**CMPE 208: Network Architecture and Protocols**

**Bitcoin Protocol Analysis**

Submitted to

Prof. Shai Silberman

Team 5

Vaishnavi Reddipalli – 010826088

Susheel KB Kaparaboyna - 010826179

Ashok Aratikatla - 010805639

Goutham Nekkalapu - 010815233

**Contribution of team members:**

Vaishnavi Reddipalli – Analyzing Bitcoin Protocol, Lab execution, Message types, Protocol components

Susheel KB Kaparaboyna – Lab execution, Message types, Documentation, Attacks

Ashok Aratikatla – Lab setup, Packet capture, Analyzing Bitcoin Protocol, Findings

Goutham Nekkalapu – Documentation, Overview, Lab execution

**Overview:**

**Bitcoin Core Introduction:**

Bitcoin is a cryptocurrency and a payment systems invented by an unidentified programmer, or group of programmers, under the name of Satoshi Nakamoto in 2008. It was released as an open source software in 2009. The system is peer-to-peer and transactions take place between users directly, without an intermediary ( a banking system, government organizations ). These transactions are verified by network nodes and recorded in a public distributed ledger called blockchain, which uses bitcoin as its unit of account. Since the system works without a central repository or single administrator it is considered as a decentralized virtual currency in contrast to a centralized currency system we have now being operated by banks and government institutions. Often bitcoin is considered as a first crypto and digital currency.

In this report we will see bitcoin terminology, how bitcoins are generated, about bitcoin network, working of bitcoin protocol and protocol message formats. We will explain the transactions over the bitcoin network with respect to protocol using wireshark captures.

**Bitcoin Generation:**

Bitcoins are created as a reward in a competition in which users offer their computing power to verify and record bitcoin transactions into the block chain. This activity is referred to as mining and successful miners are rewarded with transaction fees and newly created bitcoins. Besides being obtained by mining, bitcoins can be exchanged for other currencies,products, and services. When sending bitcoins, users can pay an optional transaction fee to the miners. This may expedite the transaction being confirmed. Although it is used to be profitable to mine bitcoins with your standard personal computer, the cost of the electricity necessary to do so is now greater than the value of the bitcoins you could mine. Profitable mining now requires specialized hardware that can perform more computations with greater power efficiency.

As we mentioned above, there is no central person or central authority in charge of Bitcoin. As it is distributed and most of the users have to approve any changes make it difficult for anyone to manipulate Bitcoin. Various programmers donate their time developing the open source Bitcoin software. The individual miners then choose whether to install the new version of the software or stick to the old one, essentially “voting” with their processing power. It is in the mine’s best interest to only accept changes that are good for the Bitcoin currency in the long run.

**Pros and Cons using of Bitcoin:**

**Pros:**

1. Payment freedom:

It is possible to send and receive bitcoins anywhere in the world at any time. No bank holidays, no borders, no bureaucracy. Bitcoin allows its users to be in full control of their money.

2. Choose your own fees:

There is no fee to receive bitcoins, and many wallets let you control how large a fee to pay when spending. Higher fees can encourage faster confirmation of your transactions. Fees are unrelated to the amount transferred, so it's possible to send 100,000 bitcoins for the same fee it costs to send 1 bitcoin. Additionally, merchant processors exist to assist merchants in processing transactions, converting bitcoins to fiat currency and depositing funds directly into merchants' bank accounts daily.

3. Fewer risks for merchants:

Bitcoin transactions are secure, irreversible, and do not contain customers’ sensitive or personal information. This protects merchants from losses caused by fraud or fraudulent chargebacks.Merchants can easily expand to new markets where either credit cards are not available or fraud rates are unacceptably high. The net results are lower fees, larger markets, and fewer administrative costs.

4.Security and control:

Bitcoin users are in full control of their transactions; it is impossible for merchants to force unwanted or unnoticed charges as can happen with other payment methods. Bitcoin payments can be made without personal information tied to the transaction. This offers strong protection against identity theft. Bitcoin users can also protect their money with backup and encryption.

5.Transparent and neutral:

All information concerning the Bitcoin money supply itself is readily available on the block chain for anybody to verify and use in real-time. No individual or organization can control or manipulate the Bitcoin protocol because it is cryptographically secure. This allows the core of Bitcoin to be trusted for being completely neutral, transparent and predictable.

**Cons**:

1.Degree of acceptance:

Many people are still unaware of Bitcoin. Every day, more businesses accept bitcoins because they want the advantages of doing so, but the list remains small and still needs to grow in order to benefit from network effects.

2.Volatility:

The total value of bitcoins in circulation and the number of businesses using Bitcoin are still very small compared to what they could be. Therefore, relatively small events, trades, or business activities can significantly affect the price. In theory, this volatility will decrease as Bitcoin markets and the technology matures.

3.Ongoing development:

New tools, features, and services are being developed to make Bitcoin more secure and accessible to the masses. Some of these are still not ready for everyone. Most Bitcoin businesses are new and still offer no insurance. In general, Bitcoin is still in the process of maturing.

**Bitcoin Terminology:**

**Block Chain:**

The block chain is a public ledger that records bitcoin transactions. A novel solution accomplishes this without any trusted central authority: maintenance of the block chain is performed by a network of communicating nodes running bitcoin software. Network nodes can validate transactions, add them to their copy of the ledger, and then broadcast these ledger additions to other nodes. The block chain is a distributed database to achieve independent verification of the chain of ownership of any and every bitcoin (amount), each network node stores its own copy of the block chain. For every 10 minutes a new group of accepted transactions are put together, a block is created, added to the block chain, and quickly published to all nodes. This allows bitcoin software to determine when a particular bitcoin amount has been spent, which is necessary in order to prevent double spending in an environment without central oversight. Whereas a conventional ledger records the transfers of actual bills or promissory notes that exist apart from it, the block chain is the only place that bitcoins can be said to exist in the form of unspent outputs of transactions.

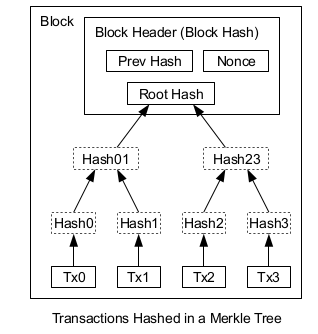


Fig: Block

The above image represents a block of a block chain. You observe various operations done for the formation of a block.

**Ownership:**

Ownership of bitcoins implies that a user can spend bitcoins associated with a specific address. To do so, a payer must digitally sign the transaction using the corresponding private key. Without knowledge of the private key, the transaction cannot be signed and bitcoins cannot be spent. The network verifies the signature using the public key.

**Transactions:**

A transaction must have one or more inputs. For the transaction to be valid, every input must be an unspent output of a previous transaction. Every input must be digitally signed. The use of multiple inputs corresponds to the use of multiple coins in a cash transaction. A transaction can also have multiple outputs, allowing one to make multiple payments in one go. A transaction output can be specified as an arbitrary multiple of satoshi. As in a cash transaction, the sum of inputs (coins used to pay) can exceed the intended sum of payments. In such a case, an additional output is used, returning the change back to the payer. Any input satoshis not accounted for in the transaction outputs become the transaction fee.

The below image depicts the transaction over bitcoin network and various operations involved in it.

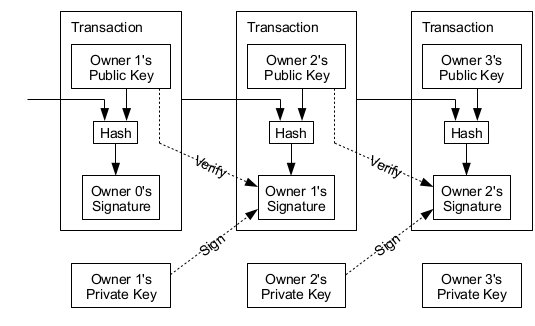


Fig: Bitcoin transaction

**Hashes:**

Usually, when a hash is computed within bitcoin, it is computed twice. Most of the time SHA-256 hashes are used.

### 

### Merkle Trees:

### Merkle trees are binary trees of hashes. Merkle trees in bitcoin use a double SHA-256, the SHA-256 hash of the SHA-256 hash of something. If, when forming a row in the tree (other than the root ), it would have an odd number of elements, the final double-hash is duplicated to ensure that the row has an even number of hashes. First form the bottom row of the tree with the ordered double-SHA-256 hashes of the byte streams of the transactions in the block. Then the row above it consists of half that number of hashes. Each entry is the double-SHA-256 of the 64-byte concatenation of the corresponding two hashes below it in the tree.This procedure repeats recursively until we reach a row consisting of just a single double-hash. This is the Merkle root of the tree.

**Signatures:**

Bitcoin uses Elliptic Curve Digital Signature Algorithm (ECDSA) to sign transactions.

Public keys (in scripts) are given as 04 <x> <y>, where x and y are 32 byte big-endian integers representing the coordinates of a point on the curve or in compressed form given as <sign> <x> where <sign> is 0x02 if y is even and 0x03 if y is odd.

**Signature Verification and transfer of Currency in a bitcoin network**

Transactions are cryptographically signed records that reassign ownership of Bitcoins to new addresses. Transactions have inputs records which reference the funds from other previous transactions and outputs records which determine the new owner of the transferred Bitcoins, and which will be referenced as inputs in future transactions as those funds are re-spent. Each input must have a cryptographic digital signature that unlocks the funds from the prior transaction. Only the person possessing the appropriate private key is able to create a satisfactory signature; this in effect ensures that funds can only be spent by their owners.

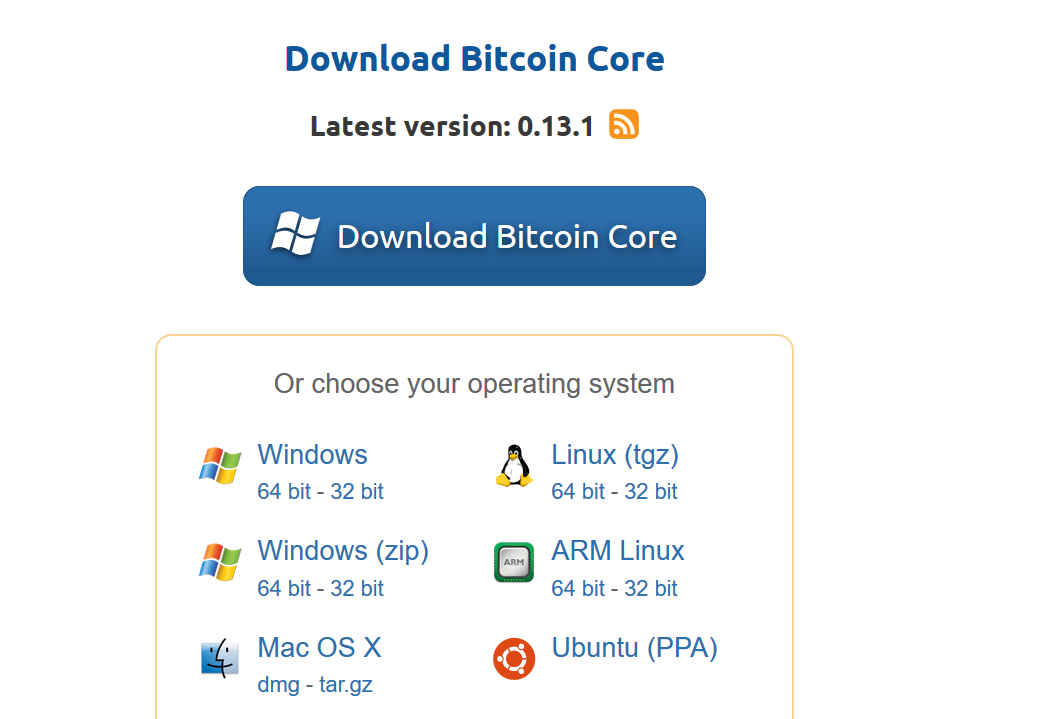
Each ouput determines which Bitcoin address (or other criteria, see script) is the recipient of the funds.In a transaction, the sum of all inputs must be equal to or greater than the sum of all outputs. If the inputs exceed the outputs, the difference is considered a transaction fee, and is redeemable by whoever first includes the transaction into the blockchain.

In addition to the newly created Bitcoins, the coinbase transaction is also used for assigning the recipient of any transaction fees that were paid within the other transactions being included in the same block. The coinbase transaction can assign the entire reward to a single Bitcoin address, or split it in portions among multiple addresses, just like any other transaction. Coinbase transactions always contain outputs totalling the sum of the block reward plus all transaction fees collected from the other transactions in the same block.

**Setup:**

Windows machine has been setup as follows to learn more about the bitcoin protocol. Following are the installation instructions:

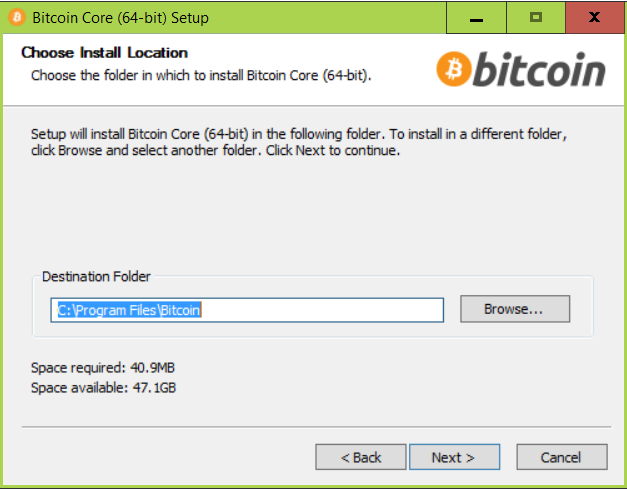
1.The latest version of Bitcoin Core(0.13.1) has been downloaded from <https://bitcoin.org/en/download> and is executed with required privileges.



