

Investigating and Analyzing Bitcoin Blockchain Protocol using Wireshark

Auqib Hamid Lone

Department of Computer Science and Engineering, NIT Srinagar, Jammu and Kashmir, India, 190006 E-mail: auqib92@gmail.com

Roohie Naaz Mir

Department of Computer Science and Engineering, NIT Srinagar, Jammu and Kashmir, India, 190006 E-mail: naaz310@nitsri.net

Received: 01 March 2018; Accepted: 07 June 2018; Published: 08 July 2018

Abstract—A bitcoin node needs to download the full block contents of the entire blockchain, before actually being able to send and receive transactions on bitcoin broadcast network, except simple payment verification clients which require only block headers and bloom filters to sync with others peers available on the network. Transactions/Blocks pass through a complex process at sender and receiver than it apparently looks to be. During transmission transactions/blocks are broken down into smaller chunks of data so that they can be carried on the wire. These chunks are given appropriate headers, encapsulated and then passed through several layers to reach the destination. In this paper we captured Bitcoin packets using Wireshark and deeply investigated and analyzed them. We investigated how bitcoin transaction/block messages work and what values and parameters are considered during this whole process.

Index Terms-Bitcoin, Blockchain, Wireshark.

I. INTRODUCTION

In recent years, there has been an immense growth of e-commerce across the world. Most of the internet transaction mediating systems like PayPal rely on the trusted third-Party mediators - banking institutions. These trusted third-party mediators are responsible for verification and authentication of e-transactions and thus provide a sense of assured security to its users. Unfortunately, all of these trusted third-party mediators have one severe disadvantage of having "Central Point of Failure". To overcome shortcomings of trusted thirdparty mediators, a concept of *Bitcoin: A Peer to Peer ecash* was proposed by Satoshi Nakamoto in 2008 [1].

Bitcoin has revolutionized the whole financial transaction system, by introducing the very first fully decentralized digital currency, that is both secure and stable. Bitcoin can be defined in various ways; it's a protocol, a digital currency, and a platform. It is a combination of peer-to-peer network, protocols, and software that facilitates the process of creation and control of the digital currency.

The bitcoin network is a P2P network [2] where nodes exchange transactions and blocks. Wireshark can be used to visualize these transactions and blocks and thus can serve as an invaluable tool to get deep insight of the Bitcoin protocol. In this paper we capture the Bitcoin packets to deeply investigate and analyze them. We investigate how bitcoin transactions and block messages work, and what values and parameters are considered during this whole process.

The rest of the paper is organized as follows: In section II, the Bitcoin blockchain protocol is explained. In section III, basic setup methodology for experimentation is presented. In section IV, the bitcoin packets are investigated using Wireshark. Finally, the conclusion is given in section V.

II. BITCOIN BLOCKCHAIN PROTOCOL

There are many possible ways to define the Bitcoin, one way is to think of Bitcoin as a *protocol* governing the overall communication on the Bitcoin network, the other possible way is to think of it as a *digital currency* which is purely P2P electronic cash, allowing payments to be sent directly, bypassing any intermediary financial institutions. In more general sense Bitcoin represents a platform comprising of P2P network, protocols and a software that eases the creation and usage of digital currency named bitcoin.

Bitcoin uses Elliptic Curve Cryptography(ECC) [3,4]based on SECP256K1 standard. Private keys are a random 256 bit numbers, mostly encoded using Wallet Import Format(WIF). Public keys are basically x and y coordinates on a elliptic curve. Public keys can be presented in compressed or uncompressed form with 33 and 65 bytes of length respectively. A bitcoin address is actually the hash of a public key, computed as under [5].

Algorithm 1: bitcoin address generation

Input: Public Key, Version

Output: bitcoin address: 26-35 alphanumeric characters.

- 1 Calculate the SHA256 hash of Public key;
- 2 Calculate RIPEMD-160 hash of output of the step1;
- 3 Key hash = Concatenation of the Version and output of step 2;
- 4 Calculate SHA256 two times of Key hash of step3;
- 5 Checksum = 1st 4 bytes of the output of step 4;
- 6 Concatenate Key hash with Cheksum;
- 7 Encoding the output of step 6 with Base58 encoding scheme;
- 8 Return output of step 7;

Version = 1 byte of 0 (zero); on the test network, this is 1 byte of 1

The heart and soul of Bitcoin is Blockchain: Technology that makes bitcoin a fully decentralized digital currency, which is both stable and secure. Blockchain can be defined as a immutable decentralized public ledger, which is both timestamped and ordered. Each block on the chain is identified by a hash and is linked to its previous block by referencing the previous block's hash.

A. Bitcoin Blockchain core Components

a. Block and Block Header

Block is one of the important and driving component of Bitcoin Blockchain. Table 1 and Table 2 presents the structure of a block and block header with size and brief description of each entry in the the block and block header respectively.

Table 1. Structure of a Block

Bytes	Name	Description
80	Block header	This includes fields from the block header.
variable	Transaction counter	The field contains the total number of transactions in the block, including the Coinbase transaction.
variable	Transactions	All transactions in the block.

Table 2. Structure of a Block Header

Bytes	Name	Description
4	Version	The block version number that dictates the block validation rules to follow.
32	previous block header hash	This is a double SHA256 hash of the previous block's header.
32	Merkle root hash	This is a double SHA256 hash of the Merkle tree of all transactions included in the block.
4	Timestamp	This field contains the approximate creation time of the block in the Unix epoch time format. More precisely, this is the time when the miner has started hashing the header (the time from the miner's point of view).
4	Difficulty Target	This is the difficulty target of the block.
4	Nonce	This is an arbitrary number that miners change repeatedly in order to produce a hash that fulfills the difficulty target threshold.

b. Bitcoin as a State Transition System

Bitcoin platform can be thought of as a state transition system, comprising of *state* and a *statet ransaction function*. *State* consists of ownership status of all bitcoins and *transition function* takes state and transaction as input and outputs a new state [6].

