Empirical Comparison of Block Relay Protocols

Muhammad Anas Imtiaz, David Starobinski, and Ari Trachtenberg

***Abstract*—Block relay protocols play a key role in the performance and security of public blockchains. As a result, several such protocols have been deployed in the context of Bitcoin and its variants (*e.g.,* legacy, *compact block relay* and *Graphene*) in an attempt to reduce bandwidth utilization. However, the relative performance of these protocols in realistic networking conditions (*e.g.,* with nodes churning - joining and leaving the network) is still not known. This paper aims to fill this key knowledge gap using an experimental testbed of twelve full nodes connected to the Bitcoin Cash blockchain. With the aid of novel logging tools, we contrast the performance of these three protocols, in realistic scenarios, with respect to communication, delay, and block decoding success. Our main findings are that Graphene generally performs the best when nodes remain connected, boasting an average propagation delay of 190 ms (*i.e.,* 29% lower than compact block and 80% lower than the legacy protocol). However, when nodes churn at a high rate, compact blocks may perform better. Through a careful temporal analysis, we identify some root causes of the protocol inefficiencies, together with potential mitigation. We have made our measurement framework and experimental logs publicly available to the broader research community.**

***Index Terms*—blockchains, Bitcoin, measurements, performance.**

I. Introduction

Blockchains and cryptocurrencies have emerged as disruptive technologies with profound financial and societal impact. Indeed, as of December 2021, the global cryptocurrency market cap exceeds two trillion dollars, with the leading cryptocurrency, Bitcoin, accounting for half of this market. The Bitcoin cryptocurrency is seeing increased adoption throughout the industry, including by financial firms and banks [1], [2], S&P 500 companies [3], [4], and small investors. Notably, the mayor-elects of New York and Miami recently announced that they will receive some paychecks in Bitcoin [5]. Bitcoin has also proven useful for maintaining communication and transaction in countries experiencing financial distress [6], [7].

As the name implies, blockchains consist of a chain of blocks, each of which is comprised of a set of transactions. The chains are maintained by consensus (rather than centralized control), which relies upon *mining* and subsequent dissemination of blocks [8]. The process of verifying the integrity and correctness of blocks is performed by *full nodes* [9], [10], which store the entire blockchain [11] and form a peer-to-peer network for updates. Full nodes verify that a newly mined block meets specifications and is valid (*e.g.,* it does not contain doubly-spent transactions). Once a full node validates a block, it relays the block's contents to peers [12] based on the specifications of an underlying *block relay protocol*. As such, full nodes and block relay protocols play a crucial role in the performance and security of the blockchain system [13]–[16].

Block relay protocols, in turn, need to be resilient to *churn* of full nodes, a phenomenon wherein a node independently departs and returns to the peer-to-peer network [17]–[19]. Churn is ubiquitous and pervasive in blockchain networks [18]–[20] and may occur due to a variety of reasons, such as the need to apply software patches or intermittency of power or network connectivity. Indeed, power outages are common in developing countries [21], [22] and not unusual in developed countries as well [23], [24].

The original (legacy) block relay protocol implemented by Satoshi Nakamoto in Bitcoin is the *normal block relay* protocol [25], in which blocks are relayed with full copies of their transactions. Subsequently, the *compact block* relay protocol [26] was developed (and is now the default) in an effort to reduce the total bandwidth required for block propagation. A compact block contains the same metadata as a normal block, but instead of sending a full copy of each transaction, it sends only a hash of the transaction. Depending on the number of inputs and outputs, a transaction may contain several hundred bytes [27], [28], whereas the hashes used for the compact block require only 6 bytes per transaction [26]. Replacing normal blocks with compact blocks should therefore result in significant bandwidth savings, provided that *the receiver already has the relevant transactions in its mempool* and just needs to know in which blocks they belong. The work in [18], [19] shows that this assumption often does not hold, however, due to churn. These findings raise the following question: *Could the compact block protocol perform worse than the normal protocol, especially when faced with node churn?*

More recently, Ozisik *et al.* [29], [30] introduced *Graphene*, a block relay protocol that uses probabilistic data structures, namely Bloom filters and inverted Bloom lookup tables (IBLTs) to relay blocks. The premise of that work is that combination of these data structures further reduces the size of the block and, consequently, the bandwidth required to relay blocks. However, the work in [29], [30] only compares the performance of Graphene with the compact block relay protocol through simulation. Chiefly, the impact of churn is ignored altogether. Since Graphene is a complex protocol, which may incur several round-trips of communication in the worst-case, it is unclear from the outset if and when this protocol is superior to compact blocks. *Is the performance gain achievable with Graphene significant enough to warrant replacement of the compact block relay protocol in Bitcoin?*

To answer the above questions, we perform a thorough and systematic comparison of the three aforementioned protocols (*i.e.,* the normal, compact block, and Graphene protocols) under various network regimes. Our comparisons are carried

This research was supported in part by NSF under grants CCF-1563753, CNS-1717858, and CNS-1908087.

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

out through the popular Bitcoin Unlimited (BU) client, which is a concrete implementation of Bitcoin Cash (a fork of the Bitcoin blockchain - see Section II-A) that can support all three protocols after some code changes. Unlike existing simulation-based evaluations, our experiments include important real-world artifacts such as the fluctuations in node connectivity that are ubiquitous for these networks. We note that while other block relay protocols have been proposed in the literature (cf. Section II), none of them are currently implemented in Bitcoin clients.

Our experimental testbed, which consists of 12 full nodes, operates in three network regimes. The first regime represents an ideal situation where the full relay nodes in our testbed are `always on`, *i.e.,* continuously connected to the BU network. In the second regime, nodes in the testbed exhibit `statistical churn`, mimicking the statistical behavior of real-life churning nodes on the Bitcoin network [18]–[20]. In the third regime, nodes experience `periodic churn`, wherein nodes cycle through "on" and "off" periods at a fixed frequency. This regime allows us to compare and contrast the impact of different churn parameters on the performance of block relay protocols, and to emulate scheduled network outages due to the aforementioned electrical power loss. We stress that all our experiments consist of full nodes that relay, but do not mine, Bitcoin blocks.

Main Contributions.

We describe and develop fine-grained measurement capabilities to log and assess the performance of three block relay protocols *in-situ*;

We set up a testbed and evaluate the performance of the protocols in three different network regimes through extensive experiments;

We report results for different metrics including the block propagation delay per hop, the communication size (total amount of communication needed to relay a block), and the probability of decoding a block at various points of execution of the protocols;

We perform an in-depth analysis of the dynamic and temporal behavior of the protocols, through which we identify inefficiencies.

Main Findings.

1) In both the `always on` and `statistical churn` regimes, Graphene performs by far the best among the protocols. The mean propagation delays of the compact block and normal block protocols are, respectively, about 1.4 times and 5 times larger than for Graphene, while the mean communication sizes are, respectively, at least 1.8 times and 15 times larger.
2) In the `periodic churn` regime, Graphene protocol still generally performs best, but there are cases where the compact block protocol is superior. The normal protocol consistently performs the worst.
3) Regardless of the location of the peers of our testbed nodes, reducing communication can significantly reduce the propagation delay, as evidenced by the computation of the Spearman Rank Correlation (SRC) coefficient [31] between these two metrics.
4) We uncover previously unknown protocol inefficiencies. For Graphene, we show that a churning node may miss many transactions upon rejoining the network, which can lead to significant delay in decoding the first few blocks. For compact blocks, we find that, excluding the coinbase transaction, *i.e.,* the transaction created by a miner to collect the reward for their work, full transactions contained in the first message sent by the protocol are often redundant or insufficient for block decoding.

In summary, under the many scenarios considered in this paper, the normal protocol never performs better than the compact block protocol, even under extreme churn. In turn, Graphene generally performs better than the compact block protocol, sometimes by a significant amount. Our analysis includes a discussion of potential solutions to the observed inefficiencies. Though these experimental results were measured on Bitcoin Unlimited clients of the Bitcoin Cash network, we expect that they should also be of interest to the related main Bitcoin network.

A. Road map

The rest of this paper is organized as follows. In Section II, we present extensive background on the three block propagation protocols discussed in this work, together with other related works. In Section III, we introduce our measurement tool, and evaluate the performance of the three block relay protocols in the `always on`, `statistical churn`, and `periodic churn` network regimes, together with in-depth analyses of the Graphene and compact block relay protocols. Section IV concludes the paper, suggests potential improvements to the protocols, and discusses areas for future work.

II. Background and Related Work

In this section, we provide relevant background material on the relay of blocks in the Bitcoin Unlimited network, followed by a discussion of related works.

A. Bitcoin Unlimited

Bitcoin Unlimited is an implementation of the Bitcoin Cash (BCH) protocol which was forked in 2017 from the reference implementation of Bitcoin, also known as Bitcoin Core [32]. The first release of BCH-compatible Bitcoin Unlimited came in the same year [33]. We next explain the reasoning behind this fork.

The problem of scalability in Bitcoin is well documented [34]. The original Bitcoin protocol placed a 1 MB size limit on a *block*, which is a collection of Bitcoin transactions. While the exact reason remains unknown, it is speculated that Satoshi Nakamoto, the creator of Bitcoin, placed this limit to prevent adversaries from creating very large blocks filled with invalid transactions and spamming the network [35]. This limitation capped the maximum rate at which transactions could be committed to the blockchain at roughly 7 transactions per second [36], which seemed sufficient at the time. However, over the years, the popularity of Bitcoin grew together with the volume of transactions. In 2010, for example, the number

Listing 1: *Workflow of the normal block relay protocol.* SRC *and* DST *are peers where the former sends a normal block which the latter receives.*

```
1 SRC: send headers to DST for block G
2 DST: request block G from SRC via getdata
3 SRC: send block G to DST via block
4 DST: process block G and relay to peers
```

of confirmed transactions (*i.e.,* added to a block) per day was recorded at less than 200. In comparison, by 2016, this number grew to around 250,000 confirmed transactions per day - a roughly three orders of magnitude increase [37]. However, roughly 25,000 transactions still remained unconfirmed (*i.e.,* waiting to be added to a block) per day. Critics of Bitcoin Core believe that the block size limit is one of the causes for the low throughput of transaction confirmation in Bitcoin.

