Digital Fountain for Multi-node Aggregation of Data in Blockchains

by

Nakul Chawla

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2018 by the
Graduate Supervisory Committee:

Dragan Boscovic, Co-Chair
Kasim Selçuk Candan, Co-Chair
Ming Zhao

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

Blockchain scalability is one of the issues that concerns its current adopters. The current popular blockchains have initially been designed with imperfections that introduce fundamental bottlenecks which limit their ability to have a higher throughput and a lower latency.

One of the major bottlenecks for existing blockchain technologies is fast block propagation. A faster block propagation enables a miner to reach a majority of the network within a time constraint and therefore leading to a lower orphan rate and better profitability. In order to attain a throughput that could compete with the current state of the art transaction processing, while also keeping the block intervals same as today, a 24.3 Gigabyte block will be required every 10 minutes with an average transaction size of 500 bytes, which translates to 48600000 transactions every 10 minutes or about 81000 transactions per second.

In order to synchronize such large blocks faster across the network while maintaining consensus by keeping the orphan rate below 50%, the thesis proposes to aggregate partial block data from multiple nodes using digital fountain codes. The advantages of using a fountain code is that all connected peers can send part of data in an encoded form. When the receiving peer has enough data, it then decodes the information to reconstruct the block. Along with them sending only part information, the data can be relayed over UDP, instead of TCP, improving upon the speed of propagation in the current blockchains. Fountain codes applied in this research are Raptor codes, which allow construction of infinite decoding symbols. The research, when applied to blockchains, increases success rate of block delivery on decode failures.

ACKNOWLEDGMENTS

Firstly, I would like to express my sincere gratitude to my advisor Dr. Dragan Boscovic for the continuous work to support my thesis study and related research. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Master's study, He has not only given me experience with research, but also helped me expose myself to Industry.

I would like to thank Dr. Kasim Selcuk Candan and Dr. Ming Zhao for their relentless support and guiding me throughout my thesis work.

I would like to thank Dr. Darren Tapp for teaching me and making me understand more about public blockchains. He helped me understand the economics for miners and the role of game theory in the existing multiple cryptocurrencies.

I would like to thank Jeremy Liu for starting having the many meaningful discussions on the current leading research on blockchains through Blockchain Innovation society that we co-founded.

Finally, I must express my very profound gratitude to my parents for providing me the unfailing support and continuous encouragement throughout my years of study and through the process of researching. This accomplishment would have not been possible without them.

Thank you.

Nakul Chawla

TABLE OF CONTENTS

Page

LIST OF FIGURES v

CHAPTER

1 INTRODUCTION 1

1.1 Scalability Challenges with blockchains 4

1.1.1 Challenge: Throughput 4

1.1.2 Challenge: Validation of large blocks 5

1.1.3 Challenge: Relay of large blocks 5

1.2 Existing Solutions 6

1.2.1 Compact Blocks 7

1.2.2 Xtreme Thin Block 9

1.2.3 Graphene blocks 11

1.3 Current Proposal 12

1.3.1 Digital Fountain 13

2 RELATED WORK AND THEORY 15

2.1 Digital Fountains Codes 16

2.1.1 Encoding and Decoding of Luby Transform Codes 16

2.1.2 Raptor Codes 18

3 DESIGN AND METHODOLOGY 21

3.1 The Simulator 21

3.1.1 Parameters 23

3.2 Hypothesis 24

3.3 The protocol specifications 26

3.4 Implementation Details 27

4 EXPERIMENTS & RESULTS 30

CHAPTER Page

4.1 Evaluation Criteria 30

4.2 Dash Core Client 30

4.2.1 Testing Environment 30

4.2.2 Observations 31

4.3 Dash Simulator 31

4.3.1 Testing Environment 31

4.3.2 Results 34

5 CONCLUSION 42

5.1 Future Work 43

REFERENCES 44

LIST OF FIGURES

Figure Page

1.1 Traditional Relay 6

1.2 Low Bandwidth Compact Relay 8

1.3 High Bandwidth Compact Relay 8

1.4 XThin Relay 10

1.5 Graphene Relay 11

3.1 Protocol 27

4.1 Orphan rate with respect to symbol Sizes 33

4.2 Block Size vs Orphan Rate k=10 34

4.3 Block Size vs Orphan Rate k=100 35

4.4 Block Size vs Orphan Rate k=1000 36

4.5 Block Size vs Orphan Rate k=10000 37

4.6 Median Block propagation k=10 38

4.7 Median Block propagation k=100 39

4.8 Median Block propagation k=1000 40

4.9 Median Block propagation k=10000 41

Chapter 1

INTRODUCTION

A blockchain, is a continuously growing list of records, called blocks, which are linked and secured using cryptography. Each block typically contains a cryptographic hash of the previous block, a time stamp and transaction data. By design, a blockchain is immutable because of the linked structure. It is ”an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way” according to [25]. In order to be used as distributed ledger, it is managed by a peer to peer network, in which all peers/nodes follow one protocol. This protocol takes care of messaging between these peers, and adding and validating the new blocks formed. Once a block is added to the blockchain, the block can’t be altered, since all blocks added after will need to be changed.

Blockchains are secure by design and exemplify a distributed computing system with high Byzantine fault tolerance, leading to consensus in a decentralized network.

This makes blockchains potentially suitable for the recording of events, medical records, and other records management activities, such as identity management, transaction processing, documenting provenance, food traceability, and voting. [7] Blockchain was invented by Satoshi Nakamoto in 2008 to serve as the public transaction ledger of the cryptocurrency bitcoin [30]. The invention of the blockchain for bitcoin made it the first digital currency to solve the double-spending problem without the need of trust. Blockchain networks essentially engage two major types of actors:

1. **Miners:** Mining is the process of adding transaction records to the public ledger. The miners, in a proof of work blockchain, solve a computationally diffi-

cult problem in order to add a block to the ledger. In bitcoin, this problem is to solve for a number (nonce) [10] that when hashed with the block data must start with a certain number of zeroes. This number of zeroes is termed the difficulty of the network. The miner who successfully adds a block to the ledger is rewarded with a block reward along with the fee on all transactions included in the block. The miners, therefore have incentive in computing the hash for the next block and propagating it to the network so that their block is accepted. In order to mine these computationally expensive hashes and propagate them through the network, miners invest heavily into high grade infrastructure, bandwidth for the node and the electricity costs of running the mentioned hardware.

2. **Propagation Nodes:** These nodes are voluntary nodes on the network that help in propagation of the blocks and transactions. Propagation nodes are of 2 types:

   (a) **Full Nodes:** These nodes help in propagating the complete blocks and fulfill all requests. In addition to propagating the blocks, it also verifies and validates the transactions by checking its inputs and outputs. They validate whether there is a double spend on the transaction by checking a complete history of the transaction. They also verify the signatures of the witnesses. In order to validate and serve all requests, these blocks store the complete copy of the blockchain.

   (b) **Light Nodes/SPV (Simplified Payment Verification):** These nodes are lightweight nodes that run on machines with less computational power and less storage. They also validate the block but not by checking the transactions in depth, but they do so by checking their merkle proofs. That is basically hashing all the transaction hashes together to see if the merkle root [8] present in the block header is what they get after hashing all their transactions. In order to do so, the SPV wallets only need the header of all blocks included in the chain and the merkle proofs. Therefore, these nodes do not store a copy of the blockchain and hence don't propagate complete blocks. These are often run on Raspberry Pis or mobile wallets.

The Dash network [21] employs an additional 2nd layer protocol, commonly referred to as the **Masternode** layer.

1. **Masternode:** These nodes are full nodes which aim to help the Dash blockchain network by providing services and, in return, get awarded by sharing a portion of the block reward with miners. The requirements for a full node to be a masternode are as follows:

   (a) The node at all times should have a 1000 dash in its wallet.

   (b) The node should run a high bandwidth connection. The current requirement is 1 Gigabit per second.

   (c) The node should run significant memory (RAM), in order to store the transaction set in memory for faster validation. It is currently 4GB but might increase based on future needs.

In order for the blockchain network to run securely and all the consensus rules be enforced, it is important that the network contains a significant number of full nodes and help verifying transactions and propagating complete block data. The current bitcoin network runs 10,000 nodes approximately.

As the blockchain decentralized network grows larger or the number of transactions on the network, the network runs into issues relating to scalability that are explained below.

1.1 Scalability Challenges with blockchains

1.1.1 Challenge: Throughput

The block interval [3] in a blockchain network is controlled by the mining difficulty [9], which approximates to 2 minutes 30 seconds in Dash network, and lives in the neighbourhood of 10 minutes in the bitcoin network. In the bitcoin network, the mining difficulty is adjusted every 2016 blocks in order to get it closer to the above mentioned intervals. The block size was initially not constrained. However, it was later capped to 1MB in 2010 [6]. The average size of transaction on the bitcoin network is 500 bytes. The maximum number of transactions that can be packed inside a block is dependent on the transaction sizes for that block, however approximating 500 bytes a transaction, the throughput is close to 3.5 transactions per second. In the bitcoin blockchain network, it is seen that with a mixed transaction size, the network roughly has a throughput of 3-7 transactions per second [40].

Transaction sizes depend on three variables :

1. List of inputs
2. List of outputs
3. A list of witnesses, that is one per each input and is dependent on a flag which if set to 0 indicates no witness.

Majority transactions, based on the data provided by block explorers, happen between two peers resulting in 1 input and 2 outputs along with 2 witnesses therefore resulting in an almost static transaction size. This results in a linear throughput increase with the increase in block size.

Public blockchains, seeing this limitation on throughput, have been slowly forking over to larger block sizes. Dash currently has a 2MB block size limit within a 2.5

minute block interval. Bitcoin Cash over the years has slowly forked the blockchain multiple times to increase the block limit to 32MB. [4] lists all forks that happened with bitcoin.

1.1.2 Challenge: Validation of large blocks

As block size gets bigger, validation of individual transactions (a 1GB block may have over a million transactions), 2 million taking average transaction size to be 500 bytes, becomes a challenge for personal computers or Raspberry Pis that are running full nodes on the network. One of the key reasons for this challenge is that although all clients employ infrastructure with multiple threads, the blockchain code does not make use of threads at its full potential. Recent research at Stanford [35] shows that even though a gigabyte block may be propagated in the block interval constraint, it will be hard to validate the number of transactions. This resulted in the changing the implementation of Bitcoin Cash cryptocurrency to handle a large number of threads in order to increase the performance of the software.

