex())
```

Wait to recieve the blocks

Proof of work

Allos to check if POW is valid!

(do not confuse with bits of a Merkle tree)

Block Headers

Block headers are sent in a headers

Field Size	Description	Data type	
4	version	int32_t	Block versi n (note, this is signed)
32	prev_block	char[32]	The hash of the previous block this particular block references
32	merkle_root	char[32]	The ref ence to a Merkle tree collection which is a hash of all transactions related to this block
4	timestamp	uint32_t	A timestamp recording when this block was created (Will overflow in 2106[2])
4	bits	uint32_t	The calculated difficulty target being used for this block
4	nonce	uint32_t	The nonce used to generate this block... to allow variations of the header and compute different hashes
1+	txn_count	var_int	Number of transaction entries, this value is always 0

Proof of work

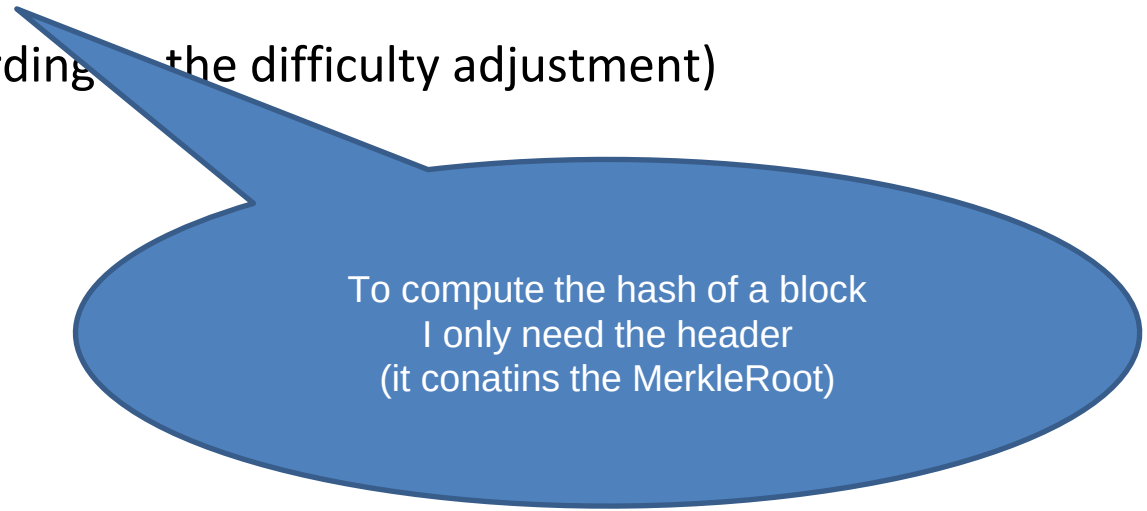
How to check that POW of a block is valid?

