Using IBLTs for Bitcoin Memory Pool Synchronization and Improving the Network

By Christopher DeLucia (ENG ‘17)
Advisor: Professor David Starobinski

Date: May 9th, 2017
Directed Study

Abstract

There have been countless proposals to help scale blockchain technologies. Specifically, Bitcoin has been pressed hardest with scalability challenges; as it has the largest market cap and deals with the highest transaction volume.[1] Of the many scalability challenges, one challenge lies in bandwidth efficiency and nodes joining and leaving the network. As Bitcoin continues to expand and support more and more transactions, every second counts. While the usage of Compact Blocks and other protocols have cut block propagation times drastically, there are still extra round trips that need to be made between nodes for missing transactions that were included in the block but didn’t exist in a receiving node’s transactional memory pool. To keep the transactional memory pools as similar as possible, we can use an Invertible Bloom Lookup Table (IBLT) to solve the set reconciliation problem between nodes when they leave the network or go offline. Through the use of the IBLT, Bitcoin mempools would be more synchronized and as a result hopefully see more bandwidth usage on the network.

Introduction

The idea of using an Invertible Bloom Lookup Table (IBLT) to improve the Bitcoin network is not new. In 2014 Gavin Andresen, former lead maintainer for Bitcoin Core, proposed the idea of using IBLTs to synchronize Bitcoin transactional memory pools to improve block propagation through the Bitcoin network.[2] The average transaction size today is about 500-600B and there’s upwards of 2000 transactions per block, so the vast majority of memory in the block is made up of transactional data.[3] At the time, miners solving a block were concerned with producing blocks close to the maximum size (1MB) due to the propagation delay of a larger block across the network and losing a “block race” against smaller blocks that could propagate faster and reach more nodes--increasing the likelihood the block would be solidified on the block chain. This was bad for Bitcoin since it was incentivizing miners to include less transactions in their blocks and transaction fees (fees on a transaction that go to the miners and are determined by the node transferring bitcoin) were inflating since nodes would have to increase their fees to attract miners to include their transaction in the current block.

[1] https://coinmarketcap.com/
[2] https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2
[3] https://tradeblock.com/blog/analysis-of-bitcoin-transaction-size-trends/

Invertible Bloom Lookup Tables sought to solve this problem by encoding transactional data in a fixed-size, compact IBLT that would be propagated with a 80B block header, salt (used for key in the table), and some information about the size of the transaction data. The transactions that were to be inserted into the IBLT would come from the transactional memory pool. When a node receives a valid transaction that was broadcasted through the network, it is added the transaction memory pool (txn-mempool) and forwarded off to other nodes.[4] When the transaction is confirmed in the node's blockchain or too much time has passed without the transaction making it onto the blockchain, the transaction is removed from the node's mempool.[5]

The idea is a miner would construct an empty IBLT and select transactions from its mempool to include in the block based on highest priority and highest fee-per-byte. The miner would then insert these transactions into the empty IBLT based on a canonical order.[6] Once the miner has solved a block, it would relay the block header, size information, etc. and (instead of all the raw transactional data) the IBLT with the encoded transactional data to its peers.[7] Once a peer receives the IBLT, the peer would construct its own IBLT with transactions from its txn-mempool (based on byte-sized hints) and perform a diff operation (xor) between its IBLT and the one it received with the block. The hope is the result from the diff operation would be an IBLT that a peer could deconstruct completely and learn about transactions that are included into the block and those which the peer did not include and need to be removed. If the differences were too great between the nodes' txn-mempools or decoding failed for some reason, nodes would have to fall back to getting the full block from another peer or some other extension of an IBLT.[8]

Compact Block Relay and the Declination of Propagation Times

In April of 2016 a Bitcoin Improvement Proposal (colloquially "bip") was put forward by developer Matt Corallo entitled "Compact Block Relay" that would later get merged into Bitcoin core on June, 2016.[9][10] Compact Block Relay would come to create a more efficient bandwidth protocol and in turn also solve longer latencies with the block propagation problem that Gavin was trying to address in 2014 with Invertible Bloom Lookup Tables.