Enter BCH-compatible Bitcoin Unlimited. It differs from Bitcoin Core mainly in the following ways: a) the block size limit is removed from consensus rules, and b) miners can freely adjust the block size [38]. Bitcoin Unlimited promises a higher transaction throughput than Bitcoin Core. While the number of transactions in a Bitcoin Core block remains strictly below 2,600 [39], Bitcoin Unlimited blocks have been known to contain over 24,000 transactions (*e.g.,* block number: 681,765) [40].

Similar to Bitcoin Core, a new block is generated, or *mined*, roughly every 10 minutes, on average, in Bitcoin Unlimited. Likewise, a new transaction, when announced, is stored in local memory, dubbed the *mempool*, of every node participating in the network, where it remains until added to a future block. In this paper, we focus on Bitcoin Unlimited because it supports not only the default and compact block relay protocol but also the newer Graphene protocol, which allows us to compare the performance of all of these protocols on a common platform.

B. Relay of blocks in Bitcoin Unlimited

When a node in the Bitcoin network receives a new block from one of its peers, it processes the block and relays it forward to its remaining peers. This relay of blocks allows the trust-less Bitcoin network to maintain consensus on valid transactions and balances available in wallets (or user accounts). We describe next the three block relay protocols of interest that are implemented in Bitcoin Unlimited. For the purpose of explaining the protocols, we consider two connected nodes SRC – a source node that relays information – and DST – a destination node that receives it.

Normal (legacy) block relay. The original block relay protocol implemented by Satoshi Nakamoto in Bitcoin is the *normal block relay* protocol. In this protocol, blocks are relayed with full transactions included, which often results in a waste of bandwidth as the receiving node most likely already received these transactions from their peers earlier.

Listing 1 shows the process of normal block relay: SRC announces the availability of a new block by sending the block's header to DST via a `headers`[1] message; DST requests the block from SRC if it is not already present in its inventory by sending a `getdata` message to the latter; SRC then responds with the full block via the `block` message; DST processes the block which includes, among other steps, validating the block header, validating the transactions within the block, removing transactions included in the block from its mempool, and so on; finally, DST propagates the block to its other peers.

Listing 2: *Workflow of the compact block relay protocol.* SRC *and* DST *are peers where the former sends a compact block which the latter receives.*

```
1  SRC: send headers to DST for block G
2  DST: request block G from SRC via getdata
3  SRC: send block G to DST via cmpctblock
4  DST: attempt to reconstruct block G
5  if reconstruct successful, i.e., no missing transactions in
     block G then
6      DST: process block G and propagate to peers
7  else
8      DST: request missing transactions from SRC via
         getblocktxn
9      SRC: send requested transactions to DST via
         blocktxn
10     DST: reconstruct block G
11     DST: process block G and relay to peers
```

Compact block relay. The *compact block relay* protocol was introduced to Bitcoin Core in 2016 [41] and implemented in Bitcoin Unlimited in 2019 [42]. Unlike a normal block which contains full transactions, the compact block only contains a 6-byte hash of each transaction with only a few full transactions (including the coinbase transaction) - we evaluate the usefulness of these extra full transactions in Section III-I. This process can reduce the bandwidth required to propagate a block by several orders of magnitude. The protocol works under the assumption that the receiver of the block already has in its mempool all transactions that are representative of the 6-byte hashes contained in the block.

Listing 2 shows the relay of a compact block. SRC announces the availability of a new block by sending a `headers` message to DST who requests the block using the `getdata` message if the block is not already present in its inventory. SRC responds with the `cmpctblock` message which contains the block header, some full transactions, and 6-byte hashes of the remaining transactions. If no transactions are *missing* from the receiver's mempool, DST is able to successfully *reconstruct* the block, and propagate it to peers (represented by lines 5-6).

If DST cannot find all transactions in its mempool that correspond to the 6-byte hashes that are included in the block, the compact block reconstruction *fails*. This is represented by lines 7-11 in Listing 2. DST requests these missing transactions by sending a `getblocktxn` message to SRC who responds with the requested transactions in a `blocktxn` message. DST is now able to successfully reconstruct the block and

[1]Note that newer versions of Bitcoin Unlimited replace the legacy `inv` message with the `headers` message for the purpose of block relay.

propagate it forward to peers. It is evident that recovering missing transactions requires extra round-trip communication, which incurs delay in propagation of the block and consumes additional bandwidth.

Note that, to the best of our knowledge, unlike Bitcoin Core, which supports both *low bandwidth* and *high bandwidth* modes in the compact block protocol, the Bitcoin Unlimited client only supports the low bandwidth mode [43], whereby a node only propagates a block forward after fully validating it locally. We refer the reader to the documentation of the compact block protocol for further details on the two modes [26]. Listing 2 depicts the low bandwidth mode of the protocol.

Graphene block relay. The *Graphene block relay* [29], [30] protocol is a more complex protocol that uses probabilistic data structures, namely Bloom filters and invertible Bloom lookup tables (IBLTs), to relay blocks (see Appendix A for background on both of these data structures). The combination of these data structures further reduces the size of the block and, consequently, the bandwidth required to relay blocks.

We next examine the Graphene block relay protocol in detail. We note in passing that this detailed description of the Graphene implementation in Bitcoin Unlimited is a contribution in its own right, since prior works provide only higher-level descriptions. Listing 3 shows the workflow of the Graphene block relay protocol. SRC announces the availability of a new block to DST via the `headers` message. If DST does not already have the block in its inventory, it sends a request to SRC with a `get_grblk` message *along with* the size m of its mempool, *i.e.,* the number of transactions in its mempool. SRC encodes the hashes of transactions in the block in a Bloom filter B and an IBLT I. It uses m as a parameter to determine the sizes of these data structures which in turn determines the number of symmetric differences that can be recovered from I. SRC then sends the Bloom filter B and the IBLT I to DST with the `grblk` message. DST collects hashes of all transactions currently in its mempool and orphan pool [44], [45] and passes them through the Bloom filter B. It creates a candidate set C of transactions whose hashes successfully pass through the filter. DST then uses C to construct a local IBLT I^0, performs the subtraction operation, *i.e.,* $I \quad I^0$, and attempts to extract the transaction hashes encoded in I by decoding the result obtained from the subtraction operation. If the decoding process is successful *and* DST has in its mempool and/or orphan pool all transactions that are included in the block, it reconstructs and processes the block, and then propagates it to its peers. This scenario is represented by the code branch marked by ¬ in Listing 3.

However, even if the subtraction operation $I \quad I^0$ and the successive decoding operation are successful, there may be some transactions in the block that are missing from the mempool and orphan pool of DST. In this case, DST needs to recover the missing transactions before it can process the block. DST does this by sending a `get_grblktx` message to SRC who responds with the requested transactions in a `grblktx` message. DST is now able to successfully reconstruct the block, process it, and propagate it to peers. Note that similar to compact block relay, recovering missing transactions requires an extra round-trip communication and delays the

Listing 3: *Workflow of the Graphene block relay protocol.* ***SRC*** *and* ***DST*** *are peers where the former sends a Graphene block that the latter receives.*

```
1  SRC: send headers to DST for block G
2  DST: request block G from SRC via get_grblk
3  SRC: encode hashes of transactions in block G into
     Bloom filter B and IBLT I
4  SRC: send B and I to DST in grblk
5  DST: create candidate set C from transactions in mempool
     and orphan pool whose hashes pass through B
6  DST: create IBLT I^0 from C
7  DST: attempt to extract encoded transaction hashes, i.e.,
     find I  I^0, and decode result
8  if IBLT subtraction I  I^0 and decode successful then
9      DST: attempt to reconstruct block G
10     if reconstruct successful, i.e., no missing transactions
         in block G then                                  ▶ ¬
11         DST: process block G and relay to peers
12     else                                               ▶ -
13         DST: request missing transactions from SRC via
             get_grblktx
14         SRC: send requested transactions to DST via
             grblktx
15         DST: reconstruct block G
16         DST: process block G and relay to peers
17 else ▶ IBLT subtraction and decode failed;
   initiate failure recovery
18     DST: create Bloom filter F of transaction set C
19     DST: request failure recovery from SRC by sending F
         via get_grrec
20     SRC: find set C^0 of transaction that are in block G but
         not in Bloom filter F
21     SRC: create IBLT J
22     SRC: send set C^0 and IBLT J to DST via grrec
23     DST: create IBLT J^0 from candidate set C [ C^0
24     DST: attempt to extract encoded transaction hashes,
         i.e., find J  J^0, and decode result
25     if IBLT subtraction J  J^0 and decode successful then
26         DST: attempt to reconstruct block G
27         if reconstruct successful, i.e., no missing
             transactions in G then                       ▶ ®
28             DST: process block G and relay to peers
29         else                                           ▶ —
30             DST: request missing transactions from SRC
                 via get_grblktx
31             SRC: send requested transactions to DST via
                 grblktx
32             DST: reconstruct block G
33             DST: process block G and relay to peers
34     else                                               ▶ °
35         DST: initiate fail-over mechanism by requesting
             normal block
```

propagation of the block in addition to consuming additional bandwidth. This scenario is represented by the code branch marked by – in Listing 3.

Next, we look at the case when the subtraction operation $I - I'$ and the successive decode operation are not successful. This may happen when DST is missing too many transactions from its mempool. It, thus, cannot create a transaction candidate set C and, consequently, an IBLT I' that is sufficient to perform the subtraction and decode operations successfully. When this happens, DST enters *failure recovery* [46] with SRC. This step requires one or more extra round-trips of communication to recover from the IBLT decode failure which further delay block propagation and consume extra bandwidth.