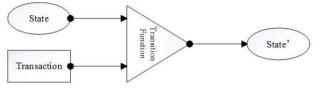


Fig.1. Bitcoin as a State Transition System.

The *state* in Bitcoin is the collection of all coins (technically, "unspent transaction outputs" or UTXO) that have been minted and not yet spent, with each UTXO having a denomination and an owner (defined by a 20-byte address which is generated as shown in algorithm 1). A transaction contains one or more inputs, with each input containing a reference to an existing UTXO and a cryptographic signature produced by the private key of the owner, and one or more outputs, with each output containing a new UTXO to be included in the state [7].

Transition function can be formally written as: $APPLY(S,TX) \rightarrow S$ or Error defined as follows:

- 1. For each input in TX:
- If the referenced UTXO is not in S, return an error.
- If the provided signature does not match the owner of the UTXO, return an error.
- 2. If the sum of the denominations of all input UTXO is less than the sum of the denominations of all outputUTXO, return an error.
- 3. Return S' with all input UTXO removed and all output UTXO added.

c. Bitcoin Mining

Mining in Bitcoin is a process through which transactions within a block are validated and blocks are added to the blockchain. Mining function is performed by special nodes called as *miners*. Mining is resource intensive process in a way as enough CPU and battery power is spent for mining a block. Roughly one new block is created (mined) every 10 minute. Miners are rewarded with new coins if and when they create new blocks and are paid transaction fees in return of including transactions in their blocks. New blocks are created at an approximate fixed rate. Also, the rate of creation of new bitcoins decreases by 50%, every 210,000 blocks, roughly every 4 years.

The major advantage of mining is that it secures the bitcoin system by guarding against frauds and double spending. Algorithm 2 presents the steps required for successfully mining a block.

Algorithm 2: The Mining Algorithm

- 1. The hash of the previous block is retrieved from the bitcoin network;
- 2. Collect the set of potential transactions broadcast on the network into a block ;
- 3. Compute the double hash of the block header with a nonce and the hash from step 1 using SHA256 algorithm;
- 4. If the result of step 3 is lower than the current difficulty target then stop the process;
- 5. Result of step 3 is greater than the current difficulty target then repeat the process by incrementing the nonce;

d. Proof of Work

As name suggests it is the proof that enough computational resources have been spent to build the valid block. Proof of Work(PoW) involves searching for a value that when hashed, such as with SHA256, the resulting hash begins with number of zero bits or is less than a specific target. The following equation sums up the Proof of Work requirement in bitcoin:

 $H(N||P_{hash}||Tx||Tx||...Tx) < Target$

Where N is a nonce, P_{hash} is a hash of the previous block, Tx represents transactions in the block, and Target is the target network difficulty value. This means that the resulting hash should be less than the target hash value. The only way to find this nonce is the brute force method.

III. METHODOLOGY

The bitcoin network is a P2P network where nodes exchange transactions and blocks. Analyzing contents of transactions, blocks and messages exchanged by peers requires capturing them, while nodes communicate with each other. For experimentation purposes, we installed Bitcoin Core(Testnet) client running on Testnet3 network [8]. The reason for using Testnet is that it allows application developers or bitcoin testers to experiment, without having to use real bitcoins or worrying about breaking the main bitcoin chain. There are few notable differences between bitcoin mainnet and testnet which are summarized in table 3.

Table 3. Bitcoin Mainnet and Testnet	Table 3.	Bitcoin	Mainnet	and	Testnet
--------------------------------------	----------	---------	---------	-----	---------

Mainnet	Testnet
The real Bitcoin network, with the block chain that everyone uses.	test network that runs in parallel with mainnet, except that the value of these coins is negligible.
Costs real money not only to buy bitcoins but also as transaction fees	Essentially free to acquire testnet bitcoins.
Bitcoin protocol works on TCP port 8333 for the main network	Works on TCP port 18333 for testnet.

For capturing Bitcoin packets, we used network protocol analyzer called Wireshark[9,10].Wireshark allowed us to visualize messages exchanged between peers and thus served as an invaluable tool to deeply investigate about the Bitcoin protocol contents. We used [11] website to recieve some free coins and then transfer some coins back to it and record the whole process for investigation with Wireshark.

IV. INVESTIGATING BITCOIN PROTOCOL USING WIRESHARK

Typically a full bitcoin node can perform four functions: wallet, miner, blockchain and network routing node. The first thing that a bitcoin core node does when it starts up is initiation of discovery of all peers. This is achieved by querying DNS seeds that are usually hardcoded into the bitcoin core client and are maintained by bitcoin community members. This lookup returns a number of DNS A records. The process of node discovery, block or header exchanges and other network functionality are summarized below:

- 1. Client sends a protocol message *Version* containing various fields such as version, services, timestamp, network address, nonce and some other fields.
- 2. Remote node responds with it's own version.
- 3. After exchanging *version* messages both nodes then exchange *verack* messages, indicating that connection has been established.
- After connection establishment *Getaddr* and *addr* messages are exchanged to find about unknown peers. Nodes can test liveliness of connection by sending ping messages. Active nodes respond with pong messages.
- 5. Now node can start downloading blocks to sync with network using *GetBlock* and *GetData* messages. If the node already has all blocks fully synchronized then it listens for new blocks using the *Inv* protocol message.



Fig.2. Bitcoin Node Discovery until version 0.9.3.