1.1.3 Challenge: Relay of large blocks

Block relay (1.1) on the blockchain network uses a gossip protocol in which a node relays the newly heard blocks to its peers which is approximately in the neighbourhood of 8-10 peers. This information is relayed using a series of messages. On hearing about a new block, the first node validates the complete block and then advertises the block its peers using an `inv`(Inventory message, which is a 64 byte hash of the received block). If the peer on the other side hasn't heard of this 64 byte hash, it sends a `getdata` message (36 bytes) to the first peer. In response to the `getdata` message, the second peer now transmits the whole block to the first peer. This is the size of a full block (current block size agreed upon by the network).

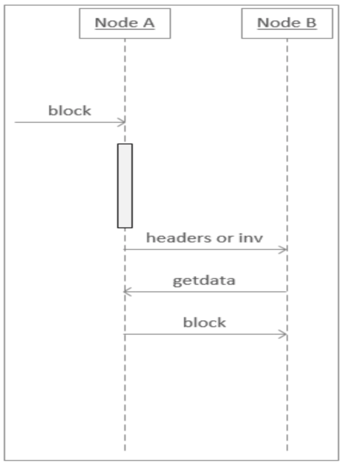

Figure 1.1: Traditional Relay

As the block size increases on the network, the last message (`getdata`) relay is bound on the upstream bandwidth of the second peer, therefore slowing down the propagation of blocks through the network.

Miners on the network although receive a transaction fee on adding a transaction to the block, a major portion of their revenue come from the block reward. If the miners are not able to propagate their blocks to the network, they lose the block reward and with that their potential profits. The block that was mined correctly but was still not added to the blockchain is referred to as an orphan block. This leads the miners to either mine smaller blocks (less revenue because of less transactions) or not mine at all. The former leads to a lower throughput and the latter leads to risks of Centralization since only a handful a miners can afford to relay larger blocks because of high bandwidth requirements.

1.2 Existing Solutions

In the recent years, the solutions to speed up propagation process have aimed at reducing the payload of the block. In the gossip protocol, transactions are relayed in the network using the same process as of blocks. Every transaction that happens on a node, is broadcasted to all its peers using an inventory message(`inv`) that is followed by a `getdata` and a transaction data transmission, if required. Since this data has already been transferred across the network, there was an opportunity to send only the information required to propagate the block. That involves a metadata of all the transactions and the full transactions that have not already been transmitted to the requesting peer. This led to developing three different solutions that manage to send a significantly smaller payload in the block, while also being able to relay the complete block. Although these existing solutions are efficient in relaying the metadata, work in a probabilistic way because of the assumption that a majority of the full transaction information is synchronized in the network.

Transaction data that has not yet been included in the network, is kept in the memory of the node and referred to as memory pool . Since these are the data that the miners use to create new blocks, synchronization of memory pool is important if we are relaying metadata for block reconstruction. However, new research in [14], discussed in details in Chapter 3 (Design and Methodology), states that, that is not always the case.

The motivation and protocol for solutions that work on decreasing the block payload are explained below in section X.

1.2.1 Compact Blocks

Bitcoin Improvement Protocol (BIP) 152 [19] introduces Compact blocks as a payload reduction technique. Compact blocks are based on the idea that most transactions on the network are already transmitted to the peers and only the difference in the information is to be sent across for efficient relay.

The process to relay a compact block is shown in 1.3 for high bandwidth relay

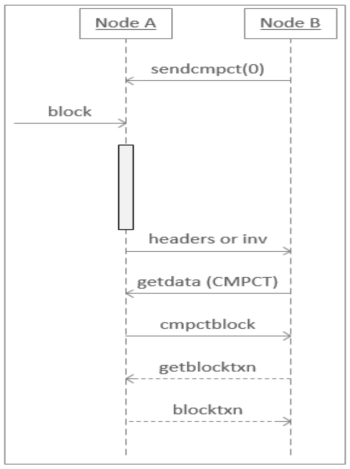

Figure 1.2: Low Bandwidth Compact Relay

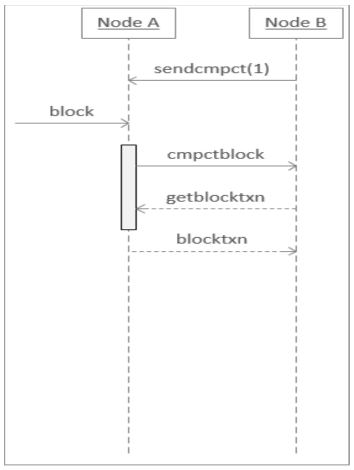

Figure 1.3: High Bandwidth Compact Relay

and 1.2 for low bandwidth compact relay and is explained below:

1. All nodes that support compact blocks broadcast `Sendcmpct(0)` or `Sendcmpct(1)` for non-expedited or expedited relay respectively.

2. A new blocks is mined, the header or `inv` is sent to the receiving node.

3. If the receiver node hasn't yet received the block, it sends a `getdata(cmpct)` request to the Sender node.

4. The sender node responds with a compact block, which is collection of 6 byte shortids of transactions and some pre-filled transactions that the node might not have heard of. (Example the coinbase transaction)

5. In case of collision of the shortid or the receiver not having the transaction information, the peer requests all the transactions (it doesn't have) in the form of `getblocktxn` request.

6. The above request is responded with a `blocktxn` message that contains the full transactions.

Based on the above process, one can understand that the compact block saves significant bandwidth by reducing the block payload to relay metadata information across the network. [2] states that the maximum sketch for a block is seen to be between 9 and 20KBs for 1MB block, and has a 90% chance of block propagation happening without having to re-request the extra transmissions. The protocol documentation does not provide any information on how it affects when the block size increases or how this technique will scale with an increased flow in transactions. There are several weaknesses that were observed after the protocol went live through a soft fork [11]:

1. The memory pools of different nodes in a high transaction flow environment are not synchronized. The node allocates a certain amount of memory to its memory pool, and after this memory pool fills up, the transactions that arrive later are dropped

1.2.2 Xtreme Thin Block

Bitcoin Unlimited Improvement Protocol (BUIP010) [1] proposed and implemented another form of payload reduction called Extreme Thin Blocks $XThin$. This

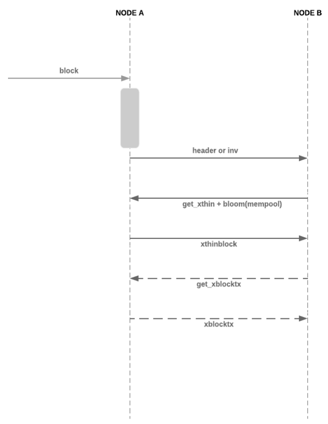

Figure 1.4: XThin Relay

technique uses a similar technique as Compact block. It does add more complications explained below in order to decrease the re-transmission rate. The process to relay an XThin block is shown in 1.4 and is explained below:

1. All nodes that support XThin Blocks, run another service `NODE_XTHIN` as a way of letting the network know the kind of blocks they support.

2. The receiver node creates a bloom filter of all the items in its memory pool. It then requests the block `getdata` along with the bloom filter.

3. The Sender then sends the block adding all transaction IDs(8 bytes) also referred as cheap hashes along with header information and full transactions that the receiver doesn't have (based on the receiver's memory pool).

4. The receiver reconstructs the block using the transactions that it has in its memory pool along with the full transactions supplied by the Sender.

The protocol documentation states that the block sizes reduce to a maximum block sketch of 17.5 KB in the experiments that were run on 1MB blocks. This was observed when the protocol went live with Bitcoin-Bitcoin Cash hard fork as well.

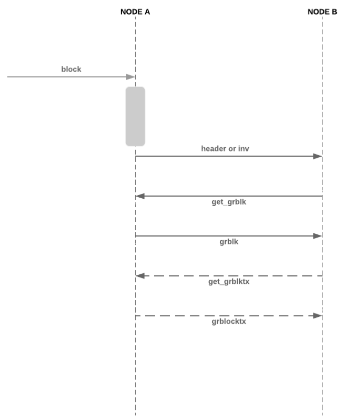

Figure 1.5: Graphene Relay

Xtreme Thin blocks face several problems as well:

1. Bloom filters although encode information in a relatively much smaller size, but have a False Positive Rate associated with it. In case of of false positives, the full transactions are re-requested by the receiver node.

1.2.3 Graphene blocks

Bitcoin Unlimited Improvement Protocol (BUIP093) [13] proposes Graphene Relay propagation in order to further decrease the payload of a block. The Graphene propagation however gets rid of the shortID list implementation and sends a bloom filter along with an Invertible Bloom Lookup table [24] in order to relay information.

The graphene protocol is shown in 1.5 and is explained below:

1. The sender sends an `inv`:inventory message to notify an new block.

2. The receiver requests the graphene block using `getgrblk` message along with the count of transactions in its mempool.

3. The sender sends the graphene payload that consists of a bloom filter of all the transactions inside the block and an Invertible Bloom Lookup Table.

4. The Invertible Bloom Lookup Table consists of a mapping of all transaction IDs along with the cheap hashes (the first 8 bytes of the Transaction ID).

5. Ranks of transaction IDs are calculated using lexicographical ordering.

6. The receiver on the other hand collects all transaction IDs in its memory pool and orphan pool and filters the transactions of the block using the Sender's bloom filter.

7. Using these filtered transactions an IBLT is constructed at the receiver's end and subtracts the sender IBLT from it.

8. IBLT decode success is based on either the subtraction to lead an empty set or a non-empty set in which the remaining transactions can be requested again.

9. IBLT decode failure can happen based on its probabilistic nature or when the receiver IBLT has very low transactions to match or a checksum failure.

The above mentioned technique is highly efficient for sending large blocks when memory pools are synced. In case of IBLT decode failures results in the node requesting a full block in order to avoid a Denial of Service attack.

1.3 Current Proposal

The techniques, mentioned in Section 1.2,

1. rely on memory synchronization of nodes communicating the blocks and the overhead of decode failures is re-transmission of full data back from the sender.

2. They also rely on a single peer to send the complete data as opposed to using multiple peers to transmit partial data.

3. They can only be used to receive blocks that are currently being propagated. They can't be used to sync new nodes.

1.3.1 Digital Fountain

This thesis aims to leverage the research of digital fountains [16] [26] [38] [39] in order to propagate block information as a fallback method for the above existing solutions.

