Churn in the Bitcoin Network

Muhammad Anas Imtiaz, David Starobinski, Ari Trachtenberg, and Nabeel Younis

***Abstract*—Efficient and reliable propagation of blocks is vital to the scalability of the Bitcoin network. As a result, several schemes, such as the compact block protocol (BIP 152), have been proposed over the last few years to speed up the block propagation. Even so, we provide experimental evidence that (i) the vast majority (97%) of Bitcoin nodes exhibit only intermittent network connectivity (*i.e.,* churn), and (ii) this churn results in significant number of unsuccessful compact blocks, roughly three times the statistic for continuously connected nodes. We conduct experiments on the Bitcoin network that show that churn results in a roughly five fold increase in block propagation time (*i.e.,* 566.89 ms vs. 109.31 ms) on average. To effect our analysis, we develop a statistical model for churn, based on empirical network data, and use this model to actuate live test nodes on the Bitcoin network. The performance of the system is measured within a novel framework that we developed for logging the internal behavior of a Bitcoin node, and which we share for public use. Finally, to mitigate the problem of missing transactions in churning nodes, we propose and implement into Bitcoin Core a new synchronization protocol, dubbed MempoolSync. Our measurements show that churning nodes implementing MempoolSync experience significantly better performance than standard nodes not implementing MempoolSync, including average block propagation delay reduced by over 50%.**

***Index Terms*—Bitcoin, blockchain, churn, propagation delay, distribution fitting, mempool, synchronization.**

I. Introduction

THE Bitcoin cryptocurrency, originally introduced by Satoshi Nakamoto in 2008 [2] as a peer-to-peer electronic payment system, is currently used for buying and selling a wide variety of goods in different markets across the globe. Together with hundreds of Bitcoin-like derivatives, cryptocurrencies sport a total market capitalization of roughly 200 billion dollars [3].

Today, Bitcoin's public ledger system, also known as the *blockchain*, records all transactions that take place in the Bitcoin network [4]. Each new transaction is broadcast over the network, and thereafter recorded by every node in its local memory pool (known as a *mempool*) for subsequent consensus-based validation. By design, a new block containing transactions is created (by a *mining* node) and propagated to the network's nodes roughly once every ten minutes. [5]

An earlier and shorter version of this paper appeared in the proceedings of the IEEE ICBC 2019 conference [1]. This research was supported in part by NSF under grant CCF-1563753.

The authors are with the ECE Department, Boston University, Boston, MA, 02215, USA. (email: {maimtiaz, staro, trachten, nyounis}@bu.edu)

A key challenge for the Bitcoin network lies in reducing the propagation time of blocks. One of the significant consequences of slower block propagation times is an increase of *forks*, wherein several blocks are mined independently and in parallel on different nodes. This issue leads to periods of ambiguity, during which different nodes in the network have different views of the blockchain, and results in *orphaning* of those blocks that do not end up in the consensus chain. Orphan blocks involve a waste of computational resources for nodes that have mined them and their descendants. Further, an adversary may leverage fork ambiguities to launch attacks (*e.g.,* double-spending [4]).

The *compact block relay* protocol [6] (described in greater detail in Section II-A) was proposed to help address this orphan challenge, and it is currently implemented on the standard Bitcoin Core reference implementation. This protocol aims to decrease block propagation time to the broader network by reducing the amount of data propagated between nodes. However, as with other peer-to-peer networks, it is also important that the Bitcoin network be able to support a high rate of *churn* [7], that being the rate at which nodes independently enter and leave the network. In fact, Satoshi's white paper envisions that Bitcoin nodes "can leave and rejoin the network at will". [2] In other words, the network should be able to quickly propagate blocks to all current nodes, even when some of these nodes frequently enter and leave the network.

Churn in different peer-to-peer networks has been widely studied, characterized and modeled [7], [8], [9], [10], [11], [12], [13]. In the context of the Bitcoin network, previous works do consider churn in their models, analysis, and simulation [14], [15], [16], [17], [18], [19], [20], but they do not measure or evaluate its impact on the live network (cf. the discussion in Section II-B). As a result, questions remain about the extent of churn in the Bitcoin network and its effect on block propagation.

A. Contributions

Our main contributions in this work are three-fold:

1) Systematically characterizing churn in the Bitcoin network;
2) Experimentally evaluate the compact block protocol under realistic churn; and,
3) Propose, implement, and evaluate a synchronization protocol as a proof-of-concept to alleviate observed issues and to highlight the benefits of synchronizing mempools of churning nodes with highly-connected nodes.

We next elaborate on each of these contributions.

Our first key contribution in this work systematically characterizes churn in the Bitcoin network. Our characterization is based on measurements of the duration of time that nodes in the Bitcoin network are continuously reachable (*i.e., up session lengths*) and continuously unreachable (*i.e., down session lengths*). Our data show that out of more than 40,000 unique nodes on the network, over 97% leave and rejoin the network multiple times over a time span of about two months. In fact, the average churn rate in the Bitcoin network exceeds 4 churns per node per day. Our statistical analysis in Section III-C identifies the *log-logistic distribution* and the *Weibull distribution* as the best fits for up session lengths and down session lengths, respectively, among several popular distributions. Next, we analyze churn at the level of IPv4 subnetworks (subnets). Over the measurement period, we find that IP addresses associated with full nodes in the Bitcoin network belong to about 29,000 unique IPv4 /24 subnets. Our analysis further shows that for over 99% of these subnets, fewer than 10 unique IP addresses from each subnet appear on the Bitcoin network over the span of two months. Lastly, we analyze churning behavior of the 10 subnets with the largest numbers of nodes. While churning behavior of nodes within subnets may be correlated, churning behavior of nodes belonging to different subnets appears largely uncorrelated.

Our second key contribution involves an experimental evaluation of the behavior of the compact block protocol under realistic node churning behavior, leveraging our statistical characterization of churn to generate samples from the above distributions. We use our samples to emulate churn on nodes under our control in the *live Bitcoin network* (*i.e.,* on the Bitcoin `mainnet`), taking these nodes off the network and bringing them back on according to the sampled session lengths over a two week period. Our analysis, compared against a control group of nodes that are continuously connected to the network, shows that the performance of the compact block protocol significantly degrades in the presence of churn. Specifically, the churning nodes see a significantly larger fraction of incomplete blocks as the control nodes (an average of 33.12% vs. 7.15% unsuccessful compact blocks). This is due to an absence of about 78 transactions on average for the churning nodes, versus less than 1 transaction for the control nodes. The end result is that, on average, churning nodes require over five times as much time to propagate a block than their continuously connected counterparts (*i.e.,* 566.89 ms vs. 109.31 ms). The largest propagation delay experienced by blocks received by churning nodes is more than twice the largest propagation delay experienced by any block received by the control nodes(*i.e.,* 105.54 s vs. 46.14 s). These results confirm that churn can have a significant impact on block propagation in Bitcoin. Note that throughout this document, we refer to the single-hop block propagation delay, *i.e.,* the time it takes to completely recover and reconstruct a block once a node receives an announcement of the block from a peer, as *block propagation delay* or *propagation delay.*

Our third key contribution is to propose and to implement into the Bitcoin Core a synchronization protocol, dubbed *`MempoolSync`*, that alleviates aforementioned issues with churning nodes. `MempoolSync` is a protocol in which a node periodically transmits top-ranked transactions of its mempool to its peers. The goal of the protocol is to provide churning nodes with those transactions that they may have missed during their down times and which are likely to be included in future blocks. Our experimental results indicate that churning nodes that accept `MempoolSync` messages are able to successfully reconstruct, on average, a larger fraction of compact blocks than churning nodes that do not accept such messages (*i.e.,* 83.19% vs. 66.88%). As a result, churning nodes that accept `MempoolSync` messages have significantly smaller block propagation delays on average (*i.e.,* 249.06 ms vs. 566.89 ms). These results show that a scheme that synchronizes mempools of churning nodes with mempools of other highly connected nodes in the Bitcoin network can overcome performance degradation issues.

B. Roadmap

The rest of this paper is organized as follows. In Section II, we cover background and related work. In Section III, we describe our methodology for obtaining and processing data on churn in the Bitcoin network, and conduct a statistical analysis of the data. In Section IV, we detail the experimental setup for evaluating the impact of churn on block propagation, and present the results. In Section V, we introduce the `MempoolSync` protocol and report experimental results on the efficiency of this synchronization protocol. We present a discussion on and limitations of our work in Section VI. Section VII concludes the paper and discusses potential areas for future work.

II. Background and Related Work

In this section, we provide relevant background material on the Bitcoin network followed by a discussion of related work.

A. Bitcoin

Bitcoin's primary record-keeping mechanism is the *block*. It is a data structure that contains metadata about the block's position in the blockchain together with a number of associated transactions (typically a couple thousands [21]). A block is generated roughly every ten minutes through the *mining* process, and, once generated, the block and its transactions become a part of the Bitcoin blockchain. It is possible that different nodes will incorporate different blocks in their version of the blockchain (a process known as a *fork*). These differences are reconciled over time in a competitive process.

In the interim time between when a transaction is announced and when it is included in a block, transactions are stored locally in the *mempool*. The mempool is a constantly changing data set that stores all the unconfirmed transactions waiting to be included in future blocks. It typically contains anywhere between 10^4 to 10^5 transactions, depending on network activity. Currently the mempool experiences as low as 1 and as high as 26 insertions per second [22]; the arrival of a new block also instigates many deletions from the mempool, between 2,000 to 2,700 transactions [23] per block.

Block Propagation is the process of communicating a newly mined block to the network, and it is the backbone of Bitcoin's ability to maintain consensus on the current balances of addresses (wallets). When a new block is discovered, each Bitcoin node advertises the block to all of its neighboring peers. There are currently two main block protocols in Bitcoin: the original protocol developed for the first implementation of Bitcoin and the *Compact Block Relay* Protocol (BIP 152 [6]). The original protocol is adequate for block propagation but may require significant network resources, typically close to 1 MB per block.[24] Since neighboring peers in the Bitcoin networks can be geographically distant, this approach is susceptible to large delays.[25]

The *compact block relay* was developed in an effort to reduce the total bandwidth required for block propagation. As the name implies, a compact block is able to communicate all the necessary data for a node to reconstruct and validate one standard block. The compact block contains the same metadata as the normal block, but it includes a hash, rather than a full copy, of each transaction. Depending on the number of inputs and outputs, a transaction may consist of between 500 and 800 bytes [26], whereas the hashes used for the compact block are only 6 bytes per transaction [6]. This resulting significant bandwidth saving relies on the assumption that the receiver already has the relevant transactions in its mempool and just needs to know to which blocks they belong. This trade-off makes a compact block much smaller in size than the original block at the cost of potentially resulting in extra round-trip communications for transactions whose hashes the receiver does not recognize (using the `getblocktxn/blocktxn` messages [27]).

The compact block relay has two modes of operation: *low bandwidth relaying* and *high bandwidth relaying* [6]. In the former mode of operation, a node announces a block to its peers only after fully validating the block itself. In the latter mode of operation, however, a node announces a block immediately to its peers upon receiving it without any validation. Regardless of the mode of operation chosen by two peers to exchange information in, if a receiver's mempool contains all the transactions whose hashes are contained in a compact block that it received, then it will be able to successfully reconstruct the original block. However, if not all transactions are already in the node's mempool then it will fail to reconstruct the block. When the compact block protocol fails, the extra round trips slow down block propagation and increases the risks of a fork in the blockchain. Fig. 1 illustrates the process.

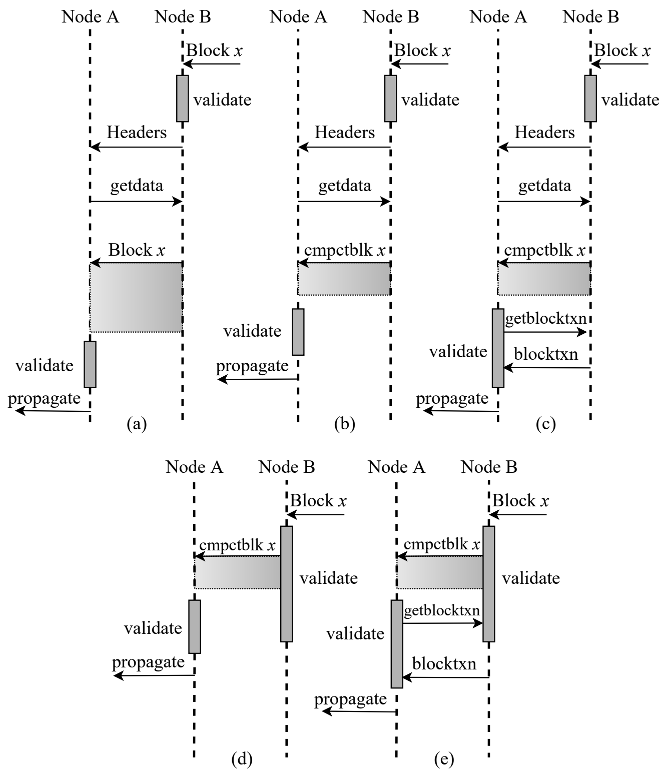


Fig. 1: Block propagation: (a) an *Original Satoshi block*, (b) a *Successful compact block* (low bandwidth relay mode), (c) an *Unsuccessful compact block* (low bandwidth relay mode), (d) a *Successful compact block* (high bandwidth relay mode), and (e) an *Unsuccessful compact block* (high bandwidth relay mode). In the cases (c) and (e), additional communications recover missing transactions from peers.