Until bitcoin core client version 0.9.3, a method called blocks-first was used for synchronization by nodes. The blocks-first method is very slow and some time takes days to complete synchronization. Consequently it was discontinued from bitcoin core client version 0.10.0. New initial block download(IBD) method named headers-first was introduced from client version 0.10.0. The headersfirst method drastically reduces synchronization time to few hours. In headers-first method, when the client starts up, it checks whether the blockchain is fully synchronized already if the header chain is already synchronized; if not, which is the case the first time the client starts up, it requests headers from other peers using the getHeaders message. If the block chain is fully synchronized, it listens for new blocks via Inv messages, and if it already has a fully synchronized header chain, then it requests blocks using Getdata protocol messages. The IBD node also checks whether there is trade-off between headers and blocks: if headers are more in header chain: then it requests blocks by issuing the Getdata protocol message.

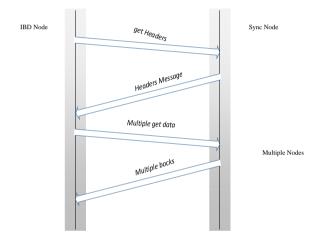


Fig.3. Header and Block Synchronization in Bitcoin Core Client>= 0.10.0.

39.31.104.14		10.10.10.17	Time
18333	version	50049	9.172562
18333	version	50049 🔫	9.446826
18333	verack	50049 🖛 🛶	9.446827
• 18333	verack	50049	9.447017
18333	getaddr	50049	9.461344
18333	[unknown command]	50049	9.581228
18333	[unknown command]	50049	9.581470
18333	[unknown command]	50049	9.581628
18333	ping	50049	9.589432
18333	[unknown command]	50049	9,725346
18333	[unknown command]	50049	9,725349
18333	[unknown command]	50049	9,725350
18333	ping	50049	9.725352
18333	addr	50049	9,725353
18333	getheaders	50049	9.725833
18333	[unknown command]	50049	9,725840
18333	addr	50049	9.811151
18333	getheaders	50049	9.811443
18333	[unknown command]	500.05	9.812228
	pong	50049	9.812228
18333	headers	50049	9.812573

Fig.4. Flow graph between two Peers on Bitcoin Network.

A. Analyzing Bitcoin Protocol Messages in Wireshark

Bitcoin dissector[12] plays an important role in analyzing the traffic and identification of Bitcoin protocol commands. Bitcoin dissector in Wireshark also provides valuable information such as the packet type, command name, and results of the protocol messages. Figure 4 provides protocol graph showing the flow of data between the two peers, providing clear picture on what messages are exchanged when a nodes starts up.

At present there are 27 types of protocol messages in total, but most likely they will increase over time as the protocol grows [4]. Table 4 lists the main ingredients of Bitcoin protocol messages with short description of each.

Table 4. Structure of a Bitcoin Protocol Message	e
--	---

Field Name	Bytes	Description
magic	4	Value indicating message origin network
command	12	String identifying messagecontent.
length	4	Length of payload in number of bytes
checksum	4	First 4 bytes of sha256(sha256(payload)).
payload	variable	The actual data.

Co Pi	acke omma aylo aylo	and ad	nan Ler	ie: igth	ver i: 1	sic 02	n	7 7 7 7 8	43								
⊳ v	ersi	on	mes	sag	e												
0000	00	04	96	1e	08	b0	18	66	da	Øa	48	8e	08	00	45	00	fHE.
0010	00															1f	@ #Y.
0200	68															18	hG6 {h.G.[P.
0600	01															00	@
0040	00															00	Z.Y
0050	00															1f	Y.
0000	68															00	h.G
070	00															ad	
0800	Зc															Зa	<:/ Satoshi:
0000	30											01					0.14.2/. ?

Fig.5. Bitcoin Protocol Message in Wireshark.

a. Version

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

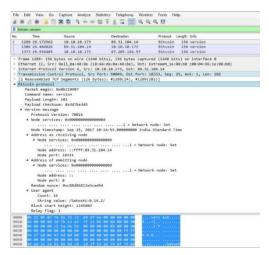


Fig.6. Version message in Wireshark.

Field Name	Bytes	Description
version	4	Identifies protocol version being used by the node.
services	8	bitfield of features to be enabled for this connection.
timestamp	8	standard UNIX timestamp in seconds.
addr_recv	26	network address of the node receiving this message.
addr_from	26	network address of the node emitting this message.
nonce	8	Randomly generated every time a version packet is sent and is to detect connections to self.
user_agent	variable	User Agent (0x00 if string is 0 bytes long).
start_height	4	The last block received by the emitting node.
relay	1	Whether the remote peer should announce relayed transactions or not

If version packet is accepted, *verack* packet will be sent.

b. Verack

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

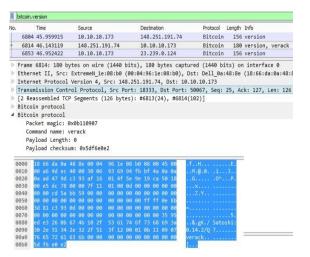


Fig.7. Verack Message in Wireshark.

c. Addr

This message is used by bitcoin nodes to provide information on known nodes of the network. If nodes don't advertise for 3-hours they are automatically forgotten. Table 6 summarizes the contents of payload of *addr* message.

Table 6. Payload Contents of addr Message

Field Name	Bytes	Description
count	More than 1	Number of address entries (max: 1000).
address list	Multiple of 30	Address of other nodes on the network.