In order to make these methods more efficient, this thesis aims to add a protocol of aggregating data taken from multiple peer nodes in the form of encoded symbols, collecting them over to the receiver end and re-constructing the block.

In case of block failure, in the current blockchains, the peer requests all the full transactions from one peer using `getdata`. In case of large blocks, this is a time consuming relay and can lead to higher orphan rates and hurt the profitability of miners. As an alternative, the receiving node can request all the peer nodes for part of data and reconstruct the block and recreate the source block. This significantly reduces the uploads per peer in the network and reduces the stress of data transmission per node. However, the overall data downloaded by the receiving peer has some overhead in case of decode failures.

There are 2 ways of implementing this idea:

1. Creating a list of transactions that are missing and dividing them among all nodes and requesting for the specific data.

2. Requesting a Forward Error Correcting fountain code, that if the receiver has enough symbols of, can recreate the block. This method is explained and is worked out in this thesis.

The reasons for using the latter technique are the following:

1. The fountain code symbols can be transmitted over both UDP and TCP. While transmitting this data in a UDP network, there will be a need to transmit additional overhead symbols in case of data loss.

2. In a blockchain network, when in the above described techniques if a decode failure happens, the whole block data needs to be sent. However, when using fountain codes, the whole data need not be sent . It is sufficient to send more repair symbols to successfully decode. These symbols are smaller in size and, thus, get transmitted faster.

3. Miners with higher bandwidth can use asynchronous multicast to reliably transmit data.

Trade offs of using the fountain code technique are as follows:

1. Extra computation is added to a node in order to create the symbols before relaying them. These computations are only done once, by the miner while relaying the symbol. The other nodes can flush the symbols to disk and use it for relaying at anytime.

2. There is overhead while reconstructing the block. In order to decrease the number of decode failures, overhead repair symbols are added. However, this burden is added on the download side and not on upload side, because the upload is shared by a set of peers.

Chapter 2

RELATED WORK AND THEORY

This thesis is inspired from the delivery methods utilized by Content delivery Networks (CDNs) [31] [34] [41] [42] and torrent networks in order to relay large files on a network in an efficient manner. CDNs transmit large file over an adaptive relay network in the form of software or media distribution. In order to transmit these large files to multiple thousands of clients, the CDN caches symbols in its memory and opens up many fountains one for each client.[32] and [33] show efficient ways of using fountain codes to deliver a continuous stream of systematic symbols (Systematic symbols : The first k that are transmitted are the encoded data part). Assuming that a part of those k symbols are dropped, repair symbols are sent till the client collects a total of k (original + repair) symbols and is able to reconstruct this data.

Torrents on the other hand use a distributed file in order to find the nodes that can provide the information. This file is called the torrent file. When we know the nodes that can provide the information, we request them for the information. These nodes respond with a part of the file that they own. When all nodes provide the known information, the file is aggregated and reconstructed into the usable file.

BitTorrents [5] [18] were experimented to use fountain codes in 2012 by [20]. However, the results in the case of torrents were almost comparable to the one with no coding technique used in the initial phase, but as enough seeders seed the data, the download with coding increases to a much faster rate. In our case, the block will be present with all the seeders, therefore the algorithm to find/disperse data is not employed as is the case with this paper.

In the context of blockchains, miners who need to send the information to many nodes and in a fast and reliable manner, stream multiple fountains for each node that they are connected to. The full nodes use the torrent approach of aggregating data and upload the partial information.

Blockchains, in the near future, in order to scale on chain will require to transmit large blocks over the network in a relatively small block interval. Since, CDNs effectively transmits their target data for efficient downloads, the idea can be abstracted and specific pieces of technology can be utilized in blockchains to synchronize large blocks over the decentralized distributed network.

2.1 Digital Fountains Codes

The common analogy to understand Fountain codes, is a water fountain. In order to fill a glass of water, one needs enough water, and not specific parts of water. Fountain codes work on the same principle. In order to reconstruct data, one needs enough symbols, here k. They can be a combination of source and repair symbols.

Fountain codes are also known as rateless erasure codes. Near optimal erasure codes [29] can transform k symbol messages into practically infinitely encoded form i.e. they can generate an arbitrary amount of redundancy symbols that can be used for error correction. **Luby Transform** [27] codes are the first class of practical fountain codes which depend on bipartite graphs to trade reception overhead for encoding and decoding speed.

These codes employ a simple $\oplus$ based algorithm to encode and decode a message. And these codes adopt only a one way communication protocol which can directly be plugged as an alternative to `getdata` request explained in the coming chapters.

2.1.1 Encoding and Decoding of Luby Transform Codes

1. The sender encodes and sends packets of information.

2. The receiver evaluates each packet. If the packet has an error, it discards that packet, otherwise it is saved as a message.

3. Eventually the receiver has enough valid packets to reconstruct the message, on which an ACK is sent after transmission.

Luby Transform Encoding Process

1. The message is divided into n blocks of roughly equal length.

2. The degree $d, 1 <= d <= n$ is chosen at random.

3. d blocks are chosen at random.

4. If M_i is the i^th block of the message, data portion of the next packet is $M_i1 \oplus M_i2 \oplus ... \oplus M_id$

5. A prefix is appended to the encoded packet: defining n, d and a list of indices $i_1, i_2, ..i_d$.

6. Some form of error detecting code (as simple as cyclic redundancy check) is applied to the packet and transmitted.

Luby Transform Decoding Process

1. The packet is first checked, if the packet isn't clean or it has already been sent, it is discarded.

2. If the cleanly received packet is of degree $d > 1$, it is first processed against all the fully decoded blocks in the message queuing area, then stored in a buffer area if its reduced degree is greater than 1.

3. If $d = 1$ is received, the packet is matched against all the packets of $d > 1$ residing in the buffer. If $d = 2$ is received, it is reduced to $d = 1$ and moved to message queuing area.

4. When all the n blocks are produced, the signal is transmitted.

2.1.2 Raptor Codes

Raptor codes are systematic fountain codes, where on receiving symbols the system can't identify whether these are original source or the repair symbols.

The way these are generated is, from the original source k symbols are constructed that can decode the entire source. Once we know that, the k can decode the original, we generate more symbols from the original symbols again and call them the repair symbols. The receiver end now gets a degree distribution in such a way that we don't actually require symbols of degree 1 to decode all the original symbols.The degree distribution employed for RaptorQ is taken from a mix of 2 or more distributions. Any mix of source and repair symbols, as long as we have enough symbols (k), we can decode the source.

The idea that makes Raptor codes work they do, is deactivation decoding. In order to decode the entire data, we find a very small fraction of symbols through Giant Component analysis and which can be later discovered using Gaussian elimination as shown in [36] and [28], that if we assume to be decoded, we can decode the rest of the data.

With this version improvised over time, if k symbols are received at the receiver's end, there is a .5% chance of failure and is independent of k, and with every overhead symbol, the chances of failure to decode reduces by a factor of 200 as stated in [37].

Therefore, for the current time, the research on erasure channels have reached to optimality with the invention of raptorQ codes with a clever use of two or more degree distributions.

The result on Raptor codes shows that a file on TCP that takes about 2min 15 seconds to downloads, downloads in about 13 seconds. Therefore, this research is further applied to recover/reconstruct data in blockchain in order to understand its implications on block propagation.

Some of the advantages of these codes, as stated in the paper [15]

1. **Scalable:** The server (miner) load remains constant if the number of leeching nodes increase. In cases of miners relaying blocks on asynchronous multicast connections, this will help increase the performance in sending data.

2. **Reliable:** An exact copy of the original file is reconstructed at the receiver end, which is very important, since we are dealing with transactional data.

3. **Reception Efficient:** The total number of packets that each receiver needs to reconstruct the file or data is minimal. In ideal cases, the length of data to be received equals the length of aggregated symbols. As we add more symbols, every symbols decreases the chance of decode failure by 200.

4. **Time Efficient:**The time spent to generate and reconstruct the packets is minimal. These symbols can be saved in a file and can be directly broadcasted in order to respond to a request.

5. **Time Independent:** Receivers may initiate the download at their discretion, implying that different receivers may start the download at widely disparate times. Receivers may sporadically be interrupted and continue the download at a later time.

6. **Server Independent:** Receivers may collect packets for the file from one or more servers that are transmitting packets. No coordination between servers

should be required for this.

7. **Tolerant:** The protocol should tolerate a heterogeneous population of receivers, especially a variety of end-to-end packet loss rates and data rates.

Chapter 3

DESIGN AND METHODOLOGY

3.1 The Simulator

In order to understand propagation and the scalability of the dash network, a simulator was developed at ASU using an already existing simulator that was developed at ETH Zurich for this research [23]

The ETH blockchain simulator is an NS3 (Network Simulator 3) based bitcoin simulator. This simulator demonstrates the workings of Bitcoin, Dogecoin and Litecoin, but bases its research on each block being the fundamental/rudimentary component.

We found that simulations that merely went down to the block level were not detailed enough for our purpose. In order to reliably model a consensus network with block propagation speedups such as compact or xthin we must understand these networks down at the level of transactions.

In order to understand the network at the level of transactions, we modeled generation of transactions using a normal distribution. Each node is has a transaction generator that creates a random transaction with a piecewise constant distribution and hashes that data.

Masternode in the current simulator is not a separate class but a check on full node. The network is randomly distributed with full nodes, masternodes and the miners. All the miners in the network are connected to each other which simulates the real world network. When establishing the connections between nodes, we create point-to-point channels between them, which abstracts away any intermediate devices (routes, switches, etc). These channels have two characteristics:

1. Latency
2. Bandwidth

Mining pools typically maintain private peering connections among themselves which we capture in our simulations.

In short, below are the modifications done to the bitcoin-simulator for it to run the current dash-simulations:

1. Transactions are added in order to add traffic and understand propagation techniques in more detail.
2. Every node now has its own mempool in order to test set reconciliation for Xthin and Compact propagation.
3. Compact block propagation is added.
4. Extreme thin block propagation is added.
5. Masternode (bool) is added to the full-node. Because this is network propagation, the bandwidth is the only differentiating factor between a full node and masternode.
6. Bloom filter using Murmur Hash library is added in order to test xthin.
7. Capability to test x-11 hashing is added, although it doesn't add value to propagation.
8. FillBlock - this parameter if true, fills the block with transactions to its maximum capacity. If this is set to false, the simulator uses a piecewise constant distribution to generate the next block size and adds transactions limiting to the new block size.