B. Related Work

Stutzbach and Rejaie [7] study churn in several peer-to-peer networks, specifically Gnutella, BitTorrent, and Kad. By inserting crawlers into each network, they characterize various metrics, such as peer inter-arrival time, session lengths, peer up time, peer down time etc., and fit distributions to the respective metrics. The authors state that "one of the most basic properties of churn is the session length distribution, which captures how long peers remain in the system each time they appear". Our work characterizes the statistics of session lengths and churn in the Bitcoin network, which to our knowledge have not been studied so far. Furthermore, our work is not limited to statistical characterization of churn, but also evaluates the impact of churn on the behavior of the Bitcoin network with respect to the efficacy of the compact block protocol.

Apostolaki et al. [15] simulate partitioning attacks on the Bitcoin network. In a partitioning attack, an attacker divides the network into multiple disjoint components, where no information flows between any two components. The authors incorporate churn in their simulations and assume that session lengths follow exponential distributions. We show in our work that, on an aggregate level, session lengths are better modeled by heavy-tailed distributions.

Decker and Wattenhofer [4] measure the time it takes for a block to propagate in the Bitcoin network. They show that the delay in propagation of blocks in the network results in forks in the Bitcoin blockchain. Since only one branch of the fork becomes part of the blockchain, nodes that create blocks in the other branch(es) essentially waste their power. Forks in the blockchain also lead to a phenomenon called *information eclipsing* which an adversary can leverage to perform a *double spending* attack. However, that work was published before the compact block protocol was implemented, *i.e.,* each block contained full transactions and no reconstruction was needed. Therefore, it does not capture the current behavior of block propagation, including additional delay incurred due to missing transactions in a compact block received by a node. We show in this work that churn can increase propagation delays of compact blocks received by a node in the Bitcoin network.

Neudecker et al. [14] study churn in the Bitcoin network from an attacker's perspective. They vary the session length of an attacking node in the Bitcoin network and, through simulations, show that a network partitioning "attack is sensitive to churn of the attacking node." However, they do not characterize churn in the network, and thus, it is unclear what is the basis for the parameters used in the simulations.

Karame et al. [16] study the security of using Bitcoin in fast payments, such as paying for a meal at a fast-food restaurant. They theorize that because of churn in the Bitcoin network, the connectivity of a victim node with the rest of the network varies with time. This gives an adversarial node considerable opportunities to connect with a victim node and perform a double spend attack. However, the authors neither characterize churn nor take it into account when performing analysis, measurements and experiments.

Augustine et al. [28] and Jacobs et al. [29] propose algorithms for efficient search and retrieval of data items in churn-tolerant peer-to-peer networks. These algorithms can help churning nodes retrieve transactions that they missed while being disconnected from the network. The algorithms assume that a churning node knows *a-priori* the ID of a peer that has the required data. Such an assumption does not hold in Bitcoin because Bitcoin is an unstructured peer-to-peer network [2]. Specifically, a node in Bitcoin does not know in advance which peer stores the data that it needs, and thus it broadcasts data requests to multiple peers [30].

Mišić et al. [31] study the improvements brought upon by the compact block relay protocol on the Bitcoin network. The queuing analysis presented by the authors shows that while the compact block relay protocol improves delivery times of blocks by up to 20% and reduces the probability of forks occurring in the network by up to 25%, it requires high transaction traffic to successfully recover transactions from their hashes in the compact blocks. However, the authors do not account for the presence of churn in the Bitcoin network when evaluating the performance of the compact block relay protocol.

Motlagh et al. [17], [18], [19], [20] present an analytical model for the churning process in the Bitcoin network using continuous time Markov chain. The authors point out that churning nodes in the Bitcoin network not only affect the propagation of blocks in the network, but also consume network resources to synchronize their local copy of the blockchain with the rest of the network upon rejoining it. While these works complements our findings in this paper, the authors present results from simulations based on an assumption that all churning nodes have the same session lengths. We present results based on data from the live Bitcoin network where session lengths of churning nodes are sampled from the actual distribution of up and down time of nodes in the network.

Ozisik et al. [32] propose the *Graphene* protocol, which couples an Invertible Bloom Lookup Table (IBLT) [33] with a Bloom filter in order to send transactions in a package smaller than a compact block. According to the authors, the size of a Graphene block can be a fifth of the size of a compact block, and they provide simulations demonstrating their system. This block propagation concept has recently been merged into the Bitcoin Unlimited blockchain. However, similar to the compact block protocol, Graphene assumes a large degree of synchronization between mempools of sending and receiving peers. In case of missing transactions, the receiving peer requests larger IBLTs from the sending peer, thus potentially adding significant propagation delay.

Mišić et al. [34] show that synchronizing mempools of churning nodes when they rejoin the network not only decreases the chances of missing transactions from compact blocks, but also reduces unnecessary network traffic when retrieving the aforementioned missing transactions from peers. While this work is complementary to our findings in this paper, the authors present simulation-based findings whereas we report results from live Bitcoin nodes with an implementation of a synchronization protocol in the Bitcoin software.

Naumenko et al. [35] propose the *Erlay* transaction dissemination protocol, the aim of which is to reduce the consumption of bandwidth due to dissemination of transactions across nodes in the Bitcoin network. The protocol uses *set sketches* to perform set reconciliation across mempools of nodes. The authors do not evaluate the protocol in the presence of churn, and it is unclear whether Erlay would perform efficiently when a node misses a large number of transactions from a block that it receives. We show in this work that a block received by a churning node can miss as many as 2,722 transactions.

A preliminary version of this work was presented at the *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* [1]. The main differences between the aforementioned prior work and this work are as follows:

1) We introduce an entirely new section, Section V, where we propose, implement and evaluate `MempoolSync`, a protocol that keeps the mempools

```
"220.75.229.130:3927": [
    70015,                  Protocol Version
    "/Satoshi:0.13.2/",     User Agent
    1526337217,             Connected Since
    13,                     Services
    165277,                 Height
    "220.75.229.130",       Hostname
    "Seoul",                City
    "KR",                   Country Code
    37.5985,                Latitude
    126.9783,               Longitude
    "Asia/Seoul",           Timezone
    "AS4766",               ASN
    "Korea Telecom"         Organization Name
]
```

Listing 1: Part of a JSON file transmitted to a Bitcoin node.

of churning nodes synchronized with highly connected nodes.

2) The current paper includes, a new fined-grain analysis of churn at the level of IPv4 /24 subnetworks (subnets) (see Section III-D).
3) The experiments of Section IV have been repeated with larger measurement datasets and reflect the current state of the Bitcoin network, as of 2020. The results that we obtained from the new round of experiments show a similar trend as those in the original paper, *i.e.,* churning nodes still perform worse than non-churning nodes.

III. Churn Characterization

Nodes on the Bitcoin network may leave and rejoin the network independently. As a result, characterizing *churn* requires observation of the node activity on the network. In this section, we first detail our methodology to obtain and process data on churn, and then we present our statistical analyses. In Section IV, we leverage this characterization to run experiments on the compact block protocol with churning nodes.

A. Obtaining and processing data

The site Bitnodes [36] continuously crawls the Bitcoin network and provides a list of all up nodes with an approximate 5 minute resolution. Each network snapshot is available for roughly 60 days[37], and the website provides a rich API interface that can be used to download each snapshot as a JavaScript Object Notation (JSON) file containing the IP address, version of Bitcoin running, geographic location etc., of the nodes on the network that are up. Listing 1 shows an example of a JSON snapshot taken by the crawler at the UNIX timestamp 1526742217.

Our analysis was based on all available JSON files from Saturday, May 19, 2018 11:03:37 AM EST (UNIX timestamp: 1526742217) to Tuesday, July 17, 2018 04:06:14 PM EST (UNIX timestamp: 1531857974) including a total of 14,674 snapshots ordered according to unique UNIX timestamps.

We parsed each JSON file and generated a dataset of all IP addresses that appear at any point on the Bitcoin network during the aforementioned time period, totalling 47,702 distinct IP addresses. We find that out of these IP addresses, 46,520 ($> 97.5\%$) announce that they have the `NODE_NETWORK` [27], [38] service enabled, *i.e.,* they are able to provide a copy of the full blockchain. Out of the about 2% remaining nodes, a large fraction ($> 60\%$) is running in pruned mode, *i.e.,* they are able to provide at least the last 288 blocks. Therefore, it is evident that almost all nodes in our data set are comprised of full nodes that partake in dissemination of information in the Bitcoin network.

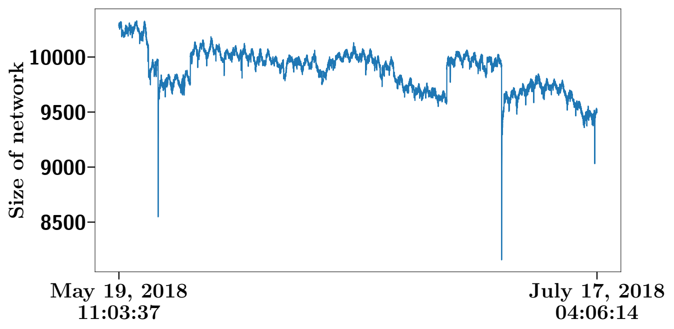


Fig. 2: Size of Bitcoin network over the measurement period.

Given the list of IP addresses, we then ran a script that looks for each IP address through each consecutive network snapshot. If an IP address was found in two consecutive network snapshots, we concluded that the IP address was *up*, and thus online, for 10 minutes (recall Bitnodes' 5 minute resolution). Similarly, if the IP address was found in only one of the two consecutive network snapshots, we inferred that the node either left or rejoined the network. Finally, if IP addresses that were not found in any of the two consecutive network snapshots were designated as *down* (*i.e.,* offline) for 10 minutes. This allowed us to record the networked behavior of a node, *i.e.,* the duration of time it is on and off the Bitcoin network over the 14,674 snapshots.

B. Churn Rate

Out of 47,702 distinct IP addresses observed on the network during the aforementioned time period, only 1,154 (*i.e.,* 2.42% of the nodes) were online at all times. Nodes corresponding to the remaining IP addresses contributed to churn in the Bitcoin network.

Prior work [7], [39] showed that the overall size of peer-to-peer networks (*i.e.,* the total number of peers) stays relatively stable over time. Indeed, Fig. 2 depicts the number of reachable nodes in the Bitcoin network extracted from successive snapshots, where, on average, there are 9,881 reachable nodes with a standard deviation of 186. The low deviation, in addition to visual inspection of Fig. 2, suggests that the size of the Bitcoin network is indeed stable over the measurement period.

Next, we evaluated the *churn rate*, namely the rate at which nodes oscillate between up and down sessions. More precisely, the churn rate can be defined as $R = 1/T$, where

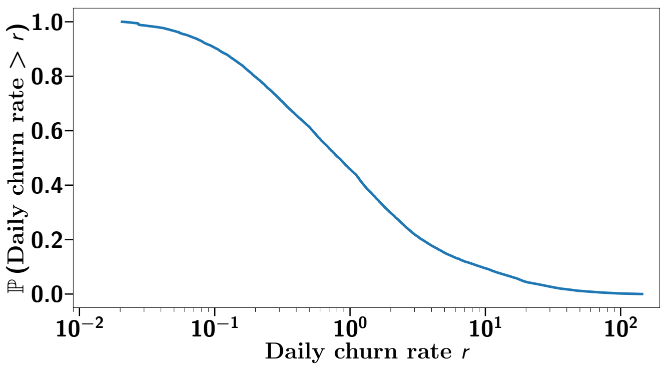


Fig. 3: Daily churn rate on the Bitcoin network.

T is a random variable corresponding to the sum of the duration of an up session and its subsequent down session. Fig. 3 shows the CCDF of the churn rate R as measured across all the observed Bitcoin nodes. Note that R exceeds one churn per node per day, with probability greater than 45%, and that there is a 10% probability of $R \geq 9$ churns per node per day. The average churn rate per node is $\bar{R} = 4.16$ per day.