To perform failure recovery, DST creates a new Bloom filter F and inserts into it hashes of the transactions from the candidate set C. This enables SRC to determine which transactions are present in DST's mempool. DST then sends F to SRC with a `get_grrec` message. SRC creates a new set of transactions C' that is comprised of the transactions that are in the block but whose hashes are not in F. It also creates a revised IBLT J adjusting for the false positives that appear in F, and then sends C' and J to DST as a `grrec` message. DST creates a new candidate set of transactions $C \cup C'$. It locally creates an IBLT J' from this candidate set and uses it to extract the transaction hashes encoded in J by first performing the subtraction operation $J - J'$ and then decoding the result of the subtraction operation.

If the subtraction and decoding operations are successful, DST attempts to reconstruct the block. If at this point, there are no transactions missing from its mempool, DST successfully reconstructs the block, processes it, and propagates it forward to peers. This scenario is represented by the code branch marked by ® in Listing 3.

If, however, there are still some transactions missing from its mempool, DST requests these transactions from SRC. Upon receiving the missing transactions, DST successfully reconstructs the block, processes it, and propagates it forward to peers. Note that this scenario requires yet another round-trip communication further delaying block propagation and consuming additional bandwidth. This scenario is represented by the code branch marked by ¯ in Listing 3.

One may wonder why there may still be missing transactions after failure recovery. This could be because the Bloom filter has a non-zero probability of false positives and SRC may falsely conclude that there already exists a transaction in Bloom filter F and not include it in the set of transactions C'. In this case, DST may end up in a situation where it needs to perform another round-trip communication to recover transactions that are included in the block but not present in its mempool.

Finally, if the subtraction operation $J - J'$ and the successive decode operation also fail, DST must fall back to a fail-over mechanism by requesting a full block (as in the normal block protocol) instead of a Graphene block from SRC. This scenario is represented by the code branch marked by ° in Listing 3.

C. Other related work

Neudecker [47] built a tool (that is not publicly available) for monitoring different aspects of the Bitcoin network. Though this tool provides valuable information on general network properties, including end-to-end propagation delays and churn, it does not provide detailed information about events related to the propagation of transactions and blocks at individual nodes, which is crucial for understanding the causes of delays and network inefficiencies.

Kalodner *et al.* [48] present *BlockSci*, a blockchain analysis tool designed to perform an analysis on transaction graphs, scripts, block indexes and other data that are view-able by the end-user. Though this tool is geared towards analysis of privacy and forensics, it lacks the ability to perform analysis on parameters such as propagation times of blocks, transactions missing from blocks, etc., which our logging system is able to achieve.

Imtiaz *et al.* [18], [19] show that churn is ubiquitous in the Bitcoin network, and they create statistical distributions that characterize this churn. They use the characterization to study the impact of churn on the propagation of compact blocks in the Bitcoin network. In our work, we present an in-depth study of the efficiency of the Graphene block relay protocol, with and without churn, and provide a comparison to the default and compact block protocols. We further identify previously unknown inefficiencies of the compact block protocol (see Section III-I). The work in [20] shows that churn is significant in the Ethereum network. This confirms that churn is a general phenomenon in blockchain systems that must be considered in the design of block relay protocols.

Rohrer *et al.* [49] present *Kadcast*, a protocol which, unlike mainstream blockchain systems such as Bitcoin or Ethereum, uses a structured overlay network to propagate blocks in the blockchain networks. The authors show, via simulations, that their protocol performs better, in terms of propagation delay of blocks, than "VanillaCast", a framework representative of most blockchain networks. To the best of our knowledge, however, the authors do not account for churn in their simulations. It is, therefore, not known how well the protocol performs in such conditions.

As the Bitcoin protocol is currently designed, a node *floods* announcements of transactions it receives to all of its peers even though it is possible that those peers already know of the transactions being announced [50]. Prior work [51] shows that transaction announcement accounts for 30 – 50% of the overall Bitcoin network traffic. Zhange *et al.* [52] and Naumenko *et al.* [51] respectively propose *RepuLay* and *Erlay* to improve the relay of transactions in the Bitcoin network which should consequently improve the relay of blocks. However, to the best of our knowledge, the former was only evaluated over simulations and not implemented in the Bitcoin software, whereas the latter is currently undergoing review and has yet to be merged in the Bitcoin software [53], [54].

III. Evaluation of Block Relay Protocols

In this section, we experimentally evaluate the performance of the Graphene, compact, and default block relay protocols in

three network regimes: (i) `always on`, which represents the ideal situation where full nodes stay continuously connected to the BU network; (ii) `statistical churn`, where nodes churn according to statistical characterization derived in the literature [18], [19]; and (iii) `periodic churn`, whereby nodes follow a periodic churn pattern with fixed duration for the "on" and "off" periods. This allows us to better isolate churn factor that affects the performance of block relay protocols. Further, this can emulate disconnections due to power outages, with nodes staying off the network over extended durations (see discussion in Section I).

The rest of this section is organized as follows. We first detail the mechanisms we employ to collect data from our experiments. Next, we describe our measurement setup. Then, in the rest of this section, we present our extensive findings on the efficiency (or lack thereof) of the aforementioned block relay protocols for various metrics and in different network regimes.

A. Data collection mechanism

Our data collection mechanism builds upon the "log-to-file system" developed in [18], [19] that produces human-friendly, easy-to-read text files. First, we port the system to the Bitcoin Unlimited software. Next, we significantly augment this logging system with new capabilities to record data relevant to Graphene blocks, including capturing various events relevant to this complex protocol, as described in Section II. We also add new functionality so that data related to compact and normal blocks can be recorded with finer granularity. The logging system allows one to (i) identify events when they occur; (ii) follow the changes in states as they take place; and (iii) record relevant data, *e.g.,* the hash of a block that is announced and the list of transactions in the block, to files which one can later use to obtain results. Note that the Graphene block relay protocol has two dozen states and numerous state transitions which requires significant effort to identify and track in the Bitcoin Unlimited software. The Appendix of the dissertation [55] expands in greater detail upon the complexity of capturing all events related to each block relay protocol.

The primary data point in our experiments is a block and we tie every data associated with the block to its unique hash. We do this for each Graphene, compact, and normal block received by our measurement nodes. This allows us to isolate, identify, and acquire enough information to get necessary results. We use this logging system as our primary method of obtaining data in our experiments. Our logging system is public and available on GitHub [56].

B. Experimental setup

The purpose of the experiments in this section is to gauge the performance of the Graphene, compact, and default block relay protocols. For this purpose, we connect 12 nodes to the live BU 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), 1 TB HDD and 12 GB RAM. The nodes are each running the Linux Ubuntu 18.04.5 LTS (Bionic Beaver) distribution. The nodes run v1.9.0.1 of BU with an implementation of a bug fix [57] and an implementation of the logging system detailed in Section III-A. We have made this version of BU public and available on GitHub [56]. We emphasize that all of the nodes in our experimental setup are on both the `DST` side of Listings. 1 - 3 when they receive blocks *from* their peers and on the `SRC` side when they relay blocks *to* their peers. However, any reference to a "node" in this paper is when it is on `DST` side and relevant information is recorded.

To study the performance of block relay protocols in the `always on` and `statistical churn` regimes, we run experiments and measure data over a period of two weeks starting from Tuesday, April 20, 2021 00:00:00 EST. Six nodes always stay connected to the BU network throughout the measurement period. Two of these nodes are configured to accept Graphene blocks only, two to accept compact blocks only, and the remaining two to accept neither, *i.e.,* accept default blocks only. Additionally, six nodes fluctuate on and off the BU network using session lengths sampled from distributions that represent churn in the Bitcoin network [19]. Specifically, the nodes stay *on* and *off* the network with session lengths sampled from the *log-logistic* (see Eq. (1) and corresponding parameter values in [19]) and the *Weibull* (see Eq. (2) and corresponding parameter values in [19]) distributions, respectively. Two of these nodes accept Graphene blocks only, two accept compact blocks only, and the remaining two only accept default blocks.

To study the performance of block relay protocols in the `periodic churn` regime, we introduce the following fluctuation periods: 20 minutes (m), 1 hour (h), 3 h, and 6 h. We chose the duration of these periods based on results of preliminary experiments: durations that are either shorter or longer do not yield results that are markedly different from ones presented in this section in the 20 minutes and 6 hours, respectively. For each of the fluctuation period duration, we further divide experiments into two cases. In the first experiment, which ran for one week starting from Thursday, March 25, 2021 22:00:00 EST, we set the off duty cycle to be 25% of the fluctuation period. That is, during each fluctuation period, the node stays off the network for the first 25% of the time, and on the network for the remaining 75% of the time. In the second experiment, which ran for one week starting from Friday, April 02, 2021 02:01:19 EST, we similarly set the off duty cycle to be 75% of the total duration of the fluctuation periods. We split the 12 nodes in our testbed into four groups of three nodes. Each group is configured with one of the four aforementioned fluctuation periods (*i.e.,* 20 m, 1 h, 3 h, and 6 h). Within each group, one node accepts Graphene blocks, one node accepts compact blocks, and the last node accepts default blocks only.