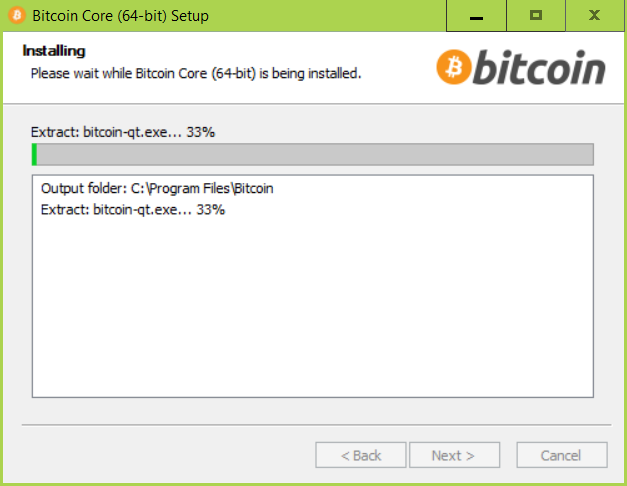
2.After running the executable select Next when below screen appears,



3. Select the installation directory for the bitcoin core to be installed



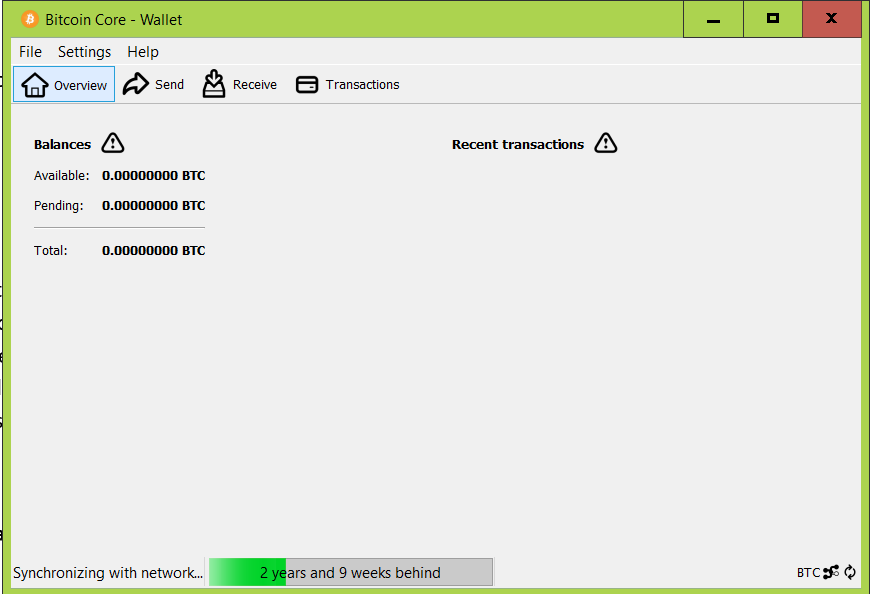
4.Let the installation install all the required things



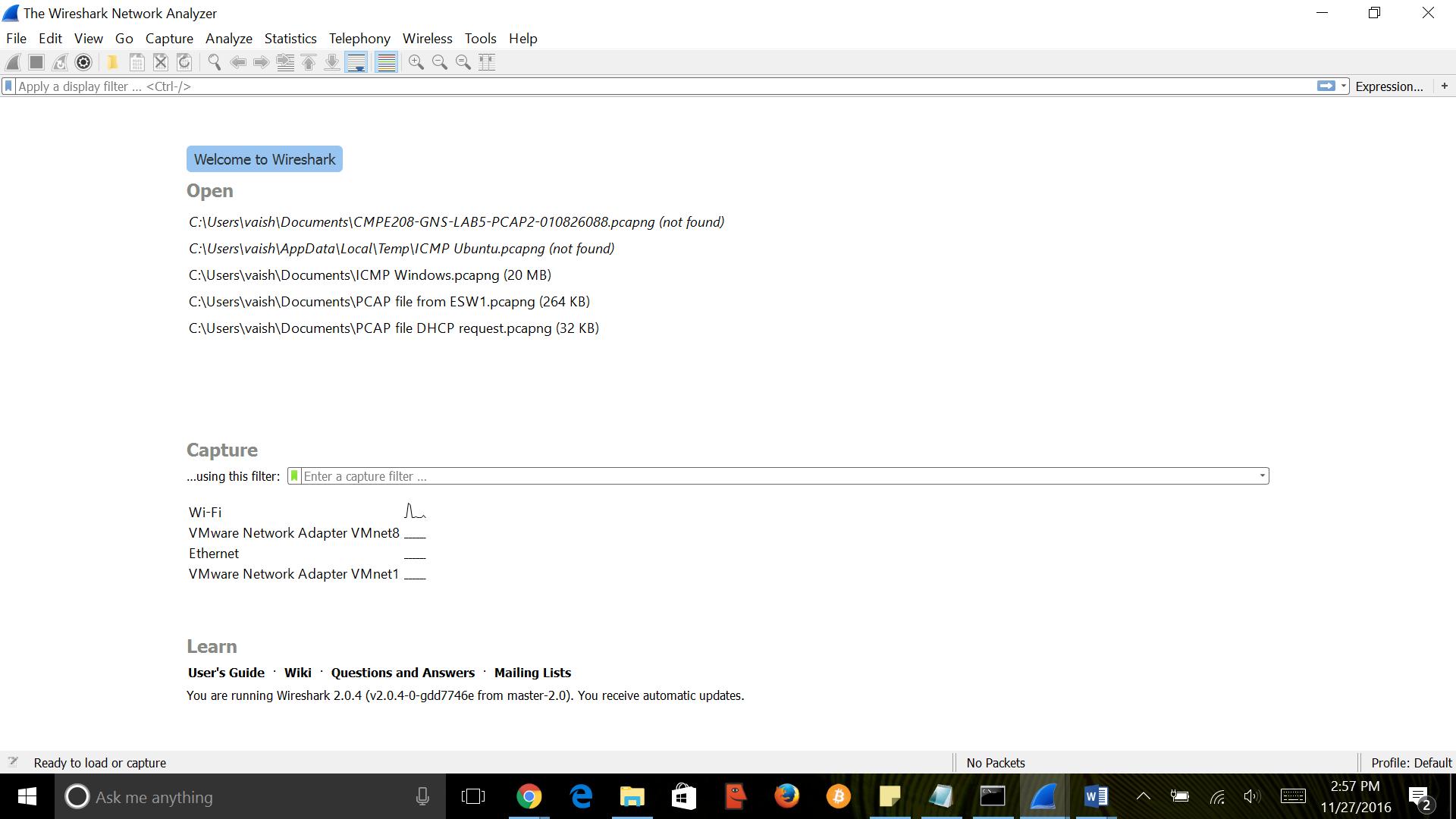
5.Once done press finish to launch the Bitcoin Core.



6.Setup the location to where the older data is to be downloaded, and the core starts syncing all the older transactions to the mentioned location. Make sure you have enough space and are connected to Internet via a high speed line, it almost requires 100Gb of download.



After installing the Bitcoin Core and syncing all the data Wireshark has been used to analyse the network traffic which is discussed in the report. Install Wireshark for analyzing packets.



**Bitcoin Protocol Implementation:**

bitcoin is the program that implements the Bitcoin protocol for remote procedure call (RPC) use. It is bundled with the Bitcoin Core. It is basically multithreaded C++ program. The major operations are as below,

**Initialization and Startup:**

On Startup, the client initializes multiple subroutines and threads to handle concurrent operations.

* The first thing main does is it calls the AppInit, which in turn calls up the GUI.
* Next it locks the bitcoin directory and if bitcoin is already running displays the same and exits.
* It now tries to listen to the bitcoin port, and if in use displays the error message and exits.
* Next loads the IP address from the database, then loads block index and loads the wallet.
* Then gets the top block number from the wallet and scans the block chain from the block and above.
* Finally, it creates two threads 1. StartNode() and if acting as server 2. ThreadRPCServer().

**Below are the major threads created and their functionalities,**

**StartNode():**

StarNode() is sort of a master networking thread, It first creates a CNode for the localhost 127.0.0.1 internal address to handle communication.

* Next gets the local IP address, then verifies the local IP address using 3rd Party verification.
* Then creates multiple threads ThreadIRCSeed (to exchange IP addresses), ThreadSocketHandler (Send and receive from sockets, accept connections), ThreadOpenConnections (to initiate Outbound connections), ThreadMessageHandler (to process messages), finally if specified it starts the bitcoin mining thread.

**Thread IRCSeed:**

* This connects to 92.243.23.21 port 6667, JOINs the channel and reads lines one at a time looking for other users.

**Thread ThreadSocketHandler:**

* This runs in an endless loop servicing the threads that needs servicing. Handles the disconnected nodes and manages the IO.

**Thread ThreadOpenConnections:**

* Gets the nodes from parameters, seeds, irc, etc, and then goes into a loop connecting to each.

**Thread ThreadRPCServer:**

* This is a rather complicated implementing the HTTP(S)+JSONRPC server using the boost classes.

**Node Discovery:**

Client uses multiple techniques to find out about the bitcoin nodes that are currently connected to the network.

* Nodes receive callback address of remote nodes that connect to them and send getaddr for more nodes.
* Nodes make DNS requests(seed.bitcoinstats.com, dnsseed.bluematt.me etc) to receive IP addresses, they even use hard coded addresses in code.
* Nodes exchange addresses with other nodes, Nodes store the IP addresses into a database which are read at startup.
* Nodes can be provided IP address as command line arguments or be provided as a text file at startup, It also self-broadcasts its own addresses to all connected nodes.

Timestamps are stored for last seen of the Nodes.

**Handling the getaddr message:**

When a node receives a getaddr request, it responds with the addresses which it found active in the last 3 hours. It limits the output IP address to 2500 if found more.

**Node Connectivity:**

The bitcoin client creates the ThreadOpenConnections, it chooses among available addresses and makes connections and disconnects nodes when appropriate.

The outbound connections are limited to a maximum value and when it exceeds goes into a 2 second delay and then retries based on active connections and the max. outbound connections.

Seed Nodes, If the Node is unable to learn about any Nodes. It will use an internal list of 320 addresses hardcoded into code.

Outbound random selection, the addresses are given a score based on their availability and other factors. The node with higher scores wins and the OpenNetworkConnection is called on the Node.

Inbound accepting and disconnecting, Sockets are disconnected if they are 60 seconds old and have not sent or received any data, if the current inbound or outbound data exceeds a buffer.

**Peer-to-Peer Network:**

The Bitcoin works as Peer-to-Peer network protocol, in which all communication occurs in TCP. The basic constants and defaults which can be found in the chainparams.cpp file are as follows,

|  |  |  |  |
| --- | --- | --- | --- |
| **Network** | **Default Port** | **Start String** | **Max nBits** |
| Mainnet | 8333 | 0xf9beb4d9 | 0x1d00ffff |
| Testnet | 18333 | 0x0b110907 | 0x1d00ffff |
| Regtest | 18444 | 0xfabfb5da | 0x207fffff |

But the default listen ports of the nodes can be changed from the command line.

**Protocol Versions:**

Below are latest few versions of the P2P network protocols with versions and major changes,

|  |  |  |
| --- | --- | --- |
| Version | Initial Release | Major changes |
| 70014 | Bitcoin Core 0.13.0 | BIP152:   * Added sendcmpct, cmpctblock, getblocktxn, blocktxn messages. * Added MSG\_CMPCT\_BLOCK inventory type to getdata message. |
| 70013 | Bitcoin Core 0.13.0 | BIP133:   * Added feefilter message |
| 70012 | Bitcoin Core 0.12.0 | BIP130:   * Added sendheaders message |

These are few latest releases/updates to the Bitcoin P2P protocol.

**Version Handshake:**When a local peer connects to a remote peer, the data transfer is initiated only when the remote peer receives a version message.

L -> R: Send version message with the local peer's version  
R -> L: Send version message back  
R: Sets version to the minimum of the 2 versions  
R -> L: Send verack message  
L: Sets version to the minimum of the 2 versions

**Bitcoin Packet Dissection:**

We start bitcoin using the GUI (Bitcoin Core) or without a graphical display. As the implementation is done on Windows Operating System

Once the Bitcoin application is launched after installation, the application downloads all the blocks which has been already created from the start date of the bitcoin. Each block and the corresponding transactions in the block are received through the bitcoin protocol. The destination/source ip address in the network packets will be host ip address in which the bitcoin application is running depending on the message type received/sent.

The packet captures below will show how protocol works. Bitcoin first will check the version of the bitcoin protocol and then starts syncing the blocks to update the database. The detailed description of each message type and how the node will communicate to its peer nodes is explained in the later sections.