We point out that the nodes that are always up do not contribute to churn in the Bitcoin network. For this reason, we filtered out data related to these nodes from our data sets on the session lengths of a node's up and down time on the network. In addition, we filtered out the first and the last session (whether up or down) of each node, because we did not know how long a node was up or down before we started and after we finished our measurement.

C. Statistical fitting of session lengths

Prior work [40], [41], [42], [43] showed that session lengths in various peer-to-peer protocols exhibit a behavior similar to a heavy-tailed distribution. Therefore, in our statistical fitting, we focus on fitting heavy-tailed distributions to the data, specifically: the *generalized Pareto distribution*, *the log-normal distribution*, the *Weibull distribution* [44] and the *log-logistic distribution.* Nolan [45] shows that maximum likelihood estimation (MLE) of heavy-tailed distribution parameters is feasible. Hence, we use MATLAB's distribution fitting capabilities [46] to fit distributions based on the MLE criterion. Finally, we also consider the exponential distribution, as a basis for comparison.

1) Up sessions: We first fit a distribution to the data representing up session lengths. Our fitting applies to the first 25,000 minutes, which roughly translates to 2.5 weeks. We used the following criteria to determine the goodness-of-fit of the various distributions to the actual data:

1) The R-squared value given by

$$\mathrm{R}^2 = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})},$$

where y is the actual value, $\hat{y}$ is the calculated value, and $\bar{y}$ is the mean of y [47]. An $\mathrm{R}^2 = 1$ suggests a perfect model [48].

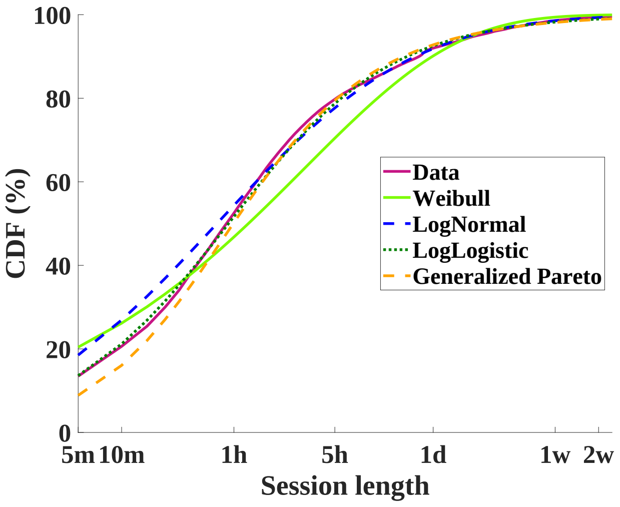


Fig. 4: Distribution fits for "up session" lengths.

Distribution	R^2	RMSE
Weibull	0.9002	2.60e−03
Log-normal	0.9939	1.29e−06
Log-logistic	0.9907	1.80e−03
Generalized Pareto	0.9856	2.20e−03
Exponential	0.4904	21.70e−03

TABLE I: R^2 and RMSE scores of distribution fits for "up session" lengths.

2) The root mean squared error (RMSE) given by

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (\hat{y}_i - y_i)^2},$$

where y is the actual value and $\hat{y}$ is the calculated value. [47] An RMSE = 0 indicates that all of the calculated values lie on the line formed by the actual values [49].

3) Visual inspection of the data.

We set the parameter values generated by MATLAB as a base and performed an exhaustive search within ±10% of the base parameters. The final results for each distribution are the highest R^2 and lowest RMSE in that range.

The results can be seen in Fig. 4, and the R^2 and RMSE scores for the fits are detailed in TABLE I. A key observation is that the *exponential distribution* is a very poor fit for the session lengths. While the *log-normal distribution* performs the best in terms of R^2 and RMSE scores, Fig. 4 indicates that the *log-logistic distribution* fits best the CDF of the empirical data, at least in the initial portion where most of the data lies. Therefore, judging from the combination of Fig. 4 and TABLE I, we conclude that the *log-logistic distribution*, given by

$$F_{(\alpha,\beta)}(x) = \frac{1}{1 + (x/\alpha)^{-\beta}}, \tag{1}$$

where $\alpha > 0$ is the scale parameter, and $\beta > 0$ is the shape parameter, is the best fit for the up sessions. The

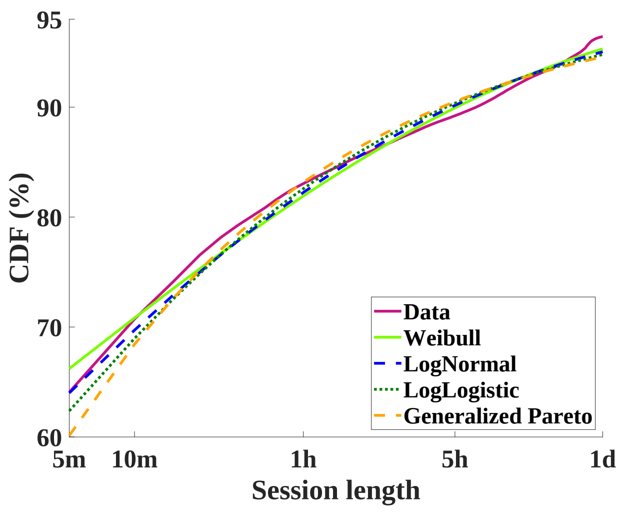


Fig. 5: Distribution fits for “down session” lengths.

Distribution	R^2	RMSE
Weibull	0.9777	3.28e−04
Log-normal	0.9694	5.36e−04
Log-logistic	0.9575	7.72e−04
Generalized Pareto	0.9429	9.55e−04
Exponential	0	1

TABLE II: R^2 and RMSE scores of distribution fits for “down session” lengths.

parameters for the *log-logistic distribution* fit in Fig. 4 are $\alpha = 11.000$ and $\beta = 0.771$.

2) Down sessions: Next, we fit distributions to the data representing down session lengths. We employed an approach similar to that in the previous section. We focused on performing a statistical fitting for sessions that are down for up to one day (representing over 93% of the cases). Note that the mempool of a node that is continuously off the network for a duration exceeding one day will largely be out of synchronization with the rest of the network.

The fitting results are shown in Fig. 5. The corresponding R^2 and RMSE scores are listed in TABLE II. Notice that the *exponential distribution* is a very poor fit and is never able to achieve an R^2 value above 0. Observing the combination of Fig. 5 and TABLE II, we conclude that in this case the *Weibull distribution*, given by

$$F_{(\lambda,k)}(x) = \begin{cases} 1 - e^{-(x/\lambda)^k} & x \geq 0 \\ 0 & x < 0, \end{cases} \tag{2}$$

where $\lambda > 0$ is the scale parameter, and $k > 0$ is the shape parameter, is the best fit for the down sessions. The parameters for the *Weibull distribution* fit in Fig. 5 are $\lambda = 0.640$ and $k = 0.183$.

TABLE I and TABLE II show that the exponential distribution is not a suitable fit for either the up session lengths or the down session lengths. This suggests that on an aggregate level, Markov process may not be suitable for performing analysis on churn in the Bitcoin network. Instead, we believe that churn should be analyzed using alternating renewal processes with heavy-tailed session lengths [50].

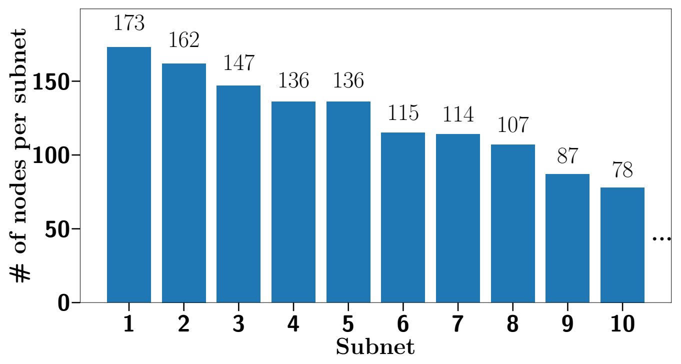


Fig. 6: Largest IPv4 /24 subnets sorted in descending order.

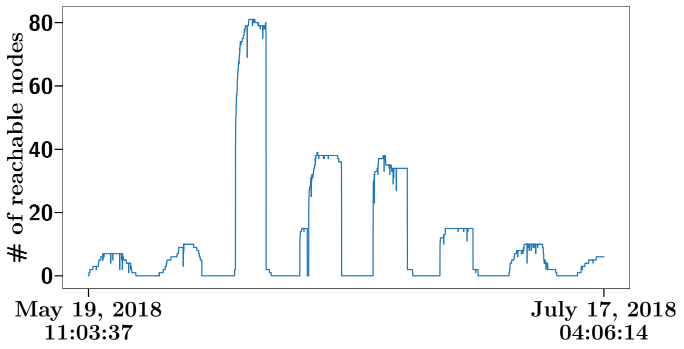


Fig. 7: Number of reachable nodes in the largest IPv4 /24 subnet in consecutive Bitcoin network snapshots.

D. Subnet Analysis

In this section, we investigate churn behavior at the level of IP subnetworks (subnets). Our dataset contains 39,574 IPv4 nodes and 7,512 IPv6 nodes; a tiny fraction of the remaining nodes use onion routing [51]. We first focus our analysis on IPv4 /24 subnets, identifying 29,036 such subnets over 99% of which contain fewer than 10 Bitcoin nodes (*i.e.,* with unique IP addresses). The average number of Bitcoin nodes per subnet is 1.36. Fig. 6 shows statistics for the 10 largest subnets, falling quickly from a maximum of 173 nodes to below 100.

Fig. 7 depicts the evolution of the number of reachable nodes over time in the largest subnet. A prominent pattern emerges: all the nodes in the subnet are periodically unreachable for roughly the same time duration. Another interesting insight is that the 173 nodes recorded in this subnet do not appear on the Bitcoin network at the same time. In fact, at most 81 unique nodes are reachable at the same time. We observe similar behavior in the next nine largest subnets.

We next study the *duty cycle*, defined as the fraction of time during which a node is reachable on the Bitcoin network. Fig. 8 shows the CDF of the duty cycle of nodes belonging to the largest subnet. The highest duty cycle of

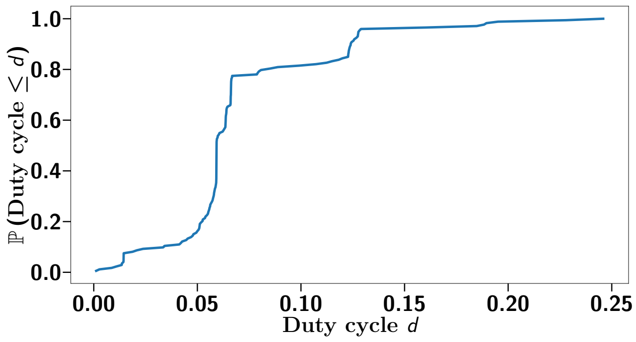


Fig. 8: CDF of duty cycle of nodes in the largest IPv4 /24 subnet. The duty cycle of a node represents the fraction of a time it is reachable during the measurement period.

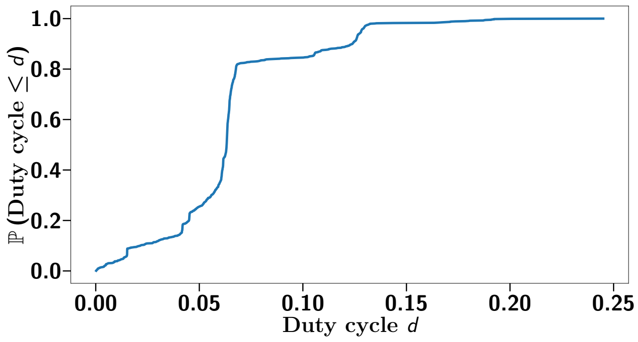


Fig. 9: CDF of duty cycle of nodes in the 2nd to 10th largest IPv4 /24 subnets.

a node in this subnet is 0.25. On average, a node in this subnet has a duty cycle of 0.07 with a standard deviation of 0.04. Fig. 9 shows the CDF of the duty cycle in the next nine largest subnets, which is very similar to that shown in Fig. 8.