The resulting simulator code is available at [17]

3.1.1 Parameters

The simulator takes in the below given parameters:

1. Block size
2. Block Interval
3. Propagation type
4. Number of blocks
5. Fill block - The block is filled to capacity if fillBlock=true.

We ran multiple experiments in order to understand and compare the below three types of propagation:

1. Traditional
2. Compact
3. Extreme Thin Blocks

The simulations were run with different network sizes and different block sizes. Through these simulations, it was realized that both compact and xthin blocks can support the Dash network with a capacity an order of magnitude larger than the traditional block propagation.

We ran simulations for both Compact and XThin in the same testing set up from block sizes of 1 MB to 10MB. Through the experiments, we were able to observe that compact blocks are not scalable above 4MB block sizes on the dash network. However, we were able to propagate XThin blocks with lower orphan rates. The reason XThin blocks gave us better results is because of the bloom filter that is sent apart from cheap hashes, which helps understand the other peer whether or not what full transactions to send.

Note: The simulator does not always present the same results every time because the propagation can take different routes, the distributions used can use different node bandwidths, blocks are generated using Geometric progression from a seed value which is generated from a random seed. Parameters and distributions used in the simulator, gives some variance to the results, although the increase/decrease slope of the results can be confidently used.

3.2 Hypothesis

The memory pools of the nodes on the network are not synchronized when there is high amount of transaction flow. This is because of the following reasons:

1. Unspent transactions that are relayed to the node are stored in-memory. When transactions flow is high, and the memory is full, the transactions are dropped.

2. Transactions use gossip protocol for propagation through the decentralized network and therefore rely on nodes that are forwarding it. It is not always possible for the transaction to have relayed to a node where the block is received. Since, compact, XThin and graphene rely on memory pool synchronization, it is not always the case that all transactions relayed in the block sketch are known to the node.

Since, the memory pools of the peers go out of sync, reducing payload by using cheap hashes often leads to re-transmissions. This re-transmission is in the form of complete block or block transactions. In order to increase the speed of this request, we use fountain codes. Using fountain codes enables reliable transmission of data between all peer nodes and then aggregating the data at the node that is requesting the block.

The reasons of using and applying them to blockchains for transmitting data are multifold:

1. The primary reason for using fountain (raptor) codes is that they work on quantity (`k`) of symbols rather than quality (what part of data is being relayed). Therefore, on receiving any `k` symbols, we can reconstruct a block with 99.5% efficiency [37] .

2. Infinite decoding symbols can be created using the data, all peers required to send the data create k/(number of peers) symbols, independent of what the other peers have created, and then send them to the receiving node to reconstruct the block.

3. On decode failures, creating more decode or repair symbols has a linear complexity therefore these symbols are not compute heavy.

4. The repair symbols are small in size and can be relayed faster rather than sending the whole block for recovery.

The most important among others for us is that they are computationally easier to encode/decode since they use binary operations for both and they are server independent. In case of blockchains, we can ask data from all connected peer nodes and collect symbols to recreate the block. This will generate a lower upload footprint for the nodes that are serving data therefore leads to a faster synchronization. The scalability can be tested by using this propagation technique for large block sizes till we lose consensus (Orphan rate = 50%). The block propagation throughput is constrained by orphan rate of the network.

3.3 The protocol specifications

Below are the specifications of the implementation of protocol used by digital fountain in blockchains:

1. The sender node, on hearing a new block, sends an inv to another peer. The whole block body is reinterpreted as a byte array and then encoded into a set of symbols of a length that was negotiated between the peers.

2. On receiving the inv/header message the receiver requests the whole blocks using an expedited `getraptor` request to **all its peers**.

3. All peers who have heard of the block respond with `raptorcode` and encoded symbols of the blocks. The other peers drop the message.

4. The receiver collects these symbols in a singleton class (map of block header ID to symbol ID and symbol pair). Note that, since this research is done for large block sizes, symbols here will be significantly large. In order to prevent DoS attacks, every symbol must contain the header. The header should not be encoded. In smaller blocks, this can be optimized to send only the 64 byte hash along with every symbol.

5. The receiver knows the amount of symbols needed to reconstruct the block and waits till it gets at least the required count of the symbols.

6. On reaching the required count, it reconstructs the block using raptor decoding and serializes the stream on byte array to form the block.

The above is also shown in the figure 3.1

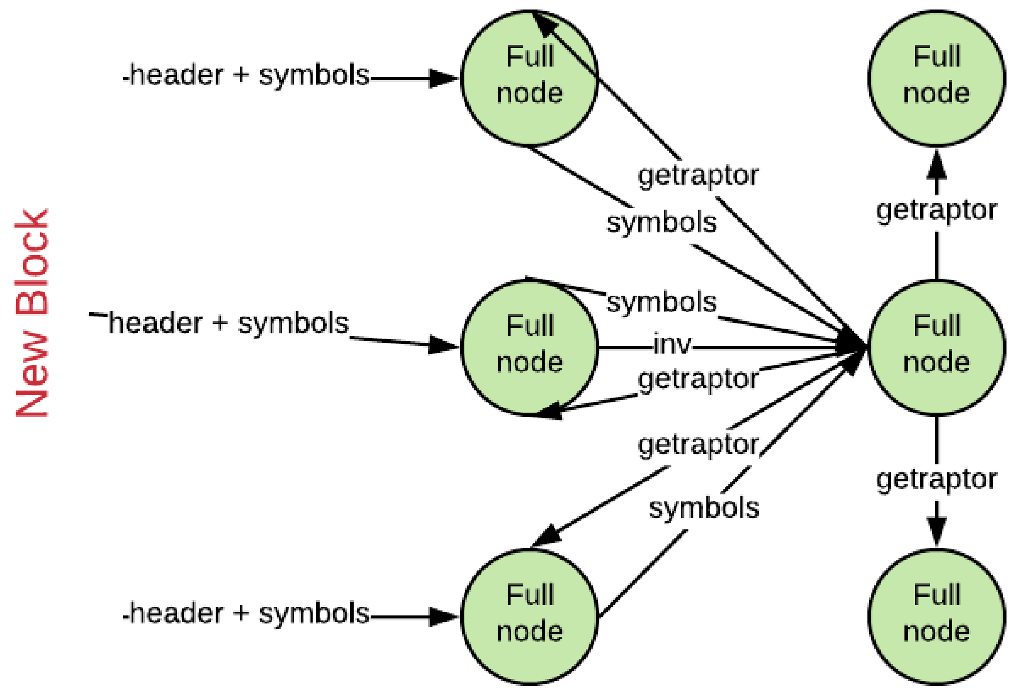

Figure 3.1: Protocol

3.4 Implementation Details

The implementation in order to test the hypothesis of data aggregation is done in 2 parts:

1. **Dash Core Client:**In this specific implementation, the following parts are implemented:

   (a) **RaptorQ (RFC6330) encoder and decoder:** This is done in c++ adhering to the Dashcore standards. The header-only C++ version of libRaptorQ is used. This is implemented in order to under to understand the encoding and decoding procedure for each custom data type (block, transaction, etc). Also, to collect data on number of symbols and symbol sizes in order to re-create the data.

(b) **Signaling:** Signals are developed in the dash client. This is primarily done to understand the use cases where a node can be automatically attacked without a malicious intention. (example: in an expedited `raptorcode` message from a non-miner node can be considered as DoS, because symbols will get recovered after we receive all symbols)

Modifications in the dash client:

(a) New classes for Raptor Symbol and Raptor Symbol data has been added. The raptor symbol class creates a vector of symbols to be sent in as the message, while the data class captures the size of each symbol, the total size, the time taken to encode and decode the data.

(b) A new singleton class is added that keeps track of all symbol data mapped to its header that a node contains. This data remains in memory for the last 6 blocks and is then flushed to disk.

(c) A symbol decoder is developed that collects the symbols in the singleton class and keeps a check on the count. It tries to decode only when the symbols are collected.

2. **Dash Simulator:** Dash client gives us a good transparency of how encoding and decoding will work, and the symbol sizes, and the constraints. However, in order to test how scalable the hypothesis is, experiments need to be run with variable block sizes, variable symbol sizes and number of connections. While testing in the Dash client, the regression test mode needs to be used but it does not add any latency between the nodes which defeats the purpose of the propagation. The testnet can be used, but large block sizes can not be tested

because of the low transaction flow on the testnet. Dash Simulator that has been described above, gives the following advantages:

(a) It helps in understanding the metrics of every node, which is not available in public blockchains. Also, it will be not be available because of security reasons.

(b) It helps us run various parameters and help model the blockchain in a way we want it to be tested.

Modifications in the Simulator:

(a) Since the testing of propagation of blocks is to bested, for this hypothesis transaction and memory pool have been removed.

(b) The masternode layer is still there.

(c) 2 new messages have been introduced, `getraptor` and `raptorcode`.

(d) The `getraptorcode` is sent to all peers as soon as one hears about the block through the `inv/header` message.

(e) The raptorcodes are uploaded in a stream of messages.

Chapter 4

EXPERIMENTS & RESULTS

The current raptor code in Dash is only developed for replacing `getdata` requests. We have compared 2 metrics in order to compare the techniques. The results shown below can efficiently be applied to compact blocks, XThin and Graphene blocks in cases where the block transactions are sent.

4.1 Evaluation Criteria

1. The orphan rate. The orphan rate tells us how many blocks weren't added to the block and how much profitable it will be for a miner to use this technique.
2. The block propagation times for median block propagation on of the nodes. The block sizes we have used are from 100MB to 1GB.
3. Orphan rates for different symbol sizes help us understand what symbol size to use.

Below is a detail of all the experiments that were run in order to test the hypothesis and understand the implications of introducing Fountain Codes into the current state of blockchains.

4.2 Dash Core Client

4.2.1 Testing Environment

1. DashCore version 12.3 is used for development.

2. libRaptorQ [22] 0.1.X branch, 697b318036a1ea4751e69cde7989ff3b3bd7c655 commit is used. The code needs to be changed in order to directly include the header files.

3. 3 local nodes in regression test mode were spawned for each experiment.

4. Each node creates its own blockchain in the assigned directory.

The tests done on the actual dash core client were used to find out how many repair symbols were used in order to complete the data.

4.2.2 Observations

1. It was observed, that for a block on regression test mode, the aggregated size of `k` encoded symbols had the same size as that of a block mined.

2. In the setup of 3 nodes and 4 nodes, we sent 1/3 and 1/4 encoded symbols from each node to the receiving node and we were able to recreate the block.