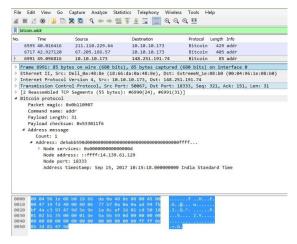


Fig.8. Addr Message in Wireshark.

d. Inv

This message is used by nodes to advertise their knowledge of one or more objects. It can be received unsolicited or in reply to *getblock* message. Payload of this message can have upto 50,000 entries corresponding to approximately 1.8 megabytes. Table 7 summarizes the contents of payload of *inv* message.

Table 7. Payload Contents of inv Message

Field Name	Bytes	Description
count	variable	Number of inventory entries.
inventory	Multiple of 36	Inventory vectors.

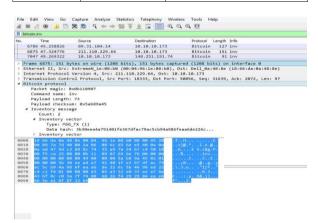


Fig.9. Inv message in Wireshark

e. Getdata

This message is used in response to *inv* for retrieving the contents of specific object. Usually *getdata* is sent after receiving *inv* packet, after filtering known objects. Payload of this message can have upto 50,000 entries corresponding to approximately 1.8 megabytes. Table 8 summarizes the contents of payload of *getdata* message.

Table 8. Payload Contents of Getdata Message
--

Field Name	Bytes	Description
count	variable	Number of inventory entries.
inventory	Multiple of 36	Inventory vectors.