The similarity between Fig. 8 and Fig. 9 raises an important question: is churn behavior in the 10 largest subnets correlated? We consider the 10 largest subnets and compute the correlation of churn for nodes within the same subnet and in different subnets. We use the Pearson correlation coefficient [52] to measure correlation between the presence of nodes on the Bitcoin network. Given two data sets D_1, D_2, the Pearson correlation coefficient, ρ, where $-1 \leq \rho \leq 1$, is given by

$$\rho_{D_1,D_2} = \frac{\text{cov}(D_1, D_2)}{\sigma(D_1)\,\sigma(D_2)},$$

where $\text{cov}(D_1, D_2)$ represents the covariance between the two data sets, D_1 and D_2, $\sigma(D_1)$ represents the standard deviation of the data set D_1, and $\sigma(D_2)$ represents the standard deviation of the data set D_2 [53]. Fig. 10 shows the results in the form of a correlation matrix. While the behavior of nodes within the same subnet may show correlation with one another, the behavior of nodes across subnets is largely uncorrelated. This finding indicates that across the 10 largest IPv4 /24 subnets nodes independently contribute to churn in the Bitcoin network.

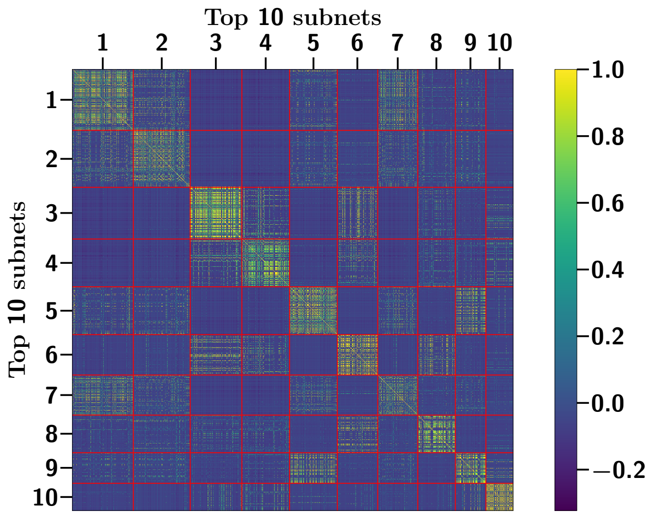


Fig. 10: Correlation matrix showing the correlation between churn behavior of nodes in the 10 largest /24 IPv4 subnets. The red line delimiters separate between different subnets.

E. Geographic Analyses

Fig. 11 shows the geographical location of the 47,702 individual nodes discovered in the Bitcoin network during the time period mentioned in Section III-A. Nodes that are always up during this time period are marked white to make them distinguishable from the remaining nodes that contribute to churn in the network. We observe that the majority of the Bitcoin nodes are located in the North America and Europe. South America, North Asia, the Far East and Oceania show a moderate presence while Africa and Central and South Asia show a very little presence of Bitcoin nodes. We note that the nodes that are always connected are not co-located in one region but rather spread out over the entire world map.

Our geographic distribution is summarized in TABLE III, which shows that the North American and European continents have the highest percentage of nodes that are always up. On the other hand, Africa has a very small percentage of nodes that are always up. These results may be related to the intermittent nature of Internet access in parts of that continent.

IV. Experimental Analysis of Compact Block Performance with Churn

In this section, we evaluate the performance of block propagation, and especially the compact block protocol, in the presence of churning nodes, to realistically reflect the behavior of the Bitcoin P2P network. The section details the mechanism that we developed to log events on the Bitcoin network, the experimental setup, the method for emulating churning nodes based on the distribution fits performed in Section III-C, and finally, the results obtained.

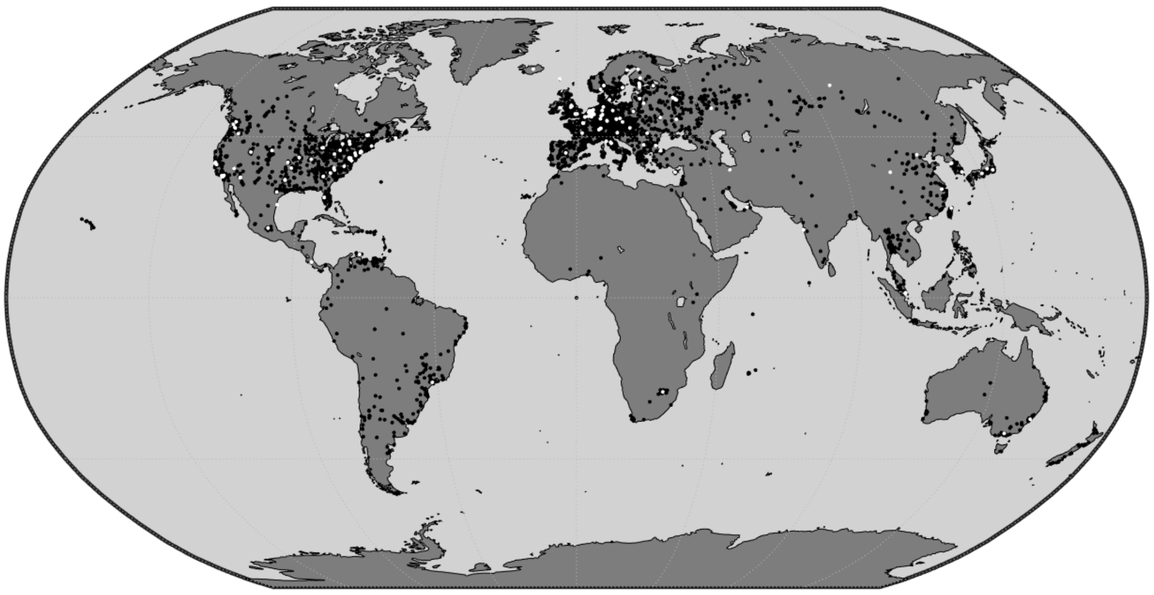


Fig. 11: Geographic location of individual nodes on the Bitcoin network. Nodes that are always up are marked white. Remaining (black) nodes contribute to churn in the network.

Continent	**Percentage**
Africa	0.051
Asia	1.138
Europe	3.239
Oceania	0.692
North America	3.414
South America	0.289

TABLE III: Percentage of continuously connected nodes in each continent.

A. Data Collection Mechanism

To aid in understanding Bitcoin Core's behavior, we have developed a new log-to-file system that produces human-friendly, easy-to-read text files. This logging system is open-source and we have made it available to the research community (`[54]/src/logFile.*`). This new logging mechanism allows one to isolate specific behaviors through select calls anywhere within the Bitcoin Core's source code, most notably information about different protocols such as the compact block. The logging system writes core data to a log file, and also can record various events and the information associated with those events. For instance, when a compact block arrives, the system logs this event and saves the transaction hashes included in the compact block in a separate file with a unique identifier tying it to a log entry. We have used this system as our primary data collection mechanism for all of our experiments.

B. Experimental setup

The aim of the experiment is to determine the efficiency of the compact block protocol in the presence of churn. We achieve this by running eight nodes in the Bitcoin network. The nodes are Dell Inspiron 3670 desktops, each equipped with an 8th Generation Intel® Core i5−8400 processor (9 mB cache, up to 4.0 GHz), 1TB HDD and 12GB RAM. The nodes are each running the Linux Ubuntu 18.04.1 LTS distribution.

We ran experiments over a period of two weeks. Four nodes (denoted by X_1, X_2, X_3, X_4) used sampled session lengths to emulate churn on the network. Specifically, we generated samples of the best fit distributions given by Eq. (1) and Eq. (2), such that the aggregate sum of the up and down sessions is at least two weeks for each node. We limited both the up and down session lengths from a minimum of 1 second to a maximum of 1 day making sure that the mean of these session lengths is within 1% of the mean of the original session lengths used to characterize churn. The remaining four nodes (denoted by C_1, C_2, C_3, C_4) acted as *control nodes* that are continuously connected to the network. Fig. 12 shows the sampled up and down session lengths for each churning node used in the experiments. It is clear from the figure that each churning node emulates up and down sessions independent from other churning nodes. In order to avoid any bias, we used the Bitcoin RPC API `setban` [55], [56] to ensure that the eight nodes are not connected to each other as peers in the Bitcoin network. This way, our nodes did not directly influence each other. Our experiment started on Wednesday, May 27, 2020 12:00:00 EST and ran without interruption for two weeks. We have made all experimental logs publicly available on GitHub [57].

C. Statistics on compact blocks

We compare the number of compact blocks that the churning nodes (denoted by X_1, X_2, X_3, X_4) and the (stable)

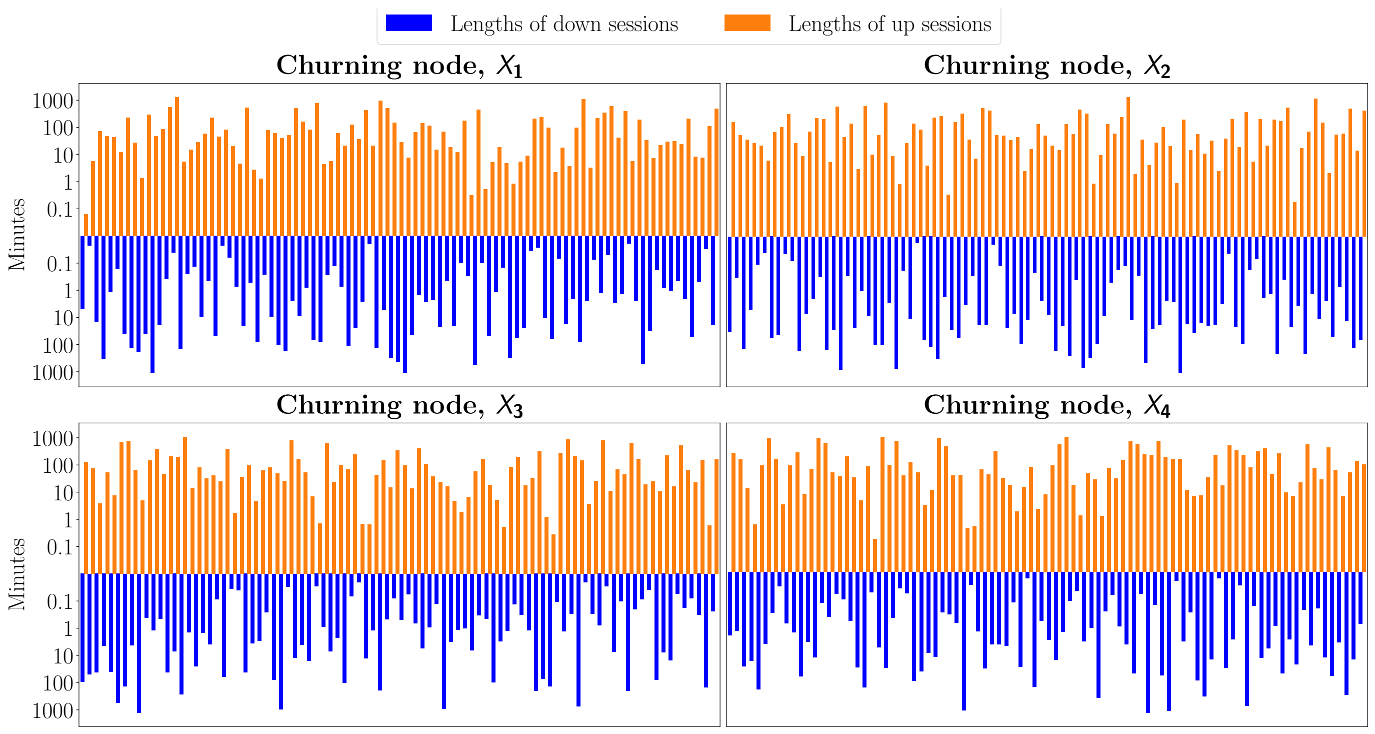


Fig. 12: Sampled up and down session lengths.

Nodes	Blocks Received	Successful Compact Blocks (%)	Unsuccessful Compact Blocks (%)
C_1	1726	93.97	6.03
C_2	1453	91.53	8.47
C_3	1751	91.55	8.45
C_4	1899	94.05	5.95
X_1	1299	66.20	33.80
X_2	1278	73.08	26.92
X_3	1279	62.16	37.84
X_4	1198	66.03	33.97

TABLE IV: Block reception statistics for control nodes C_1, C_2, C_3, and C_4, and churning nodes X_1, X_2, X_3 and X_4.

control nodes (denoted by C_1, C_2, C_3, C_4) fail to reconstruct. TABLE IV shows the results. The churning nodes are unable to reconstruct a larger fraction of compact blocks that they received as compared to the control nodes. Indeed, of the blocks they receive, the control nodes are able to reconstruct successfully on average 1,585.25 blocks out of 1,707.25 blocks (*i.e.,* 92.85% of the blocks), while the churning nodes are able to reconstruct successfully on average only 845.00 blocks out of 1,263.50 blocks (*i.e.,* 66.88% of the blocks). The results are quite consistent across both the control and churning nodes.