3. For small block sizes, we used 4 byte and 16 byte symbols. But for larger blocks, symbols can be as long as we want, with the constraint that symbol size is divisible by encoded array sent. If this is not the case, the decode fails. Therefore, symbol size and encoded vector size is negotiated in the `getraptor` request.

4.3 Dash Simulator

4.3.1 Testing Environment

1. The simulator is run with 24 nodes, with 8 mining pools. 8 full nodes and 8 masternodes. 24 nodes here represent the whole network because:

(a) For this protocol to work, memory pool synchronization is not required.

(b) Both traditional and the fountain protocol are compared on a 24 node network in order to have a fair comparison.

(c) Traditional propagation is used, because the messages - `blocktxn`, `xblocktx` and `grblocktx` in compact, XThin and graphene respectively are similar to full blocks except they are smaller in size (Since only missing transactions are propagated.

2. The masternodes although in the real network have higher bandwidth, in this experiment we have kept it the same as full nodes. The reason for this is to extrapolate these results to other public cryptocurrencies.

3. The average bandwidth taken for each node is taken from [12], where the paper talks about the latency and the bandwidth distribution for each continent.

4. The simulations for run for 100 blocks each.

5. Different symbol sizes have been used to understand the dependency of symbol sizes on orphan rate. `k` varies between 10 and 10000 i.e the number of symbols required to reconstruct the block. Orphan rates are calculated for each of these `k` to understand the ideal size for raptor code symbol size.

6. Each node only forwards 50% symbols, except the miners, who stream the all the symbols. In case if only one node is streaming the block, the data is re-requested and the rest 50% are streamed.

7. The average bandwidth distributions are taken from [12] which for these simulations ranged below 50 MB per second.

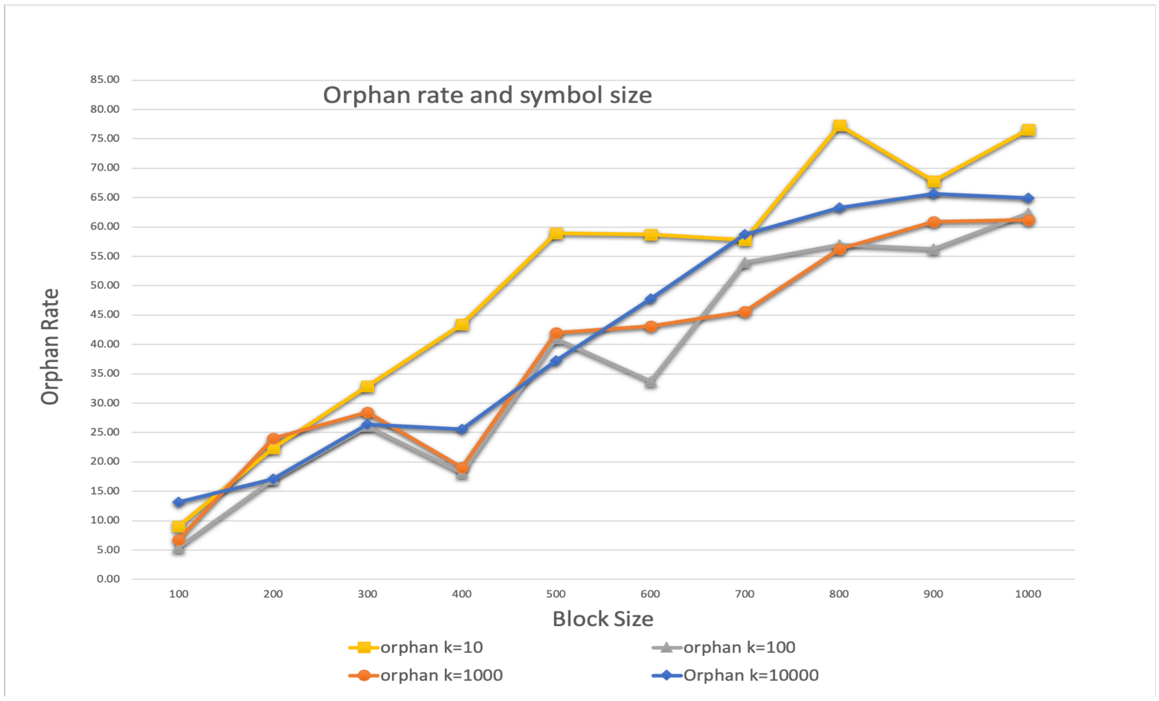

Figure 4.1: Orphan rate with respect to symbol Sizes

Choosing the symbol size

In order to aggregate the symbols from the peer nodes, we need to understand what ratio of the block size should the symbol size be in order for the propagation to be optimal for the bandwidth of the nodes that we have tested.

The following are the observations based on 4.1. In the graph, `k` refers to the number of symbols required to reconstruct the block. All symbols are equal in size.

1. For $k = 10$, we see that for 100MB blocks although the orphan rate starts with a low value of 9%, it increases fast as the block size increases. This is because the symbols sizes become larger than the upload bandwidth for the node and therefore take time to propagate.

2. For $k = 100$ and $k = 1000$, the orphan rate increases at almost the same rate. It can be noticed that although the symbol sizes differ by a magnitude of 10,

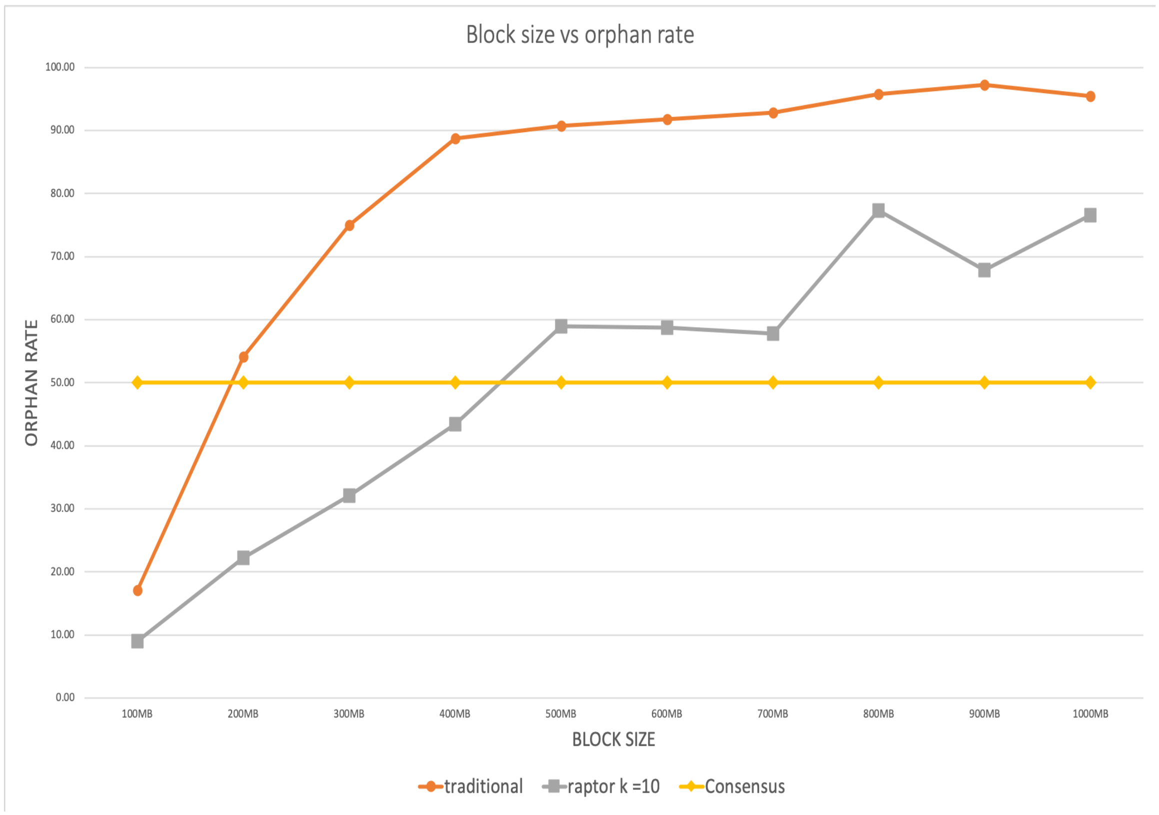

Figure 4.2: Block Size vs Orphan Rate k=10

for block sizes between 100MB and 1000MB, symbols will vary between 1MB and 10MB, which are still not utilizing the complete bandwidth of the node and hence take lesser time to propagate and therefore less orphanage.

3. For $k = 10000$ in smaller block sizes, between 100MB and 400MB, the performance is optimal, although when the block sizes grow, it is seen that the orphan rate is similar to that of $k = 100$ and $k = 1000$, which is still significantly less than the original block propagation.