	on.getdata			
-				
lo.	Time	Source	Destination	Protocol Length Info
	289 39.286774	10.10.10.173	89.31.104.14	Bitcoin 91 getdata
	806 45.966382	10.10.10.173	78.46.68.181	Bitcoin 163 getdata
65	986 49.092299	10.10.10.173	89.31.104.14	Bitcoin 91 getdata
Eth	ernet II, Src:	Dell_0a:48:8e (18:	66:da:0a:48:8e), Dst:	ExtremeN_1e:08:b0 (00:04:96:1e:08:b0)
			.10.10.173, Dst: 78.4	
				18333, Seq: 2067, Ack: 1716, Len: 109
		Segments (133 byt	es): #6805(24), #6806	[109)]
	coin protocol			
	Packet magic: 0			
	Command name: g			
	Payload Length:			
	Payload checksu			
	Getdata message			
	Count: 3			
	Inventory ve			
	Type: Unk	nown (1073741825)		
	Data hash		367dfac79ac5cb94a9861	eaa6de226c
	Data hash 4 Inventory ve	ctor	367dfac79ac5cb94a9861	eaa6de226c
	Data hash 4 Inventory ve Type: Unk	ctor nown (1073741825)		
	Data hash 4 Inventory ve Type: Unk Data hash	ctor nown (1073741825) : 1e4ab2e22caae015	367dfac79ac5cb94a9861	
	Data hash Inventory ve Type: Unk Data hash Inventory ve	ctor nown (1073741825) : 1e4ab2e22caae015 ctor		
	Data hash Inventory ve Type: Unk Data hash Inventory ve Type: Unk	ctor nown (1073741825) : 1e4ab2e22caae015 ctor nown (1073741825)	655f446d67fd55aa3270a5	bflc6c50dl
	Data hash Inventory ve Type: Unk Data hash Inventory ve Type: Unk	ctor nown (1073741825) : 1e4ab2e22caae015 ctor nown (1073741825)		bflc6c50dl
0000	Data hash Inventory ve Type: Unk Data hash Inventory ve Type: Unk Data hash	ctor nown (1073741825) : 1e4ab2e22caae015 ctor nown (1073741825) : d305a352e033eea78	365f446d67fd55aa3270a 3e43bf8cc03a7f79806826	bflc6c50d1 f4292984ea
	Data hash Inventory ve Type: Unk Data hash Inventory ve Type: Unk Data hash 00 04 95 1e 08	ctor nown (1073741825) : 1e4ab2e22caae0155 ctor nown (1073741825) : d305a352e033eea78 b0 18 66 da 0a 4	655f446d67fd55aa3270a5 8e43bf8cc03a7f79806826 8 8c 05 00 45 00	bf1c6c50d1 f4292984ea
0010	Data hash 4 Inventory ve Type: Unk Data hash 4 Inventory ve Type: Unk Data hash 500 64 96 1c 98 600 95 14 19 40	ctor nown (1073741825) : 1e4ab2e22caae0155 ctor nown (1073741825) : d305a352e033eea73 : b0 18 66 da 0a 4 : 00 80 06 3e b0 0	565f446d67fd55aa3270a5 3e43bf8cc03a7f7980682c 8 8c 05 00 45 00 9a 03 ad 4c 2c	bf1c6c50d1 f4292984ea
0010 0020	Data hash Inventory ve Type: Unk Data hash Inventory ve Type: Unk Data hash 06 64 96 1e 08 08 95 14 19 48 44 b5 c3 87 47	ctor nown (1073741825) : 1e4ab2e22caae0155 ctor nown (1073741825) : d305a352e033eea78 b0 18 66 da 0a 4	565f446d67fd55aa3270a5 3e43bf8cc03a7f79806826 8 8e 08 00 45 00 1 04 09 ad 4e 2e 1 7f bb 42 50 18 0.	bf1c6c50d1 f4292984ea
0010 0020 0030	Data hash Inventory ve Type: Unk Data hash Inventory ve Type: Unk Data hash 06 64 96 1e 08 08 95 14 19 48 44 b5 c3 87 47	ctor nown (1073741825) : 1=4ab2e22caae015! ctor nown (1073741825) : d305a352e03aea76 b0 18 66 da 0s 4 06 88 06 3e b0 6 9d 80 63 3e b0 c 2 0 0 03 01 00 00 4	565f446d67fd55aa3270a5 3643bf8cc03a7f7980682c 8 Be 98 00 45 00 1 00 08 ad 4e 2e 1 7f bb 42 59 18 0. 9 3b 30 ee ad a7	bfic6c50d1 f4292984ea g g
0010 0020 0030 0040	Data hash Inventory ve Type: Unk Data hash Inventory ve Type: Unk Data hash Noventory ve Type: Unk Data hash 00 04 95 10 03 00 51 12 40 10 257 06 08 91 40 1f 63 67 42 65 55 46	ctor nowm (10737741825) i 1e4ab2e22caae015 ctor i 3d95a52e033eea77 b0 18 66 da 0a 4 00 80 96 3e b0 0 9d 80 27 a0 cc 2 9d 80 27 a0 cc 2 9d 80 27 a0 cc 4 60 63 01 06 06 4 df ac 79 ac 5c b	565f446d67fd55aa3270a5 3e43bf8cc03a7f7980682c 8 8e 08 00 45 00 1 8 08 08 ad 4c 2e 1 7f bb 42 50 18 0 3b 30 ee ad a7 9 4a 98 6f ee ad a7 0 30 00 00 40 1e	612665901 f4292984ea 6
0010 0020 0030 0040 0050 0060	Data hash # Inventory ve Type: Unk Data hash # Inventory ve Type: Unk Data hash %6 43 96 1e 07 66 95 14 19 46 44 05 c3 87 47 61 92 67 0e 66 91 48 1f c3 67 62 26 c5 b4 66 64 52 22 cc 35	ctor nown (1073741825) ; 1e4ab2e22caae015; ctor nown (1073741825) ; d305a352e033eea73 b0 18 66 da 0a 4 60 80 60 5a b0 6 94 80 27 80 cc 2 60 03 01 00 00 4 97 ac 79 a0 cc 2 60 6 30 1 00 00 4 97 ac 79 ac 5c b 98 ed 22 c4 c1 f 98 64 22 64 c4	565f446d67fd55aa3270a5 5643bf8cc03a7f79806826 5 0e 06 00 45 00 a 0a 0a ad 4e 2e 1 7f bb 42 50 18 0 35 0 ee a4 a7 b 35 30 ee a4 a7 4 03 00 00 40 1e 4 03 00 00 40 1e	bf1c6c50d1
0000 0010 0020 0030 0040 0050 0050 0060 0070	Data hash # Inventory ue Type: Unk Data hash # Inventory ue Type: Unk Data hash 000 04 96 1e 00 00 95 14 19 40 04 4 b5 c3 87 47 01 02 67 0e 00 91 40 1f e3 67 49 52 c2 2c as 49 62 c2 b6 1c 66	Ctor nown (1073741825) ; 1e4eb2e2caae015 Ctor nown (1073741825) ; d305a3552e033eea72 ; d305a3552e033eea72 ; d305a3552e033eea72 ; d305a352e033eea72 ; d305a352e033eea72 ; d305a352e032 ; d305a352e032 ; d305a42 ; d305a42	565f446d67fd55ma3270m1 1e43bf8cc03a7f798066226 1	bf1665801 f4292984ca f429.0984ca g, b, b, b, b, g, b, b, b, b, g, g, b, b, c, f, b, b, g, b, c, f, b, c,
0010 0020 0030 0040 0050 0060	Data hash J Inventory ve Type: Unk Data hash J Inventory ve Type: Unk Data hash On 04 06 1e 00 04 95 14 19 46 44 b5 c3 87 47 61 02 67 0e 06 14 01 1f e3 07 de 22 cc 5b 44 cb 22 c2 cc 70 a5 bf 1c 60 00 06 44 a3 03	ctor nown (1073741825) ; 1e4ab2e22caae015; ctor nown (1073741825) ; d305a352e033eea73 b0 18 66 da 0a 4 60 80 60 5a b0 6 94 80 27 80 cc 2 60 03 01 00 00 4 97 ac 79 a0 cc 2 60 6 30 1 00 00 4 97 ac 79 ac 5c b 98 ed 22 c4 c1 f 98 64 22 64 c4	665f446d67fd55m3270m5 8e43bf8cc03a7f7980682z 0 0 0 0 42 00 45 00 1 7 10 42 00 14 2 1 7 10 42 00 14 2 1 7 10 42 00 14 2 1 4 30 0 0 4 14 2 1 8 30 0 0 4 14 1 1 8 20 0 0 4 14 1 1 8 20 10 0 4 14 1 1 8 4 30 18 0 7 6 1 1 8 4 30 18 0 0	bf1c6c50d1

Fig.10. Getdata Message in Wireshark.

f. Notfound

This message is used in response to *getdata* when contents of specific object can't relayed. Table 9 summarizes the contents of payload of *notfound* message.

Table 9. Payload	Contents of Notfound	Message
------------------	----------------------	---------