The protocol introduces a new way to send blocks in the form of a compact block. The protocol optimizes the sending of transactions by generating short transaction ids for transactions in the

[4] Note: Valid means syntactically correct, size requirements, input hasn't been referenced yet, etc. This is not the same as confirmed--which means a txn has made it onto a block on the blockchain. See full list of txn validity (https://en.bitcoin.it/wiki/Protocol_rules#cite_note-7)
[5] https://bitcointalk.org/index.php?topic=1845013.msg18357378#msg18357378
[6] Note: For definition and details of the "canonical order", see the original proposal under "Canonical Ordering of Transactions" (https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2)
[7] Note: This would be for peers supporting the IBLT version of the implementation. "Older" nodes you would still need to send the full block with the transactional data in it.
[8] https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2
[9] https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki
[10] https://github.com/bitcoin/bitcoin/pull/8068

node's mempool that would normally be included in a block. These short transaction IDs are generated by doing a SHA256 hash on the block header (with nonce appended) and running a SipHash-2-4 on the full transaction ID with results from the SHA256 hash. The end result is a 6B txn-id as opposed to the normal 32B. Compact blocks contain mostly the block header, the shortened txn-ids, and some other prefilled transactions explicitly that you presume your peer is missing.[11] The idea is that peer receiving the compact block would be able to reconstruct the full block on their own and proceed to validate it (*See Figure 1.1 in Appendix for protocol)*. This removes a lot of the raw transactional data that occupies the most memory during the relaying process.

The prefilled transactions are determined when a node receives the full block and it contains transactions that were missing from local sources (i.e mempool). This prefilled section can hold up to 6 transactions and is important because if a node receives a compact block and is missing more transactions than were fit into the short tx ids and prefilled sections, it will need to undergo an additional round trip 'getblocktxn/blocktxn' exchange.[12][13] This threshold difference between nodes mempools adds round trips for the network and can increase network latency and bandwidth. This is a detail that will become important later on in the IBLT protocol development.

FIBRE Network

Matt Corallo (the same author of the Compact Block BIP) did not stop at just at the Compact Block Relay. While working on the way to compress the blocks, he was also working on what would come to be known as the Fast Internet Bitcoin Relay Engine (FIBRE). FIBRE is a relay network that improves the Bitcoin Relay network by eliminating latency spikes due to missing packets as well as fully utilizing the Compact Block Relay.[14] The network uses a UDP based protocol (instead of TCP used in the Bitcoin network), taking advantage of forward error correction for errors that occurred in transmission. Currently, there are 6 relay nodes that are distributed across the world and act as dedicated servers to rapidly relay compact blocks across the globe. These relay nodes are extremely fast and can relay compact blocks over long distances in milliseconds.[15] The FIBRE Network has significantly reduced latencies and has proved to be successful for allowing blocks to propagate across the network. And while this adds some centralization to Bitcoin (since this network and servers are maintained by Corallo), it can be argue it allows for smaller miner pools to have a fairer chance at earning their revenue proportional to their work. Many large mining pools already have their own protocol in place for relaying blocks to each other, so this initiative can be viewed as a way to "level the playing field".

[11] Note: Namely the coinbase transaction as well as other transactions that were missing from receiving node's transaction mempool but were included in the block. If you had missing transactions in your mempool is possible your neighbor nodes are also missing them.
[12] https://bitcoincore.org/en/2016/06/07/compact-blocks-faq/
[13] *See Figure 2.1a in Appendix for visual*
[14] http://bitcoinfibre.org/
[15] Note: Full stats on the FIBRE Network can be found http://bitcoinfibre.org/stats.html

IBLT and Compact Block Relay

The Invertible Bloom Lookup Tables solve the set reconciliation problem well--that is, determining differences between sets with mostly similar data. The Compact Block Relay in combination with FIBRE solved the problem of efficiently propagating blocks across the network, however, its optimal efficiency relies on nodes having seemingly small differences of transactions in their respective mempools. This leads to a couple of questions: how often are these mempool differences a problem and is there a way IBLTs can be used to synchronize mempools amongst peers that would improve the relay, bringing it closer to maximum efficiency?

Part of the uniqueness of the Bitcoin network is that it's constantly changing. At this second a estimated number of reachable nodes is about 7000.[16] However, this number fluctuates by the thousands as nodes can leave the network for an hour, day or even months and rejoin. Full Nodes and some miners maintain a local copy of the blockchain as well as maintain a mempool and unspent transaction output (UTXO) data. This is because they need to enforce the rules of the network by validating transactions (checking syntax, referenced outputs, size requirements, etc.).[17] When these nodes leave the network and return, they must first obtain missing blocks from their peers in a "getdata/block" exchange and validate those blocks from the latest block height they left off at.[18]

When nodes have been online pretty consistently, the network does pretty well with transmitting a compact block on the first try. According to Corallo, when nodes are online for about 3-4 weeks they can receive these compact blocks without having to do additional round trip for missing transactions about 50% of the time. He mentions almost always when you miss it's between 1-5 txns each time and not a large chunk of a block.[19] This is pretty impressive considering that a transaction mempool can contain on the order of 5K-65K transactions, with a couple of thousand being removed roughly every ten minutes when included in a block and some expiring.[20] Yet despite this impressive difference given the number of transactions in the pool, this begs the question what happens when nodes are not yet warmed up for a few weeks?