4.3.2 Results

Orphan Rates

From graphs 4.2, 4.3, 4.4, 4.5 we can observe,

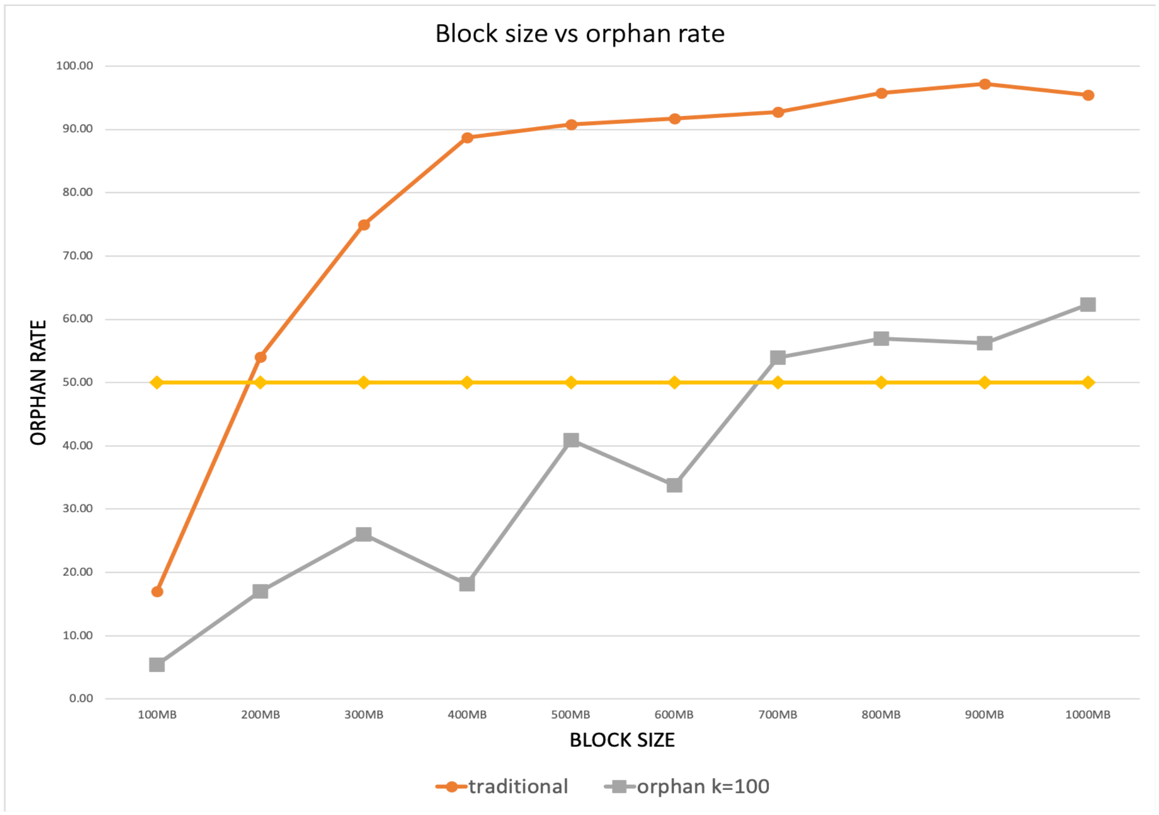

Figure 4.3: Block Size vs Orphan Rate k=100

1. The orphan rate in traditional relay reaches close to 97%, that is that almost all blocks produced get orphaned when we run a 1 GigaByte simulation. That said, for all block sizes between 100MB and 1GB, the network is unable to maintain consensus.

2. On the contrary, we see that when the data is aggregated from multiple peers, the orphan rate is contained below 50% for a 400MB block with `k` = 10 and for `k` between 100 and 10000, the network maintains consensus below 600MB.

Bandwidth

1. **average bandwidth** Average bandwidth used by both cases is similar in our simulations, however we are sending no overhead symbols. In case of lesser than

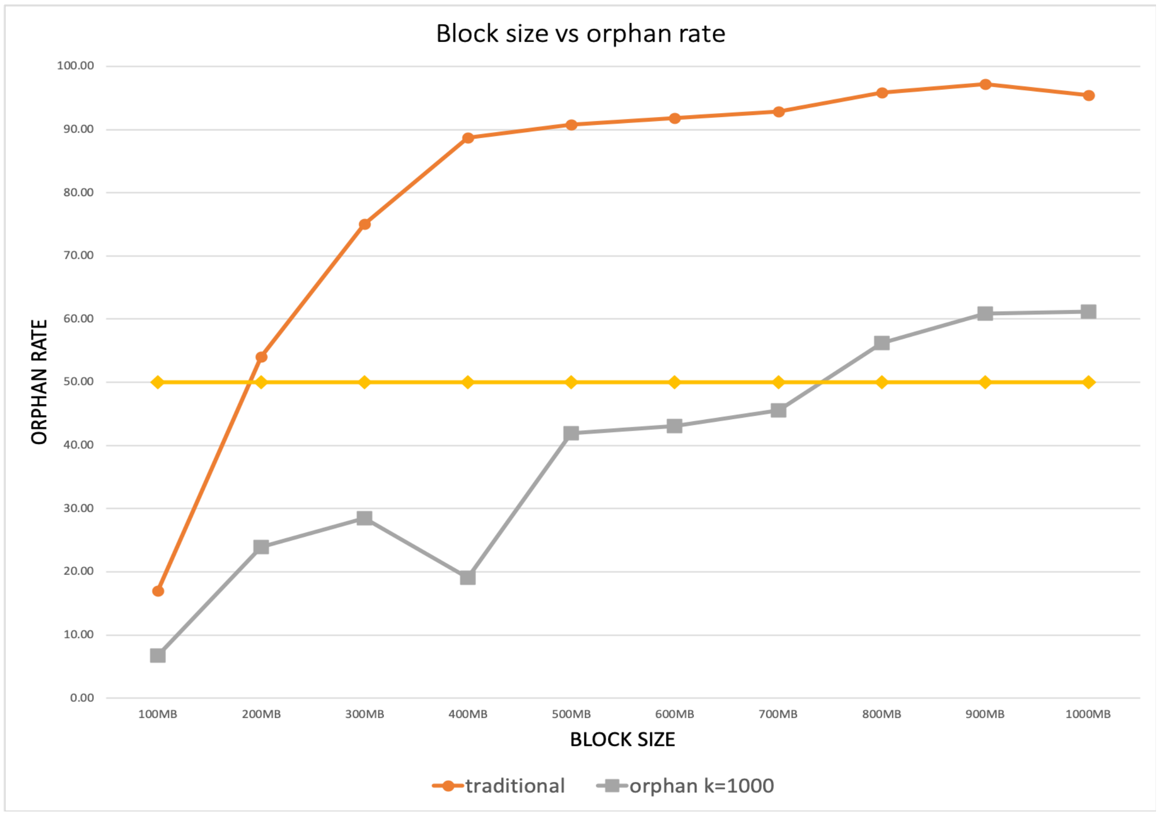

Figure 4.4: Block Size vs Orphan Rate k=1000

k symbols received, we ask the peers again to send more symbols.

2. The total number of `getraptor` are significantly higher, by a factor of 2, in some cases 3. That is because of us sending multiple requests to aggregate one block. This doesn't affect the overall bandwidth utilization because the size of the request is 36 bytes each.

Median Block Propagation

From graphs 4.6, 4.7, 4.8, 4.9, we can observe,

1. The median block propagation for block sizes in traditional block propagation elevate above 2.5 minutes in 200MB block sizes, and that is noticeable in 4.2

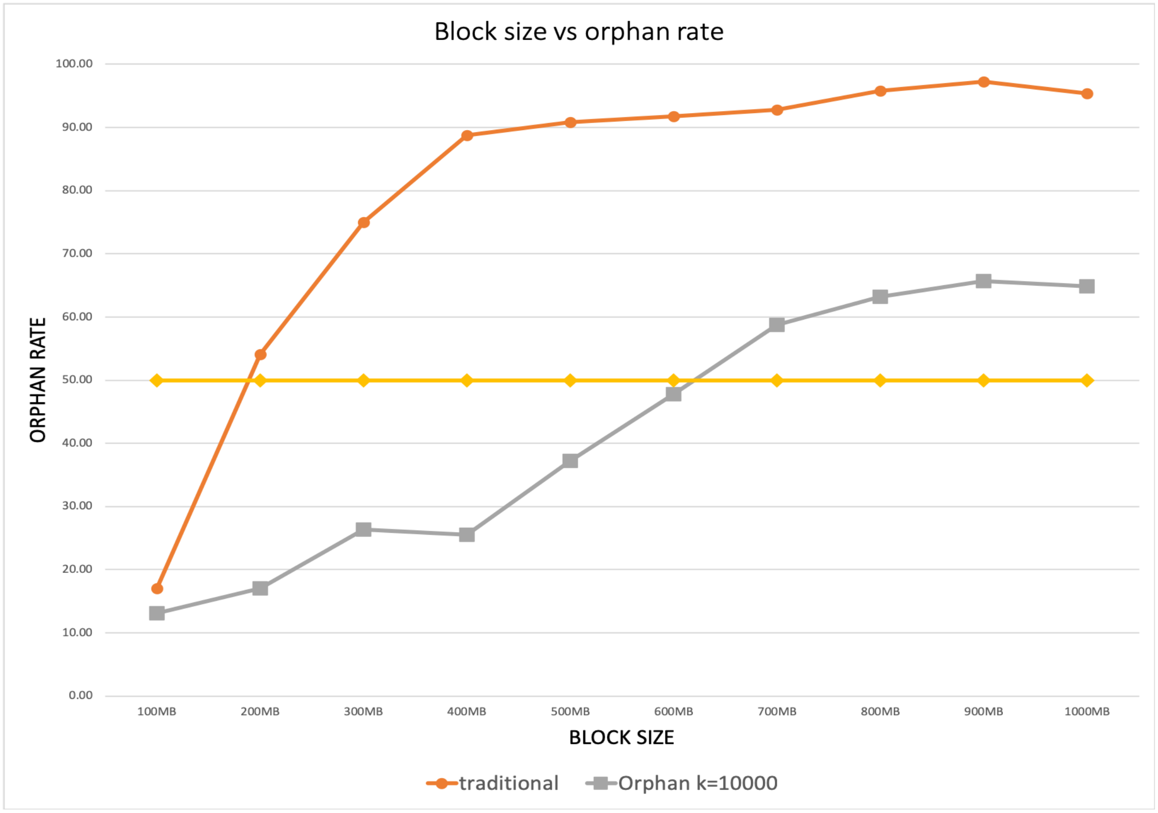

Figure 4.5: Block Size vs Orphan Rate k=10000

that 200MB blocks are not in consensus.

2. The median block propagation remains below 2.5 minutes for Raptor Code Propagation till 400MB for `k=10` and remain below below 2.5 minutes for 600MB blocks for `k` between 100 and 10000. Therefore 4.2 shows that consensus is maintained till 400MB blocks for `k=10` and 4.3, 4.4 and 4.5 show consensus is maintained for 600MB blocks for `k` values between 100 and 10000.