Field Name count inventory			l Name Bytes							Description													
				variable					Number of inventory entries				ries.										
				Multiple of 36					6	Ir	vento	or	γv	ec	tor	s.							
1															eless Tor Q Q I		Hel	p					
	bitcoin.				_																		
No		Time				iourc		8.181				estina	ation			tco			Info				
h	7538									++>	01	har	*** (antur	ed /728 1	h.i.*.		. 10	tarf		0		
0.0	Frame Ether Inter	753 net	8: 9 II, 9 Prote	1 by Src: bcol	Ext Ver	on i rem	wiren_	: (72 Le:08	B bi :b0 : 78	(00:	04: 68.	96:	1e:08	:: 10.	ed (728 Dst: De 10.10.17	11_	3a:4	8:8e	(18	:66	:da:@		
	Frame Ether Inter Trans	753 net net miss	8: 9 II, 9 Prote	1 by Src: Dcol	Ext Ver	on i rem	wiren_	: (72 Le:08	B bi :b0 : 78	(00:	68.	96:	1e:08	:: 10.	Dst: De	11_	3a:4	8:8e	(18	:66	:da:@		
	Frame Ether Inter Trans Bitco Pa Co Pa	753 net miss in p cket yloa yloa tdat	8: 9 II, 1 Proto ion 0 mag: d nam d Len d che	1 by 5rc: ocol Contr Contr Col Ic: (me: r ngth: cksu ssage	tes Ext Ver nol 3x0b notf 37 um:	on i rem sion Pro 1109 ound	wire eN_i n 4 toco 907 d	e (72 Le:08 , Src ol, S	B bi :b0 : 78	(00:	68.	96:	1e:08	:: 10.	Dst: De	11_	3a:4	8:8e	(18	:66	:da:@		
	Frame Ether Inter Trans Bitco Pa Co Pa Pa d Ge	753 net miss in p cket yload yload tdati Cour Inve 18	8: 9 II, 1 Proto mag: d name d che a mes a m	1 by Src: col Contr col Lc: 0 me: r ngth: cksu ssage L 'y ve a 0a	tes Ext Ver rol 3x0b botf : 37 um: 2 ecto 48	on i rem sion Pro- 1109 ound 0xb2	wir eN n 4 toco 007 d 7a80	e (72 Le:08 , Src ol, S 07a4	8 bi: : b0 : 78 rc Pe	(00: .46. ort:	64: 68. 18	96: 181 333	1e:08 , Dst , Dst	::b0), :: 10. : Port	Dst: De 10.10.17 : 50055,	11_0 3 Se	9a:4	8:8e	(18	:66	:da:@		
	Frame Ether Inter Trans Bitco Pa Co Pa Pa d Ge	753 net miss in p cket yloa tdati Cour Inve 18 00	8: 9 II, 9 Proto mag: d nam d Len d chi a mes nt: 1 entor 66 d 4d e	1 by 5rc: col Contr col ic: 0 me: r ngth: sage 1 y ve a 0a 6 9a	tes Ext Ver rol 3x0b botf : 37 um: 2 ecto 48 40	on i rem sio Pro 0xb? c 8e 00	wir eN n 4 toco 907 d 7a80 7a80 900 30	e (72 Le:08 Srcol, Srcol, S	6 1e c 76	(00: .46. ort: 08 4e	64: 68. 18 b0 2e	96: 181 333 08 44	1e:08 , Dst , Dst , Dst	: b0), : 10. : Port	Dst: De 10.10.17 : 50055, .fH .M@.0	se	3a:4	8:8e	(18	:66	:da:@		
	Frame Ether Inter Trans Bitco Pa Co Pa Pa d Ge	753 net miss in p cket yloa yloa yloa tdat Cour Inve 18 00 0a	8: 9 II, 1 Prote ion (mag d nam d Len d chi a men t: 1 entor 66 d 44 e ad 44	1 by 5rc: cool Contr col Contr col col col col col col col col	tes Ext Ver rol 37 Jum: 48 40 c3	on i rem sio Pro 00xb 8e 8e 87	wird eN 907 d 7380 900 300 21	e (72 Le:08 Srcol, S 07a4 07a4 06 b 7f c	6 1e c 76 6 77	(00: .46. ort: 08 4e 80	64: 68. 18 b0 2e 27	96: 181 333 333 08 44 a3	1e:08 , Dst , Dst , Dst , Dst , Dst	: b0), : 10. Port	Dst: De 10.10.17 : 50055,	11_1 3 Se	9a:4	8:8e	(18	:66	:da:@		
	Frame Ether Inter Trans Bitco Pa Co Pa Pa d Ge	753 net miss in p cket mman yloa tdat Cour Inve 00 00 00 55	8: 9 II, 1 Proto roto mag d nar d Ler d chi a men t: 1 entor 66 d 4 4 e ad 4 fb 8 ad f	1 by Src: cool Contr col ic: (me: r mgth: ecksu ssage 1 'y ve a 0a 6 9a 7 9d 0 477 4 bd	tes Ext Ver rol 3x0b notf : 37 	on i rem sion Pro 00000 00000 00000 00000 880 000 830 830	wirren	e (72 Le:08 , Src Dl, S 07a4 84 9 84 9 86 b 97 b 1 0 81 9 81 9 81 9 81 9 81 9 81 9 81 9 81 9	6 1e c 76 6 77 0 00 9 26	(00: .46. ort: 08 4e 80 40 1d	64: 68. 18 b0 2e 27 c8	96: 181 333 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8	00 45 00 45 05 00 3c 50 da 21	: b0), : 10. : Port : Port : 00 : 00 : 00 : 03 : 51	Dst: De 10.10.17 : 50055, .fH .M@.0	11_13 Se	vN.C w.'. .@.e	.E.	(18	:66	:da:@		

Fig.11. Notfound Message in Wireshark.

g. Getblocks

This message is sent by a node requesting other node to return *inv* packet containing list of blocks starting from the last known hash in the block locator object to stop_hash or 500 blocks whichever is earlier. Table 10 summarizes the contents of payload of *getblocks* message.

Field Name	Bytes	Description
version	4	The protocol version.
hash count	1 or more	Number of block locator hash entries.
Block locator hashes	32 or more	Block locator objects from newest block to genesis block.
hash_stop	32	Hash of the last desired block. Set 0 to get as many blocks possible (500).