In the Bitcoin Core version 0.14.0 release a change was added where a node will save its current transaction mempool in a 'mempool.dat' file prior to shutting down.[21] Since this file is saved on disk, when the node rebooted it could load its previous saved mempool as opposed to starting with an empty mempool. This is an improvement because now it is less likely for there to be a greater

[16] https://bitnodes.21.co/
[17] http://chimera.labs.oreilly.com/books/1234000001802/ch06.html (From *Mastering Bitcoin* by Andreas Antonopoulos)
[18] http://chimera.labs.oreilly.com/books/1234000001802/ch06.html#spv_nodes (From *Mastering Bitcoin* by Andreas Antonopoulos)
[19] Matt Corallo (matt@mattcorallo.com) and Chris DeLucia (deluciac@bu.edu) personal email correspondence on March 24, 2017.
[20] http://statoshi.info/dashboard/db/memory-pool
[21] https://bitcoin.org/en/release/v0.14.0

difference in nodes mempools when they restart which translates to a greater probability of relaying a compact block without requiring extra round trips.

Protocol Development

I present three scenarios that a node may experience and put forth a proposal using IBLTs at an attempt to reduce the overhead on the network and make for a more efficient use of bandwidth. To reiterate, the problem I am trying to solve is when a node has either restarted or has been absent from the network for a part of time and thus faces additional RTs for some period of time due to transactions missing in their mempool (unable to generate the short txn ids for relay and block reconstruction).

Let's define X, Y, Z as time variables (in sec) such that: $X > Y > Z$. These variables represent a time that a node has "left" the network or hasn't been accumulating transactions in their mempool.

Additionally, we can define N, M as transactional differences between mempools of nodes (in number of txns) with $N > M$. Later, we can attempt to come up with values for these variables, but for now variables will serve as place holders.

Scenario 1

The booting node has been offline for X time s.t. their mempool has overturned; that is almost all transactions stored in the latest mempool are no longer in peers' current mempools.

If a node has gone offline for some longer period of time X and their latest stored mempool contains transactions that were either confirmed or expired, it can use the current 'mempool' message to obtain a latest mempool from another peer.

Scenario 2

The booting node has been offline for Z time s.t. the mempool mostly differed by a value M; that is the latest transactions stored in the mempool do not differ much from current online peers' mempools and the booting node can reliable perform compact block relays without roundtrips >50% of the time.

Suppose that a node has only left the network for some small fraction of time and is still "warm". Their mempool doesn't differ very much from the neighboring peers and so they don't miss many transactions in the Compact Block Relay and can reliably transmit compact blocks without requiring addition RT most of the time. The IBLT in the scenario would be too much overhead for what it's worth. In this case, the node doesn't need to request for additional transactions and so they just resume the usual activity of accepting transactions and relaying blocks.

Scenario 3

The booting node has been offline for some time Y s.t. the difference between the stored mempool and online peers is N; that is it would take some significant amount of time with a node online before the node can relay about 50% of the compact blocks without additional roundtrips.

This is the scenario in which I feel the IBLT can be utilized. Based on the current system, nodes that fell in this category would have to send a 'getblocktxn' request to the peer it received the compact block from and receive 'blocktxn' data whenever it was missing transactions from its mempool that were included in the block. This could go on for a few weeks, missing a few transaction most of the time until the mempool has turned over enough.

Instead of putting up with these additional trips, I propose using an IBLT to encode mempool transactions and find out which transactions the booting node has missed. Instead of accepting a $<$50% successful relay rate for weeks, the booting node would send a request (ex: send_IBLT) for a peer to reply with an IBLT of the transactions in its current mempool. The peer would send the IBLT with the encoded data (ex: $\text{IBLT}_{\text{peer}}$). Upon receiving $\text{IBLT}_{\text{peer}}$, the boot node would construct its own IBLT with transactions from its latest mempool and perform a diff operation between the peer's IBLT and its IBLT ($\text{IBLT}_{\text{peer}}$ - $\text{IBLT}_{\text{boot}}$). The purpose of this is to attempt to recover the missing transactions resulting from this diff and add them to their mempool. The hope would be sending the $\text{IBLT}_{\text{peer}}$ once would be enough to surpass a significant time of unnecessary data transfer due to these missing transactions.

Quantitative Analysis

The above is a general protocol specified with variable parameters. This protocol still leaves out key questions such as what would be the size of an IBLT to successfully recover transactions with *N* differences? How long would it take to overturn a certain mempool or have too many differences? At what point is this approach better than the additional roundtrips? To answer some of these questions, we can do some back-of-the envelope estimations to come up with approximate values and have an idea of the metrics of the IBLT approach.