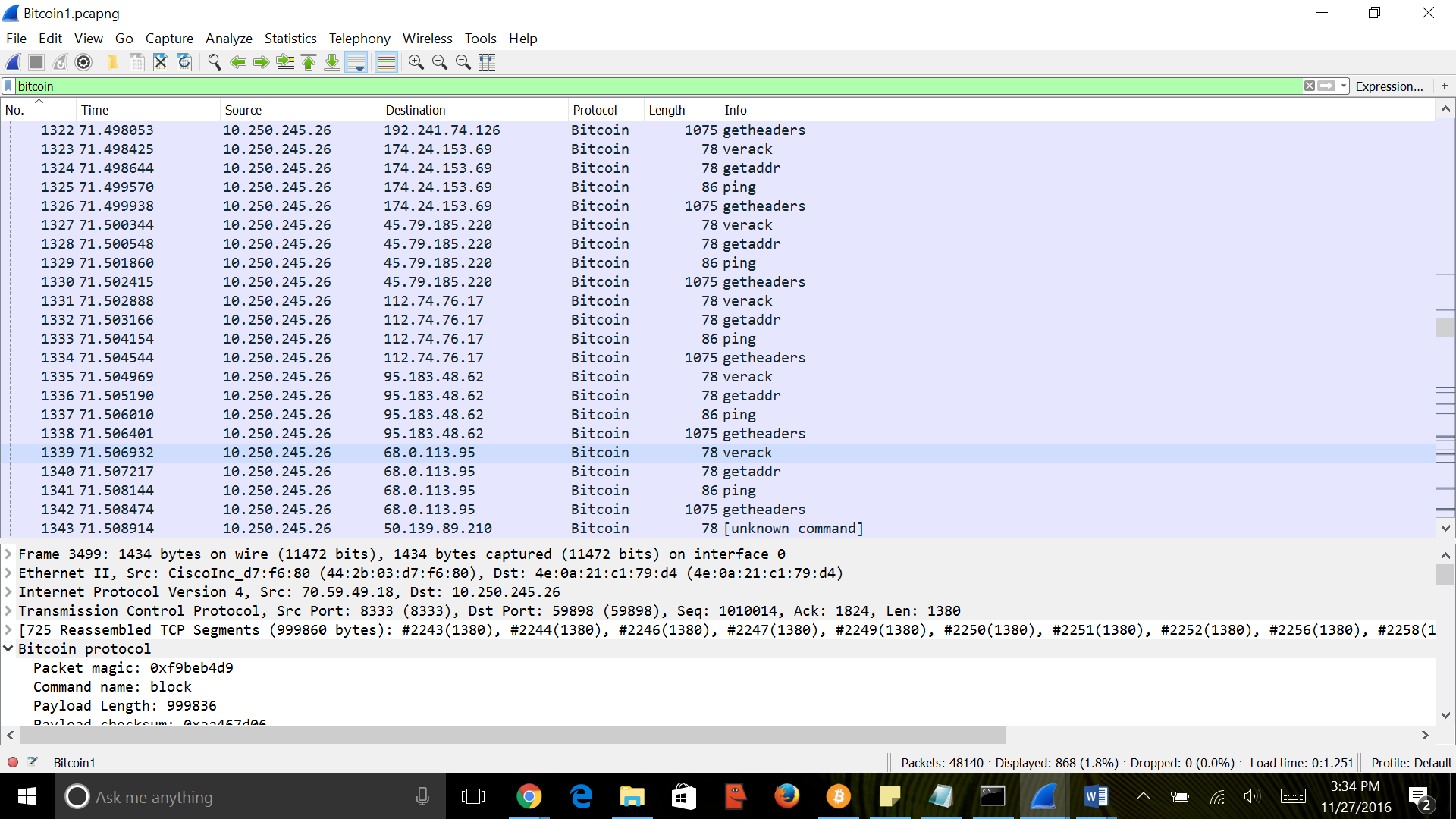


Fig: Packet capture

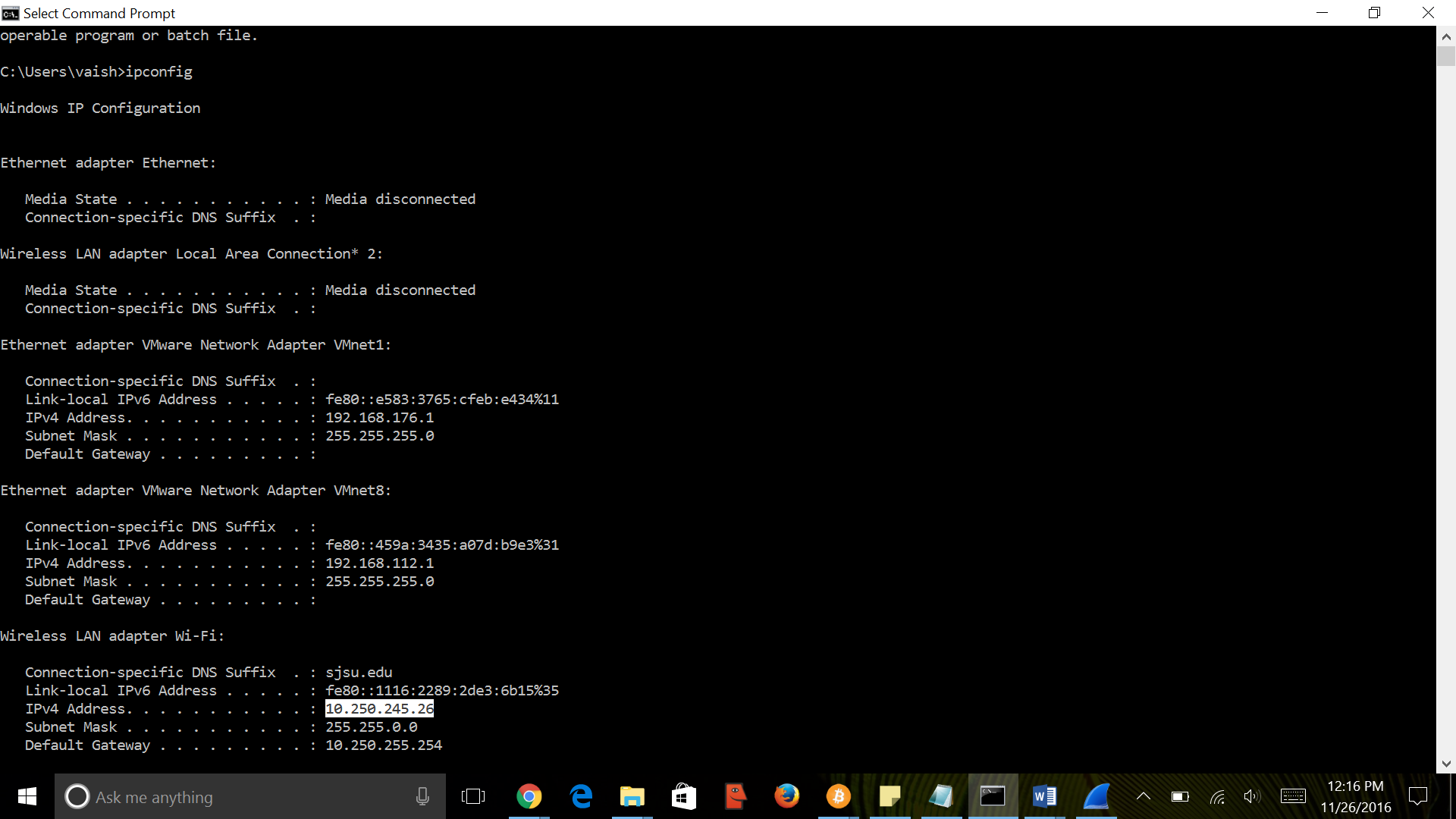


Fig: Destination/host ip configuration

The Bitcoin core wallet is set to send and receive bitcoins. Each bitcoin will have unique bitcoin address through which it can be transferred to other person. The update of the transaction will be sent to all the nodes connected to the network.

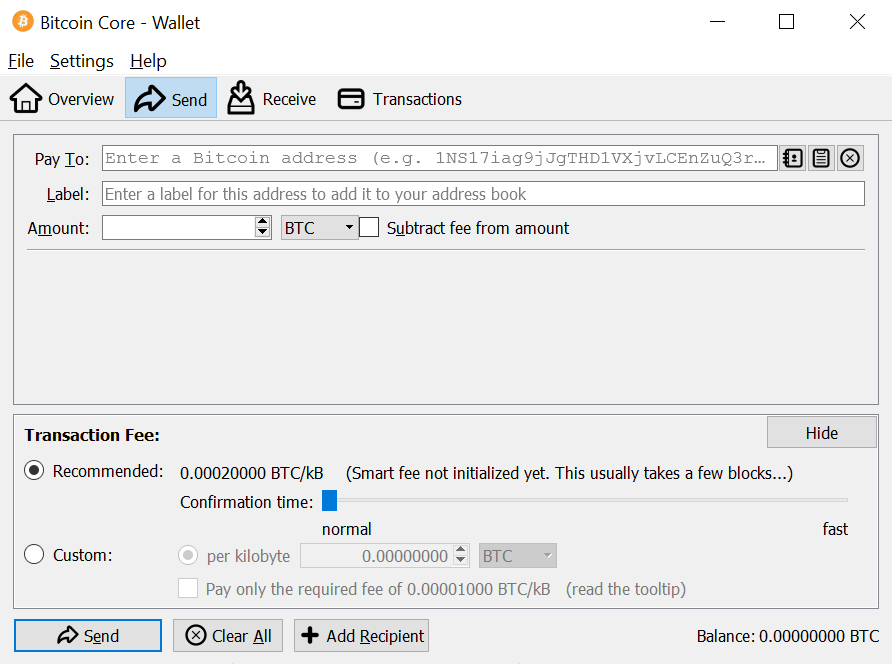


Fig: Transferring a bitcoin

There are two types of messages in bitcoin protocol:

1. Data message

Examples: Block, getblock, getdata, getheaders, headers, inv, mempool, merkleBlock, NotFound, Tx

1. Control message

Examples: Addr, Alert, FilterAdd, FilterClear, FilterLoad, GetAddr, Ping, Pong, Reject, SendHeaders, VerAck, Version

**Data messages:**

The flow of data messages between bitcoin application server and destination host takes place in an order. The below figure represents the network messages list for request and reply of data which are related to transactions and blocks update.

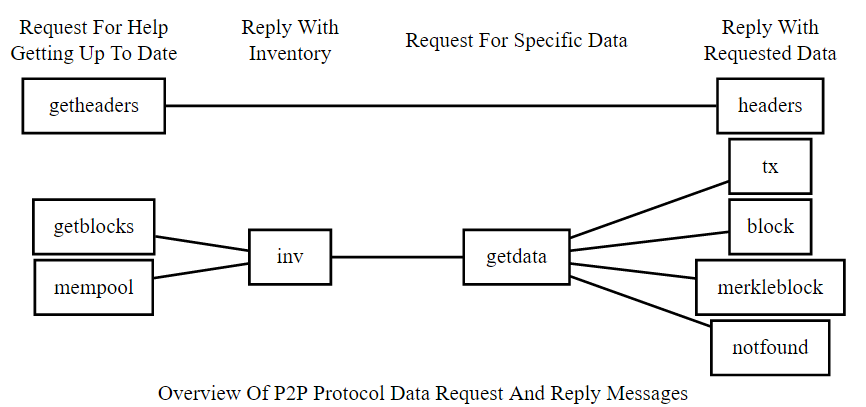


Fig: Data messages flow

The description for each message format and the content of each message is given below:

**Getheaders message:**

Getheader message is used to request header messages which will provide the header of the blocks in the block chain starting from a particular point. The headers which will come as a reply to the getheader

message will contain as many as 2000 headers of the blocks.



Fig: getheaders message

**Headers message:**

One or more than one block header will be sent to the receiving node according to the request sent by the node through getheader message.

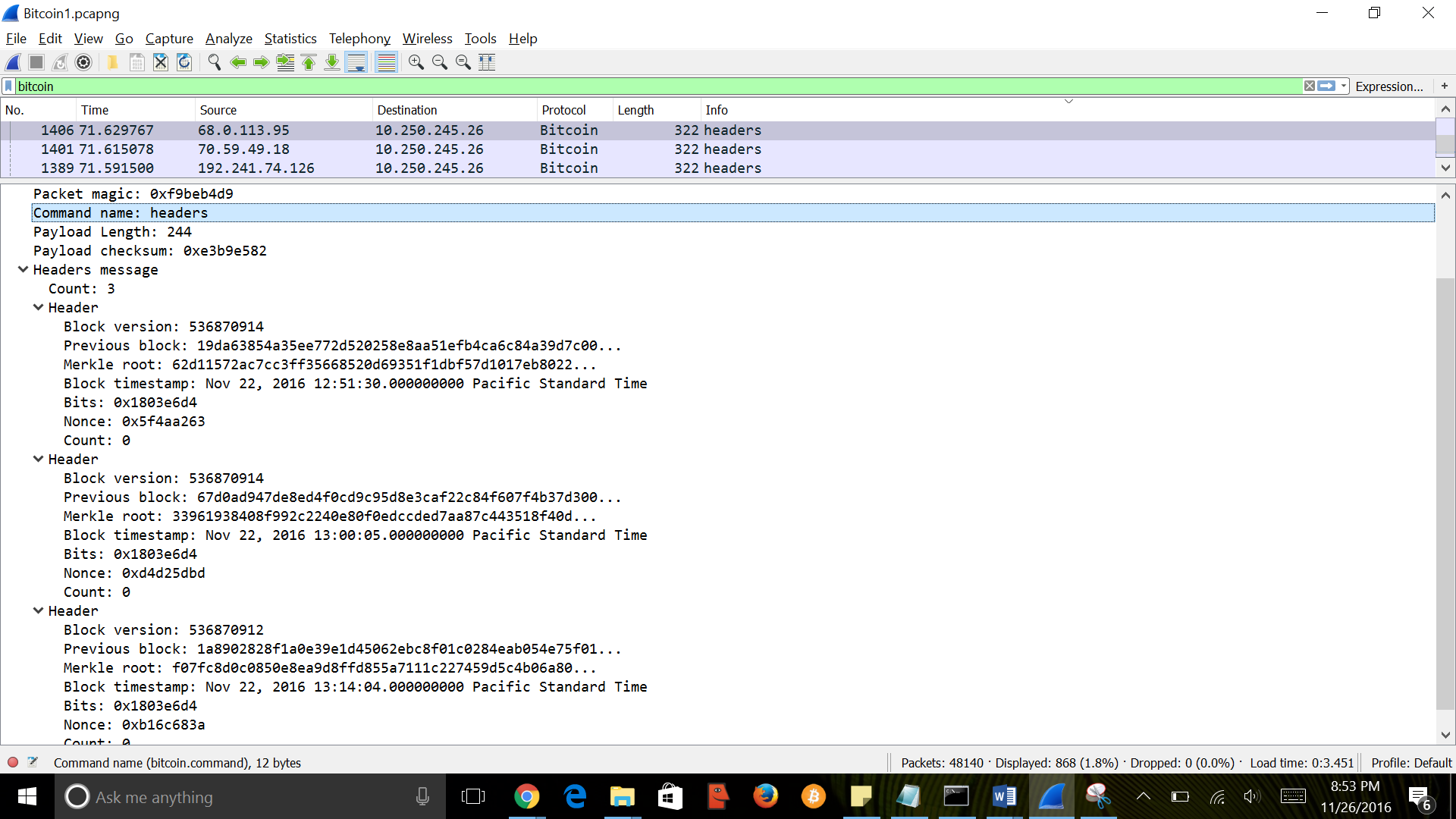
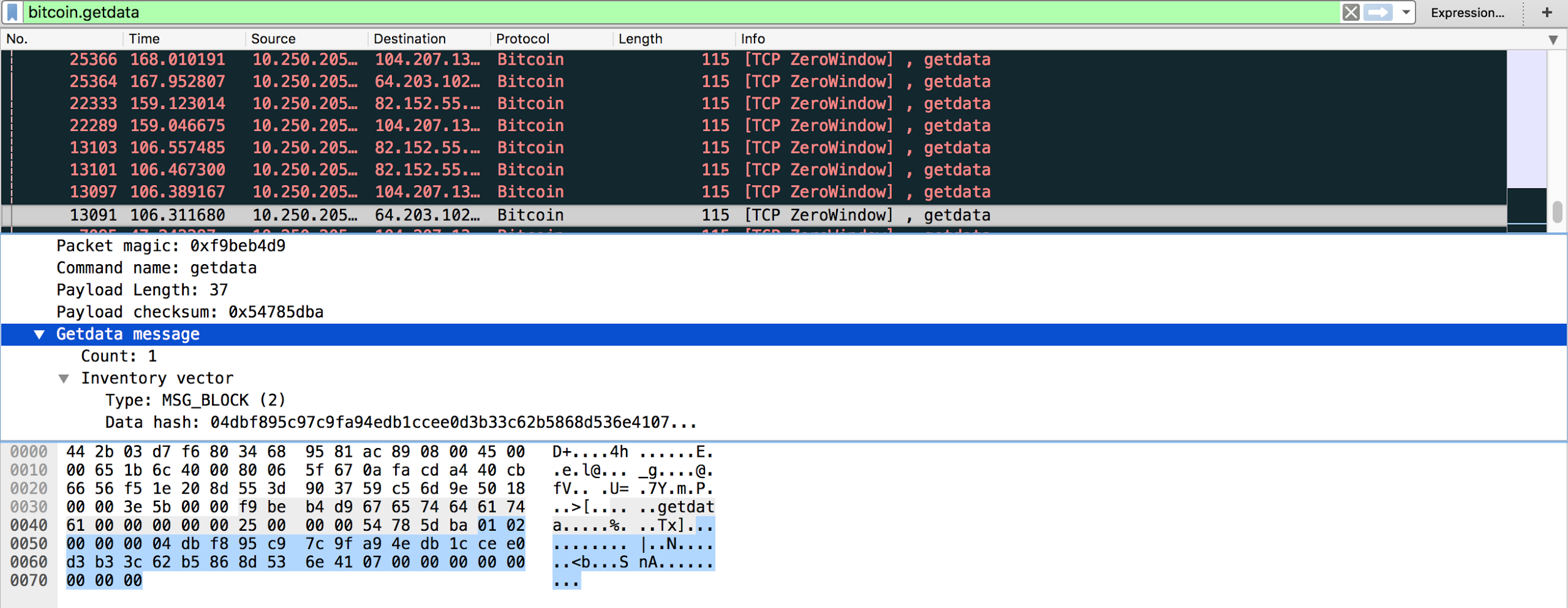