D. Statistics on missing transactions

Churning nodes are generally missing far more transactions in blocks they are unable to reconstruct than the control nodes. We identify transactions missing from blocks by recording the requests for missing transactions that a node makes, upon receiving a new block. This

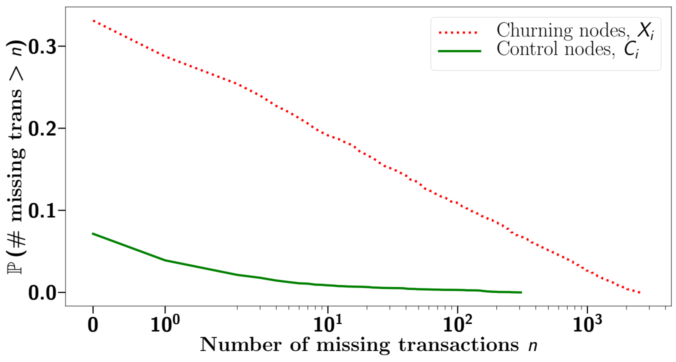


Fig. 13: CCDF of number of missing transactions in churning and control nodes.

is done using the log-to-file system described earlier (cf. Section IV-A).

We find that on average a churning node misses 78.08 transactions from a block with a standard deviation of 288.04 transactions, whereas a control node misses on average 0.87 transactions with a standard deviation of 10.78 transactions. Fig. 13 shows the CCDF of the number of missing transactions. From the figure, we observe that churning nodes may be missing up to thousands of transactions from a block they receive, while control nodes may miss at most a few hundred transactions. Roughly 11% of blocks received by churning nodes miss more than 100 transactions up to as many as 2,722 missing transactions in a block. On the other hand, only about 0.3% of blocks received by control nodes miss more than 100 transactions up to a maximum of only 307 missing transactions in a

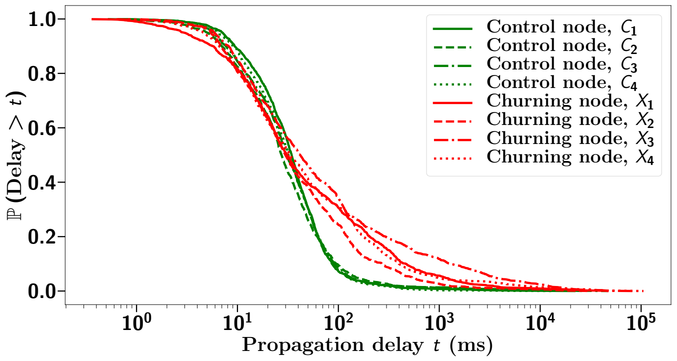


Fig. 14: Propagation delay across all blocks for both churning and control nodes.

block. Therefore, our results clearly indicate that churning nodes need to request a high number of transactions from their peers to successfully reconstruct a block.

E. Statistics on propagation delay

Next, we investigate whether and how transactions missing in a block delay the block's propagation. We measure propagation delay as the difference between the time at which a measurement node receives an announcement of a block, *i.e.,* an `inv` message with the hash of the block, from one of its peers and the time at which the node is able to successfully collect all missing transactions that are included in the block.

We compare the propagation delay of blocks received by churning nodes with the propagation delay of blocks received by control nodes. Blocks received by the control nodes experience an average propagation delay of 109.31 ms with a standard deviation of 1,066.15 ms. Blocks received by the churning nodes, on the other hand, experience an average propagation delay of 566.89 ms with a standard deviation of 3,524.78 ms.

Fig. 14 shows the CCDF of propagation delays of blocks received by all nodes. From the figure, we observe that blocks received by control nodes rarely have large propagation delays. On an aggregate level, only about 7% of blocks received by control nodes have a propagation delay larger than 100 ms with a maximum block propagation delay of 46.14 s. By comparison, on an aggregate level, roughly 30% of blocks received by churning nodes experience a propagation delay larger than 100 ms with a maximum propagation delay 105.54 s, more than twice that of any block received by control nodes.

V. MempoolSync

The experimental results from Section IV make it clear that missing transactions add significant delay to the propagation of blocks. This problem is especially acute for churning nodes since their mempools often miss transactions [1]. To address this issue, we propose, implement and evaluate a new protocol to keep the mempools of churning nodes synchronized with the rest of the network. We call this protocol `MempoolSync`. The main

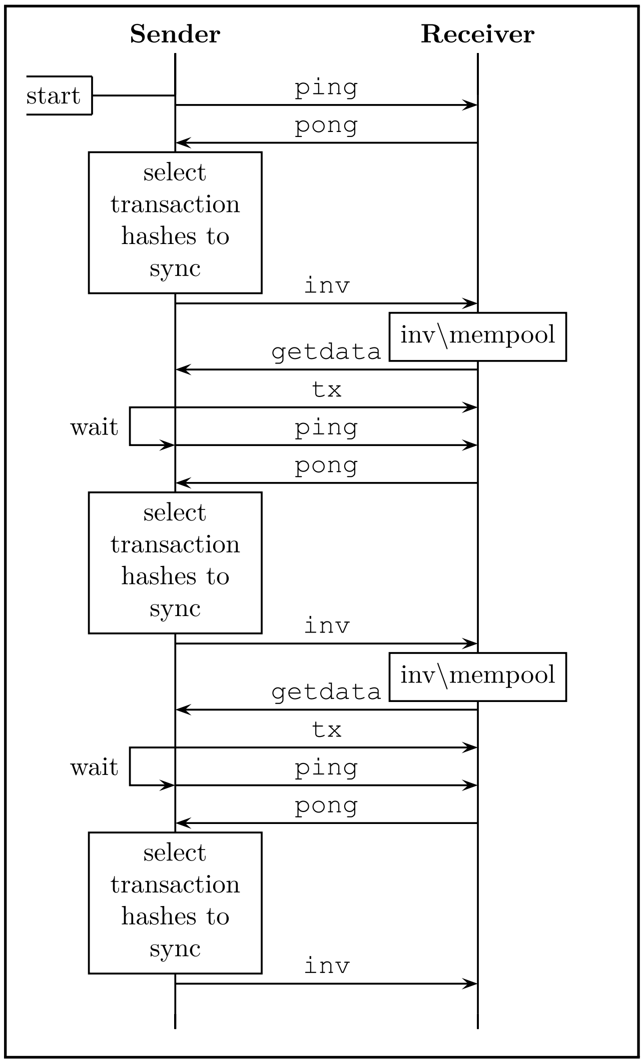


Fig. 15: Exchange of messages between the non-churning node (*sender*) and the churning node (*receiver*) in the `MempoolSync` protocol.

goal of `MempoolSync` is to reduce the number of missing transactions in mempools and, consequently, the propagation delay of blocks. Note that `MempoolSync` does not attempt to minimize communication complexity, a well-known problem in the distributed computing literature [58], [59], [33] whose implementation we leave for future work. Rather, our implementation of MempoolSync into the Bitcoin Core demonstrates the key benefits of synchronizing the mempools of churning nodes with the rest of the network.

A. Design of MempoolSync

`MempoolSync` is designed to periodically synchronize the mempool of a churning node (*receiver*) with the mempool of a non-churning node (*sender*). Fig. 15 shows an overview of the synchronization protocol. The protocol leverages Bitcoin's existing functionality to package and send inventory (`inv`) messages, as well as request and propagate transactions. The sender selects transaction hashes from its mempool and packages them in a message (*inv*). The sender then sends the message to the receiver who, upon receiving the message, computes which of the hashes in the message are not present in its mempool. The receiver then requests the respective transactions from the sender (`getdata`), who in turn responds with the requested transactions (`tx`). Note that `MempoolSync`, does not require additional steps to ensure a receiving node actually receives all transactions that it requested. Instead, the protocol relies on the default Bitcoin behavior to send transactions. The sender then waits for a configurable amount of time before repeating the process.

`MempoolSync` is a one-way synchronization protocol, *i.e.,* the sender has no prior knowledge of the state of the receiver's mempool. Hence, an important question arises here: which transaction hashes should the sender select in each synchronization round? Our solution is based on the reference implementation of the algorithm for miners [60] in the Bitcoin Core. This reference suggests that miners should prioritize transactions based on their `ancestor_score` [61]. The `ancestor_score` is an internal Bitcoin scoring mechanism that ranks a transaction according to the total unconfirmed transaction fees in its ancestor tree. The sender in the `MempoolSync` protocol mimics this prioritization and likewise selects transaction hashes based on the respective transactions' `ancestor_score`. Indeed, one can expect that these transactions are the most likely to be included in upcoming blocks.

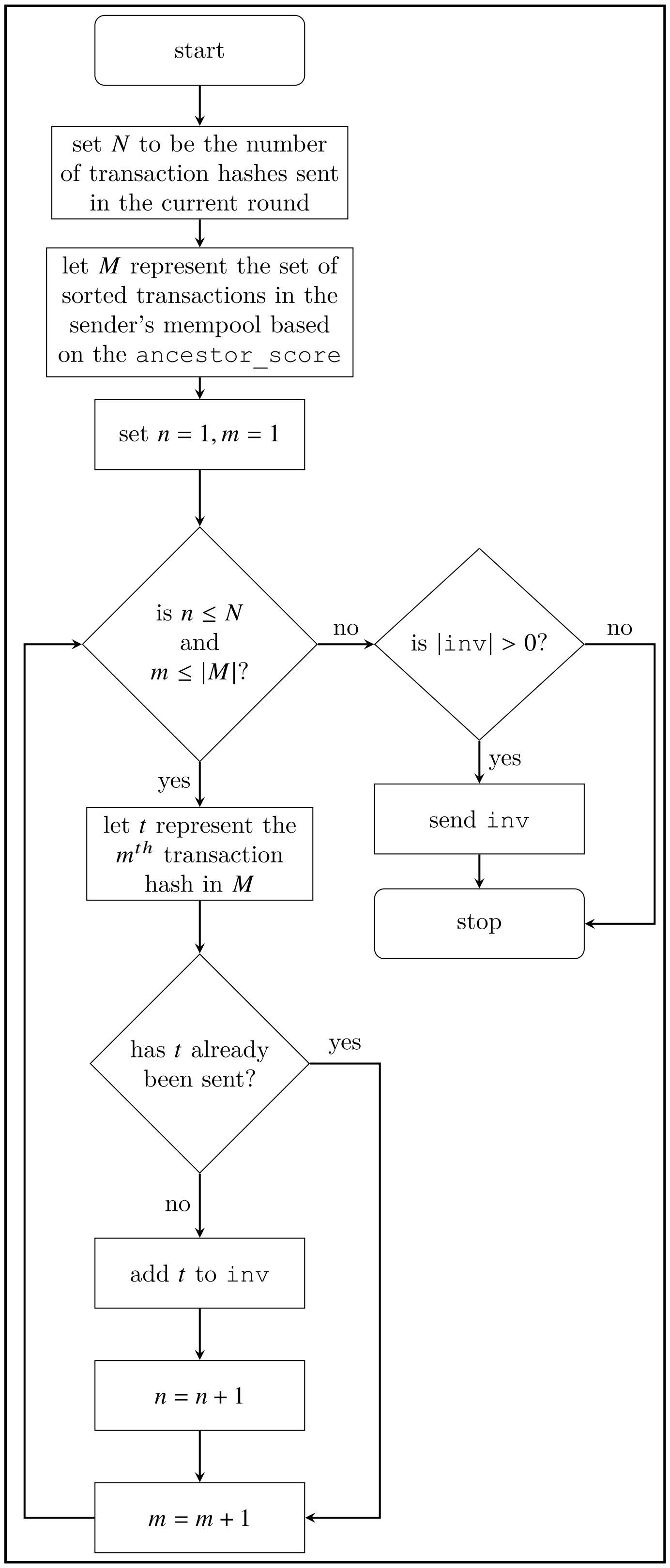
Fig. 16 shows a general overview of how transaction hashes are selected and inserted in an `inv` message in each round of `MempoolSync`. For the sake of efficiency, the protocol ensures that the sender does not send the same transaction hash more than once. Indeed, the sender keeps track of the hashes it has previously sent, and omits re-sending them again in future rounds. The sender achieves this by storing hashes of transactions already sent to a peer in a C++ `std::map` [62] data structure. Hashes that are no longer in the mempool of the sender are periodically removed from the data structure to avoid memory overhead.

We next detail how the sender smartly decides the number of transaction hashes to include in an `MempoolSync` `inv` message. Denote by `N` the number of transactions that are packaged into the `MempoolSync` `inv` message. Next, denote the number of transactions in the sender's mempool by `NumTXsMP`, and the default number of transaction hashes to be sent in a single `MempoolSync` `inv` message by `DefTXtoSync`. Let `Y` represent a fraction of the mempool size (*i.e.,* a number between between 0 and 1).