It is important to have an idea of how long it would take the mempool to overturn. We can roughly estimate this by calculating the following:

$$\#\ of\ txns\ in\ mempool = txns\_entering - txns\_leaving + C_i \qquad (Eq.\ 1.1)$$

Where C_i is the number of txns already in the mempool. But let's ignore that for now and just assume $C_i = 0$. If this is the case, the above formula can be equated to

$mempool\ txns = TPS*t - t/T_C$

Where TPS is the number of transactions added to the mempool per second (txns/s), T_C is the median time it takes to confirm a transaction into the block (i.e leave the mempool) in seconds, and *t* is the time on the network in seconds. By factoring out *t* we can finally come up with a general expression

$mempool\ txns = [(TPS*T_C - 1)/T_C]*t$ *(Eq. 1.2)*

or solving for *t*

$t = (mempool\ txns)/[(TPS*T_C - 1)/T_C]$ *(Eq. 1.3)*

For example (1.1), suppose the following conditions[22]:

E1.1

$TPS = 4.30\ txns/sec$
$T_C = 960\ sec\ (16\ min)$
$Mempool\ txns = 15000$

Using Eq. 1.3:

$t = (15000)/[(4.3*960 - 1)/960] = 3489\ sec$ *or approx. 58 minutes*

In this case, if a node was offline for about 58 minutes (our *X* value in Scenario 1) and had 15K txns in its mempool, the node may presume the mempool has overturned a lot of its transactions and use a 'mempool' message if desired to obtain a peer's mempool.

Parameter Values for IBLT

Exploring a value for when it is not worth the overhead (*Z* time offline w/ M differences from Scenario 2) is something that would require more guess-work and would vary depending on the network conditions. It's probably safe to say some milliseconds offline or disconnected from the

[22] Note: It would be a better estimate to use averages from when the node was online, as opposed to the current network conditions

network would not be worth sending the IBLT. However, exploring the upper limit is more challenging. It is unclear at what point we would stop scaling the IBLT to recover differences (i.e if we were missing 10k transactions from a 60k pool, what size IBLT would we need to recover that?).

Rusty Russell is an Infrastructure Tech Engineer at Blockstream--a company that provides blockchain solutions.[23] In 2014 he did some experimenting with Gavin's idea of the IBLT and looked at some stats about IBLT sizing and recoverability with real transaction data from his Bitcoin node. He came up with a lot of interesting data and graphs to show the range of the IBLT, experimenting with factors like table and bucket sizing. He started testing with a 1 MB table, but also looked at the recoverability of a 30 KB table. He showed that such a table could still fully recover transactional data with 30 transaction difference between two peers (with 64 byte slices or buckets for the table).[24] If we were adding transactions to our mempool at a rate of 3 tps, it would take roughly 10 seconds to build up this difference. With a 1MB table, he was able to recover the transactions if there was a difference of 600-700 transactions.[25] It's worth noting that doubling the size of the table gives us less than double the amount of recoverability.[26]

Using Russell's data as an example, we could come up with the following for an implementation of using IBLTs (for simplicity, assume we don't want a peer to send over a 1MB table).

Mempool Differences (# of txns missing)	Resolution
$0 < N < 30$	Continue validating transactions and cmpctblks, occasionally requiring additional RT for missing txns
$30 \leq N \leq 600$	Request 30 KB - 1MB IBLT from peer
$N > 600$	Request 'mempool' from peer

It would require more research and simulation to come up with the limits, but the main takeaway is there's some differences that we can use to reconcile with an IBLT. And the hope would be that this would be more efficient in bandwidth than sending the round trip data involved with 'getblocktxn/blocktxn'.

[23] https://blockstream.com/about/#mission

[24] http://rustyrussell.github.io/pettycoin/2014/11/05/Playing-with-invertible-bloom-lookup-tables-and-bitcoin-transactions.html

[25] Note: This presumes your peer had your transactions and 600-700 more. For our scenarios we can presume a peer who has been offline has all the missing transactions relative to a peer that was online

[26] http://rustyrussell.github.io/pettycoin/2014/11/05/Playing-with-invertible-bloom-lookup-tables-and-bitcoin-transactions.html

Criteria For Evaluation

With more research to be done with mempool difference estimation and IBLT specifics, it is possible to come up with better defined boundaries. Until a personal full node is running, we must rely on public stats that are available for information.

Statoshi.info (a pun on Bitcoin creator "Satoshi Nakamoto") was created by an engineer called Jameson Lopp. This is an aesthetically beautiful site that publishes node statistics with tons of graphs depicting things like bandwidth consumption and mempool memory size.[27] Using information from his site, I was able to gather some statistics on bandwidth usage for the 'getblocktxn/blocktxn' exchange.