Fig: header message

**GetData**

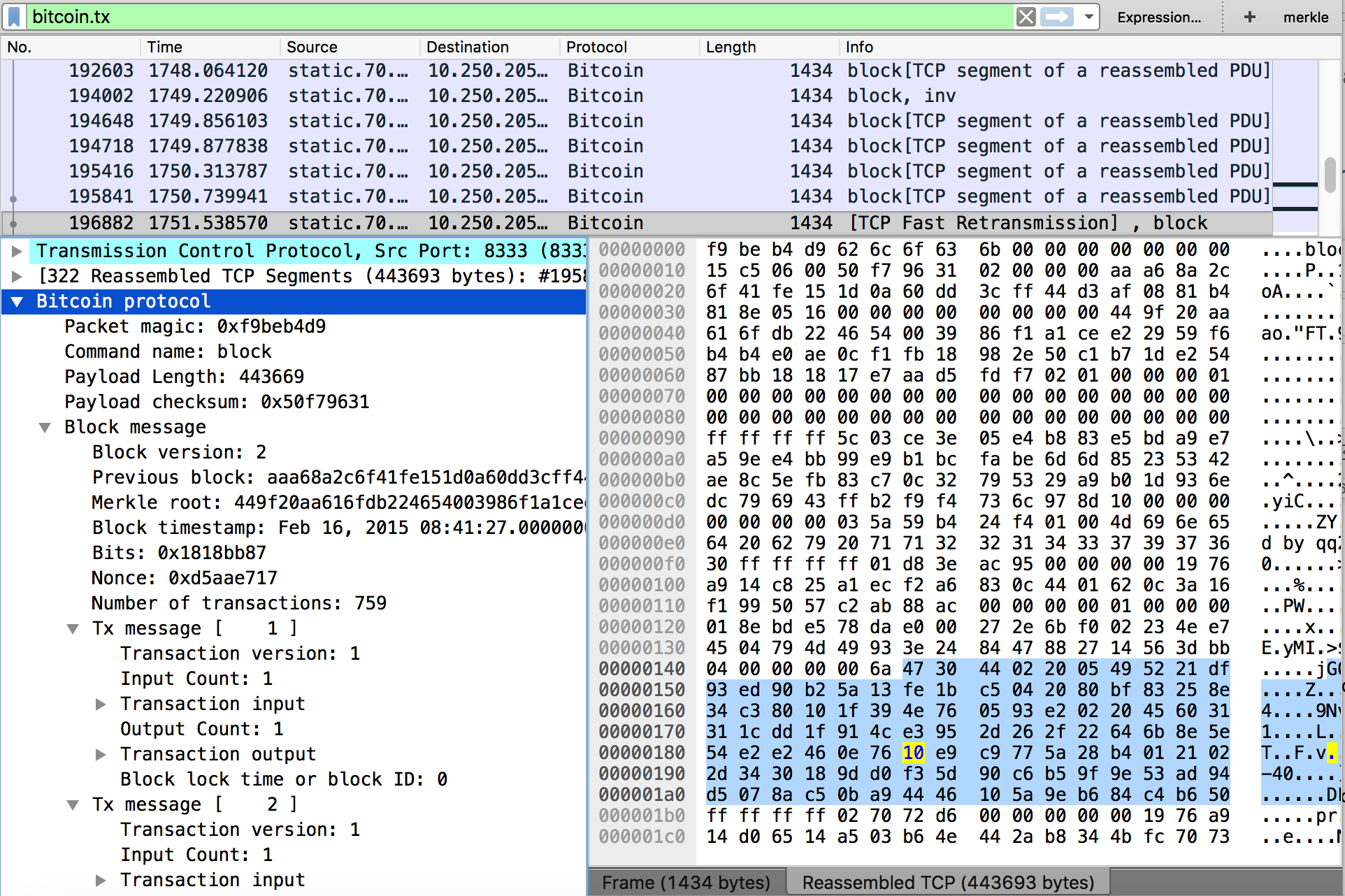
This message requests one or more data objects from another nose. The objects are requested by an inv message

Response varies from being a tx message, a block message, a merkle block message and a not found message



**TX:**

The tx message transmits a single transaction in the raw transaction format. It can be sent as a getdata response as well as a merkle block response. If there is a new transaction originating at the source it will be unsolicited as well

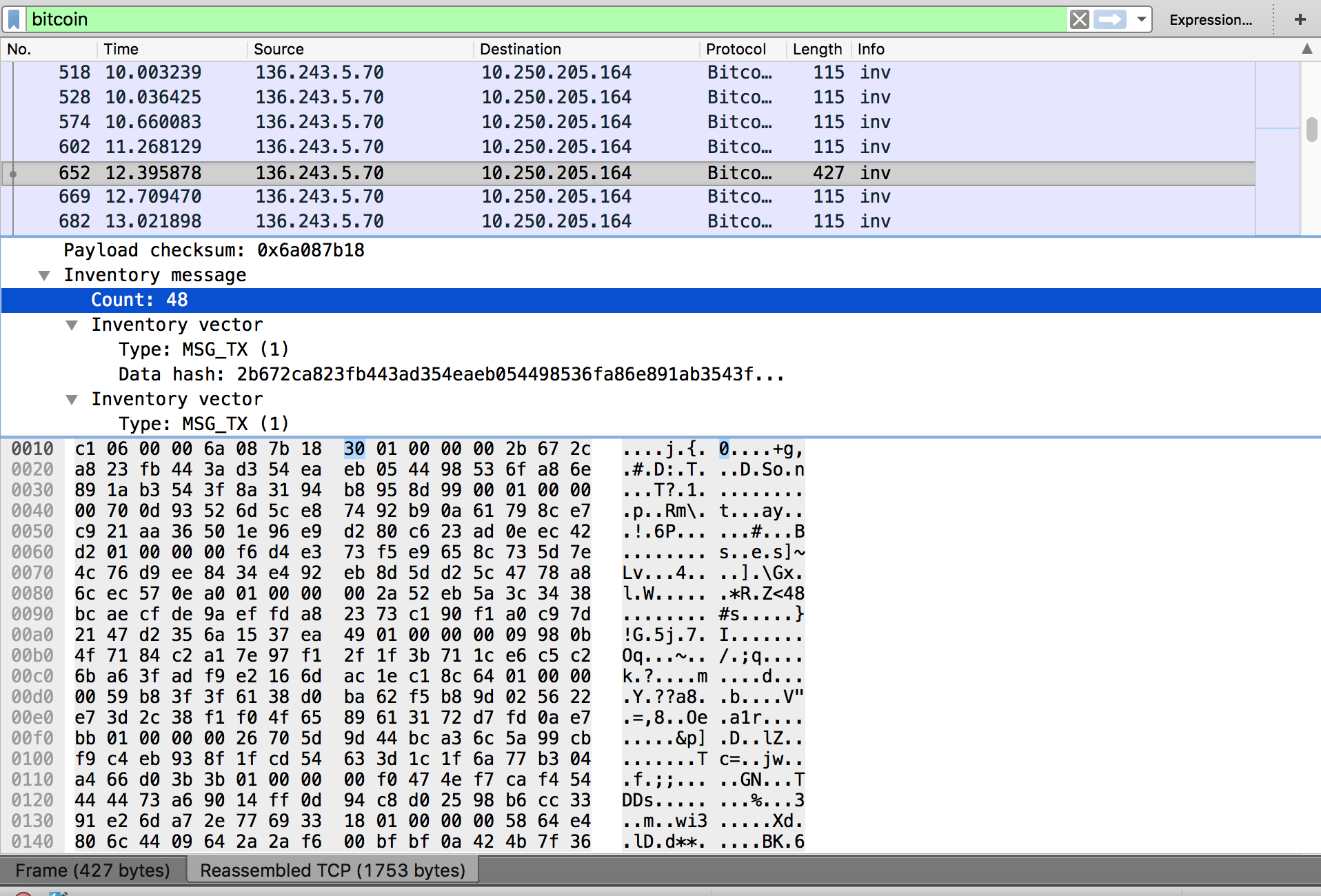


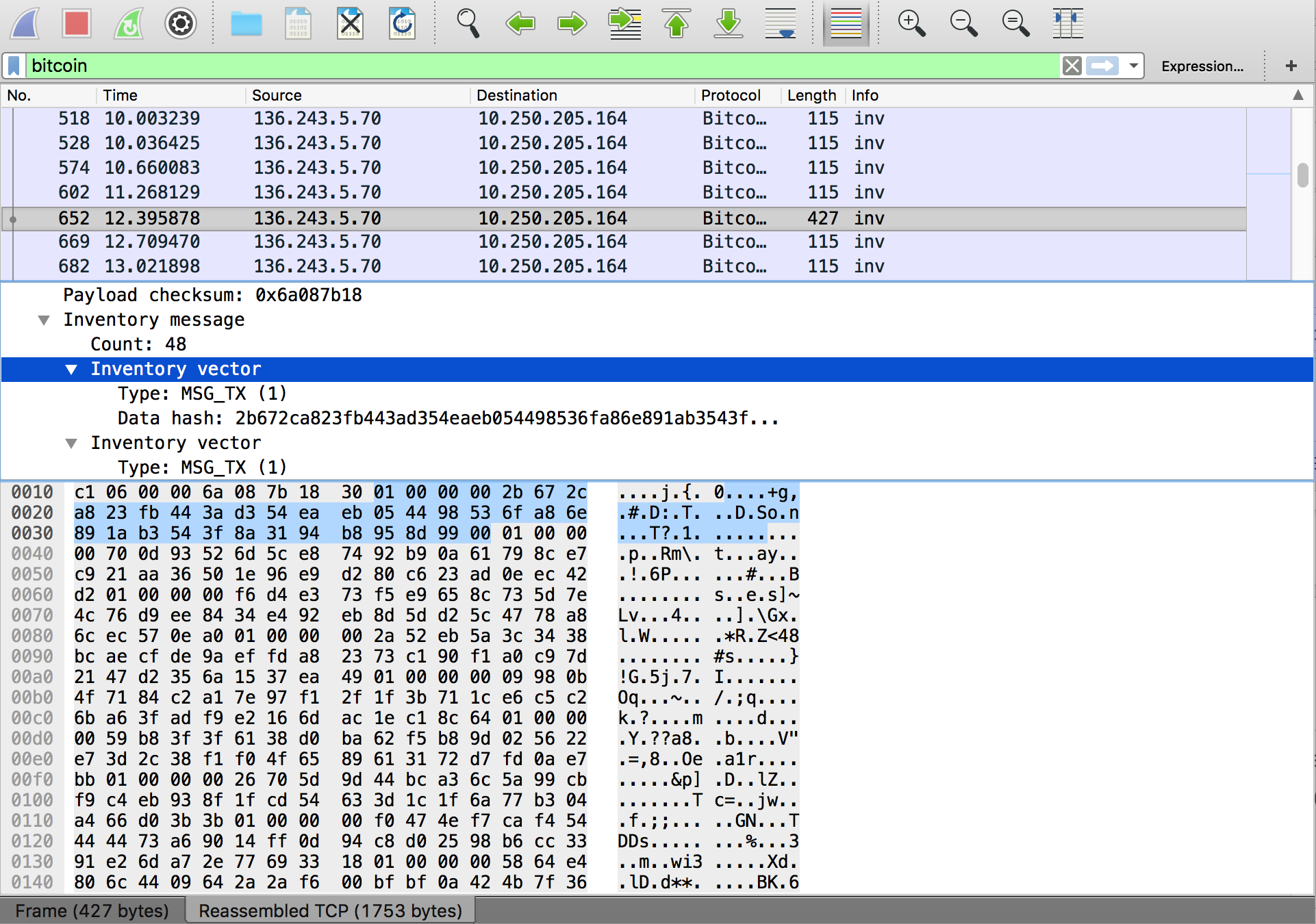
**INV Message:**

Its the inventory message that transmits single or multiple inventories of objects known to the source. It can be sent without request from other nodes to broadcast new transactions or blocks.

It can also be used to send as a reply to “getblocks” or “mempool” message

|  |  |  |  |
| --- | --- | --- | --- |
| **Bytes** | **Name** | **Data Type** | **Description** |
| Varies | count | [compactSize uint](https://bitcoin.org/en/glossary/compactsize) | The number of [inventory](https://bitcoin.org/en/glossary/inventory) entries. |
| Varies | [inventory](https://bitcoin.org/en/glossary/inventory) | [inventory](https://bitcoin.org/en/glossary/inventory) | One or more [inventory](https://bitcoin.org/en/glossary/inventory) entries up to a maximum of 50,000 entries. |





**Block message:**

A block message is sent with a block header followed by transaction details.

**Block header:**

The header of the block is serialized as an 80-byte format as shown below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | version | previous block header hash | merkle root hash | time | nBits | nonce |
| **Data type** | uint32\_t | char[32] | char[32] | uint32\_t | uint32\_t | uint32\_t |
| **Bytes** | 4 | 32 | 32 | 4 | 4 | 4 |

* Version - Version number - To identify the set of rules to follow for block validation.
  + Block versions:

Version 1 is the genesis block.

Version 2, Version 3, Version 4 are introduced for the respective bitcoin core releases.

* Previous block header - To ensure that no previous block is changed.
* Merkle root - To ensure no transaction is modified without header modification. Merkle root is generated using TXIDs of all the transactions which are present in the block.

To form a Merkle root, TXIDs of coinbase transaction and the other transactions are placed together and made as 64 raw bytes and hashed together.

* Time - To indicate the unix epoch time at which header hashing is started by the miner.
* nBits - To indicate the target threshold encoded version.
* Nonce - To indicate an arbitrary number which can be changed by miners to maintain the hash below threshold target.