To get an in-depth view of additional transactions in `cmpctblock` messages, we performed another experiment which ran for two weeks starting from Friday, September 3, 2021 02:00:00 EST. In this experiment, three of the 12 nodes are always on, and the remaining nine statistically churn according to distributions presented in prior work [18], [19]. Since this is a study on additional transactions in

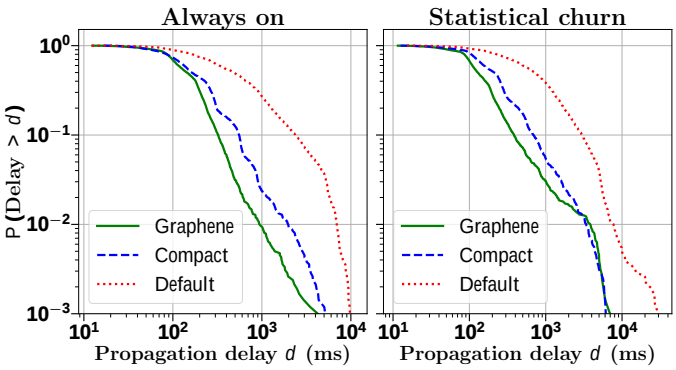


Fig. 1: *Complementary cumulative distribution functions (CCDFs) of block propagation delays in Graphene, compact, and normal block relay protocols in the* `always on` *and* `statistical churn` *regimes. The Graphene block relay protocol performs best in roughly* 99% *of blocks whereas the normal block relay protocol always performs worst.*

Fluctuating period	Graphene		Compact		Default	
	100 ms	1,000 ms	100 ms	1,000 ms	100 ms	1,000 ms
20 m	83.65%	10.57%	86.25%	16.57%	98.16%	69.19%
1 hr	76.56%	15.65%	84.10%	15.42%	95.88%	51.67%
3 hr	75.26%	8.85%	79.22%	8.62%	93.55%	37.67%
6 hr	75.60%	3.30%	79.03%	3.41%	93.60%	36.65%

(a)

Fluctuating period	Graphene		Compact		Default	
	100 ms	1,000 ms	100 ms	1,000 ms	100 ms	1,000 ms
20 m	95.66%	42.02%	96.72%	49.13%	99.19%	86.54%
1 hr	75.35%	28.46%	78.96%	21.99%	97.24%	46.81%
3 hr	78.86%	16.61%	77.99%	14.92%	96.34%	48.32%
6 hr	77.52%	9.22%	76.02%	9.03%	92.04%	38.75%

(b)

TABLE I: *Fraction of blocks that have propagation delay larger than* 100 *ms, and* 1,000 *ms in Graphene, compact and default block relay protocols over varying fluctuating periods with* (a) 25%*, and* (b) 75% *off duty cycles, in the the* `periodic churn` *regime.*

`cmpctblock` messages, all 12 nodes are configured to accept compact blocks only.

Note that nodes configured to accept Graphene or compact blocks only *must* also accept default blocks as a fail-over mechanism in case the aforementioned protocols fail drastically. Therefore, our testbed nodes could connect to peers that are not configured with the same relay protocol. In such cases, a testbed node and its peer will relay default blocks only. To make sure that we do not introduce any bias in our results, we do not force our nodes to drop connections with peers with whom they can only communicate via the default block relay. Thus, our nodes connect to peers following the protocol implemented in the Bitcoin software by default. Results obtained from our experiment are detailed in the sections that follow. We have made our experimental logs publicly available for use by the wider research community [58].

C. Statistics on the propagation delay of blocks

In this section, we present statistics on the one-hop block propagation delays in different network regimes. We measure propagation delay as the difference between the time the header of a block, *i.e.,* the `headers` message, is received by a measurement node and the time at which the block is fully reconstructed and processed.

Fig. 1 shows the complementary cumulative distribution functions (CCDFs) of block propagation delays for nodes in the `always on` and `statistical churn` regimes configured with the Graphene, compact, and normal block relay protocols. In nodes in the `always on` regime, Graphene, compact, and normal blocks have mean propagation delays of 190.67 ms, 268.34 ms, and 974.11 ms, respectively. On the other hand, in nodes in the `statistical churn` regime, Graphene, compact, and normal blocks have mean propagation delays of 259.13 ms, 364.19 ms, and 1287.22 ms, respectively.

These statistics show that i) Graphene blocks have the smallest average propagation delays out of the three protocols, whereas normal blocks have the largest average propagation delays across both regimes, and ii) blocks across the three different protocols in the `statistical churn` regime always have, on average, larger block propagation delays as compared to blocks across the respective protocols in the `always on` regime, which is explained by nodes not receiving several transactions from their peers while they are off the network. Upon receiving a block that may contain many of these missing transactions, the nodes must perform round-trip communication to recover the transactions (in the case of Graphene or compact block relay protocols) and/or to recover from block decode failure (in the case of Graphene block relay protocol) which also results from missing transactions. The extra communication adds to the delay in reconstructing blocks and, consequently, the delay in propagation of these blocks.

Interestingly, even though the absolute performance of each of the three protocols is different in the `always on` and `statistical churn` regimes, their *relative* performance remains the same, *i.e.,* in both regimes the average propagation delay of Graphene is about 29% lower than that of the compact block protocol and 80% lower than that of the normal block protocol.

TABLE I shows the fraction of blocks that have a propagation delay exceeding 100 ms and 1,000 ms respectively, across the block relay protocols in nodes in the `periodic churn` regime, for different fluctuating periods and off duty cycles. A key takeaway from the table is that the default block relay protocol always performs worse, in our experiments, than both the Graphene and compact block relay protocols. The default block contains full transactions as compared to Graphene and compact blocks that contain short hashes of a majority of transactions, and the result is that default blocks take longer to propagate.

A trend that can be observed across all fluctuating periods is that both Graphene and compact block relay protocols perform worse when they are off the network for 75% of the fluctuating periods. Similar to the case with `always on` and `statistical churn` regimes, this can be attributed

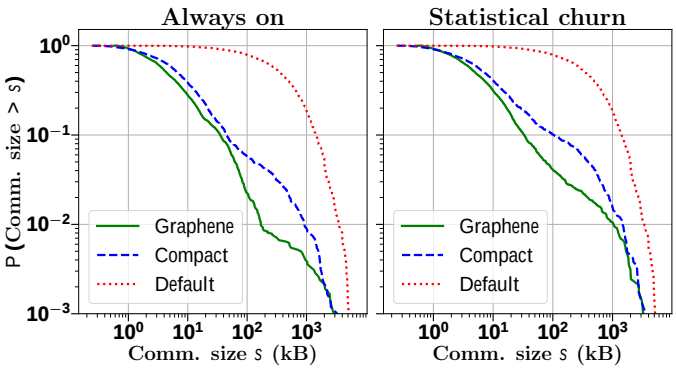


Fig. 2: *CCDFs of block communication sizes in Graphene, compact, and default block relay protocols in the* `always on` *and* `statistical churn` *regimes. Graphene block relay protocol performs best in both regimes whereas default block relay protocol performs worst.*

to extra round-trip communication for recovering missing transactions and performing failure recovery.

Next, it is apparent that as the lengths of the fluctuating periods increase, the performance of both the Graphene and compact block relay protocols improves. That is, a smaller fraction of blocks has large propagation delays. We theorize that this is because as the nodes stay off the network for longer, a large number of the transactions that they miss receiving from their peers is already included in blocks that they also miss receiving while they are off the network. Therefore, once they rejoin the network, they miss fewer transactions and suffer from fewer failures, thereby requiring fewer additional round-trip communications. Therefore, they experience shorter block propagation delays.

On the other hand, we note that the Graphene block relay protocol generally outperform the compact block relay protocol except for a few cases, especially when the fluctuating period is 1 hr and the off duty cycle is 75% of the fluctuating period. This is likely because the node misses several transactions from its peers that are going to be included in the next few blocks that it will receive. Therefore, the node requires additional round-trip communication to recover these transactions adding to the propagation delay of blocks.

D. Statistics on the communication size per block

Next, we look at statistics on the communication size per block received by different types of nodes. When calculating the total communication size per block, we take into account the serialized size of the initial block received (*i.e.,* `grblk`, `cmpctblock`, and `block` for Graphene, compact and default blocks, respectively), and the serialized sizes of all follow up round-trip messages sent and received to and from peers. These messages could, *e.g.,* be sent to recover missing transactions from peers or to perform failure recovery. For detail on all possible message exchanges between nodes in BU, please refer to Section II-B.

Fig. 2 shows the CCDFs of block communication sizes for the nodes in the `always on` and `statistical churn` regimes configured with the Graphene, compact and normal block relay protocols. Again, Graphene performs best. In

Block relay protocol	Comm. size	Fluctuating period			
		20 m	1 hr	3hrs	6 hrs
Graphene	10 kB	57.20%	69.41%	50.82%	51.54%
	100 kB	47.72%	54.45%	23.04%	13.06%
	1,000 kB	23.90%	23.54%	9.04%	4.70%
Compact	10 kB	76.65%	64.40%	57.92%	51.50%
	100 kB	52.29%	52.05%	26.76%	13.53%
	1,000 kB	18.27%	13.93%	9.18%	5.10%
Default	10 kB	98.56%	98.50%	98.46%	99.23%
	100 kB	82.71%	81.05%	81.37%	81.03%
	1,000 kB	30.51%	25.98%	26.62%	28.63%

(a)

Block relay protocol	Comm. size	Fluctuating period			
		20 m	1 hr	3hrs	6 hrs
Graphene	10 kB	26.93%	29.91%	41.41%	31.74%
	100 kB	11.20%	18.83%	12.46%	7.77%
	1,000 kB	3.39%	11.07%	4.75%	2.13%
Compact	10 kB	63.20%	60.83%	57.26%	53.20%
	100 kB	25.10%	24.46%	19.45%	9.55%
	1,000 kB	5.87%	4.77%	5.53%	1.81%
Default	10 kB	98.15%	98.15%	98.09%	98.05%
	100 kB	83.03%	82.46%	82.14%	81.72%
	1,000 kB	31.54%	28.97%	28.57%	27.79%

(b)

TABLE II: *Fraction of blocks that have communication sizes larger than* 10 *kB,* 100 *kB, and* 1,000 *kB in Graphene, compact and default block relay protocols over varying fluctuating periods with* (a) 25%, *and* (b) 75% *off duty cycles, in the the* `periodic churn` *regime.*

nodes in the `always on` regime, Graphene, compact, and normal blocks have mean communication sizes of 24.53 kB, 46.17 kB, and 599.93 kB, respectively. On the other hand, in nodes in the `statistical churn` regime, Graphene, compact, and normal blocks have mean communication sizes of 40.58 kB, 73.37 kB, and 592.64 kB, respectively.