By default, `MempoolSync` `inv` message should contain `DefTXtoSync` transaction hashes. That is,

$$\texttt{N} = \texttt{DefTXtoSync}.$$

However, the sender must take care of a couple of edge cases:



start
set N to be the number of transaction hashes sent in the current round
let M represent the set of sorted transactions in the sender's mempool based on the ancestor_score
set $n = 1, m = 1$
is $n \le N$ and $m \le |M|$?
no
is |inv| > 0?
no
yes
send inv
stop
yes
let t represent the m^{th} transaction hash in M
has t already been sent?
yes
no
add t to inv
$n = n + 1$
$m = m + 1$

Fig. 16: Procedure for selecting transaction hashes to be included in the `inv` message in each round.

1) It is possible that the number of transactions in the sender's mempool far exceeds the default number of transaction hashes that the `inv` message should

contain, *i.e.,* NumTXsMP ≫ DefTXtoSync. When this happens, it makes sense to synchronize a larger fraction Y of transactions hashes from the sender's mempool. That is,

$$\mathtt{N} = \max\left(\mathtt{DefTXtoSync}, \mathtt{Y} \times \mathtt{NumTXsMP}\right).$$

2) Similarly, it is possible that the sender does not have enough transactions in its mempool, *i.e.,* NumTXsMP < DefTXtoSync. This could happen when the sender has just joined the network, or it has just received a new block which causes removal of transactions from its mempool. When this happens, the sender synchronizes its *entire* mempool with the churning node. That is,

$$\mathtt{N} = \min\left(\mathtt{DefTXtoSync}, \mathtt{NumTXsMP}\right).$$

Ignoring this edge case would cause exceptions when running the Bitcoin software if the node tries to retrieve more transactions than available in the mempool.

We next provide a simple example to illustrate how transactions are chosen to be sent in a MempoolSync message. In this example, there are ten transactions in the sender's mempool, *i.e.,* NumTXsMP = 10. The protocol has smartly chosen the number of transactions to be included in the MempoolSync message to be five, *i.e.,* N = 5. TABLE V(a) shows the hashes of transactions in the mempool of the sender along with their ancestor_score before they are sorted. TABLE V(b) shows the same hashes sorted according to their ancestor_score in a descending order. The sender now has to pick five hashes from this sorted list of hashes. However, it also has to make sure it does not re-send any hashes that have already been sent to the receiver as shown in TABLE VI. It can be seen that some of these hashes are in the top five positions in the sorted list of hashes. Therefore, while picking five transaction hashes from this list, the sender skips over any hashes that it finds in TABLE VI, resulting in a list of hashes as shown in TABLE VII. These hashes are packed into a MempoolSync message and sent to the receiver.

We have added an implementation of the MempoolSync protocol to a fork of the Bitcoin Core software as a proof-of-concept [54]. To make sure that the protocol does not interfere with, or worse, stall the main thread in the software, our implementation spawns a new thread when a Bitcoin client is started up. All operations related to the protocol strictly take place within this new thread.

Our implementation of the MempoolSync protocol relies on a connection manager maintained by each node. The connection manager, among other attributes, contains a list of addresses of peers to which the node is connected. In the MempoolSync protocol, all participating nodes are identified by their IP addresses. A node acting as sender transmits MempoolSync inv messages to peers listed in the connection manager. By default, when a peer disconnects from a Bitcoin node, the former remains in the latter's connection manager for up to 20 minutes even after it has disconnected [63]. Note that a sender always *pings* a receiver before sending to it the MempoolSync inv message containing transaction hashes. This way, the sender will not send inv messages to nodes that are down or unreachable.

TX hash	Ancestor Score
69dc6c	586
9d9816	34
ea844d	440
fa8082	495
a31fa4	592
824da7	16
4c09b6	212
d5a820	474
28d3b6	833
fa8ffc	504

(a)

TX hash	Ancestor Score
28d3b6	833
a31fa4	592
69dc6c	586
fa8ffc	504
fa8082	495
d5a820	474
ea844d	440
4c09b6	212
9d9816	34
824da7	16

(b)

TABLE V: An illustration of (a) *Unsorted* transactions in the mempool with their ancestor scores (in *satoshis)*, and (b) *Sorted* transactions in the mempool with their ancestor scores (in *satoshis).*

TX hashes		
69dc6c	d5a820	4c09b6

TABLE VI: An illustration of hashes of transactions already sent to a peer.

TX hashes				
28d3b6	a31fa4	fa8ffc	fa8082	ea844d

TABLE VII: An illustration of transaction hashes sent in MempoolSync message when $N = 5$.

B. Experimental evaluation of MempoolSync in the presence of churn

We performed an empirical evaluation of the MempoolSync protocol in the presence of churn. In this section, we describe our experimental setup and then present the results in the following sections.

We ran this experiment in parallel with the experiment in Section IV-B by adding four additional nodes (denoted by M_1, M_2, M_3, M_4) with similar hardware capabilities to the experimental setup. Nodes M_i, where $i \in \{1, 2, 3, 4\}$, emulated churn with up and down session lengths independently sampled from the distributions obtained in Section III-C for each node. Fig. 17 illustrates the sampled session lengths. In addition, these nodes were also configured to accept MempoolSync messages. Note that nodes can be configured as either senders or receivers in the MempoolSync protocol by setting the appropriate preprocessor macros to 1 as documented in the file src/logFile.h available in our GitHub repository [54].

The control nodes C_i from Section IV-B acted as the *sending* nodes in the MempoolSync protocol. Each churning node M_i was connected to a different sending node

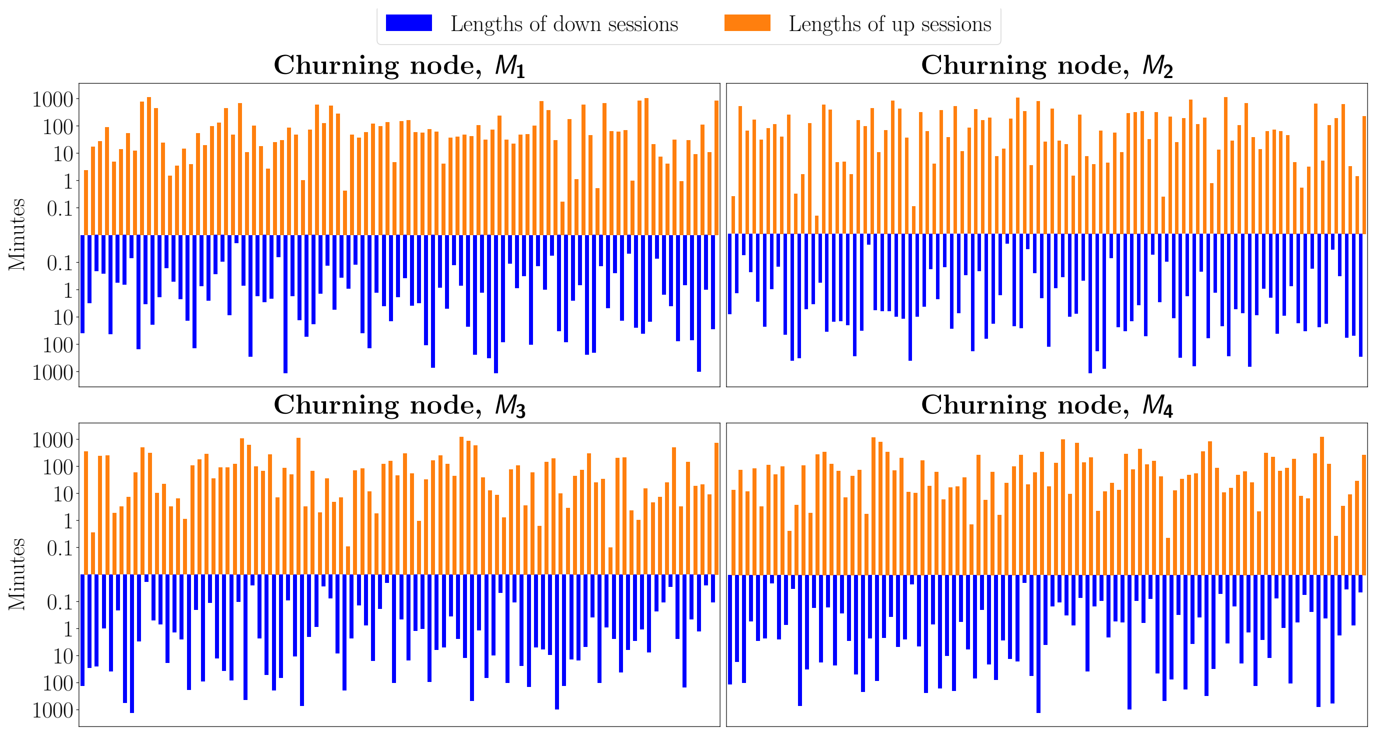


Fig. 17: Sampled up and down session lengths.

C_i. A preliminary measurement shows that a waiting time of 30 seconds in the `MempoolSync` protocol produces the best results. Therefore, we configured all sending nodes C_i to send a `MempoolSync` message after every 30 seconds. The parameter `Y` (which controls the size of the `MempoolSync` message as a fraction of the mempool size in each control node) was set to 0.1 and the parameter `DefTXtoSync` (which controls the default number of transactions sent in a single `inv` message) was set to 1,000. We found from a test measurement that some of the transactions sent as part of the `MempoolSync` protocol may end up as orphan. To make sure that `MempoolSync` does not cause unnecessary eviction of transactions already in the orphan pool, we increased the orphan pool size to 1,000 transactions. Prior work [64] shows that the chances of orphan transactions getting evicted with an orphan pool of this size are quite low.

To avoid biases, none of the nodes connected to one another as peers in the Bitcoin network (except obviously for the pairs (C_i, M_i)). The experiments ran without interruption from Wednesday, May 27, 2020 12:00:00 EST for two weeks. To avoid sending unnecessary traffic to other peers in the Bitcoin network, we made sure that each node C_i only sends `MempoolSync inv` messages to node M_i. We have made all experimental logs publicly available on GitHub [57].

Note that the statistics for the sending nodes are the same as nodes C_i in Section IV. Similarly, statistics for churning nodes that do not accept `MempoolSync` are the same as nodes X_i in Section IV. Therefore, in the following sections, we only compare the statistics between churning nodes that accept `MempoolSync` messages, *i.e.*, nodes M_i, and churning nodes that do not accept `MempoolSync` messages, *i.e.*, nodes X_i.

Nodes	**Blocks Received**	**Successful Compact Blocks (%)**	**Unsuccessful Compact Blocks (%)**
M_1	1142	80.82	19.18
M_2	1184	84.54	15.46
M_3	1247	80.91	19.09
M_4	1270	86.30	13.70
X_1	1299	66.20	33.80
X_2	1278	73.08	26.92
X_3	1279	62.16	37.84
X_4	1198	66.03	33.97

TABLE VIII: Block reception statistics for churning nodes M_1, M_2, M_3, and M_4 that accept `MempoolSync` messages, and churning nodes X_1, X_2, X_3, and X_4 that do not accept such messages.

C. Experimental Results

1) Statistics on compact blocks: TABLE VIII compares the percentage of successful compact blocks between the churning nodes M_i and X_i. The data in the table shows that churning nodes that accept `MempoolSync` messages always reconstruct a larger proportion of compact blocks that they received as compared to churning nodes that do not accept `MempoolSync` messages. The churning nodes X_i that do not implement `MempoolSync` successfully reconstruct, on average, only 66.88% of the compact blocks that they receive. By comparison, churning nodes M_i, that do implement `MempoolSync` successfully reconstruct, on average, more compact blocks *i.e.*, 83.19%. This finding

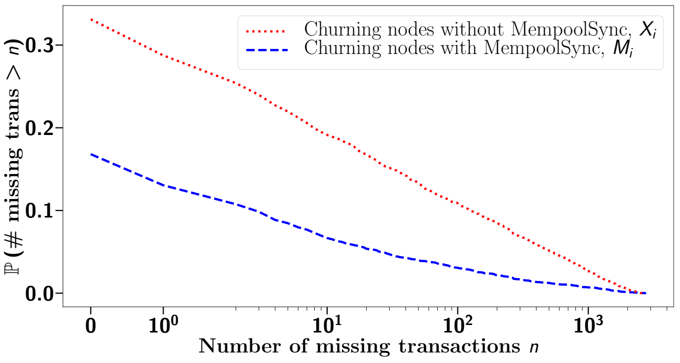


Fig. 18: CCDF of number of missing transactions across all blocks for all nodes.