Fig.12. Getblocks Message in Wireshark.

h. Getheaders

This command is used by thin clients to quickly download the block chain to sync with the network. The response to *getheaders* message is a header packet containing the headers of the block starting right after the last known hash in the block locator object, hash stop or 2000 blocks, whichever comes first. Table 11 summarizes the contents of payload of *getheaders* message.

Table 11. Payload Contents of Getheaders Message

Field Name	Bytes	Description
version	4	The protocol version.
hash count	1 or more	Number of block locator hash entries.
Block locator hashes	32 or more	Block locator objects from newest block to genesis block.
hash _stop	32	Hash of the last desired block. Set 0 to get as many blocks possible (2000).

Getheaders message	
Block version:	70014
Count: 32	
Starting hash:	32fe63e49c19488fa125e0d9bcb473164c7b7e94a39de58f
Starting hash:	262d326a4ce0d7bd311f38612bde1760e1a3c9cde0de652a
Starting hash:	873b4fc90d34da01c0cc769b78c2553a16f101bf9076a6ae
Starting hash:	2e6281e22571fe3479a94a2e41155ab966ae2150b2fef0cd
Starting hash:	a317db1f29d7cfe2483851ecb390df31c30532f10bd23ef8
Starting hash:	82f33e58bf96fd466a127c14b747c49d07ec4caa5a676df3
Starting hash:	12f919162aacfdb46de8174717348cfa438ebc43e520780b
Starting hash:	bafa60ea3d583a3623c7b5460a7a0a973595ba43e33bd826
Starting hash:	d78ca4ad8fec0bdf56a6c156972f478b30b38d186c59dbb6
Starting hash:	30e9b71fb27c204b7ad576b0e57d4701bddbc3f68d9de884
Starting hash:	39febce43104a269650017e1b64483a46e79ed7d96ea2ece
Starting hash:	65b042c3bf9f50b294986980eae2e95c8b468415e596145d
Starting hash:	ba4a5b684798c24b3fa7d1d3ea980a21ab094e8c68debd52
Starting hash:	fc48d6fefcad15614868945cec6a4c02a4c723b8ac9d0ce8
Starting hash:	429c18baf86977e278c0b1c37412ca294ab98400830cbc8f
Starting hash:	016644edc1bf809a0e67c8de504f25e28270b3ad3fecd8d7
Starting hash:	d5a184cb396a2f25347ff5900d9315f9fbbcdc795cc98cbe
Starting hash:	2952d787e977a87a1caf704b2b6cd9985888d69234d701f8
Starting hash:	12d7f7b59ecd72ad062f3ad032157f13a5321a3193c656ec
Starting hash:	
Starting hash:	d93ddc60259eade9704e9e4f221eafadfba012e4e803b975
Starting hash:	6c343a34ce967648af604b292cd0c2080cc32955fe906742
Starting hash:	30a7c7cce7976156389e0a89235a1963f121a605311b6a31
Starting hash:	bff14f830ff5ef16e86dc2327e11354c4960026edf2814e9
Starting hash:	e5591dd9360c39dbe6468ed2f68a0728be953c5b5115246e
Starting hash:	35576471350c8a75ea01a4cf50db2dca164471b3ebd5469f
Starting hash:	e4496aebab793ddb9cb554beb3e39a7a668c3133a9fc0175
Starting hash:	f0454fe9f088492332a6e5ca6db1ea0e584daab06f3ec101
Starting hash:	f44b7e5f048c255835bd68a234f52ae86b06ed148ebda4e9
Starting hash:	253c819b314aa2ef67d82dd37e61bbcc9a8c3333c0485a64
Starting hash:	d9e767720db97f7516cfd5d98760b5fe2d129cc37459cad0
Starting hash:	43497fd7f826957108f4a30fd9cec3aeba79972084e90ead
Stopping hash:	000000000000000000000000000000000000000

Fig.13. Getheaders Message in Wireshark.

i. Block

This message is send in response to *getdata* message requesting transaction information from a block hash. Table 12 summarizes the contents of payload of *block* message.

Table 12. Payload Contents of Block Message

Field Name	Bytes	Description
version	4	Signed Block version information.
prev_block	32	The hash value of the previous block this particular block references.
merkle_root	32	The reference to a Merkle tree collection which is a hash of all transactions related to this block.
timestamp	4	A Unix timestamp recording when this block was created.
bits	4	The calculated difficulty target being used for this block .
nonce	4	The nonce used to generate this block.
txn_count	variable	Number of transaction entries.
txns	variable	Block transactions, in format of "tx" command.

Bitcoin protocol Packet magic: 0x0b110907 Command name: block Payload Length: 674 Payload checksum: 0xea71606d # Block message Block version: 536870912 Previous block: 3a9ed9e47930973f6632ef874b7c02283451a49a9d0a0d55... Merkle root: 5620db1ace096948528facb560e74cd56ab16c99e5f15a90... Block timestamp: Sep 13, 2017 16:39:11.000000000 India Standard Time Bits: 0x1b03fffc Nonce: 0x5b4c8045 Number of transactions: 2 ▲ Tx message [message [1] Transaction version: 1 Input Count: 0 Output Count: 1 Transaction output Value: 1 Script Length: 0 Script: <MISSING> Block lock time or block ID: 0 4 Tx message [2]
Transaction version: 0 Input Count: 0 Output Count: 0 Block lock time or block ID: 0

Fig.14. Block Message in Wireshark.

j. Headers

This message is send in response to *getheader* message returning desired block headers. Table 13 summarizes the contents of payload of *headers* message.

Table 13	Pavload	Contents of	f Header	Message

Field Name	Bytes	Description
count	variable	Number of Block headers.
headers	More than 81 per header * No. of head-ers	Block header.
txns	variable	Block transactions, in format of "tx" command.