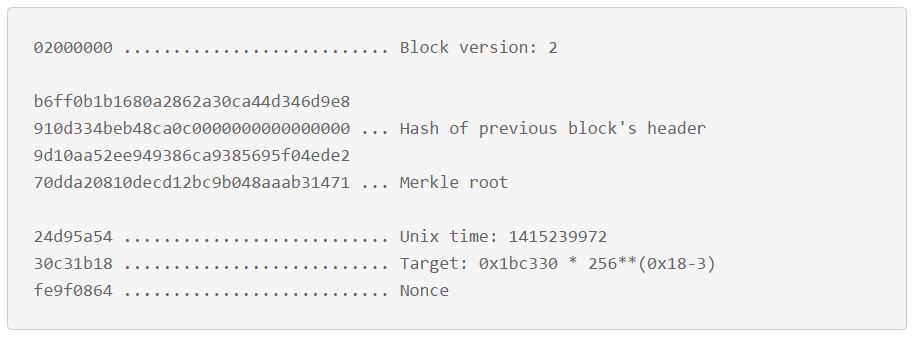


Fig: A header sample in hex

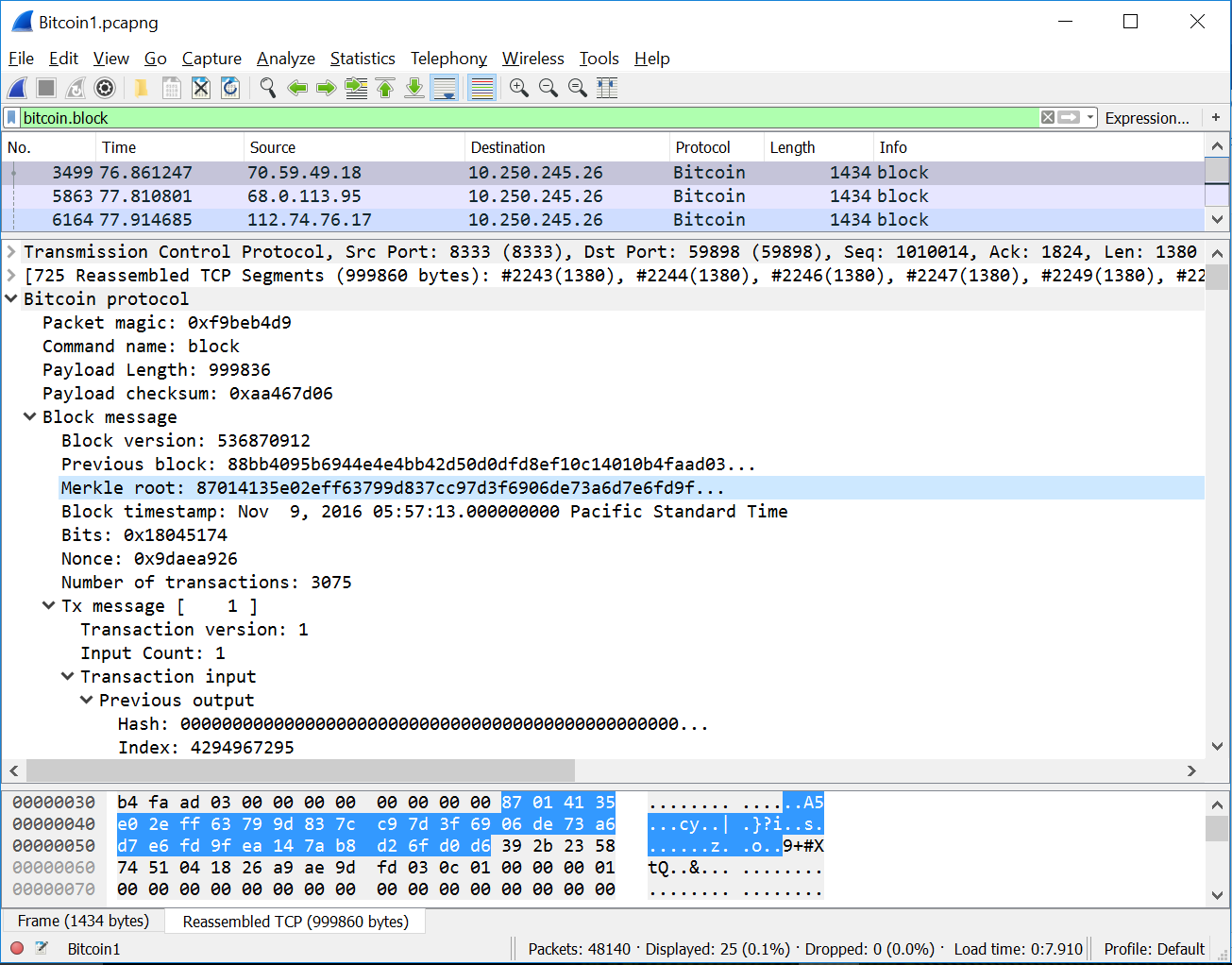
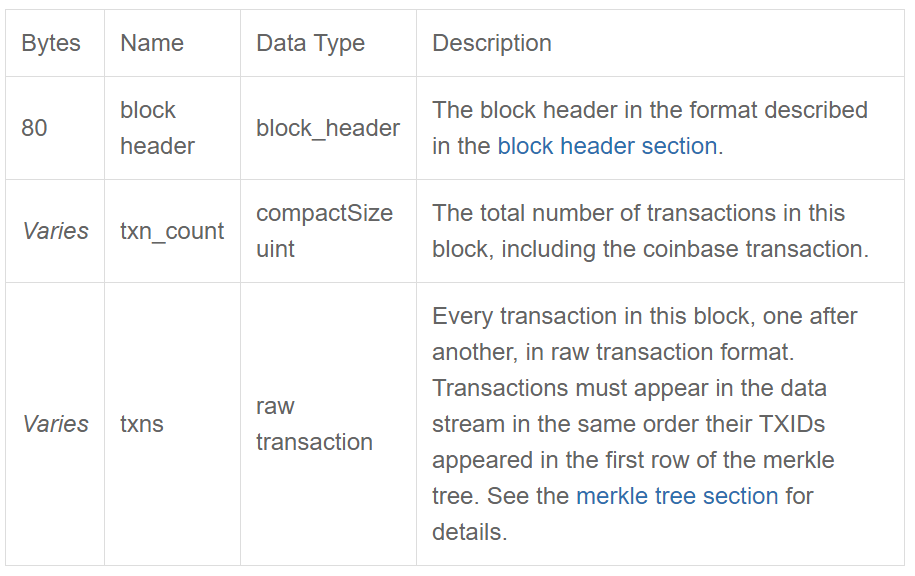


Figure: Block header info – Bitcoin protocol packet capture

**Serialized blocks:**

In the serialized block, block header is followed by transaction count and the list of all transactions happened in that block.



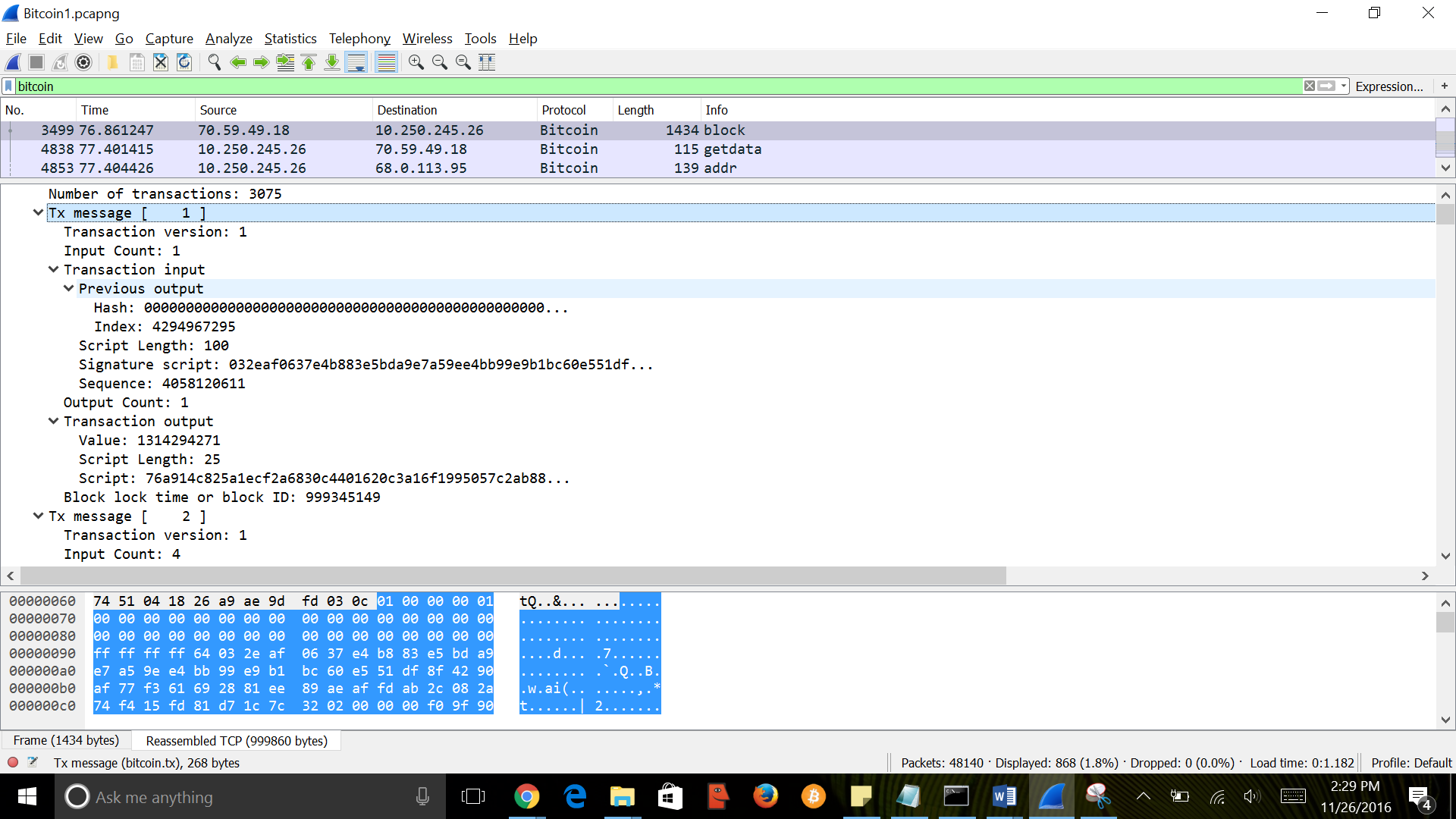


Fig: Raw transaction packet capture

The above packet capture represents a serialized block in which block header is continued with total number of transactions in the current block which included coinbase transaction followed by the raw transaction.

**Raw transaction format – tx message:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| version (uint32\_t)  4 Bytes | tx\_in count  (compactSizeuint)  Variable size | tx\_in (txIn)  Variable size | tx\_out count (compactSizeuint)  Variable size | tx\_out (txOut)  Variable size | lock\_time (uint32\_t)  4 Bytes |

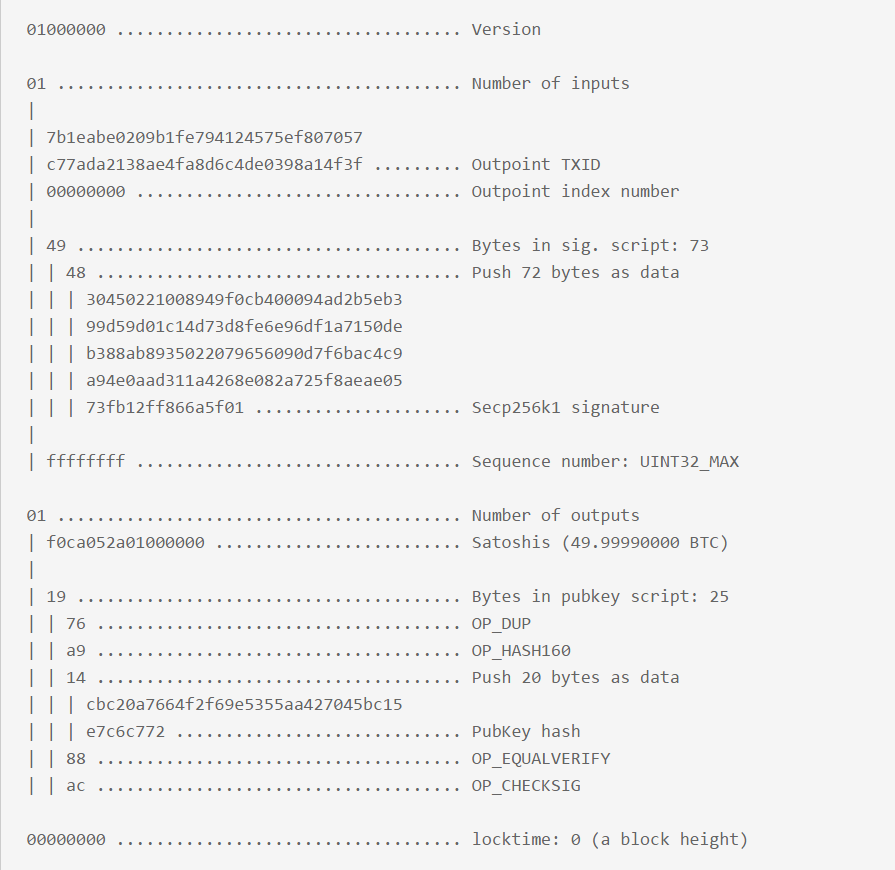


Fig: Raw transaction format

* Version - Version number - To identify the set of rules to follow for block validation.
* Tx\_in count - To identity number of inputs in the corresponding transaction.
* Tx\_in - Transaction inputs.
  + Below is the format for transaction input:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Name | Previous\_output | Script bytes | Signature scripts | Sequence |
| Bytes | 36 | Varies | Varies | 4 |
| Data Type | Contains char[32] of 32bytes hash value and uint32\_t type index value of 4 bytes | Compact size uint | Char[] | uint32\_t |

* Tx\_out count - To identify number of outputs in the corresponding transaction.
* Tx\_out - Transaction outputs.
  + Below format represent the transaction output:

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Value | Script length | Script |
| Byte | 8 | 1+ | Varies |
| Data type | Int64\_t | Compactsize uint | Char[] |

* Lock\_time - unix epoch time.

**Control messages:**

Below are the control messages sent in network to control two peers’ connections and to advise each other with respect to rest of the network.



Fig: Control messages

**Getaddr message:**

From the receiving node, getaddr message requests a message type of addr. The transmitting node of getaddr message uses those IP addresses to update its database with available nodes quickly. For getaddr message there is no payload and the same will be seen in the below packet capture stating payload length as zero.