Statistics taken on April 23rd 2017 at around 8:00PM EST showed that the 'blocktxn' message had a max bandwidth of about 13 KBps and an average bandwidth of 155 Bps.[28] This high max, but low average indicates this message is mostly a high spike when transmitting and low otherwise. If we use this average of 155 Bps and estimate the range of data being used over 3 weeks, we can calculate its somewhere between 140 MB (50% success of no additional RT) and 281 MB (0% success rate of no additional RT)[29]

Through the use of an IBLT it is possible to save bandwidth in the long term. If we use the same network statistics from above and assume that after the 3-4 weeks we would have a successful no additional RT rate of $\geq 50\%$, we could make a comparison. For the 3 weeks, our range for data would be between $(140 + N)$ MB (50% success of no additional RT) and $(28 + N)$ MB (90% success of no additional RT) where N is the size of the $\text{IBLT}_{\text{peer}}$ in megabytes.[30] Of course there is more to be done but with these rudimentary estimations it seems we have the potential to save a decent amount of bandwidth.

[27] http://statoshi.info/
[28] *Note: The 'getblocktxn' message occupied negligible bandwidth*
[29] *See Figure 2.1b from Appendix*
[30] *See FIgure 3.2 from Appendix*

Queue Model for Mempool

As mentioned previously, the information available for Bitcoin node activity only comes from a few sources. The only way to see what is actually going on between peers and data usage is to run a full node for yourself. Until that day comes, we can still come up with basic models to simulate certain network conditions influence.

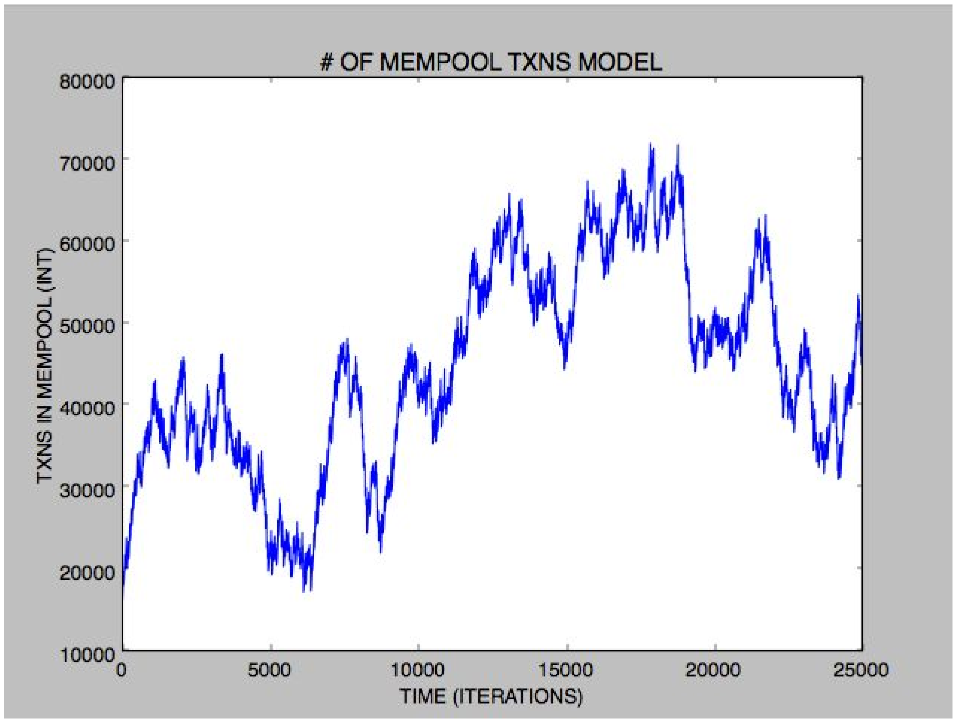
I have developed a simple Python-based transaction mempool model for simulations. We can use this model to look at which network conditions caused spikes or drops in the mempool. Additionally, we can use the model to specifically look at changes in the mempool overtime to simulate how a mempool might change while a node is offline. This model serves as a basic building block and could be developed further to test implementations that rely on changes in the mempool.