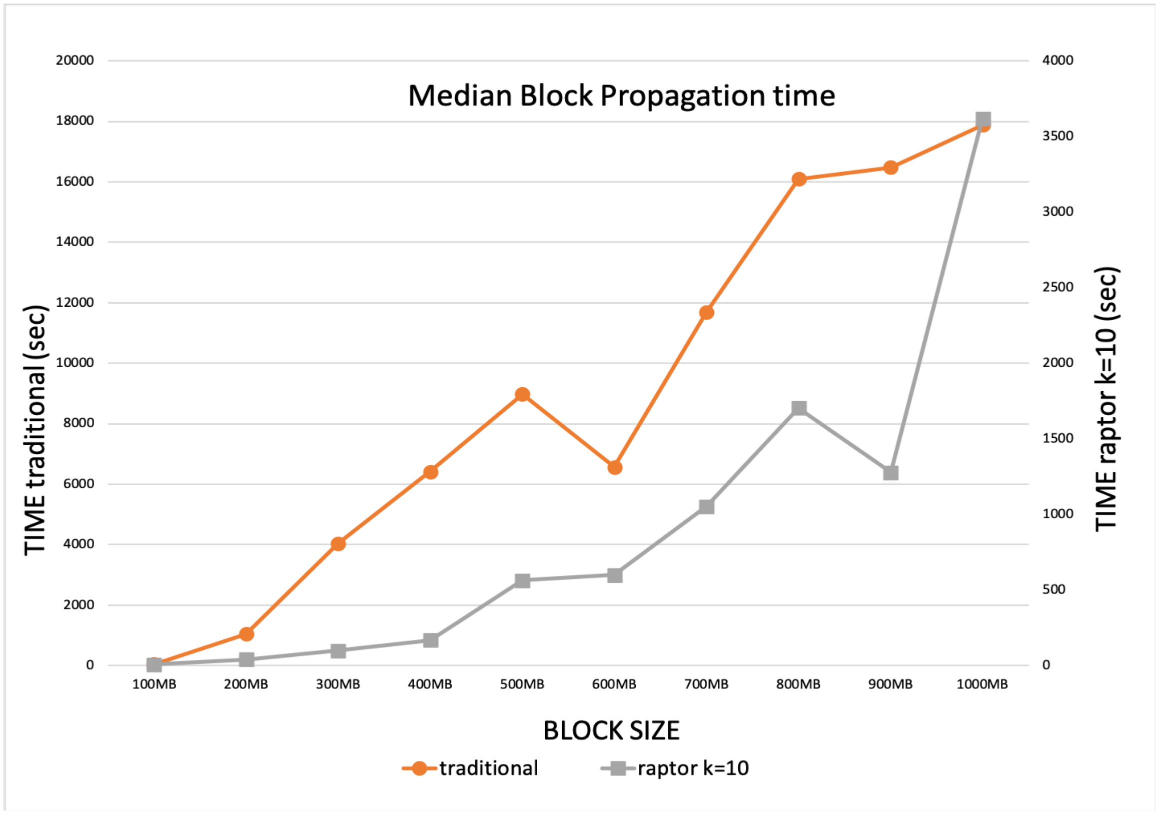

Figure 4.6: Median Block propagation k=10

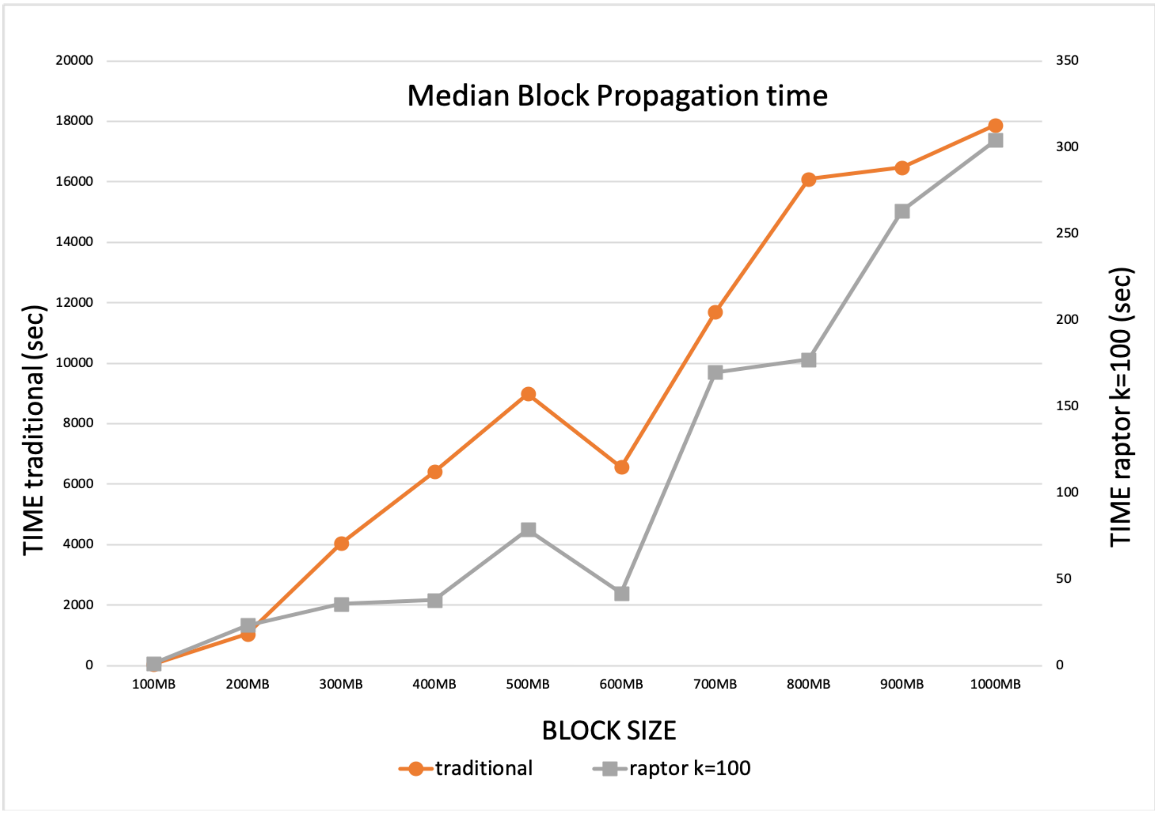

Figure 4.7: Median Block propagation k=100

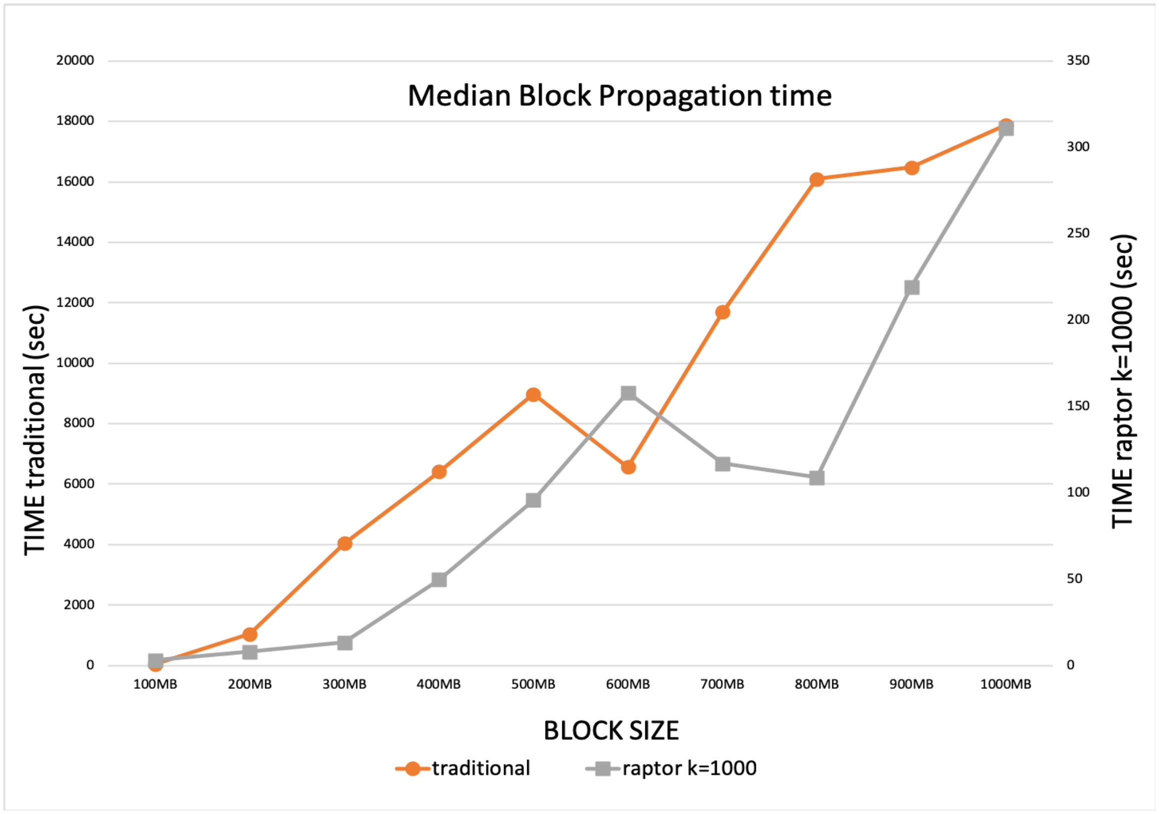

Figure 4.8: Median Block propagation k=1000

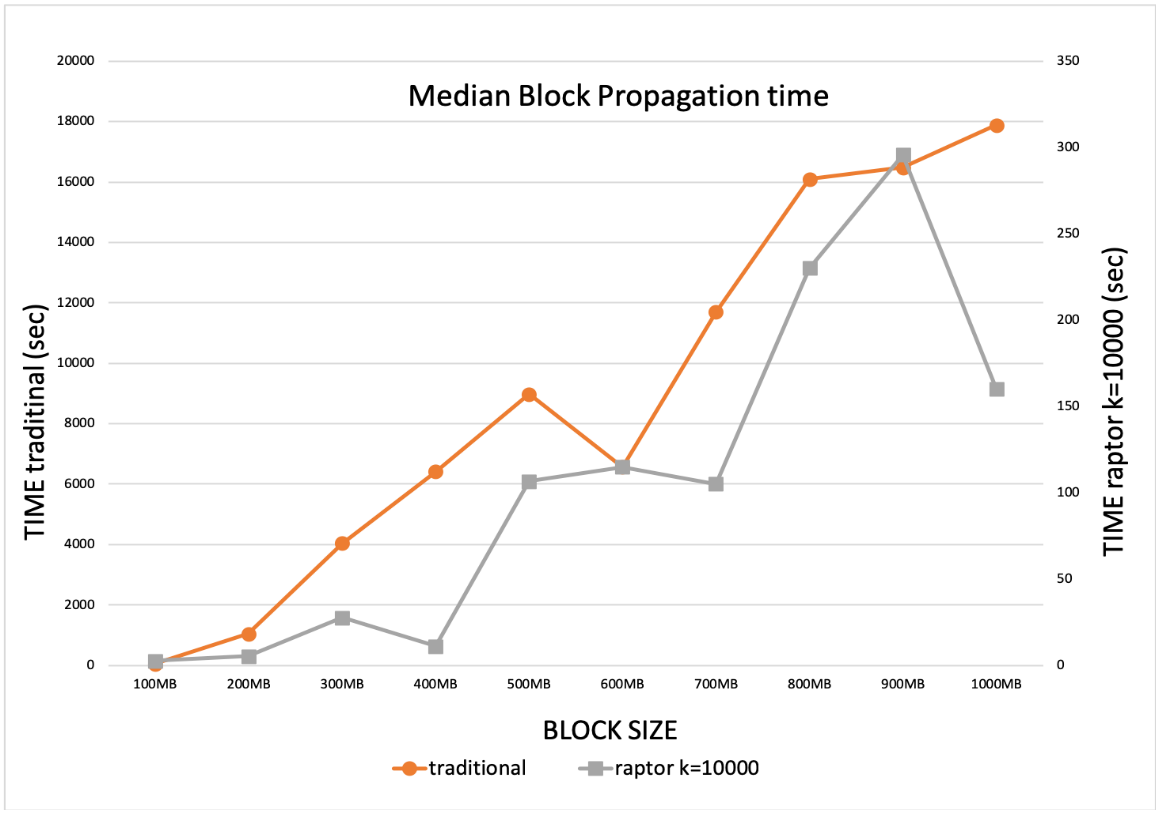

Figure 4.9: Median Block propagation k=10000

Chapter 5

CONCLUSION

Analyzing the above results, we can conclude that utilizing data aggregation techniques on a node for data propagation works better than syncing block using one peer. It must be noted that full nodes are personal machines that do not have high bandwidth. They get several advantages when only relaying symbols instead of the full blocks:

1. Miners have the capability of multi-casting. In the experiments, we have used TCP connections, but we have scheduled these uploads simultaneously. They can simultaneously serve large blocks to many nodes by continuous asynchronous streams, which wasn't possible in the traditional relay.

2. The data received by a full node is downloaded through multiple peers in parallel which results in faster downloads.

The above advantages, help nodes to propagate large blocks faster. Although, The current propagation techniques result in much faster propagation and lower orphan rates. Those (compact, XThin, Graphene) techniques have been tested on smaller blocks and they are highly dependent on memory pool synchronization. If memory pools (of the peers propagating blocks) do not synchronize, the methods fail to relay blocks faster and rely back on the traditional propagation. This thesis is intended to put the idea that today's state of art of propagation techniques (compact, XThin and Graphene) can take advantage of fountain codes for relay as a complimentary technology to make propagation faster.

5.1 Future Work

We have seen from the above experiments that the orphan rate is significantly reduced when sending the Fountain codes instead of leeching the block from a single client. The orphan rate for a 100 MB blocks reduces from 17% to 9% which is very profitable for a miner. We can also conclude that the network is able to maintain consensus upto 600MB blocks as compared to 100MB in traditional. We also know that orphan rates is much lower and transmission speed is higher in techniques like Compact blocks, XThin blocks and graphene blocks because they efficiently reduce the payload while sending the draft/sketch of the whole block. The one major area where fountain codes can make these techniques better is when there is a need for re-transmission of block data (`blocktxn` ,`xblocktx` and `grblocktx` which block transactions for compact block, xthin blocks and graphene blocks respectively).

In order to show the advantage of using fountain codes in increasing the performance of compact, XThin and Graphene blocks, the protocol will be more complicated because memory pool of each nodes are being used for encoding and decoding.

REFERENCES

[1] "Bitcoin unlimited improvement proposal 10", https://github.com/BitcoinUnlimited/BUIP/blob/master/010.mediawiki, accessed : 2018-10-21 (2016).

[2] "Compact blocks faq", https://bitcoincore.org/en/2016/06/07/compact-blocks-faq/, accessed : 2018-10-21 (2016).

[3] "Bitcoin confirmation", Wikipedia https://en.bitcoin.it/wiki/Confirmation, accessed : 2018-10-20 (2018).

[4] "Bitcoin forks", Wikipedia https://en.wikipedia.org/wiki/List_of_bitcoin_forks, accessed : 2018-10-20 (2018).

[5] "Bittorent specification", Wiki = https://wiki.theory.org/index.php/BitTorrentSpecification, accessed : 2018-10-20 (2018).

[6] "Block size limit controversy", Wikipedia https://en.bitcoin.it/wiki/Block_size_limit_controversy, accessed : 2018-10-20 (2018).

[7] "Blockchain wiki", Wikipedia https://en.wikipedia.org/wiki/Blockchain, accessed : 2018-10-21 (2018).

[8] "Merkle tree", Investopedia =https://www.investopedia.com/terms/m/merkle-root-cryptocurrency.asp, accessed : 2018-10-20 (2018).

[9] "Mining difficulty", Wikipedia https://en.bitcoin.it/wiki/Difficulty, accessed : 2018-10-20 (2018).

[10] "Nonce", Wikipedia https://en.wikipedia.org/wiki/Nonce, accessed : 2018-10-21 (2018).

[11] "Soft fork", Wikipedia https://en.bitcoin.it/wiki/Softfork, accessed : 2018-10-20 (2018).

[12] "Testmy country", Online https://testmy.net/country, accessed : 2018-10-20 (2018).

[13] Bissias, G. and B. Levine, "Bitcoin unlimited improvement proposal 093", https://github.com/BitcoinUnlimited/BUIP/blob/master/093.mediawiki, accessed : 2018-10-21 (2018).

[14] Boscovic, D., N. Chawla and D. Tapp, "Block propagation applied to nakamoto networks", (2018).

[15] Byers, J. W., M. Luby and M. Mitzenmacher, "A digital fountain approach to asynchronous reliable multicast", IEEE Journal on Selected areas in Communications **20**, 8, 1528–1540 (2002).

[16] Byers, J. W., M. Luby, M. Mitzenmacher and A. Rege, "A digital fountain approach to reliable distribution of bulk data", ACM SIGCOMM Computer Communication Review **28**, 4, 56–67 (1998).

[17] Chawla, N., "Dash simulator", Github `https://github.com/thenakulchawla/dash-simulator.git`, accessed : 2018-10-20 (2017).

[18] Cohen, B., "Incentives build robustness in bittorrent", in "Workshop on Economics of Peer-to-Peer systems", vol. 6, pp. 68–72 (2003).

[19] Corallo, M., "Compact block relay", `https://github.com/bitcoin/bips/wiki/Comments:BIP-0152`, accessed : 2018-10-21 (2016).

[20] Costa-Montenegro, E., P. S. Rodríguez-Hernández, C. López-Bravo and A. B. Barragáns-Martínez, "Analysis of the bittorrent protocol modified with fountain code", in "P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2012 Seventh International Conference on", pp. 33–40 (IEEE, 2012).

[21] Duffield, E. and D. Diaz, "Dash whitepaper", Accessed : 2018-10-10 (2018).

[22] Fulchir, L., "libraptorq", Github `https://github.com/LucaFulchir/libRaptorQ`, accessed : 2018-06-01 (2018).

[23] Gervais, A., G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf and S. Capkun, "On the security and performance of proof of work blockchains", in "Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security", pp. 3–16 (ACM, 2016).

[24] Goodrich, M. T. and M. Mitzenmacher, "Invertible bloom lookup tables", arXiv preprint arXiv:1101.2245 (2011).

[25] Iansiti, M. and K. R. Lakhani, "The truth about blockchain", Harvard Business Review **95**, 1, 118–127 (2017).

[26] Karp, R., M. Luby and A. Shokrollahi, "Finite length analysis of lt codes", in "Information Theory, 2004. ISIT 2004. Proceedings. International Symposium on", p. 39 (IEEE, 2004).

[27] Luby, M., "Lt codes", in "null", p. 271 (IEEE, 2002).

[28] Luby, M., A. Shokrollahi, M. Watson, T. Stockhammer and L. Minder, "Raptorq forward error correction scheme for object delivery", Tech. rep. (2011).

[29] Luby, M. G., M. Mitzenmacher, M. A. Shokrollahi and D. A. Spielman, "Efficient erasure correcting codes", IEEE Transactions on Information Theory **47**, 2, 569–584 (2001).

[30] Nakamoto, S., "Bitcoin: A peer-to-peer electronic cash system", (2008).

[31] Pallis, G. and A. Vakali, "Insight and perspectives for content delivery networks", Communications of the ACM **49**, 1, 101–106 (2006).

[32] Parisis, G., V. Sourlas, K. V. Katsaros, W. K. Chai and G. Pavlou, “Enhancing multi-source content delivery in content-centric networks with fountain coding”, in “Proceedings of the 1st Workshop on Content Caching and Delivery in Wireless Networks”, p. 4 (ACM, 2016).

[33] Parisis, G., V. Sourlas, K. V. Katsaros, W. K. Chai, G. Pavlou and I. Wakeman, “Efficient content delivery through fountain coding in opportunistic information-centric networks”, Computer Communications **100**, 118–128 (2017).

[34] Pathan, A.-M. K., J. A. Broberg, K. Bubendorfer, K. H. Kim and R. Buyya, “An architecture for virtual organization (vo)-based effective peering of content delivery networks”, in “Proceedings of the second workshop on Use of P2P, GRID and agents for the development of content networks”, pp. 29–38 (ACM, 2007).

[35] Rizun, P. and A. Stone, “Scaling bitcoin stanford”, Youtube `https://www.youtube.com/watch?v=5SJm2ep3X_M`, accessed : 2018-10-20 (2017).

[36] Shokrollahi, A., “The development of raptor codes”, in “2008 IEEE Information Theory Workshop”, (2011).

[37] Shokrollahi, M. A., “Advanced raptor codes”, Online `https://algo.epfl.ch/_media/en/output/presentations/raptor-bangalore.pdf`, accessed : 2018-10-20 (2006).

[38] Shokrollahi, M. A., S. Lassen and R. Karp, “Systems and processes for decoding chain reaction codes through inactivation”, US Patent 6,856,263 (2005).

[39] Shokrollahi, M. A. and M. G. Luby, “Systematic encoding and decoding of chain reaction codes”, US Patent 6,909,383 (2005).

[40] Sompolinsky, Y. and A. Zohar, “Accelerating bitcoin’s transaction processing”, Fast Money Grows on Trees, Not Chains (2013).

[41] Vakali, A. and G. Pallis, “Content delivery networks: Status and trends”, IEEE Internet Computing **7**, 6, 68–74 (2003).

[42] Verma, D. C., *Content distribution networks: an engineering approach* (John Wiley & Sons, 2003).