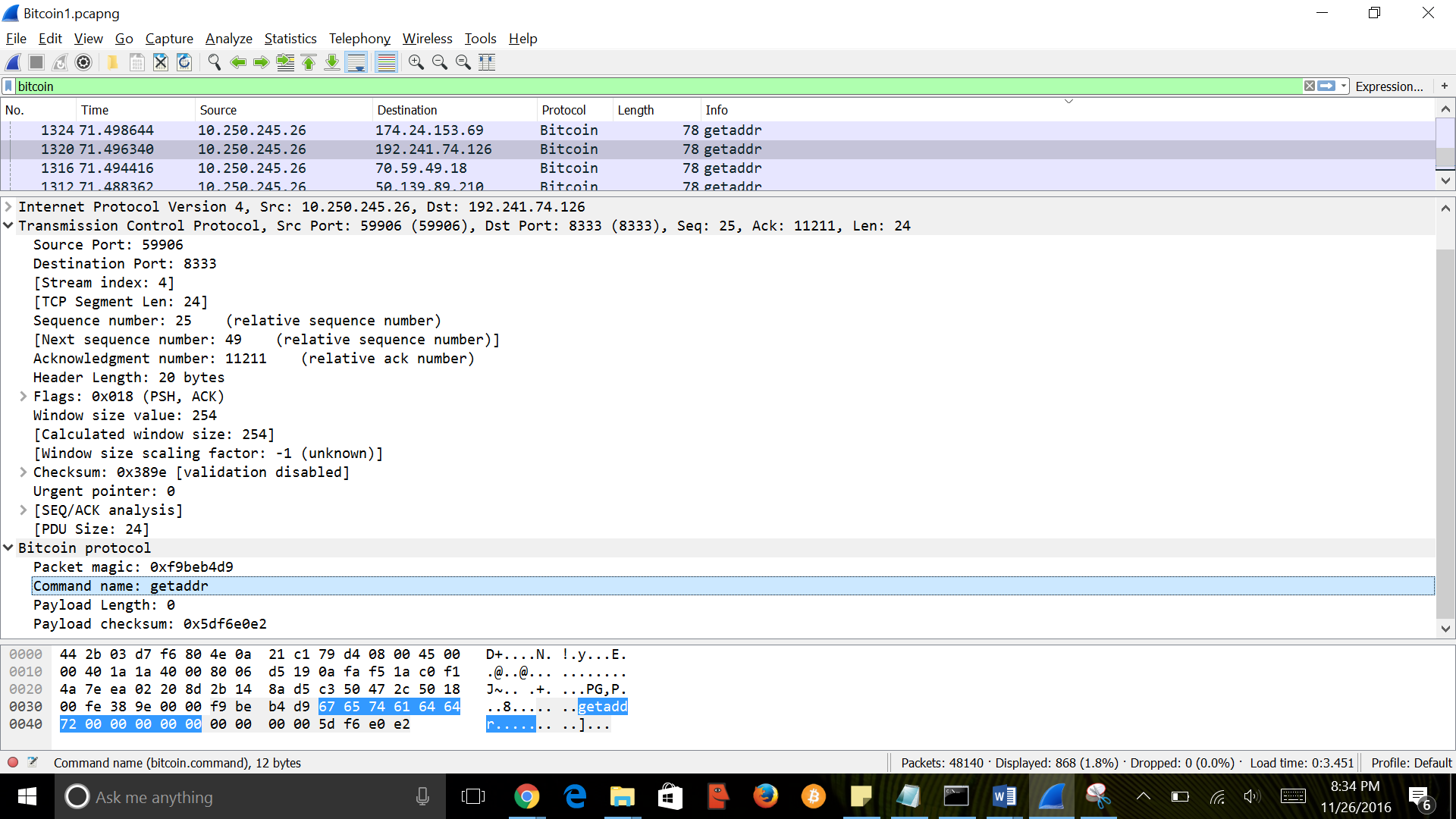


Fig: getaddr message

**Addr message (IP address message):**

Addr message relays connection information between peers connected on the network. If a peer wants to accept incoming connection creates a addr message. In response to getaddr message an addr message will be sent.

|  |  |  |
| --- | --- | --- |
| Name | IP address count | IP addresses |
| Byte | Varies | Varies |
| Data byte | Compactsize uint | Network IP address |
| Description | Number of IP address up to a maximum of 1,000 | IP address entries |

Each IP address encapsulated will follow below address format:

|  |  |  |  |
| --- | --- | --- | --- |
| Services  8 byte  uint64\_t | IP address  16 byte  char | Port  2 byte  uint16\_t | Time  4 byte  uint32 |

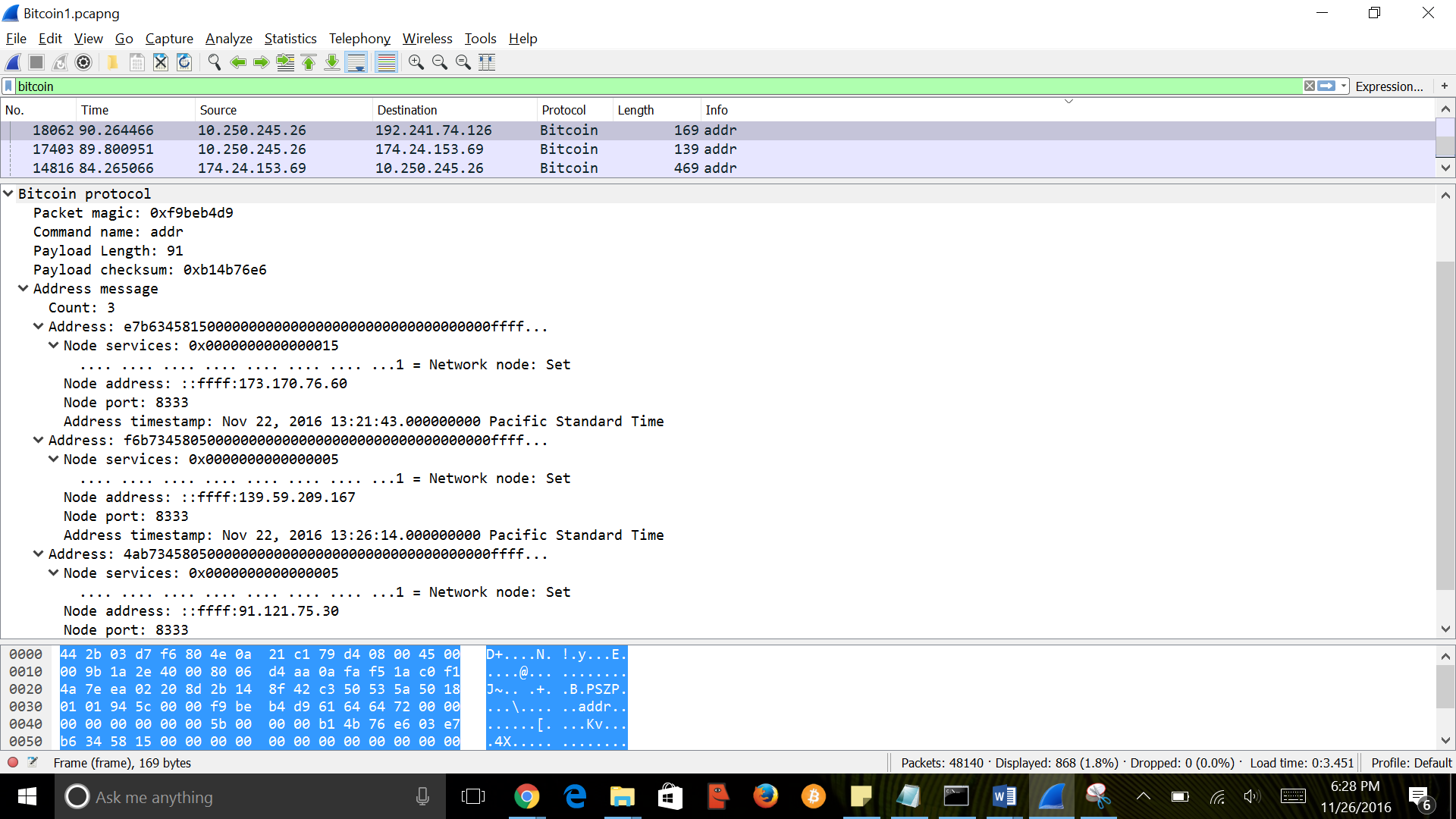


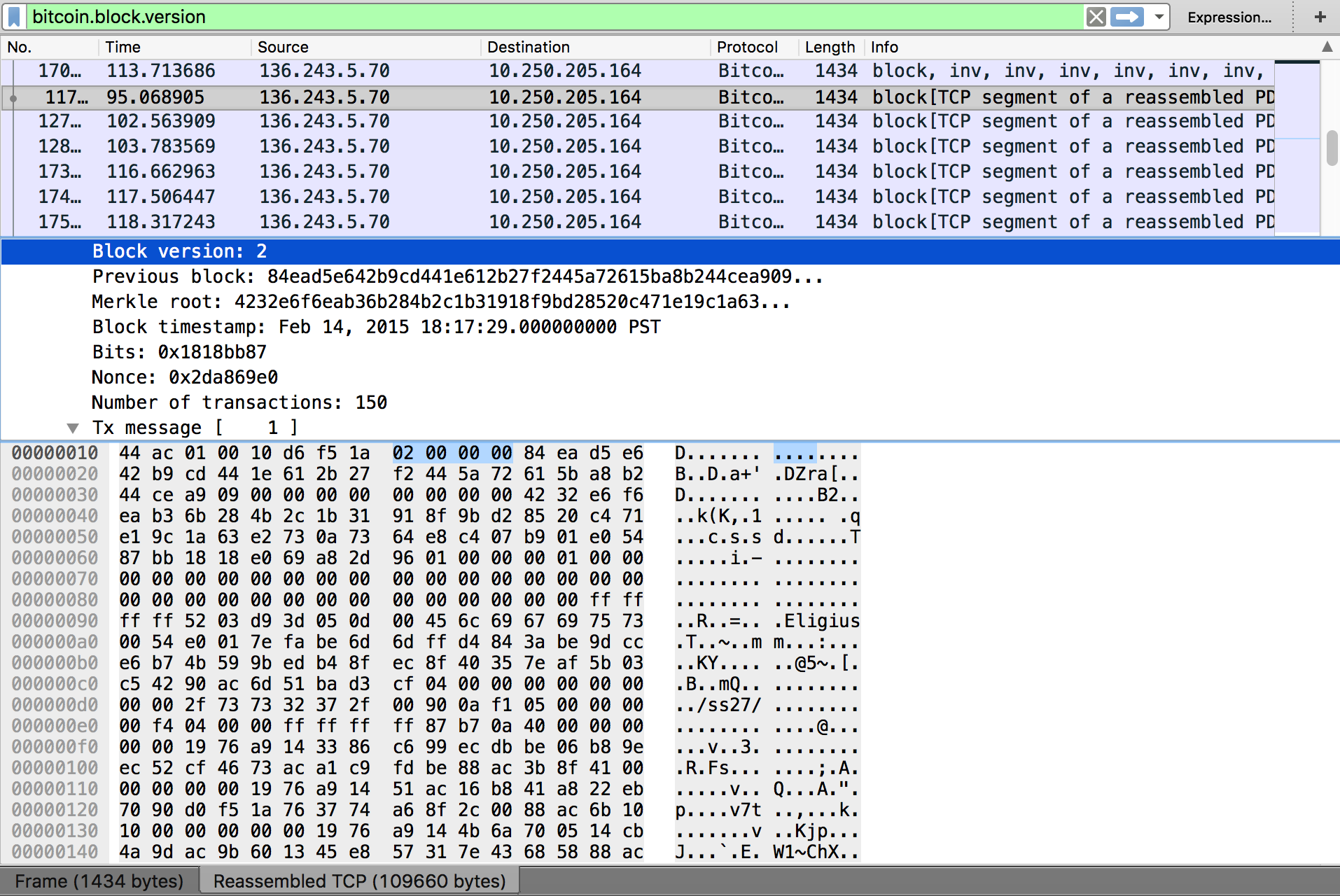
Fig: addr message

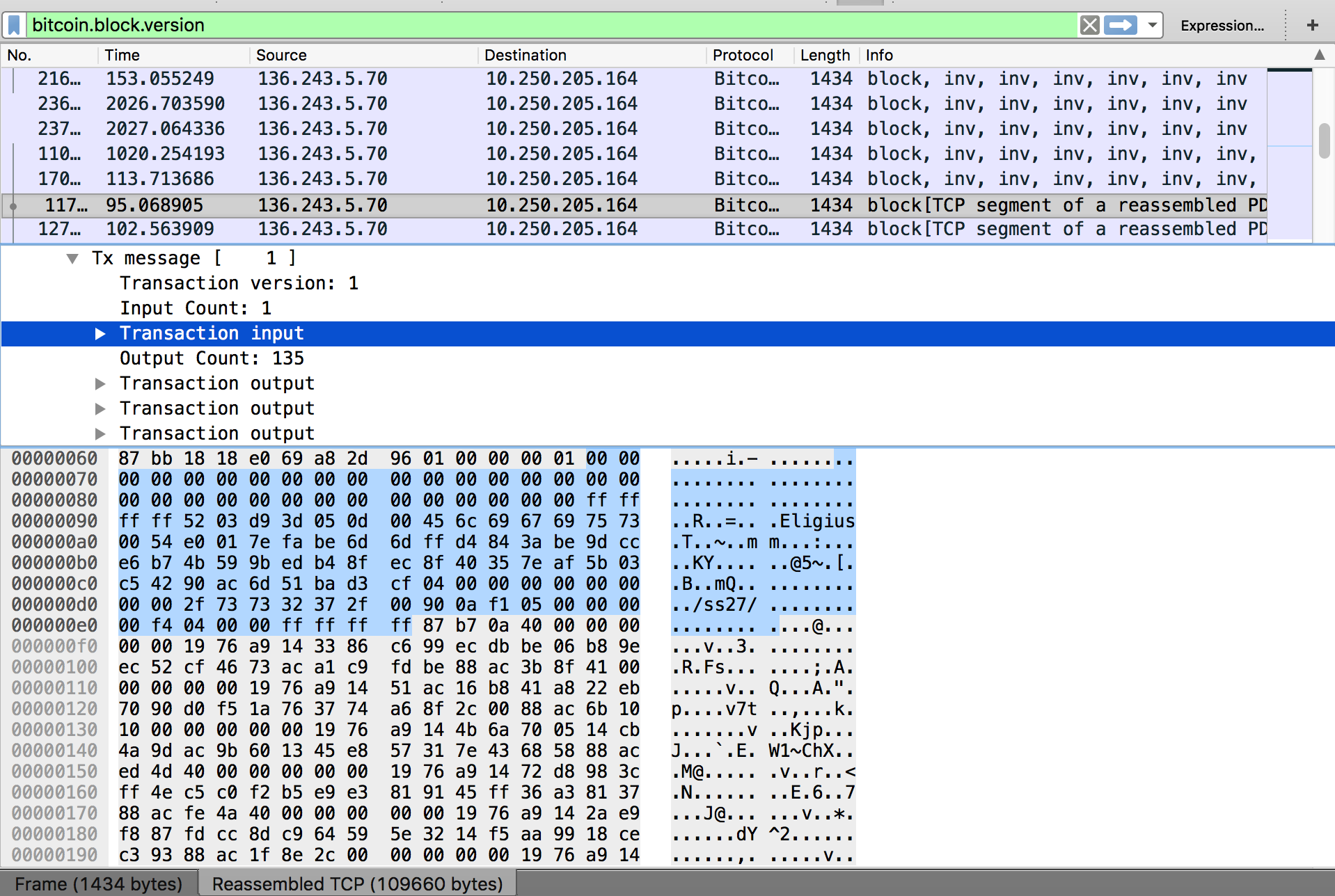
**Version message:**

Version message provides info about the source node to the receiving node at the opening of a connection. In order to receive other messages, both the peers should have exchanged version methods.