1. hash(header) < target
2. bits are correct (according to the difficulty adjustment)

Proof of work

How to check that POW of a block is valid?

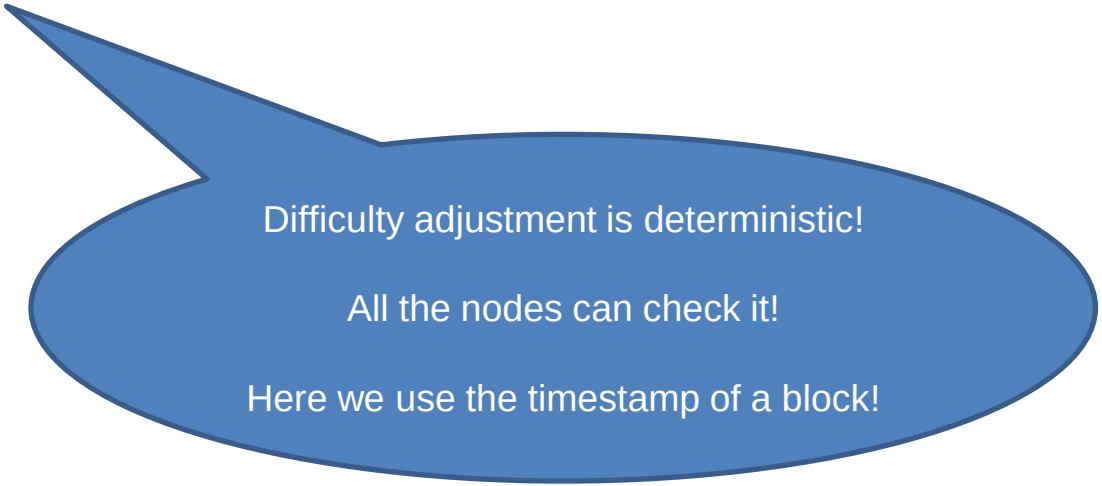
1. hash(header) < target
2. bits are correct (according to the difficulty adjustment)



Proof of work

How to check that POW of a block is valid?

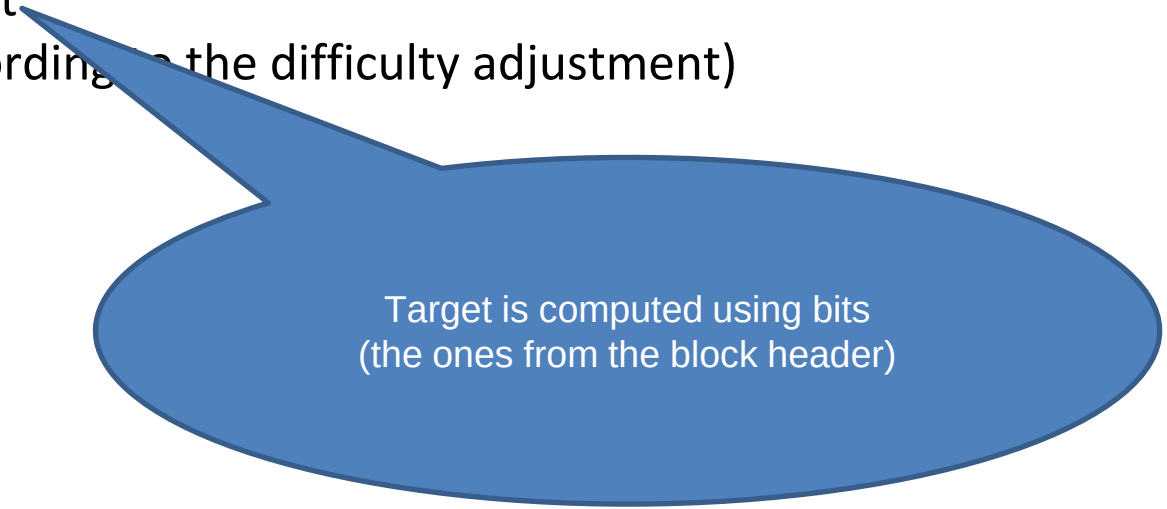
1. hash(header) < target
2. bits are correct (according to the difficulty adjustment)



Proof of work

How to check that POW of a block is valid?

1. hash(header) < target
2. bits are correct (according to the difficulty adjustment)



Proof of work

Target

How to computetarget?

- **Usando bits = 4 bytes**

bits are two different numbers:

1. **Byte 4 is the exponent: exp**
2. **Bytes 1,2,3 are the coefficient: coeff (in Little-endian)**

target = coeff · 256^{exp-3}

Proof of work

Target

$$\text{target} = \text{coeff} \cdot 256^{\text{exp}-3}$$