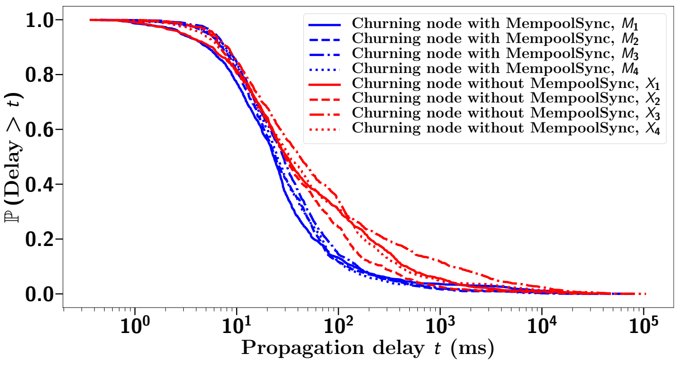


Fig. 19: CCDF of propagation delay across all blocks for all nodes.

indicates that MempoolSync leads to significant performance improvement.

2) Statistics on missing transactions: We next compare the number of transactions missing from compact blocks received by the churning nodes M_i and X_i. Fig. 18 shows the results obtained from the measurement. We find that churning nodes M_i that accept MempoolSync messages miss, on average, 21.30 transactions from blocks they received, with a standard deviation of 155.00 transactions. Churning nodes X_i that do not accept MempoolSync messages, on the other hand, miss, on average, 78.07 transactions from blocks they received, with a standard deviation of 288.04 transactions. While roughly 11% of blocks received by churning nodes X_i miss more than 100 transactions, just below only 3% of blocks received by churning nodes M_i miss more than 100 transactions. Similarly, a smaller fraction of blocks received by churning nodes M_i miss more than 1,000 transactions as compared to blocks received by churning nodes X_i. Thus, to a large degree, MempoolSync successfully synchronizes the mempools of churning nodes. This synchronization results in far fewer missing transactions in compact blocks.

3) Statistics on propagation delay: With a smaller number of transactions missing from the compact blocks, one can expect that churning nodes implementing MempoolSync will have a smaller block propagation delay than churning nodes not implementing MempoolSync. Fig. 19 confirms this intuition. Blocks received by churning nodes M_i experience, on average, a propagation delay of 249.06 ms with a standard deviation of 2,193.32 ms. On the other hand, blocks received by churning nodes X_i experience, on average, a propagation delay of 566.89 ms with a propagation delay of 3,524.78 ms.

On an aggregate level, roughly 80% of blocks received by churning nodes X_i have a propagation delay larger than blocks received by churning nodes M_i. Indeed, on an aggregate level, roughly 30% of blocks received by churning nodes X_i experience a propagation delay larger than 100 ms with a maximum block propagation delay of 105.54 s. Comparatively, only about 12% of blocks received by churning nodes M_i experience a propagation delay larger than 100 ms with a maximum propagation delay of 78.83 s.

VI. Discussions and Limitations

Characterization of churn. Our characterization of churn in the Bitcoin network relies on the data obtained from Bitnodes. To the best of our knowledge, Bitnodes does not discover nodes behind NAT or firewalls, and, therefore, the characterization is limited to behavior of nodes reachable by Bitnodes. Furthermore, it is not known what the intention of these reachable nodes is. However, our data indicates that a large majority (> 97.5%) announces access to the entire blockchain. More than 60% of the remaining nodes run in pruned-mode. Therefore, it is evident that these nodes take part in disseminating blocks through the network which can be affected when these nodes churn. Note that the Bitcoin core is only *one* implementation of the Bitcoin protocol. It is worth noting that there are other implementations of the protocol, such as btcd [65], Bitcoin Knots [66], [67], libbitcoin [68], bitcoinj [69], etc., that do not fork the Bitcoin blockchain [70] but “speak” Bitcoin and are theoretically indistinguishable from one another.

We stress that archival nodes are necessary to allow new nodes to download the blockchain when they rejoin the network. Without such nodes, an adversary could force a new node to download a false blockchain. In addition, without non-miner nodes present in the network, it would become centralized in the sense that miners would have authority over consensus of new blocks.

It may be interesting to study the effects of churn in miners (specifically, mining pools) who may implement their own internal networks for faster block dissemination. Nodes operating in these mining pools, therefore, may not be reachable by Bitnodes and are, consequently, excluded from our characterization of churn. Churn could also be modeled as a function of the number of connections that a node has. A node with higher number of connections may affect more peers when it churns. However, we note that it is not easy to measure the number of connections of a node in the Bitcoin network without knowledge of the full network topology, which is kept intentionally hidden. Moreover, in our experiments, we do not artificially modify the number of connections of nodes from the default to avoid undesirable bias without prior knowledge of number of connections of other nodes in the network. These may be interesting follow ups to work presented in this paper which we leave for future work.

Sampled session lengths. The session lengths sampled for our experiments are capped at a maximum of 1 day. However, to make sure that our results are statistically accurate, we sampled session lengths until we obtained a set of session lengths that had a mean within 1% of the mean of the original data set. We assume that nodes in the Bitcoin network exhibit a homogeneous churn behavior and follow the distributions obtained in Section III-C. Note that the session lengths are independently sampled for all nodes in our experiment, and each node's sampled session lengths are also independent from one another as illustrated in Fig. 12 and Fig. 17.

MempoolSync. We have implemented `MempoolSync` as a *proof-of-concept* to highlight the benefits of synchronizing mempools of churning nodes with highly-connected nodes in the Bitcoin network.

In our evaluation of `MempoolSync`, only one receiver was connected to a sender. We notice that with our current implementation, Bitcoin can easily handle the load of `MempoolSync` on a peer-to-peer basis. However, it is evident that in case of many churning nodes, one would need to implement a load balancing mechanism to avoid overload and network overhead at a single sender. While a majority of nodes in the network churn, we find that a large fraction of nodes do not churn as often as other nodes as shown in Fig. 3. Nonetheless, there still remains a question of how churning nodes can identify highly-connected nodes in the network in a trustless and decentralized manner.

We specifically did not connect the churning nodes not configured with `MempoolSync` to control nodes as opposed to connecting churning nodes configured with `MempoolSync` to control nodes in our experiments. This is because we wanted to obtain data for regular churning nodes without interfering with how they discover and connect to peers. Connecting churning nodes not configured with `MempoolSync` to control nodes also creates an edge between the former and churning nodes configured with `MempoolSync` connected to the same control node. This may introduce undesirable bias in our data. It is also evident that since `MempoolSync` is not a two-way synchronization protocol, it may cause unnecessary network overhead if a receiver does not churn.

Finally, note that Bitcoin is not a stationary but dynamic system and overtime statistics *will* change. Hence, it is unclear whether running experiments over longer periods would provide more statistically meaningful data. Therefore, we believe our choice of running experiments over a period of two weeks is adequate. It should be noted that results obtained from our experiments are quite consistent across different categories of nodes, *i.e.,* control nodes, churning nodes not configured with `MempoolSync`, and churning nodes configured with `MempoolSync` as shown in Sections IV and V-B.

VII. Conclusion

In this paper, we identified and empirically demonstrated the heretofore undocumented effect of *churn* on the Bitcoin network. We performed a thorough characterization of churn, including the daily churn rate and statistical fitting of the distributions of the lengths of up and down sessions. This statistical characterization should prove useful to other researchers, for the purpose of analyzing, simulating, and emulating the behavior of the Bitcoin network.

We also used the statistical characterization to evaluate the impact of churn on the propagation delay of blocks in the live Bitcoin network. In the process of this research, we developed a logging mechanism for tracing events in Bitcoin nodes, which we have released for public use [54]. Our experiments showed that churn produces a marked degradation in the performance of the delay-optimized compact block protocol. This is because unsuccessful compact blocks are much more prevalent in churning nodes, and the associated incomplete blocks often miss a large number of transactions (78.08 on average). As a result, the propagation delay of blocks processed by churning nodes is substantially larger, on average, than that of nodes that are always connected. In fact, occurrences of propagation delays that exceed one second are common. Our measurements show that more than 6% of the blocks processed by the churning nodes have a propagation delay exceeding one second, compared to less than 1% of the blocks processed by the control nodes. Note that this corresponds to the delay over a single hop on the Bitcoin network, and hence the end-to-end delay would be even larger.

We have also proposed and implemented into Bitcoin Core a proof-of-concept synchronization scheme, `MempoolSync`, that sends transactions to peers in an effort to alleviate the impact of churn and keep mempools of nodes synchronized. Our experimental results show that churning nodes that accept `MempoolSync` messages are able to successfully reconstruct, on average, a larger fraction of compact blocks that they receive as compared to churning nodes that do not accept such messages. This happens because the former miss far fewer transactions (about 3 times less on average) from the compact blocks that they receive. As a result, the churning nodes that accept `MempoolSync` messages experience block propagation delay that is, on average, slightly more than twice smaller than that of churning nodes that do not accept such messages.

As an outcome of this work, it is evident that there is significant benefit in implementing efficient synchronization of the mempools of Bitcoin nodes, thus keeping them up-to-date with transactions that they might have missed while being disconnected. We believe that it should be possible to further improve the performance of `MempoolSync` by engineering transaction synchronization based on prioritization metrics by utilizing difference-finding algorithms and reconciliation-optimized data-structures from the synchronization literature. Potential candidates include IBLT [33], due to its high tolerance for differences, and CPISync [71], [72], [73], due to its near-optimal communication complexity.

Acknowledgment

The authors would like to acknowledge Daniel Wilson for help with setting up container environment for Bitcoin experiments.

References

[1] M. A. Imtiaz, D. Starobinski, A. Trachtenberg, and N. Younis, "Churn in the bitcoin network: Characterization and impact," 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC 2019), IEEE, 2019.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 03 2009.

[3] "Cryptocurrency market capitalizations." `https://coinmarketcap.com/all/views/all/`, 2018. Online; Accessed: May 24, 2018.

[4] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pp. 1–10, IEEE, 2013.

[5] B. Wiki, "Block." `https://en.bitcoin.it/wiki/Block`, 2016. Online; Accessed: November 17, 2018.

[6] M. Corallo, "Compact block relay." `https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki`, 2016.

[7] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, pp. 189–202, ACM, 2006.

[8] D. Stutzbach and R. Rejaie, "Towards a better understanding of churn in peer-to-peer networks," *Univ. of Oregon, Tech. Rep*, 2004.

[9] O. Herrera and T. Znati, "Modeling churn in p2p networks," in *Simulation Symposium, 2007. ANSS'07. 40th Annual*, pp. 33–40, IEEE, 2007.

[10] Z. Yao, D. Leonard, X. Wang, and D. Loguinov, "Modeling heterogeneous user churn and local resilience of unstructured p2p networks," in *icnp*, pp. 32–41, IEEE, 2006.

[11] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross, "A measurement study of a large-scale p2p iptv system," *IEEE transactions on multimedia*, vol. 9, no. 8, pp. 1672–1687, 2007.

[12] D. Yang, Y.-x. Zhang, H.-k. Zhang, T.-Y. Wu, and H.-C. Chao, "Multi-factors oriented study of p2p churn," *International Journal of Communication Systems*, vol. 22, no. 9, pp. 1089–1103, 2009.

[13] F. Lin, C. Chen, and H. Zhang, "Characterizing churn in gnutella network in a new aspect," in *Young Computer Scientists, 2008. ICYCS 2008. The 9th International Conference for*, pp. 305–309, IEEE, 2008.

[14] T. Neudecker, P. Andelfinger, and H. Hartenstein, "A simulation model for analysis of attacks on the bitcoin peer-to-peer network," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pp. 1327–1332, IEEE, 2015.

[15] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *Security and Privacy (SP), 2017 IEEE Symposium on*, pp. 375–392, IEEE, 2017.

[16] G. O. Karame, E. Androulaki, and S. Capkun, "Double-spending fast payments in bitcoin," in *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 906–917, ACM, 2012.

[17] S. G. Motlagh, J. Misic, and V. B. Misic, "Modeling of churn process in bitcoin network," in *2020 International Conference on Computing, Networking and Communications (ICNC)*, pp. 686–691, IEEE, 2020.

[18] S. G. Motlagh, J. Mišić, and V. B. Mišić, "Impact of node churn in the bitcoin network," *IEEE Transactions on Network Science and Engineering*, 2020.

[19] S. G. Motlagh, J. Mišic, and V. B. Mišic, "Impact of node churn in the bitcoin network with compact blocks,"

[20] S. G. Motlagh, J. Mišić, and V. B. Mišić, "An analytical model for churn process in bitcoin network with ordinary and relay nodes," *Peer-to-Peer Networking and Applications*, pp. 1–12, 2020.