TABLE II shows the fractions of blocks that have a total communication size larger than 10, 100, and 1,000 kBs across the block relay protocols in nodes in the `periodic churn` regime for different fluctuating periods and off duty cycles. Similar to the statistics presented in Section III-C, a key takeaway from the table is that the default block relay protocol always performs worse than Graphene and compact block relay protocols. That is, default blocks are always larger in size than Graphene and compact blocks and any additional round-trip communication *combined*. This is because default blocks contain full transactions each of which can be several hundred bytes in size [27], [28]. By comparison, Graphene and compact blocks contain only short hashes representing transactions which considerably reduce the overall size of the blocks.

We observe more trends similar to those highlighted in the Section III-C: across all fluctuating periods, both Graphene and compact block relay protocols perform worse when their off duty cycle is 75% of the fluctuating period; as the length of the fluctuating periods increase, the performance of the Graphene and compact block relay protocols improves. Finally, the Graphene block relay protocol almost always outperforms the compact block relay protocol except in a few cases where our

Block relay protocol	Graphene	Compact	Default
Correlation coefficient ρ	0.51	0.61	0.68

(a)

Fluctuating period	Correlation coefficient ρ					
	Graphene		Compact		Default	
	25%	75%	25%	75%	25%	75%
20 m	0.69	0.76	0.68	0.78	0.67	0.50
1 hr	0.65	0.85	0.69	0.84	0.50	0.42
3 hr	0.60	0.70	0.72	0.79	0.61	0.32
6 hr	0.56	0.61	0.72	0.71	0.61	0.74

(b)

TABLE III: *Coefficients for Spearman Rank Correlation between the block propagation delays and block communication sizes in Graphene, compact, and default block relay protocols in nodes in the* (a) `statistical churn`*, and* (b) `periodic churn` *regimes. In general, the propagation delays and communication size are moderately to highly correlated.*

proposed theory from the previous section applies.

E. Correlation between propagation delay and communication per block

As observed from the previous two sections, the communication and delay performance of the various protocols follow similar trends. We next rigorously quantify the correlation between these metrics by calculating the Spearman Rank Correlation (SRC) coefficient ρ [31]. Provided two data sets D_1 and D_2 of equal size n, the SRC coefficient is given by

$$\rho = 1 - \frac{6 \sum_{i=1}^{n} \left(D_{1_i} - D_{2_i} \right)^2}{n \left(n^2 - 1 \right)},$$

where $-1 \leq \rho \leq 1$, and D_{1_i} and D_{2_i} are the *ranks* of the i^{th} data point in sets D_1 and D_2, respectively [59]. Values of $\rho = -1$ and $\rho = 1$ imply an exact monotonic relation between data sets D_1 and D_2 where the former implies that D_1 increases as D_2 increases and the latter implies the opposite [60]. The ranks in SRC are determined as follows: the data sets are sorted in ascending order and the values are replaced by their corresponding ranks [61].

Our results are summarized in TABLE III(a) and TABLE III(b). We find that there generally exists moderate, *i.e.,* $\rho^2 \approx [0.4, 0.6)$, to strong relationship, *i.e.,* $\rho^2 \approx [0.6, 0.8)$ [62], between block propagation delays and block communication sizes in all three block propagation relay protocols. That is, as churning nodes exchange additional messages to recover missing transactions in blocks, their propagation delay can be expected to increase regardless of the geographical locations of neighboring peers in the Bitcoin network.

F. Graphene in depth

Our findings in Sections III-C and III-D indicate that the performance of the Graphene block relay protocol degrades in some cases. In this section, we take a deeper look into the potential reasons for this degraded performance.

Recall from Section II-B that there are several scenarios where the Graphene protocol requires extra round-trip communication, which includes recovering missing transactions

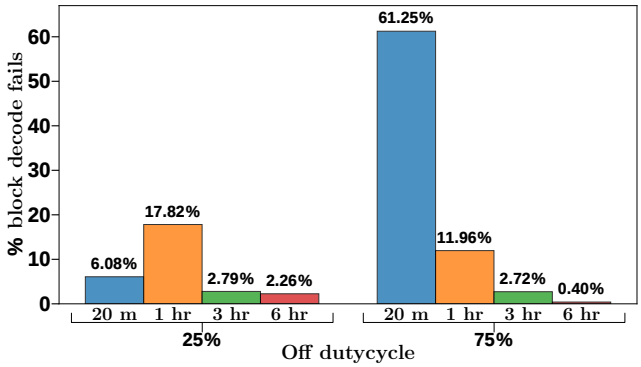


Fig. 3: *Proportion of block decode failures,* i.e., *scenarios* Ⓡ, ‾, *and* ° *in Listing* 3*, over different fluctuation periods with* 25% *and* 75% *off duty cycles, in the the* `periodic churn` *regime. Block decode failure rates are higher when nodes churn more often and stay off the network longer thereby not being able to recover. This is prominent in fluctuating periods of* 20 *m and* 1 *hr.*

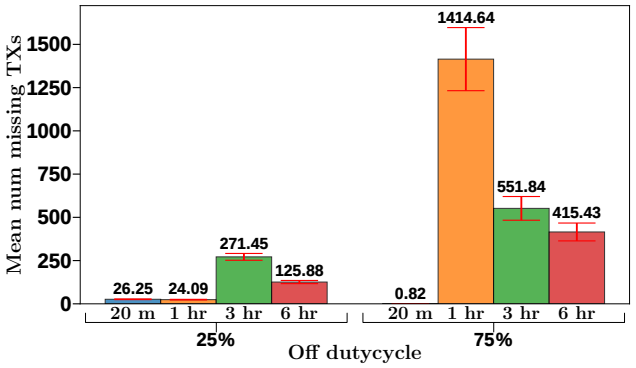
from peers and performing failure recovery when block decode fails. The Graphene block relay protocol is complex: for example, block decode could be successful but there might be transactions missing from the mempool. On the other hand, block decode could fail and there may or may not still be transactions missing from the mempool even after failure recovery.

Block decode failure occurs when the condition in line 8 of Listing 3 returns `false` (*i.e.,* the subtraction operation $I - I'$ fails). When this happens, failure recovery is performed as depicted by lines 17 onward (*i.e.,* scenarios Ⓡ, ‾, and °) in the Listing 3.

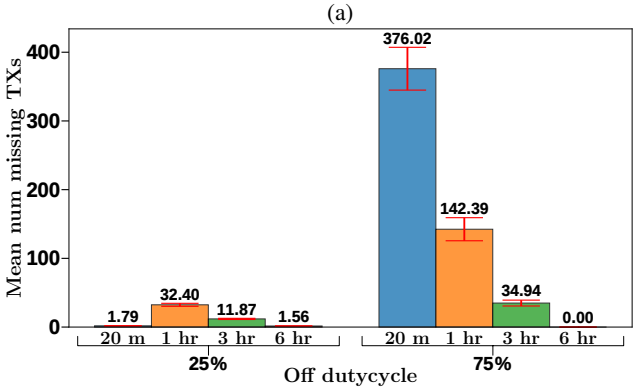
Fig. 3 shows the proportion of Graphene blocks that suffer from decode failure when the off duty cycle is 25% and 75%, respectively, and for different fluctuation periods. The figure shows that block decode failure rates are higher when nodes churn more often *and* stay off the network longer. They are, hence, unable to recover from staying off the network. As the nodes churn less frequently as well as stay on the network longer, the block decode failure rates significantly drop.

Next, we investigate the case when block decode is successful but transactions are missing from the mempool of the node when a new block is received. This is represented as Scenario ‒ in Listing 3 on lines 12-16.

Fig. 4(a) shows the mean number of transactions missing from mempool when a block is received and successfully decoded with 25% and 75% off duty cycle. The combination of Fig. 3 and Fig. 4(a) provide a thought-provoking insight: when a node churns frequently, it misses receiving enough transactions that will result in a higher fraction of block decode failures and the missing transactions will be recovered via failure recovery. On the other hand, when a node churns less frequently, it still misses transactions which are not enough to cause block decode failure. These transactions are then recovered by sending transaction recovery requests to peers. Additionally, as the nodes stay on the network for longer, they miss fewer transactions. In either cases, *both* failure recovery and recovering missing transactions require an extra round-trip



(a)



(b)

Fig. 4: *Average number of missing transactions (with* 95% *confidence intervals) from blocks that are* (a) *decoded successfully,* i.e., *scenario* ‑ *in Listing* 3*, and* (b) *not decoded successfully and require failure recovery,* i.e., *scenario* ‾ *in Listing* 3*, over different fluctuation periods with* 25% *and* 75% *off duty cycles, in the the* `periodic churn` *regime*

communication.

Finally, we take a look at the case when block decode is unsuccessful *and* there are still missing transactions even after failure recovery is performed. This is represented as scenario ‾ in Listing 3 on lines 29-33. Interestingly, Fig. 4(b) reveals that when nodes fluctuate frequently *and* stay off the network longer, they may *still* miss transactions after failure recovery. This requires an additional extra round-trip of communication *on top of* that needed to perform failure recovery.

The insights presented in this section explain the cause behind degraded performance of the Graphene block relay protocol in the case when nodes churn frequently and stay off the network longer. In some cases, up to two extra round-trips of communication are required resulting in higher delays in propagation and larger communication sizes.

G. Temporal analysis of the Graphene block relay protocol

In this section, we study the prevalence of scenarios discussed in Section II in Graphene blocks received by churning nodes. More precisely, we consider what happens in the Graphene block relay protocol when a churning node rejoins the network. For this purpose, we create collections of blocks received in each interval for which the churning nodes are connected to the Bitcoin Unlimited network. We then identify the scenarios that each block goes through.