|  |  |  |  |
| --- | --- | --- | --- |
| Bytes | Name | Data Type | Description |
| 4 | version | int32\_t | The highest protocol version understood by the transmitting node. |
| 8 | services | uint64\_t | The services supported by the transmitting node encoded as a bitfield. |
| 8 | timestamp | int64\_t | Help other nodes to determine if their clock is wrong as nodes will reject [blocks](https://bitcoin.org/en/glossary/block) with timestamps more than two hours in the future |
| 8 | addr\_recv services | uint64\_t | The services supported by the receiving node as perceived by the transmitting node. |
| 16 | addr\_recv IP address | char | The IPv6 address of the receiving node. IPv4 addresses can be provided as IPv4-mapped IPv6 addresses. |
| 2 | addr\_recv port | uint16\_t | The port number of the receiving node as perceived by the transmitting node. |
| 8 | addr\_trans services | uint64\_t | The services supported by the transmitting node. Should be identical to the ‘services’ field above. |
| 16 | addr\_trans IP address | char | The IPv6 address of the transmitting node in big endian byte order. IPv4 addresses can be provided as IPv4-mapped IPv6 addresses. Set to ::ffff:127.0.0.1 if unknown. |
| 2 | addr\_trans port | uint16\_t | The port number of the transmitting node in big endian byte order. |
| 8 | nonce | uint64\_t | A random nonce which can help a node detect a connection to itself. If the nonce is 0, the nonce field is ignored. If the nonce is anything else, a node should terminate the connection on receipt of a [version message](https://bitcoin.org/en/developer-reference#version) with a nonce it previously sent. |
| Varies | user\_agent bytes | [compactSize uint](https://bitcoin.org/en/glossary/compactsize) | Number of bytes in following user\_agent field. If 0x00, no user agent field is sent. |
| Varies | user\_agent | string | User agent as defined by [BIP14](https://github.com/bitcoin/bips/blob/master/bip-0014.mediawiki). Previously called subVer. |
| 4 | start\_height | int32\_t | The [height](https://bitcoin.org/en/glossary/block-height) of the transmitting node’s [best block chain](https://bitcoin.org/en/glossary/block-chain) or, in the case of an [SPV client](https://bitcoin.org/en/glossary/simplified-payment-verification), best [block](https://bitcoin.org/en/glossary/block) [header chain](https://bitcoin.org/en/glossary/header-chain). |
| 1 | relay | bool | Transaction relay flag. If 0x00, no inv messages or tx messages announcing new transactions should be sent to this client until it sends a filterload message or message. If 0x01, this node wants inv messages and tx [messages](https://bitcoin.org/en/developer-reference#tx) announcing new transactions. |

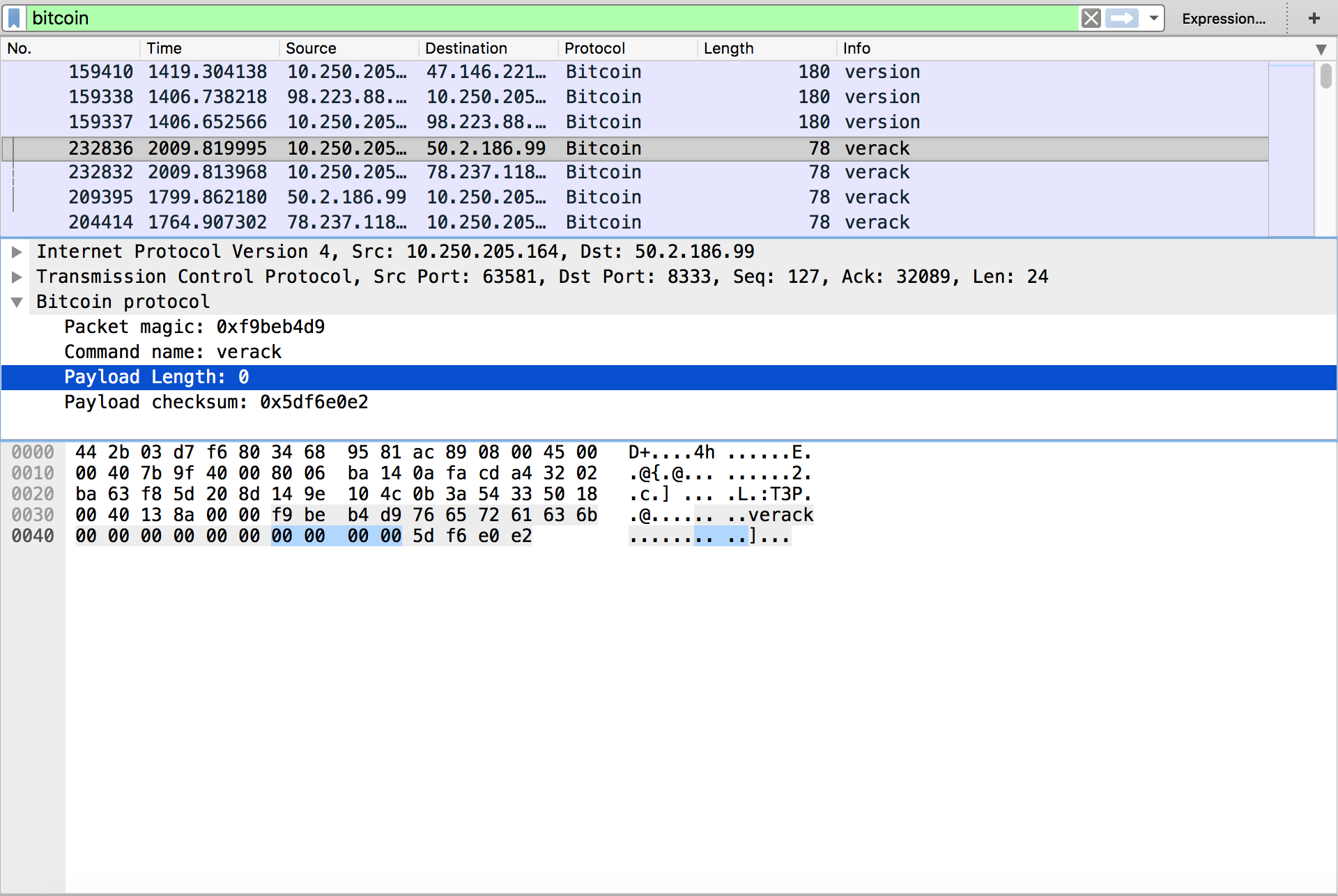
We use bitcoin.block.version filter to see the version packets captured





**Ver Ack:**

It is the acknowledge message sent for the version message, informing the transmitting node that it can begin to send other messages. It has no payload, therefore occupies less space.



**Ping message:**

The name itself reflects the meaning of the ping message. Ping message is sent to confirm whether the receiving peer is connected in the network. Upon receipt of a TCP/IP error while sending a ping message, the node which is transmitting assumes that it is disconnected with the receiving node.



Fig: Ping message

**Pong message:**

Pong message is used to send a reply for a ping message to state that the ponging node is still connected to the pinging node. If a client doesn’t not respond in 20 minutes it will be automatically disconnected. The message format of ping and pong messages are similar.

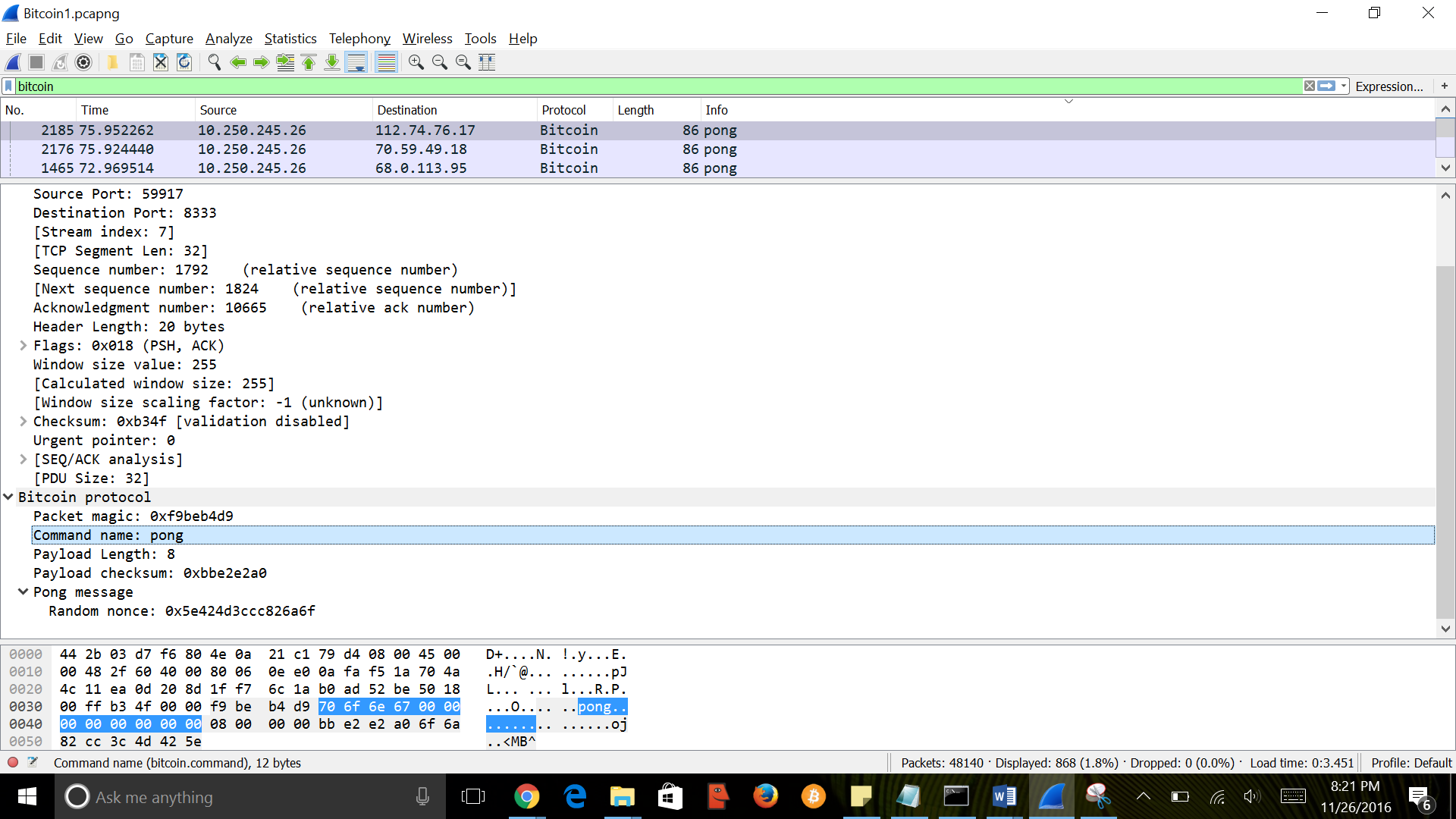


Fig: Pong message

**Protocol Components:**

All the integers in the message are encoded in little endian but only port number and IP are encoded in big endian. Below table represents the protocol message structure of bitcoin.

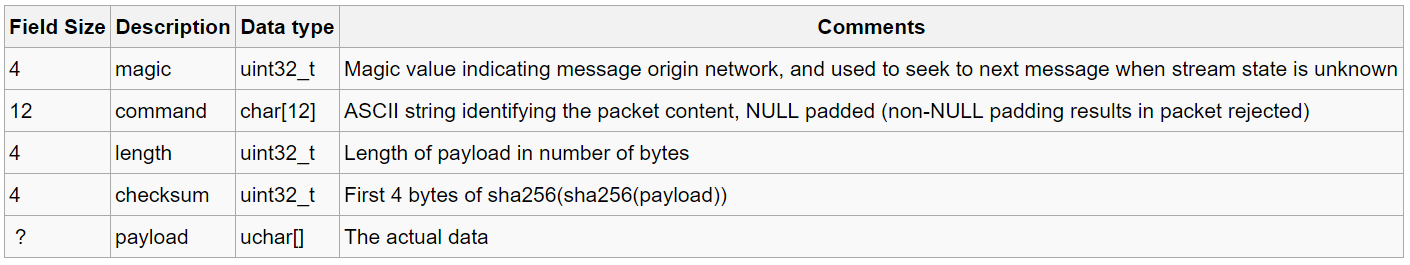


Fig: Protocol structure

The bitcoin protocol contains packet magic, Command which is sent in protocol, length of the payload, checksum value and is followed by actual payload data. Some of the packet magic values are stated below.



Fig: Packet magic values

The details of packet capture through wireshark are show in the below figure. The frame details, TCP protocol will be followed by bitcoin protocol. The packet magic value is sent over wire as 0xf9 be b4 d9 which represents the main network, the commands such as block, getdata, addr, ping, pong, ver, inv etc will be sent which is followed by corresponding payload length and checksum value and actual payload.