Pseudo Code for Model

The actual code for this model has been implemented in Python using the 'matplotlib' to generate plots and the 'random' library to generate random *int* values. The model works as the following:

```
C_i = 15000 #initial number of txns in mempool
value = C_i #variable for num of txns in mempool
TPS = randint(2, 4) #txn/sec
N_AVG = 2000 #avg number of txns included in blocks
MINE = 60 #Interval to "mine" on
M = 25000 #Number of max time or iterations

i = 1 #time after "block" generated

mempool_set = [ ]

for t in range(1, M):
      if(t%MINE==0):
            value = value + TPS*i - N_AVG
            i = 0
            MINE = randint(55, 65)
            N_AVG = randint(2000, 2200)
      else:
            value = value + TPS*i
            i += 1

      randomize TPS every few t

      if (value < 0):
            mempool_set append 0
      Else
            Mempool_set append value


Plot mempool_set
Xlabel = "TXNS IN MEMPOOL"
Ylabel = "TIME IN ITERATIONS"
```

This model allows us to generate plots that look like the following:



Even though it is a very simplified model it fairly stable for large iterations and has bounds on the order of magnitude that is reasonable compared to real mempool data (*See Figure 4.1a-b in Appendix)*. We can use plots like this to generate data and simulate a node going “offline” (i.e at some *t* set TPS = 0 and no “mine”) and then returning online and evaluating the mempool differences.

Conclusion

Compact Block Relay has made significant improvements to the Bitcoin network. However, if mempools were perfectly synchronized, there wouldn't be redundant roundtrips and bandwidth to obtain missing transactions. And while there's still a lot to be done with exploring different values and analyzing data, it seems like an IBLT could be a solution to obtaining these missing transactions between mempool sets. If we could learn about missing transactions earlier on we can help nodes that are missing these transactions get back up-to speed on the network and push Compact Block Relay closer to maximum levels of efficiency.

There's still some mysteries to be solved; specifically why exactly it takes so long with current methods to reconcile missing transactions. This could have to do with transaction chaining originating with these missing transactions. More speculation could be done to answer this question but ultimately the best way to explore this behaviour would be to run an instance of a full node. This way network traffic could be analyzed, experiments run, and empirical information gathered on how long it takes a node to leave and return to the network until it stops sending and receiving a lot of 'getblocktxn/blocktxn' exchanges.

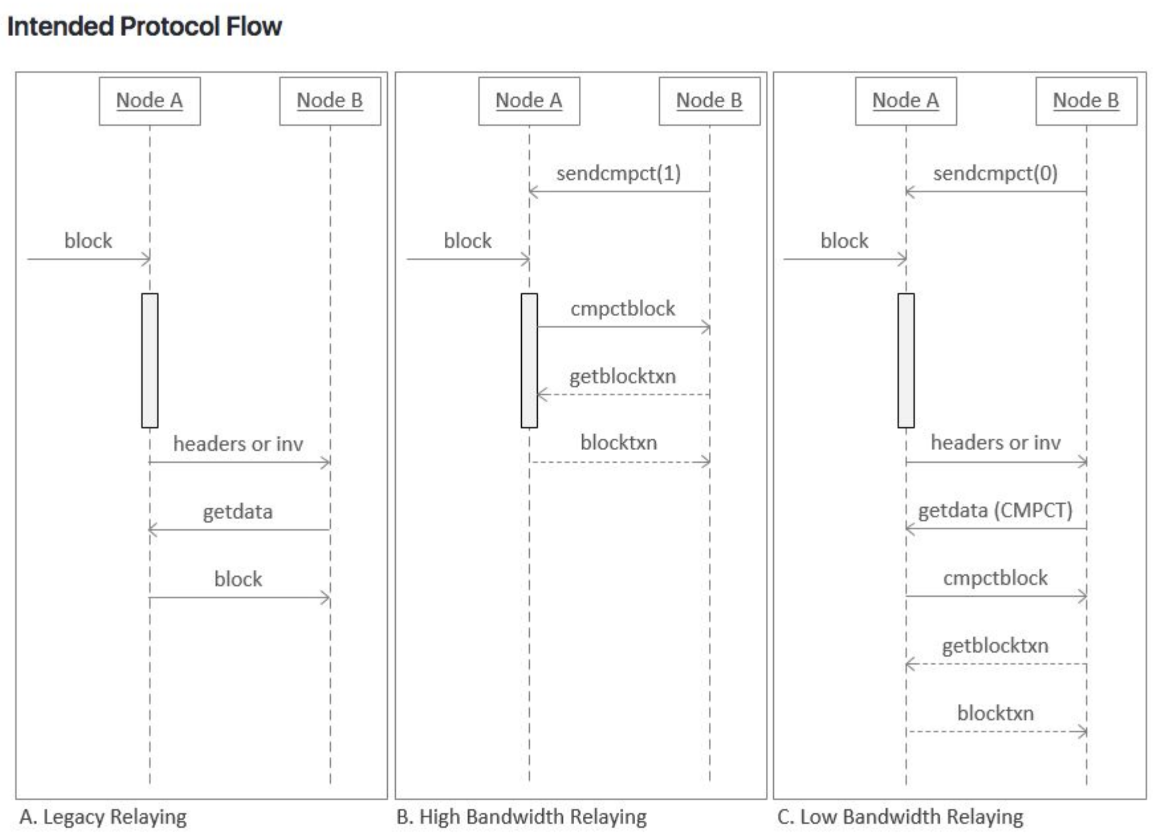
The Bitcoin network has undergone many changes over the years. Whether or not Bitcoin will eventually collapse can be up to speculation and debate. There is comfort in knowing that there are many passionate and intelligent people researching and maintaining the software everyday. There is a "Scaling Bitcoin" conference held in recent years where prominent members in the field discuss issues from wallet security to social problems facing the community.[31]