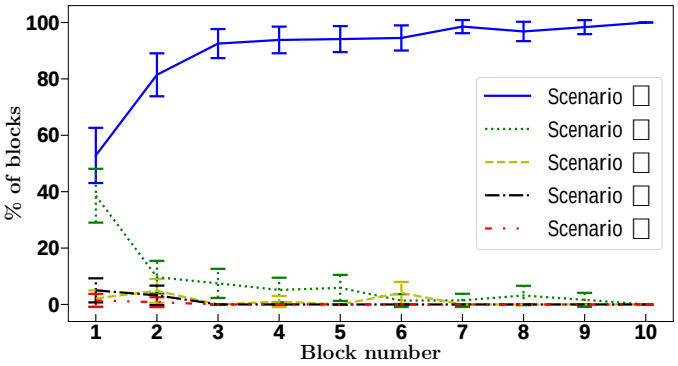


Fig. 5: *Percentage of blocks (with* 95% *confidence intervals) received in the* `statistical churn` *regime that face the five scenarios after the churning nodes rejoin the network. The longer a node stays on the network, the more scenario* ¬ (i.e., *no extra round-trip) prevails whereas the scenarios* Ⓡ, ‾, *and* ° *do not occur very often.*

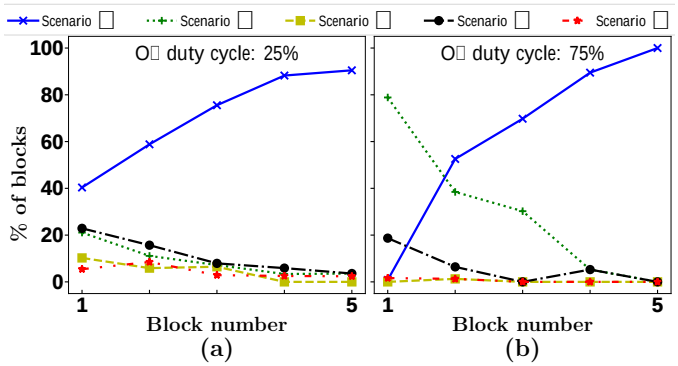


Fig. 6: *Percentage of blocks received in the* `periodic churn` *regime with a fluctuation period of* 1 *hr and off duty cycle of* (a) 25% *and* (b) 75% *that face the five scenarios after a node rejoins the network. In either case, scenario* ¬ *does not represent the majority of cases for the first block and scenarios* Ⓡ *and* ° *occur infrequently.*

We first consider the `statistical churn` regime. Fig. 5 shows findings (with 95% confidence intervals) for the first 10 Graphene blocks received after rejoining the network. Roughly 54% of the Graphene blocks received by a node immediately after it rejoins the network are successfully decoded and have no missing transactions. A significant portion, *i.e.,* roughly 39%, of the Graphene blocks are successfully decoded but have missing transactions. This shows that nodes need to perform round-trip communication with their peers to recover missing transactions immediately after they rejoin the network. While there are some blocks that suffer from scenarios involving block decode failures, their proportion is relatively small. As the node stays connected to the network and receives further blocks, the chances of transactions missing from the block significantly decrease. This trend continues on albeit some random off shoots with small probability (depicted by small peaks in the figure) in blocks with missing transactions.

We observe similar behavior in the `periodic churn` regime. For example, Fig. 6(a) and Fig. 6(b) show statistics for Graphene blocks received by nodes with a fluctuating period of

1 hr and off duty cycles of 25% and 75%, respectively. In both cases, when nodes rejoin the network, they see a large portion of blocks with missing transactions regardless of whether the IBLT decoding process is successful or not. This behavior is worse in nodes that stay off the network longer. However, similar to the `statistical churn` regime, the performance of the Graphene block relay protocol improves over time, and the proportion of blocks with missing transactions decreases significantly.

These results show that the performance of Graphene degrades when a node rejoins the network, often resulting from missing transactions. Prior work [19] shows that synchronizing mempools of churning nodes with mempools of highly-connected nodes in the Bitcoin network helps improve the performance of the compact block relay protocol in churning nodes. Therefore, we believe similar mempool synchronization can improve the performance of the Graphene block relay protocol as well – although we leave the evaluation of this hypothesis to future work.

H. Size of first message across block relay protocols

In this section, we investigate the size of the first block message for each of the relay protocols, as a function of the number of transactions included in a block. Initial messages are important because they are transmitted regardless of the specific scenario that ends up happening with future messages. We are specifically interested in identifying trends as well as outliers. Recall from Section II that the first messages are `grblk`, `cmpctblock`, and `block` for the Graphene, compact, and default block relay protocols, respectively.

Fig. 7 shows on the x-axis the number of transactions in a block received by nodes and on the y-axis the size of the first block message in bytes, in the `statistical churn` regime. *Notice that both the x- and y-axes are plotted on a log scale*. The figure shows that for default blocks, the `block` message is almost always the largest. This is because in default blocks, the first message *is* the entire block and contains full transactions included in the block. Hence, as the number of transactions in the block increase, so does the size of the `block` message. The `block` message has, on average, a size of 5.93 10^5 bytes with a standard deviation of 7.02 10^5 bytes.

Next we compare the sizes of `grblk` and `cmpctblock` messages. We observe that when the number of transactions in a block is small (*i.e.,* up to 60 transactions), the `cmpctblock` message *usually* has a smaller size than the `grblk` message. Further, there is a visibly direct relationship between the number of transactions in a compact block and the size of the `cmpctblock` message in the shape of an almost straight line. This line also forms a lower bound for the size of the `cmpctblock` messages, because for every transaction in a block, the `cmpctblock` message contains a 6-byte hash for the transaction. Therefore, as the number of transactions in a block increase, so does the size of the `cmpctblock` message. Note, however, that there are instances in which the size of the `cmpctblock` message deviates from the straight line. This is due to the additional transactions included in the `cmpctblock` message.

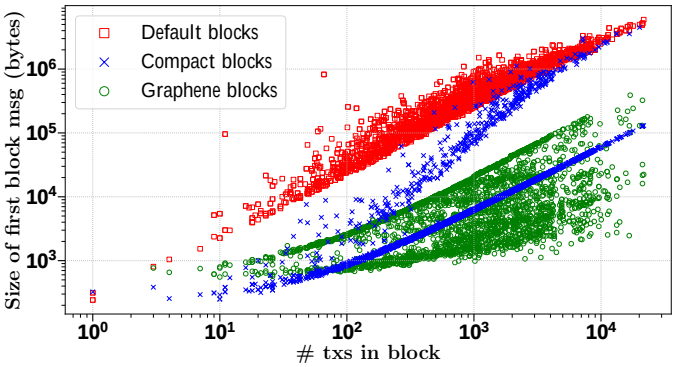


Fig. 7: *Sizes of the first block messages,* i.e., `block`, `cmpctblock`, *and* `grblk`, *against the number of transactions in the respective compact blocks. The* `block` *messages almost always have the largest sizes.*

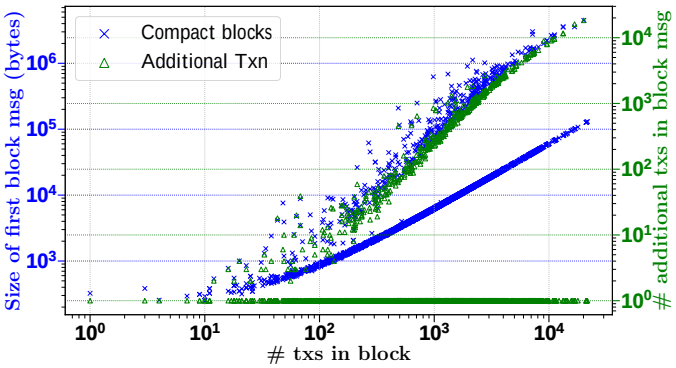


Fig. 8: *Sizes of the* `cmpctblock` *messages (left y-axis) and the number of additional transactions in blocks (right y-axis) against the number of transactions in the respective compact blocks. There exists a direct correlation between the number of additional transactions in and the sizes of the* `cmpctblock` *messages*

While it may appear from Fig. 7 that `cmpctblock` are smaller in size than the `grblk` messages, we emphasize that the former are only bounded from below by the straight line marked by crosses in Fig. 7. Sizes of `grblk` messages, on the other hand, appear to be bounded from above by a curve, marked by green circles, that appears to be asymptotically linear. As the number of transactions in blocks increase, the sizes of the initial `grblk` message tend to significantly deviate from the curve. Contrary to `cmpctblock` messages, the sizes of `grblk` messages do not depend on the number of additional transactions in the message. Upon examining the software implementation of the Graphene protocol in Bitcoin Unlimited, we find that `grblk` messages always contain only one additional transaction: the coinbase transaction. Therefore, we conjecture that the deviation from the curve is best explained by the size of the mempool sent by the `SRC` node to the `DST` node (see the discussion in Section II-B). This parameter sets the sizes of the Bloom filter and IBLT included in the `grblk` message, which in turn determines the overall size of the message. We leave further examination of this conjecture to future work.