```python
def bits_to_target(bits):
    '''Turns bits into a target (large 256-bit integer)'''
    # last byte is exponent
    exponent = bits[-1]
    # the first three bytes are the coefficient in little endian
    coefficient = little_endian_to_int(bits[:-1])
    # the formula is:
    # coefficient * 256**(exponent-3)
    return coefficient * 256**(exponent - 3)
```

Function implemented in helper.py

Proof of work

How to check that POW of a block is valid?

1. hash(header) < target
2. bits are correct (according to the difficulty adjustment)

Proof of work

```
class Block:

    def __init__(self, version, prev_block, merkle_root,
                 timestamp, bits, nonce):
        self.version = version
        self.prev_block = prev_block
        self.merkle_root = merkle_root
        self.timestamp = timestamp
        self.bits = bits
        self.nonce = nonce
```

Proof of work

```
class Block:

    def serialize(self):
        '''Returns the 80 byte block header'''
        # version - 4 bytes, little endian
        result = int_to_little_endian(self.version, 4)
        # prev_block - 32 bytes, little endian
        result += self.prev_block[::-1]
        # merkle_root - 32 bytes, little endian
        result += self.merkle_root[::-1]
        # timestamp - 4 bytes, little endian
        result += int_to_little_endian(self.timestamp, 4)
        # bits - 4 bytes
        result += self.bits
        # nonce - 4 bytes
        result += self.nonce
        return result
```

Block header already has everything needed!!!

Proof of work

```
class Block:

    def hash(self):
        '''Returns the hash256 interpreted little endian of the block'''
        # serialize
        s = self.serialize()
        # hash256
        h256 = hash256(s)
        # reverse
        return h256[::-1]
```

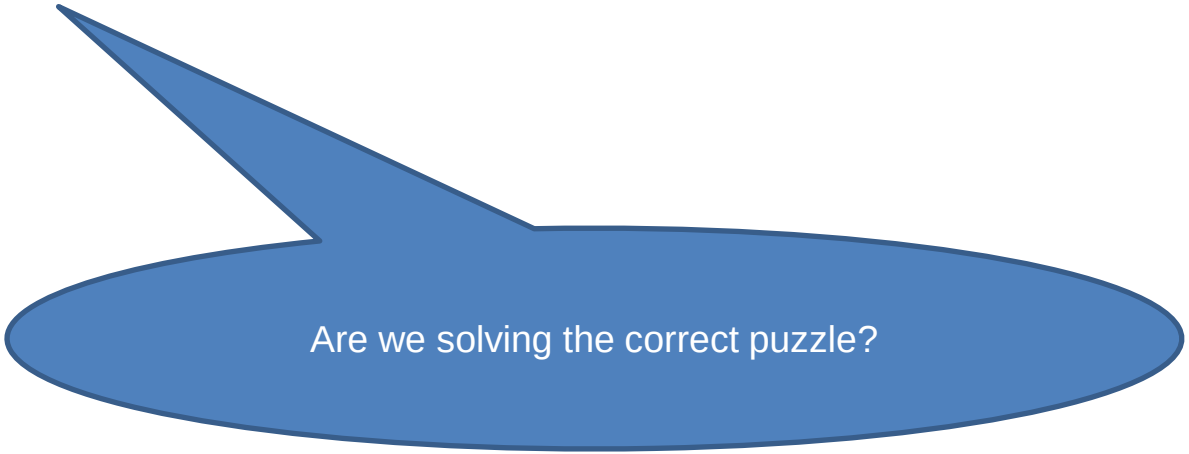
Proof of work

```
class Block:

    def check_pow(self):
        '''Returns whether this block satisfies proof of work'''
        # get the hash256 of the serialization of this block
        h256 = hash256(self.serialize())
        # interpret this hash as a little-endian number
        proof = little_endian_to_int(h256)
        # return whether this integer is less than the target
        return proof < self.target()
```

Proof of work

How to check that POW of a block is valid?

1. hash(header) < target
2. bits are correct (according to the difficulty adjustment)



Proof of work

Difficulty adjustment

bits are correct (according the difficulty adjustment)

- Each group of 2016 blocks in Bitcoin is called **difficulty adjustment period**
- **Target** is adjusted according to the formula:

time_diff = (timestamp of the last block) – (timestamp of the first block)

new_target = prev_target ·time_diff/(2 weeks)

Proof of work

...fficulty adjustment

E.g. for the period of blocks: 2017 until 4032
new_target is computed using the difference
between blocks 2016 and 1

bits are corre...

- Each group of 201... ... **difficulty adjustment period**
- **Target** is adjusted ...ding to the formula:

time_diff = (timestamp of the last block) – (timestamp of the first block)

new_target = prev_target ·time_diff/(2 weeks)

Proof of work

...fficulty adjustment

E.g. for the period of blocks: 2017 until 4032
This is the target fo the period 1 - 2016

bits are corre...

- Each group of 2016 ... **difficulty adjustment period**
- **Target** is adjusted a... to the formula:

time_diff = (timesta... of the last block) – (timestamp of the first block)

new_target = prev_target ·time_diff/(2 weeks)

Proof of work

...fficulty adjustment

Remark:
If time_diff>8 weeks we will use 8 weeks
If time_diff<3.5 days we will use 3.5 days

bits are corre

- Each group of 201... **difficulty adjustment period**
- **Target** is adjusted ...ding to the formula:

time_diff = (timestamp of the last block) – (timestamp of the first block)

new_target = prev_target ·time_diff/(2 weeks)

Proof of work

...fficulty adjustment

Remark:
Another one-off bug from Satoshi: the difference is between 2015 blocks, so not precisely 2 weeks!

bits are corre...

- Each group of 201... **...difficulty adjustment period**
- **Target** is adjusted ...ding to the formula:

time_diff = (timestamp of the last block) – (timestamp of the first block)

new_target = prev_target ·time_diff/(2 weeks)

Proof of work

Difficulty adjustment

```
def calculate_new_bits(previous_bits, time_differential):
    '''Calculates the new bits given
    a 2016-block time differential and the previous bits'''
    # if the time differential is greater than 8 weeks, set to 8 weeks
    if time_differential > TWO_WEEKS * 4:
        time_differential = TWO_WEEKS * 4
    # if the time differential is less than half a week, set to half a week
    if time_differential < TWO_WEEKS // 4:
        time_differential = TWO_WEEKS // 4
    # the new target is the previous target * time differential / two weeks
    new_target = bits_to_target(previous_bits) * time_differential // TWO_WEEKS
    # if the new target is bigger than MAX_TARGET, set to MAX_TARGET
    if new_target > MAX_TARGET:
        new_target = MAX_TARGET
    # convert the new target to bits
    return target_to_bits(new_target)
```

Function implemented in helper.py

Proof of work

...fficulty adjustment

Target of the genesys block; it can never be lower than this!

```
def calc
    '''Cal
    a 2016-bloc
    # if the time differen                    8 weeks, set to 8 weeks
    if time_differential >                  * 4:
        time_differential                   EKS * 4
    # if the time differe                   s less than half a week, set to half a week
    if time_differential                    WEEKS // 4:
        time_differentia                    WO_WEEKS // 4
    # the new target is                     previous target * time differential / two weeks
    new_target = bits_t                     arget(previous_bits) * time_differential // TWO_WEEKS
    # if the new target is bigger than MAX_TARGET, set to MAX_TARGET
    if new_target > MAX_TARGET:
        new_target = MAX_TARGET
    # convert the new target to bits
    return target_to_bits(new_target)
```

Function implemented in helper.py

Proof of work

Difficulty adjustment

```python
def target_to_bits(target):
    '''Turns a target integer back into bits, which is 4 bytes'''
    raw_bytes = target.to_bytes(32, 'big')
    # get rid of leading 0's
    raw_bytes = raw_bytes.lstrip(b'\x00')
    if raw_bytes[0] > 0x7f:
        # if the first bit is 1, we have to start with 00
        exponent = len(raw_bytes) + 1
        coefficient = b'\x00' + raw_bytes[:2]
    else:
        # otherwise, we can show the first 3 bytes
        # exponent is the number of digits in base-256
        exponent = len(raw_bytes)
        # coefficient is the first 3 digits of the base-256 number
        coefficient = raw_bytes[:3]
    # we've truncated the number after the first 3 digits of base-256
    new_bits = coefficient[::-1] + bytes([exponent])
    return new_bits
```

Function implemented in helper.py

Proof of work

With this, our implementation can download the block headers, and verify that they all satisfy the proof-of-work for this block!

As an exercise, you can use our implementation to verify this!

References

You can read more in:

- Programming Bitcoin, chapters 9,10
- https://en.bitcoin.it/wiki/Protocol_documentation