Despite the ongoing dialogue, one thing that is undeniable is that the value of bitcoins have grown substantially over the past few years. Currently 1 BTC is exchanged at over 1700 USD compared to a year ago when it was roughly 450 USD.[32]

[31] https://scalingbitcoin.org/
[32] http://www.coindesk.com/price/

APPENDIX

Fig. 1.1



Source: https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki

Fig 2.1a

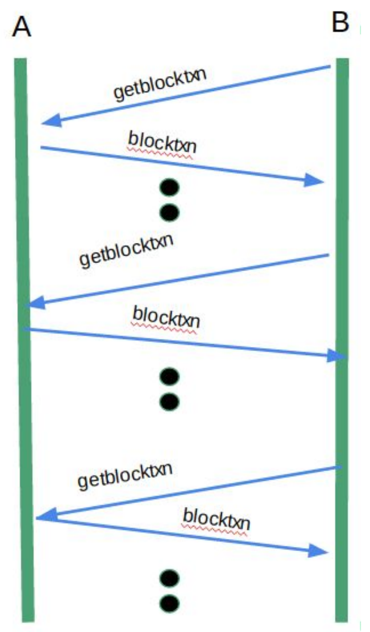


Fig 2.1b

3 WEEKS	4 WEEKS	SUCCESS %
281 MB	374 MB	0% (Always RT)
140 MB	187 MB	50% (RT half the time)

Approx. bounds: 140MB - 374MB

Fig 3.1

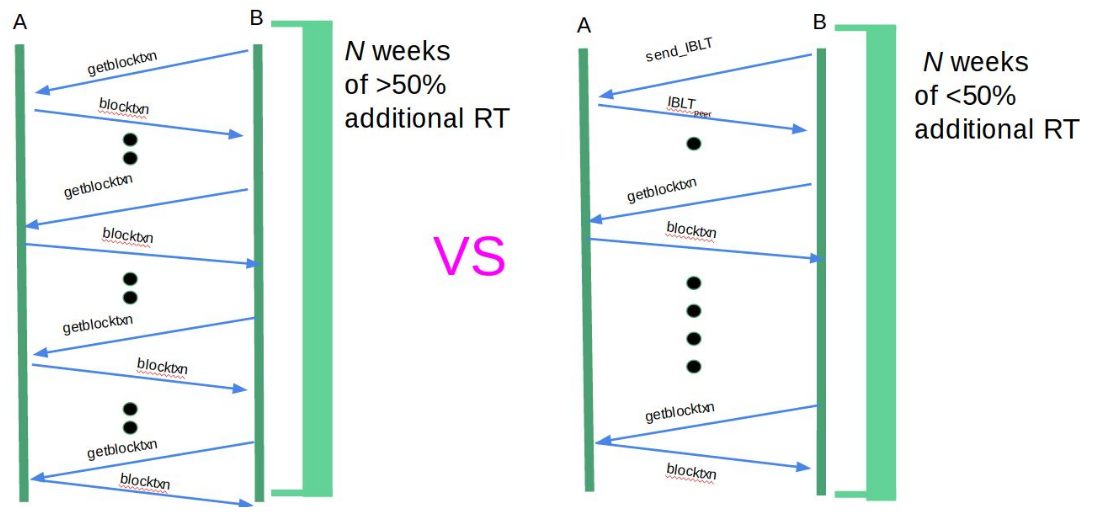


Fig 3.2

Legacy

3 WEEKS	4 WEEKS	SUCCESS %
281 MB	374 MB	0%
140 MB	187 MB	50%

IBLT

3 WEEKS	4 WEEKS	SUCCESS %
(140 + N) MB	(187 + N) MB	50%
(28 + N) MB	(37 + N) MB	90%

where N = sizeof($IBLT_{peer}$)