Overall, the `grblk` message has an average size of 1.21

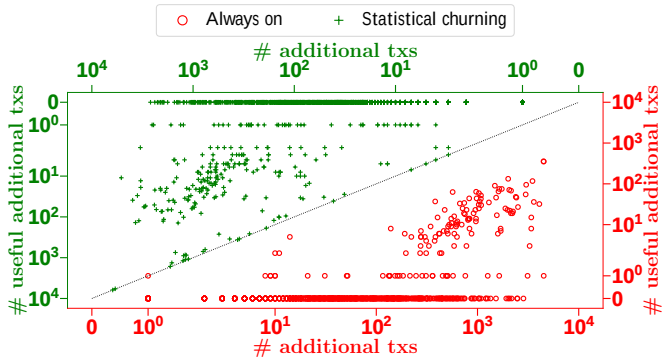


Fig. 9: *Number of useful additional transactions in* `cmpctblock` *messages against the total number of additional transactions in respective* `cmpctblock` *messages in (lower half) always on and (upper half) statistically churning nodes. The diagonal ($x = y$) represents the case when* 100% *of the additional transactions in* `cmpctblock` *messages are useful.*

10^4 bytes while the `cmpctblock` message has an average size of 6.24 10^4 bytes. Thus, the first message in Graphene is significantly smaller. Still, the `cmpctblock` message is much smaller than the `block` message of the normal block protocol, the latter having an average size of 5.93 10^5 bytes.

I. On the usefulness of additional transactions in the compact block relay protocol

Recall that the compact block protocol sends additional transactions as part of the `cmpctblock` messages. These are *full* transactions that the source node `SRC` predicts the receiving node `DST` may be missing from its mempool. In this section, we examine if these additional transactions are useful.

Denote by S_1 the set of additional transactions included in a block received by a node, and by S_2 the set of transactions contained in the mempool of the node when the block is received. The *number* of *useful* additional transactions, *i.e.,* transactions included in S_1 but not in S_2, is given by $| S_1 \setminus S_2 |$ where $\setminus$ denotes set subtraction.

The statistics on number of useful additional transactions in `cmpctblock` messages are shown in Fig. 9. The x-axes show the number of additional transactions in a `cmpctblock` message and the y-axes show the number of useful additional transactions. The straight $x = y$ line represents 100% useful additional transactions in the `cmpctblock` message. The figure is divided into two halves where the upper half represents statistics in the `statistical churn` regime and the lower half in the `always on` regime. *Note that both x-axes and y-axes are plotted on a log scale.*

We observe from the figure that churning nodes have *relatively* more instances of useful additional transactions compared to always on nodes. This is because the former are likely to miss more transactions from their mempool than the latter, since they were off the network. However, in both type of nodes, `cmpctblock` messages rarely have 100% useful additional transactions. In fact, in many cases, the *only* useful additional transaction in both churning and always connected nodes is the coinbase transaction, regardless of the number of additional transactions in the `cmpctblock` message. The data point clusters in the middle of the two halves of the figures show that in a few cases, some additional transactions other than the coinbase transactions are useful. However, even in those cases, the difference between the number of additional transactions and the number of useful additional transactions is usually high (up several orders of magnitude).

We next investigate whether useful additional transactions in `cmpctblock` messages save round-trip communication. That is, does the `DST` still miss transactions after receiving useful additional transactions included in a `cmpctblock` message?

Denote by M the set of transactions in a node's mempool when it receives a block, by T the set of missing transactions in the block, and by A the set of additional transactions in the `cmpctblock` message for that block. Then $X = T \cap M$ is the set of transactions in the block that are missing from the node's mempool, and $Y = A \setminus {}^1T \cap M^0$ is the set of additional transactions that help recover the missing transactions.

Our analysis shows that in both churning and always connected nodes, roughly 87% of `cmpctblock` messages contain only *one* additional transaction: the coinbase transaction that is always helpful. In the remaining `cmpctblock` messages, we exclude the coinbase transaction for further analysis. There are now three cases: (a) the additional transactions are not helpful at all, *i.e.,* $|Y| = 0$; (b) additional transactions are *partially* helpful, but not enough to recover all transactions in X, *i.e.,* $Y < X$ given $|Y| > 0$; or (c) additional transactions are *completely* helpful and recover all transactions in X, *i.e.,* $Y \quad X$. We find that there are no instances where the additional transactions are *completely* helpful to recover all transactions in X (*i.e.,* case (c)). On the other hand, additional transactions in roughly 84% of `cmpctblock` messages are *not* helpful *at all* (case (a)). In the remaining roughly 16% of `cmpctblock` messages, additional transactions are *partially* helpful (case (b)).

Our findings in this section show that though additional transactions (excluding the coinbase transaction) are sometimes helpful, in many instances they are either duplicates and thus end up wasting bandwidth, or not enough to completely recover transactions in blocks that may be missing in the node's mempool. Our calculations show that in the `always on` and `statistical churn` regimes, roughly 90% and 96%, respectively, of bandwidth consumed by additional transactions is unnecessary and wasted.

IV. Conclusion

We have methodically studied the empirical performance of three popular block relay protocols (Graphene, compact blocks, and the Bitcoin default) on a live blockchain, through the Bitcoin Unlimited (BU) client and in a variety of network regimes. Our experiments have identified regimes in which the different protocols excel.

For nodes that are `always on` or have `statistical churn`, the Graphene block relay protocol performed the best and the Bitcoin default block relay protocol performed the worst in terms of average block communication sizes

and propagation delays, with the compact block relay protocol performing in-between. More precisely, compared to Graphene, compact and default blocks have roughly 40% and 400% higher propagation delays, and over 80% and 1400% larger communication sizes. As a result, it seems *preferable to configure nodes with the Graphene* block relay protocol under *typical network conditions*.

For nodes that are periodically churning, our results were more nuanced. We found that the default block relay protocol performed worse than Graphene and compact blocks, in terms of propagation delay and communication sizes, regardless of the off duty cycle used in our experiments. Graphene generally performed better than compact blocks in nodes configured with a 25% off-duty cycle, and vice versa in a 75% off-duty cycle regime. More precisely, Graphene performance significantly degrades when the destination misses many transactions as this causes additional rounds of communication. As a result, *if nodes churn frequently* or are off the network for long periods of time, *it may be preferable to configure them with the compact block protocol*.

In summary, our work methodically demonstrates the benefit regime of the Graphene protocol, and suggest that it should be of interest to other blockchains, including BTC (*i.e.,* the main Bitcoin blockchain). We thus encourage integrating and evaluating Graphene within Bitcoin Core, the client software for BTC.

A. Inefficiencies and potential solutions

Our paper provided a detailed description of the implementation of the Graphene block relay protocol in Bitcoin Unlimited. We showed that there exist five scenarios for the completion of block relay between two nodes. In general, scenarios ¬ and - are the most common and complete within one and two round-trips, respectively. Yet, when nodes are churning, our fine-grained temporal analysis of Graphene identified an inefficiency. Specifically, consider a churning node that has just rejoined the network. Our analysis showed that for the first received block in the `periodic churn` regime, scenarios ¯ and/or ° occur more often than scenario Ⓡ in most considered cases. This suggests that the block failure recovery procedure in the protocol could be optimized to avoid unnecessary rounds of communication. For example, one could send transaction *hashes* when initiating failure recovery (rather than a Bloom filter) so that there is no chance of false positives. This should in principle remove the occurrence of scenarios ¯ and °. Indeed, the node would not need to decode IBLTs and probabilistically reconstruct the Graphene blocks while incurring an additional round-trip to recover transactions corresponding to the aforementioned hashes. The infrequent occurrence of the scenario suggests that the overhead of this approach should be negligible. Indeed, after the first three blocks are received, scenario Ⓡ occurs with a probability smaller than 10% in both the `statistical churn` and `periodic churn` regimes.

We also studied the practice of including additional transactions (*i.e.,* full transactions predicted by the `SRC` node to be missing from the mempool of the `DST` node) in the initial message of the compact block relay protocol. Our analysis shows that in our experimental nodes in both `always on` and `statistical churn` regimes, a large portion, *i.e.,* roughly 87%, of the `cmpctblock` messages only contain the coinbase transaction. Of the remaining portion of the `cmpctblock` messages, over 90% of the additional transactions result in wasted bandwidth. Our analysis shows that this respectively accounts on average for 10 MB and 34 MB of communication in nodes in the `always on` and `statistical churn` regimes, aggregated over the measurement period. As such, there may be benefits from limiting these additional transactions from the `cmpctblock` messages.

APPENDIX
EXTENDED BACKGROUND

A *Bloom filter* [63] is essentially a bit-vector, *i.e.,* an array A of size m where each element in the array can only be either 0 or 1. Given a set S of n objects, the Bloom filter allows testing objects s_i, where $i \in [1, n]$, for membership in S. The bit-vector A representing a bloom filter is first initialized such that all bits are set to 0, *i.e.,* $\forall j \in [1, m], A[j] = 0$. To represent S as a Bloom filter, each object $s_i \in S$ is hashed with predefined hash functions $h_1, h_2, \ldots, h_K$. The output of each hash function determines a location in the bit-vector, the bit corresponding to which is set to 1, *i.e.,* $\forall i \in [1, n], \forall k \in [1, K], A[h_k(s_i)] = 1$. The object r that is to be tested for membership in S is hashed against the same hash functions. r is *most likely* a member of S if *all* bits corresponding to the locations in A obtained by hashing it are set to 1, *i.e.,* when the result of

$$\bigcup_{k \in [1,K]} A[h_k(r)]$$

is 1. r is *definitely* not a member of S even if a single bit corresponding to the locations in A obtained by hashing it is set to 0, *i.e.,* when the result of

$$\bigcup_{k \in [1,K]} A[h_k(r)]$$

is 0.

The Bloom filter is a *probabilistic* data structure. Given a set of objects S and a Bloom filter A of size m, denote with $\mathbb{P}_{fp}$ the probability that a false positive and with $\mathbb{P}_{fn}$ the probability that a false negative occurs as a result of the test of membership of an object r in S. While a Bloom filter *does* allow false positives, *i.e.,* $\mathbb{P}_{fp} \neq 0$, false negatives can *never* occur, *i.e.,* $\mathbb{P}_{fn} = 0$. Further, $\mathbb{P}_{fp}$ is configurable and depends on the number of objects in the set, *i.e.,* $|S|$, the size of the Bloom filter, *i.e., m*, and the number of predefined hash functions, *i.e., K* [64], [65].