[21] "Average number of transactions per block." `https://blockchain.info/charts/n-transactions-per-block?timespan=2years`, 2017. Online; Accessed: November 17, 2018.

[22] "Transaction rate." `https://www.blockchain.com/charts/transactions-per-second?timespan=30days`, 2017. Online; Accessed: November 17, 2018.

[23] "Average number of transactions per block." `https://www.blockchain.com/en/charts/n-transactions-per-block?timespan=60days`, 2017. Online; Accessed: November 17, 2018.

[24] "Block size limit controversy." `https://en.bitcoin.it/wiki/Block_size_limit_controversy`, 2017. Online; Accessed: November 17, 2018.

[25] G. Pappalardo, T. Di Matteo, G. Caldarelli, and T. Aste, "Blockchain inefficiency in the bitcoin peers network," *arXiv preprint arXiv:1704.01414*, 2017.

[26] "Analysis of bitcoin transaction size trends." `https://tradeblock.com/blog/analysis-of-bitcoin-transaction-size-trends`, 2015. Online; Accessed: November 17, 2018.

[27] "Protocol documentation." `https://en.bitcoin.it/wiki/Protocol_documentation`, 2018. Online; Accessed: May 17, 2018.

[28] J. Augustine, G. Pandurangan, and P. Robinson, "Distributed algorithmic foundations of dynamic networks," *ACM SIGACT News*, vol. 47, no. 1, pp. 69–98, 2016.

[29] T. Jacobs and G. Pandurangan, "Stochastic analysis of a churn-tolerant structured peer-to-peer scheme," *Peer-to-Peer Networking and Applications*, vol. 6, no. 1, pp. 1–14, 2013.

[30] "How does a node get information from other nodes?." `https://bitcoin.stackexchange.com/questions/70621/how-does-a-node-get-information-from-other-nodes/70623`. Online; Accessed: October 17, 2019.

[31] J. Mišić, V. B. Mišić, and X. Chang, "On the benefits of compact blocks in bitcoin," in *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2020.

[32] A. P. Ozisik, G. Andresen, G. Bissias, A. Houmansadr, and B. Levine, "Graphene: A new protocol for block propagation using set reconciliation," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pp. 420–428, Springer, 2017.

[33] M. T. Goodrich and M. Mitzenmacher, "Invertible bloom lookup tables," in *Communication, Control, and Computing (Allerton), 2011 49th Annual Allerton Conference on*, pp. 792–799, IEEE, 2011.

[34] J. Mišić, V. B. Mišić, and X. Chang, "Performance of bitcoin network with synchronizing nodes and a mix of regular and compact blocks," *IEEE Transactions on Network Science and Engineering*, 2020.

[35] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh, "Erlay: Efficient transaction relay for bitcoin," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 817–831, 2019.

[36] A. Yeow, "Bitnodes." `https://bitnodes.earn.com`. Online; Accessed: November 15, 2018.

[37] A. Yeow, "Bitnodes api v1.0." `https://bitnodes.earn.com/api/`. Online; Accessed: December 17, 2018.

[38] "Full node, node_network_limited (1037) what does it mean?." `https://www.reddit.com/r/Bitcoin/comments/8wkuod/full_node_node_network_limited_1037_what_does_it/`. Online; Accessed: September 19, 2020.

[39] J. Falkner, M. Piatek, J. P. John, A. Krishnamurthy, and T. Anderson, "Profiling a million user dht," in *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pp. 129–134, ACM, 2007.

[40] F. E. Bustamante and Y. Qiao, "Friendships that last: Peer lifespan and its role in p2p protocols," in *Web content caching and distribution*, pp. 233–246, Springer, 2004.

[41] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 314–329, 2003.

[42] J. Li, J. Stribling, R. Morris, and M. F. Kaashoek, "Bandwidth-efficient management of dht routing tables," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pp. 99–114, USENIX Association, 2005.

[43] S. Sen and J. Wang, "Analyzing peer-to-peer traffic across large networks," in *Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pp. 137–150, ACM, 2002.

[44] S. Foss, D. Korshunov, S. Zachary, *et al.*, *An introduction to heavy-tailed and subexponential distributions*, vol. 6. Springer, 2011.

[45] J. P. Nolan, "Maximum likelihood estimation and diagnostics for stable distributions," in *Lévy processes*, pp. 379–400, Springer, 2001.

[46] T. M. Inc., "Fit probability distribution object to data - MATLAB fitdist." `https://www.mathworks.com/help/stats/fitdist.html`. Online; Accessed: November 06, 2018.

[47] "Evaluating goodness of fit." `https://www.mathworks.com/help/curvefit/evaluating-goodness-of-fit.html#bq_5kwr-7`. Online; Accessed: November 18, 2018.

[48] "Coefficient of determination (r-squared) explained." `https://towardsdatascience.com/coefficient-of-determination-r-squared-explained-db32700d924e`. Online; Accessed: December 2, 2018.

[49] "Rms error." `http://statweb.stanford.edu/~susan/courses/s60/split/node60.html`. Online; Accessed: December 2, 2018.

[50] S. Ross, *Stochastic Processes*, p. 114. Wiley series in probability and mathematical statistics, Wiley, second ed., 1995.

[51] M. G. Reed, P. F. Syverson, and D. M. Goldschlag, "Anonymous connections and onion routing," *IEEE Journal on Selected areas in Communications*, vol. 16, no. 4, pp. 482–494, 1998.

[52] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise reduction in speech processing*, pp. 1–4, Springer, 2009.

[53] "Pearson correlations." `https://www.spss-tutorials.com/pearson-correlation-coefficient/`. Online; Accessed: September 23, 2020.

[54] M. A. Imtiaz, "bitcoin-releases." `https://github.com/nislab/bitcoin-releases/tree/tnsm-churn`, 2020.

[55] "Bitcoin developer reference." `https://bitcoin.org/en/developer-reference#remote-procedure-calls-rpcs`, 2017. Online; Accessed: November 17, 2018.

[56] "Bitcoin core :: setban (0.16.0 rpc)." `https://bitcoincore.org/en/doc/0.16.0/rpc/network/setban/`. Online; Accessed: November 17, 2018.

[57] M. A. Imtiaz, "bitcoin-logs." `https://github.com/nislab/bitcoin-logs/tree/tnsm-churn`, 2020.

[58] Y. Minsky, A. Trachtenberg, and R. Zippel, "Set reconciliation with nearly optimal communication complexity," *IEEE Transactions on Information Theory*, vol. 49, no. 9, pp. 2213–2218, 2003.

[59] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese, "What's the difference? efficient set reconciliation without prior context," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 218–229, 2011.

[60] "`miner.cpp`." `https://github.com/bitcoin/bitcoin/blob/master/src/miner.cpp#L321`. Online; Accessed: May 11, 2020.

[61] "`txmempool.h`." `https://github.com/bitcoin/bitcoin/blob/master/src/txmempool.h`, 2018.

[62] "`std::map`." `https://en.cppreference.com/w/cpp/container/map`. Online; Accessed: September 23, 2020.

[63] "`net.h`." `https://github.com/bitcoin/bitcoin/blob/master/src/net.h#L50`. Online; Accessed: May 12, 2020.

[64] M. A. Imtiaz, D. Starobinski, and A. Trachtenberg, "Characterizing orphan transactions in the bitcoin network," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, IEEE, 2020.

[65] "btcd." `https://github.com/btcsuite/btcd`. Online; Accessed: September 23, 2020.

[66] "Bitcoin knots." `https://bitcoinknots.org/`. Online; Accessed: September 23, 2020.

[67] "Bitcoin knots (source)." `https://github.com/bitcoinknots/bitcoin/tree/v0.20.1.knots20200815`. Online; Accessed: September 23, 2020.

[68] "Libbitcoin node." `https://github.com/libbitcoin/libbitcoin-node`. Online; Accessed: September 23, 2020.

[69] "bitcoinj." `https://bitcoinj.org/`. Online; Accessed: September 23, 2020.

[70] "5 bitcoin core alternatives that don't fork the blockchain." `https://bitcoin.eu/bitcoin-core-alternatives-dont-fork-blockchain/`. Online; Accessed: September 23, 2020.

[71] J. Jin, W. Si, D. Starobinski, and A. Trachtenberg, "Prioritized data synchronization for disruption tolerant networks," in *MILITARY COMMUNICATIONS CONFERENCE, 2012-MILCOM 2012*, pp. 1–8, IEEE, 2012.

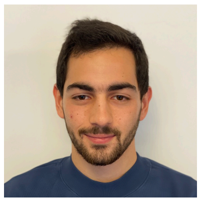
[72] D. Starobinski, A. Trachtenberg, and S. Agarwal, "Efficient pda synchronization," *IEEE Transactions on Mobile Computing*, vol. 2, no. 1, pp. 40–51, 2003.

[73] A. Trachtenberg, D. Starobinski, and S. Agarwal, "Fast pda synchronization using characteristic polynomial interpolation," in *IEEE INFOCOM*, vol. 3, pp. 1510–1519, INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 2002.

Muhammad Anas Imtiaz is a Ph.D. candidate and Doctoral Research Fellow in Electrical and Computer Engineering at Boston University, where he joined in 2017. At present, his research investigates the presence and effects of churn in the Bitcoin network, and possible improvements in the Bitcoin protocol to help mitigate such issues. His work on orphan transactions in the Bitcoin network won a best paper award at the IEEE ICBC 2020 conference. He participated as a reviewer at IEEE TNSM and as a Shadow PC at the ACM IMC 2019 conference. Prior to joining Boston University, Anas worked as a Software Development Engineer (2014-2016), and Senior Software Development Engineer (2016-2017) at Mentor Graphics (now a Siemens business). At Mentor, his responsibilities included development, maintenance and upgrade of several automotive software and legacy products offered by the company. He graduated cum laude with a silver medal in BS Computer Engineering from the National University of Computer & Emerging Sciences, Lahore, Pakistan in 2014.

David Starobinski is a Professor of Electrical and Computer Engineering, Systems Engineering, and Computer Science at Boston University. He received his Ph.D. in Electrical Engineering from the Technion - Israel Institute of Technology, in 1999. He was a visiting post-doctoral researcher in the EECS department at UC Berkeley (1999-2000), an invited Professor at EPFL (2007-2008), and a Faculty Fellow at the U.S. DoT Volpe National Transportation Systems Center (2014-2019). Dr. Starobinski received a CAREER award from the U.S. National Science Foundation (2002), an Early Career Principal Investigator (ECPI) award from the U.S. Department of Energy (2004), BU ECE Faculty Teaching Awards (2010, 2020), and best paper awards at the WiOpt 2010, IEEE CNS 2016, and IEEE ICBC 2020 conferences. He is on the Editorial Board of the IEEE Open Journal of the Communications Society and was on the Editorial Boards of the IEEE Transactions on Information Forensics and Security and the IEEE/ACM Transactions on Networking. His research interests are in cybersecurity, wireless networking, and network economics.

Ari Trachtenberg is a Professor of Electrical and Computer Engineering, Computer Science, and Systems Engineering at Boston University, where he has been since September 2000. He received his PhD and MS in Computer Science (2000,1996) from the University of Illinois at Urbana-Champaign, and his SB in 1994 from MIT. He has also been a Distinguished Scientist Visitor at Ben Gurion university, a visiting professor at the Technion - Israel Institute of Technology, and worked at TripAdvisor, MIT Lincoln Lab, HP Labs, and the Johns Hopkins Center for Talented Youth, and has been awarded ECE Teaching Awards (BU, 2013/2003), a Kern fellowship (BU 2012), an NSF CAREER (BU 2002), and the Kuck Outstanding Thesis (UIUC 2000). His research interests include Cybersecurity (smartphones, offensive and defensive), Networking (security, sensors, localization), Algorithms (data synchronization, file edits, file sharing) and Error-correcting codes (rateless coding, feedback).



Nabeel Younis is a recent graduate of computer engineering from Boston University's Electrical and Computer Engineering department. As an undergraduate at BU he piloted the early work of studying the effects of churn on Bitcoin's network. Afterwards Nabeel was a research assistant at MIT Media Lab's Digital Currency Initiative and developed a Non-Interactive Zero-Knowledge proof (NIZKs) library called zkSigma and worked on a use case project called zkLedger. Currently working at Boston-based startup Arwen, Nabeel heads API design and R&D of new trust-less (non-custodial) trading protocols for cryptocurrencies. Recently Nabeel was the Content Director of MIT Bitcoin Expo 2020: Building the Stack, a 2-day conference bringing together leaders in the Bitcoin and cryptocurreny spaces from research labs, universities and governments.