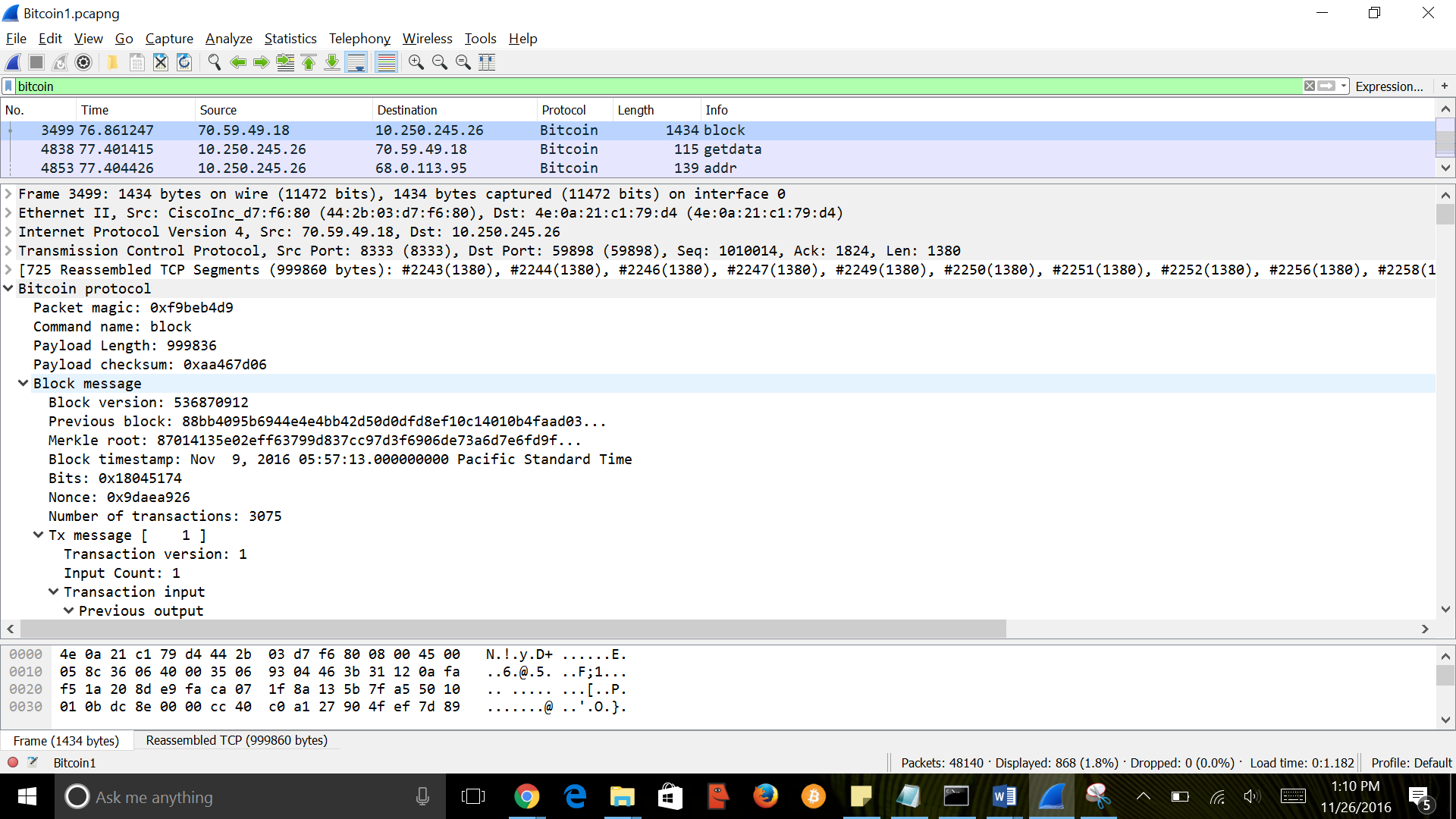


Fig: Bitcoin protocol packet capture

**Port details:**

The default port for mainnet is set to port 8333.



**The features Bitcoin Protocol depends upon:**

Consensus: When several nodes (usually most nodes on the network) all have the same blocks in their locally-validated best block chain.

**Block Header:**

An 80-byte header of a block which is hashed consecutive times to create a proof of work

**Proof of Work:**

The blockchain is maintained in an anonymous peer to peer network. So there has to be a mechanism which ensures that the blocks which are maintained will have more cost to modify. It is to ensure that untrusted peers need to work harder than other peers to modify the chain. When you chain blocks together, you have to modify all the blocks in order to modify a particular transaction.

Block headers:

4 bytes for version

32 bytes for previous block header hash

32 bytes for merkle root hash

4 bytes for time block

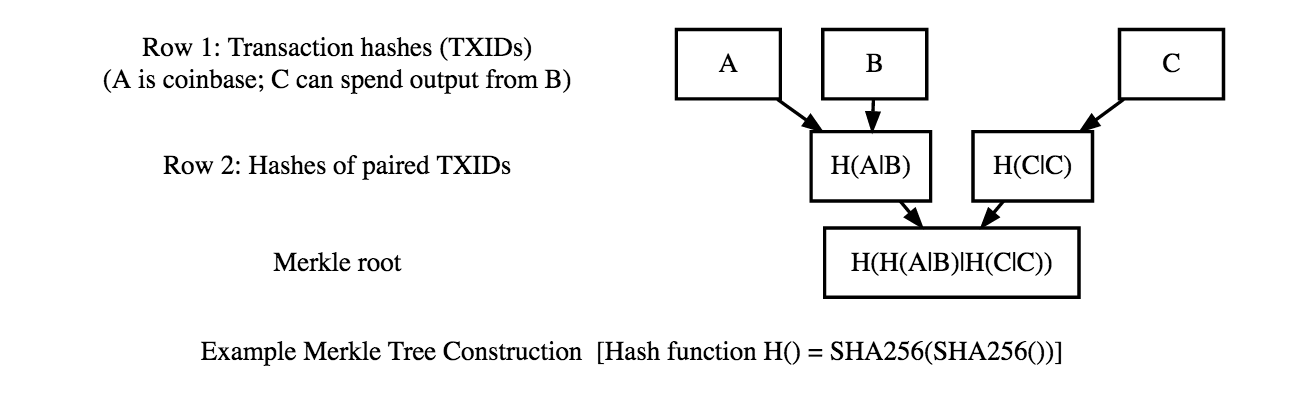
4 bytes for nBits

4 bytes for nonce

The merkel root is built using all the TXIDs of transactions in this [block](https://bitcoin.org/en/glossary/block). The arrangement of the transaction IDs are placed in order as required by the following consensus rules:

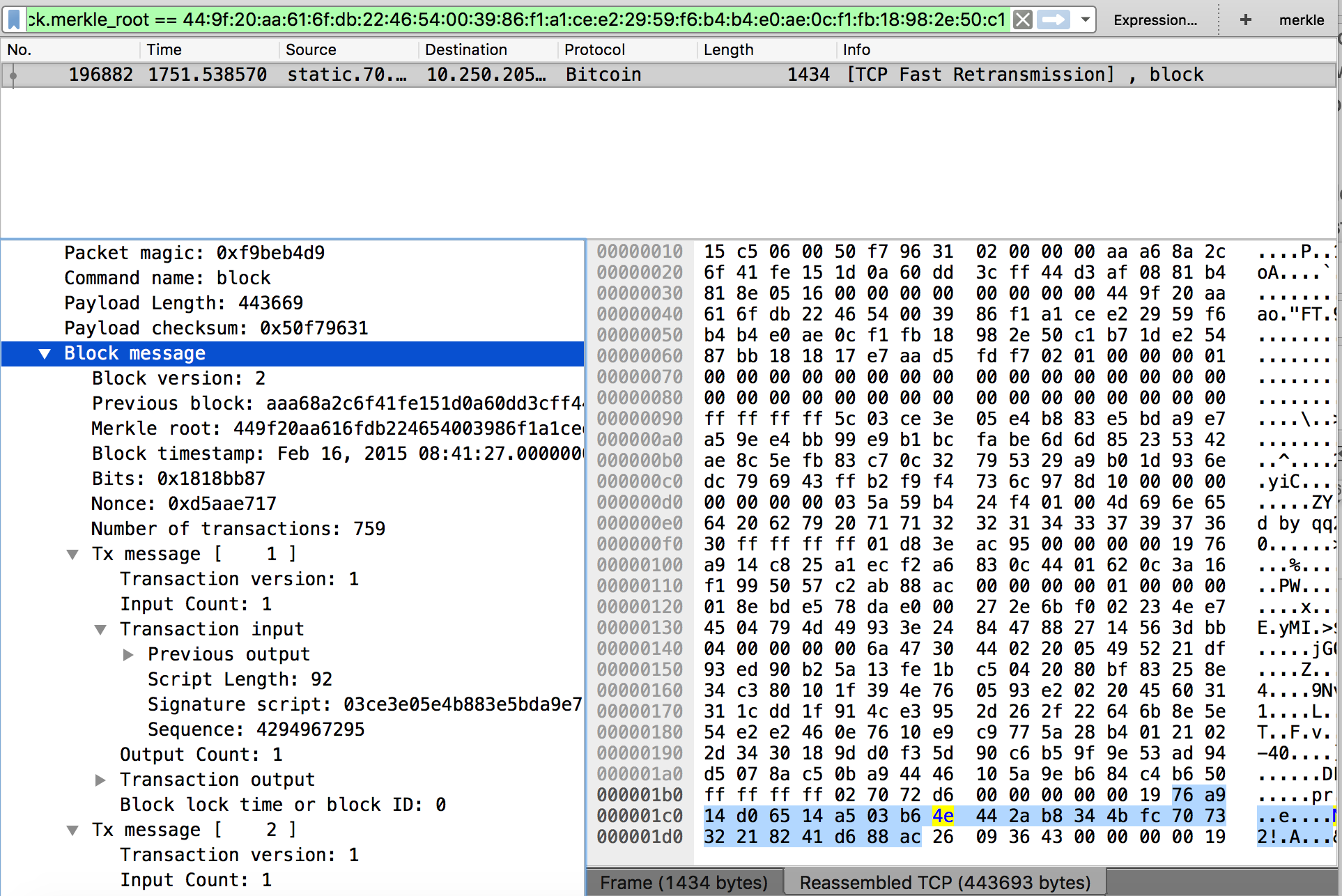
If a block only has a [coinbase transaction](https://bitcoin.org/en/glossary/coinbase-transaction), the [coinbase](https://bitcoin.org/en/glossary/coinbase) TXID is used as the [merkle root](https://bitcoin.org/en/glossary/merkle-root) hash.

If a [block](https://bitcoin.org/en/glossary/block) only has a [coinbase transaction](https://bitcoin.org/en/glossary/coinbase-transaction) and one other transaction, the TXIDs of those two transactions are placed in order, hashed together to form the merkle root.



(source: <https://bitcoin.org/en/developer-merkle-trees>)

nBits:  
Also called as target is the threshold below which a block header hash must be in order for the block to valid, and nBits is the encoded form of the target threshold as it appears in the block header.



**Operating modes and weaknesses**:

There are other packets like filter add which can be viewed in SPV mode (Simple Payment Verification). It is another alternative to the current set-up we implemented.

There are two modes of Operation:

Full node mode- Where blocks are downloaded all the way from the genesis block itself which is followed by bitcoin core. So the block depth here would be the genesis block formed in 2009. So to snoop in an invalid transaction the user has to give an alternative definition to the full chain which is computationally expensive if not impossible.

Simple Payment Verification Mode:

Download only the block headers initially and then download full blocks upon request

The SPV client knows the merkle root and associated transaction information, and requests the respective merkle branch from a full node. Once the merkle branch has been retrieved, proving the existence of the transaction in the block, the SPV client can then look to block depth as a proxy for transaction validity and security.

**Attacks and contingencies:**



RED: Vulnerable GREEN: Protected

**Direct Theft:**

The owner of a bitcoin bank site can disappear with all the user’s money because the users can’t control their own private keys

**Lightweight (SPV) wallet** users and **Bitcoin Core** users are not vulnerable because they control their own private keys.

Bait and Switch:

If you install a bitcoin wallet whose open source code is audited but the program source code isn’t. This makes the users vulnerable to theft as the application managers can push code to steal all the bitcoins.

A bitcoin bank site users are vulnerable because they can only spend their bitcoins when they use the bank’s approved software. Users are vulnerable with most software because auditors can’t easily verify the software you run (the executable) is the same as the program source code, called a deterministic build.

**Fabricated Transactions**: **Bank** users depend on the information reported by the bank, so they can be manipulated into accepting fabricated transactions.

**Lightweight (SPV) wallet** users depend on full nodes and miners to validate transactions for them. If a full node is itself belonging to a dishonest user then the cost of bringing up a fabricated transaction is zero.

**Bitcoin Core** has an advantage of validating every transaction before displaying it on the block so it’s protected.

**Chain Hijacking:**

Users have to use whatever blockchain the bank uses. So when banks and lightweight spv clients accidentally create a new false block the user’s node might accept them putting their bitcoins at risk.

Transaction Withholding:

Bank can choose what transactions it has to show to the users. This is true even in the case of SPV clients where full nodes can only send whatever they choose to send irrespective of the request. This makes the user think that the transaction didn’t happen or it did.

Chain rewrites:

Every mechanism is vulnerable to this attack. But the more confirmations a transaction in a block receives the more proof of work it gets, and it’s even more difficult to rewrite it. So we can say it’s computationally difficult if not impossible to rewrite the entire chain.

**QoS:**

Bitcoin Core includes a script for Linux to configure QoS on an individual host. Windows users can also use third-party software such as Netbalancer to throttle the application's upload bandwidth and ensure that one has enough upload bandwidth available for regular computer and internet use to be unaffected.

**Findings:**

* Bitcoin is a well-documented protocol.
* Manages synchronized records all over the network.
* Bitcoin provides a secured transaction in a distributed network
* Built in support for encryption and strong authentication.

**Learning Outcomes:**

The following were the outcomes of the project:

The quality of network directly influences the quality of service of the block chain. The latency of a network results in a temporary fork (temporary accepting the new version of the block).

Adding a new node is easy as it has to join a random subset of nodes; each node attempts to make 8 outgoing connections and is prepared to receive up to 125 incoming connections.

Bitcoin achieves network discovery through the use of dedicated directory servers or “seed nodes,” the values are hardcoded into the reference client; thereafter, each node maintains a list of peer addresses it knows about.

Nodes send INV messages to all of their peers containing the hashes of new blocks or pending transactions whenever they first hear of them. Peers can respond by requesting the full contents of these blocks or transactions if they have not yet seen them (via a GETDATA message).

By default, nodes will only forward new data once, preventing infinite propagation; only relay transactions and blocks that are valid; only relay the first block they hear of when two blocks are found in a temporary fork; and will not broadcast pending transactions which conflict (double-spend) with pending transactions they have sent.

**What does it mean?**

Bitcoin can be stated as a distributed database network which keeps a record of continuously growing data/transaction in blocks which are not allowed to be tampered/revised. Each block contains timestamp and link to the previous block. A single block can be considered as a code unit which is comprised in block chain.

It is the underlying technology for the bitcoin (A digital asset/payment system) implementation, which makes it more secure and decentralized (Not requiring a trusted administrator). There are few other alternative currencies which have bitcoin as base code such as litecoin, PPcoin, namecoin and Ripple.

**Conclusion:**

The scope of the project is to know how the bitcoin protocol operates and what type of network is necessary. It is a decentralized, ad hoc peer-to-peer broadcast network used to announce new transactions and proposed blocks. The other functional components like the blockchain and the validation rules form the most innovative part of the bitcoin. We can use blockchain and extend its functionality; for example, to enable a security feature only when validated by multiple parties. We can also use the decentralization and smart contracts to eliminate supervising parties.

**References:**

1. <https://bitcoin.org/en/developer-reference#message-headers>
2. <https://en.bitcoin.it/wiki/Protocol_documentation>
3. http://bitcoinsimplified.org/
4. <https://en.bitcoin.it/wiki/Main_Page>
5. https://www.weforum.org/agenda/2015/11/how-will-blockchain-technology-transform-financial-services/