Fig 4.1a

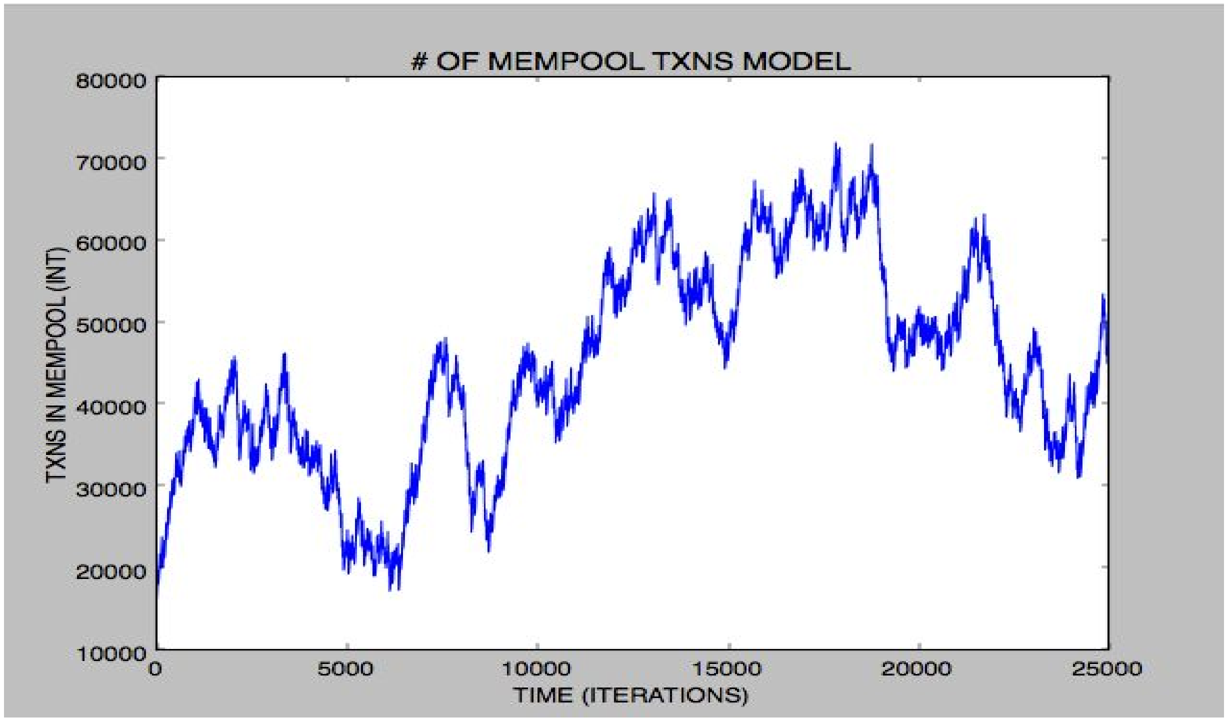
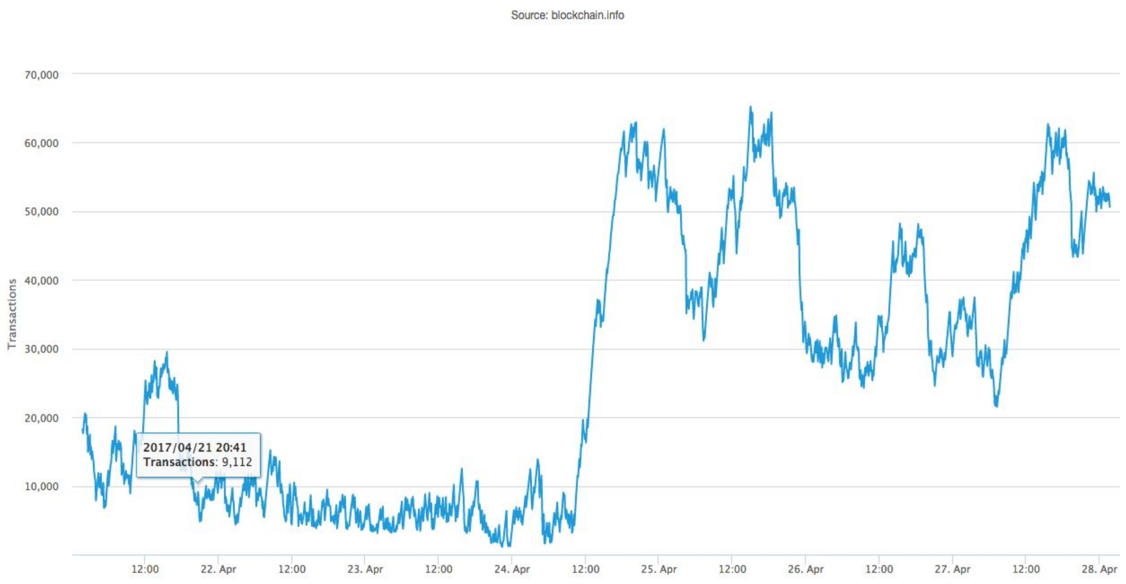


Fig 4.1b



Source: https://blockchain.info/charts/mempool-count