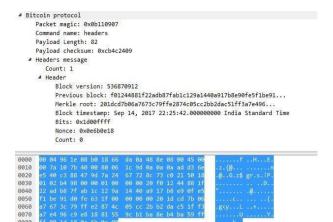


Fig.15. Headers Message in Wireshark.

k. Ping

This message is send by nodes for testing TCP/IP connection liveliness. Any connection error is presumed to be a closed connection and address is removed as current peer. Table 14 summarizes the contents of payload of *ping* message.

Table 14. Payload Contents of Header Message

Field Name	Bytes	Description
nonce	8	Random nonce.

	Packet magic: 0x0b110907
	Command name: ping
	Payload Length: 8
	Payload checksum: 0x6b4eb2b8
3	Ping message
	Random nonce: 0x51f037f900d7aedb

Fig.16. Ping Message in Wireshark.

l. Pong

This message is send in response to *ping* message by a node to prove its liveliness on network. Modern protocol versions, a *pong* response is generated by using a nonce included in the *ping* message. Table 15 summarizes the contents of payload of *pong* message.

Table 15	. Payload	contents of	header	message
----------	-----------	-------------	--------	---------

Field Name	Bytes	Description
nonce	8	nonce from ping.

-			1	1	
Bitcoin proto	col				
Packet mag	ic: 0x0b11090	37			
Command na	me: pong				
Payload Le	ngth: 8				
Payload ch	ecksum: 0x6b4	1eb2b8			
▲ Pong messa	ge				
Random	nonce: 0x51f0	37f90	0d7ae	db	
	Packet mag Command na Payload Le Payload ch Pong messa	Packet magic: 0x0b11090 Command name: pong Payload Length: 8 Payload checksum: 0x6b4 Pong message	Packet magic: 0x0b110907 Command name: pong Payload Length: 8 Payload checksum: 0x6b4eb2b8 Pong message	Packet magic: 0x0b110907 Command name: pong Payload Length: 8 Payload checksum: 0x6b4eb2b8 Pong message	Packet magic: 0x0b110907 Command name: pong Payload Length: 8 Payload checksum: 0x6b4eb2b8

Fig.17. Pong Message in Wireshark.

V. CONCLUSION

In this research work, we have deeply studied and investigated what exactly happens in the back screen when new peer joins the bitcoin network and starts syncing with network by downloading the blockchain. Firstly, we introduced core components of Bitcoin and then by using computer network protocol analyzer tool i.e. Wireshark, we have captured Bitcoin packets that are exchanged when peers communicate and deeply observed and analyzed their contents to make the whole communication process easy to understand for the readers.

REFERENCES

- [1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
- [2] K. A and J. J. S, "Recent Replica Placement Algorithms in P2P Networks – A Review", *International Journal of Computer Network and Information Security*, vol. 5, no. 5, pp. 55-63, 2013.
- [3] N. Koblitz, "Elliptic Curve Cryptosystems", *Mathematics* of *Computation*, vol. 48, no. 177, p. 203, 1987.
- [4] S. V, H. Shanavas.I, N. V and B. M, "Hardware Implementation of Elliptic Curve Cryptography over Binary Field", International Journal of Computer Network and Information Security, vol. 4, no. 2, pp. 1-7, 2012.
- [5] "Protocol documentation." [Online]. Available: https://en.bitcoin.it/wiki/Protocol_documentation
- [6] V. Buterin, "Ethereum: A next-generation smart contract and decentralized application platform," https://github.com/ethereum/wiki/wiki/White-Paper, 2013.
- [7] G. Wood, "Ethereum: a secure decentralised generalised transaction ledger," http://gavwood.com/paper.pdf, 2014.
- [8] "Testnet." [Online]. Available: https://en.bitcoin.it/wiki/Testnet
- [9] "the wireshark wiki." [Online]. Available: https://wiki.wireshark.org/
- [10] A. Turner and A. Irwin, "Bitcoin transactions: a digital

discovery of illicit activity on the blockchain", Journal of Financial Crime, vol. 25, no. 1, pp. 109-130, 2018.

- [11] "Tp's testnet faucet." [Online]. Available: http://tpfaucet.appspot.com/
- [12] "Wireshark display filter." [Online]. Available: https://www.wireshark.org/docs/dfref/b/bitcoin.html

Authors' Profiles



Auqib Hamid Lone I did my Bachelors in Information Technology and Engineering with distinction, post B. Tech my interest and passion towards information security took me into master's and I completed M. Tech in Information Security & Cyber Forensics

from Jamia Hamdard University, New Delhi with university rank. Currently I am Research Scholar at NIT Srinagar J&K. My areas of interest are Blockchain Technology, Cryptography, Network Security, Web Application Security and Digital Forensics.



Roohie Naaz Mir is professor and Head of Department Computer Science & Engineering at National Institute of Technology Srinagar, India. She received B.E. (Hons) in Electrical Engineering from University of Kashmir (India), M.E. in Computer Science & Engineering from IISc

Bangalore (India) and PhD from University of Kashmir, (India). She is a Fellow of IEI, IETE, a senior member of IEEE, member of IACSIT and IAENG. She is the author of many scientific publications in international journals and conferences. Her current research interests include Network Security, Reconfigurable computing, mobile computing, security and routing in networks.

How to cite this paper: Auqib Hamid Lone, Roohie Naaz Mir,"Investigating and Analyzing Bitcoin Blockchain Protocol using Wireshark", International Journal of Computer Network and Information Security(IJCNIS), Vol.10, No.7, pp.36-43, 2018.DOI: 10.5815/ijcnis.2018.07.04