A Bloom filter, however, does not allow listing of all members of the set that it represents. This operation can be performed with an *inverted Bloom lookup table (IBLT)* [66], [67] which, in addition to the insertion and lookup operations, also supports listing members of a set. Given two sets of objects S_1 and S_2 where the contents of the sets are not identical, IBLTs can be used to efficiently find the symmetric

differences between the two sets. We represent the two sets S_1 and S_2 each with an IBLT I_{S_1} and I_{S_2}, respectively. The symmetric difference between the two sets, *i.e.,* $D_{S_1\ S_2}$, can then be found by first subtracting the IBLTs from one another using a process dubbed as the *peeling* process, *i.e.,* $I^0 = I_{S_1} \quad I_{S_2}$, and then finally decoding the difference I^0 [68]. The decoding process is *successful* if all symmetric differences between S_1 and S_2 are found. Otherwise, the decode process has *failed*. Similar to a Bloom filter, an IBLT is a probabilistic data structure, and there is a non-zero probability that the decode process will fail. When this happens, the process can be repeated by modifying the parameters that govern the size of the IBLTs.

References

[1] "Guggenheim funds trust post-effective amendment." https://sec.report/Document/0001628280-20-016852/. SEC filing, November 27, 2020.

[2] "Bitcoin to Come to America's Oldest Bank, BNY Mellon." https://www.wsj.com/articles/bitcoin-to-come-to-america-s-oldest-bank-bny-mellon-11613044810. Wall Street Journal, February 11, 2021.

[3] "Bitcoin looks to gain traction in payments." https://www.wsj.com/articles/bitcoin-looks-to-gain-traction-in-payments-11609237801. Wall Street Journal, December 31, 2020.

[4] "Tesla buys $1.5 billion in Bitcoin." https://www.wsj.com/articles/tesla-buys-1-5-billion-in-bitcoin-11612791688. Wall Street Journal, February 8, 2021.

[5] "Incoming New York mayor Eric Adams vows to take first three paychecks in Bitcoin." https://www.cnbc.com/2021/11/04/new-york-mayor-elect-eric-adams-to-take-first-3-paychecks-in-bitcoin.html.

[6] "Inside Afghanistan's cryptocurrency underground as the country plunges into turmoil." `https://www.cnbc.com/2021/08/21/bitcoin-afghanistan-cryptocurrency-taliban-capital-flight.html`, 2021. Online; Accessed: October 26, 2021.

[7] R. Chandran, "FEATURE-Salaries to remittances: Afghans embrace crypto amid financial chaos." `https://www.reuters.com/article/crypto-currency-afghanistan/feature-salaries-to-remittances-afghans-embrace-crypto-amid-financial-chaos-idUSL8N2QU39A`, Oct 2021. Online; Accessed: December 22, 2021.

[8] P. Zheng, Z. Zheng, X. Luo, X. Chen, and X. Liu, "A Detailed and Real-Time Performance Monitoring Framework for Blockchain Systems," in *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pp. 134–143, 2018.

[9] D. Mechkaroska, V. Dimitrova, and A. Popovska-Mitrovikj, "Analysis of the Possibilities for Improvement of BlockChain Technology," in *2018 26th Telecommunications Forum (TELFOR)*, pp. 1–4, 2018.

[10] M. Essaid, H. W. Kim, W. Guil Park, K. Y. Lee, S. Jin Park, and H. T. Ju, "Network Usage of Bitcoin Full Node," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, pp. 1286–1291, 2018.

[11] D. Perard, J. Lacan, Y. Bachy, and J. Detchart, "Erasure Code-Based Low Storage Blockchain Node," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pp. 1622–1627, 2018.

[12] T. Dryja, "Utreexo: A dynamic hash-based accumulator optimized for the Bitcoin UTXO set." Cryptology ePrint Archive, Report 2019/611, 2019. `https://ia.cr/2019/611`.

[13] M. Florian, S. Henningsen, S. Beaucamp, and B. Scheuermann, "Erasing Data from Blockchain Nodes," in *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS PW)*, pp. 367–376, 2019.

[14] F. Tschorsch and B. Scheuermann, "Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2084–2123, 2016.

[15] E. developers, "NODES AND CLIENTS." `https://ethereum.org/en/developers/docs/nodes-and-clients/#network-benefits`. Online; Accessed: Aug 18, 2021.

[16] Y. Wang, "A Blockchain System with Lightweight Full Node Based on Dew Computing," *Internet of Things*, vol. 11, p. 100184, 2020.

[17] 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.

[18] M. A. Imtiaz, D. Starobinski, A. Trachtenberg, and N. Younis, "Churn in the Bitcoin Network: Characterization and Impact," in *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 431–439, 2019.

[19] M. A. Imtiaz, D. Starobinski, A. Trachtenberg, and N. Younis, "Churn in the Bitcoin Network," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1598–1615, 2021.

[20] L. Kiffer, A. Salman, D. Levin, A. Mislove, and C. Nita-Rotaru, "Under the Hood of the Ethereum Gossip Protocol," in *Proceedings of the 2021 International Conference on Financial Cryptography and Data Security (FC'21)*, 2021.

[21] O. I. Oluwasuji, O. Malik, J. Zhang, and S. D. Ramchurn, "Solving the fair electric load shedding problem in developing countries," *Autonomous Agents and Multi-Agent Systems*, vol. 34, p. 12, Dec 2019.

[22] T. H. Meles, "Impact of power outages on households in developing countries: Evidence from Ethiopia," *Energy Economics*, vol. 91, p. 104882, 2020.

[23] P. Cramton, "Lessons from the 2021 Texas electricity crisis," *Utility Dive*, May 2021.

[24] J. Bialek, "What does the GB power outage on 9 August 2019 tell us about the current state of decarbonised power systems?," *Energy Policy*, vol. 146, p. 111821, 2020.

[25] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system." `https://bitcoin.org/bitcoin.pdf`, 2008. Online; Accessed: Nov 27, 2021.

[26] M. Corallo, "Compact Block Relay." `https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki`, 2016. Online; Accessed: May 24, 2021.

[27] "Transaction Size." `https://bitcoinvisuals.com/chain-tx-size`. Online; Accessed: Oct 25, 2021.

[28] "Mempool Summary." `https://explorer.bitcoinunlimited.info/mempool-summary`. Online; Accessed: Oct 25, 2021.

[29] A. Pinar 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* (J. Garcia-Alfaro, G. Navarro-Arribas, H. Hartenstein, and J. Herrera-Joancomartí, eds.), (Cham), pp. 420–428, Springer International Publishing, 2017.

[30] A. P. Ozisik, G. Andresen, B. N. Levine, D. Tapp, G. Bissias, and S. Katkuri, "Graphene: Efficient Interactive Set Reconciliation Applied to Blockchain Propagation," in *Proceedings of the ACM Special Interest Group on Data Communication*, SIGCOMM '19, (New York, NY, USA), p. 303–317, Association for Computing Machinery, 2019.

[31] C. Spearman, "The Proof and Measurement of Association between Two Things," *The American Journal of Psychology*, vol. 100, no. 3/4, pp. 441–471, 1987.

[32] "THE HISTORY OF BITCOIN CASH." `https://bitcoincash.org/`, 2021. Online; Accessed: October 20, 2021.

[33] "Release Notes for Bitcoin Unlimited Cash Edition 1.1.0.0." `https://github.com/BitcoinUnlimited/BitcoinUnlimited/blob/release/doc/release-notes/release-notes-bucash1.1.0.0.md`, 2017. Online; Accessed: October 20, 2021.

[34] C. Decker, *On the Scalability and Security of Bitcoin*. PhD thesis, ETH Zurich, Zurich, 2016.

[35] B. Magazine, "What is the block size limit?." `https://bitcoinmagazine.com/guides/what-is-the-bitcoin-block-size-limit`, 2020. Online; Accessed: April 7, 2021.

[36] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer, "On Scaling Decentralized Blockchains," in *Financial Cryptography and Data Security* (J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, eds.), (Berlin, Heidelberg), pp. 106–125, Springer Berlin Heidelberg, 2016.

[37] Blockchain.com, "Confirmed transactions per day." https://www.blockchain.com/charts/n-transactions. Online; Accessed: April 7, 2021.

[38] B. Unlimited, "Bitcoin Unlimited FAQ." https://www.bitcoinunlimited.info/faq/what-is-bu. Online; Accessed: April 7, 2021.

[39] Blockchain.com, "Average transactions per block." https://www.blockchain.com/charts/n-transactions-per-block. Online; Accessed: April 7, 2021.

[40] B. C. Explorer, "Block stats." https://explorer.bitcoinunlimited.info/block-stats. Online; Accessed: April 7, 2021.

[41] "Compact Blocks FAQ." https://bitcoincore.org/en/2016/06/07/compact-blocks-faq/. Online; Accessed: April 24, 2022.

[42] A. Suisani, "Bitcoin Unlimited - Bitcoin Cash release 1.6.0.0." https://github.com/BitcoinUnlimited/BitcoinUnlimited/releases/tag/bucash1.6.0.0, 2019. Online; Accessed: April 9, 2021.

[43] "net_processing.cpp." https://gitlab.com/bitcoinunlimited/BCHUnlimited/-/blob/BCHunlimited1.9.0.1/src/net_processing.cpp#L399-408. Online; Accessed: April 9, 2021.

[44] 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)*, pp. 1–9, 2020.

[45] M. A. Imtiaz, D. Starobinski, and A. Trachtenberg, "Investigating Orphan Transactions in the Bitcoin Network," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1718–1731, 2021.

[46] G. Bissias, "graphene-specification-v2.2.mediawiki." https://gitlab.com/bitcoinunlimited/BCHUnlimited/-/blob/dev/doc/graphene-specification-v2.2.mediawiki, 2019. Online; Accessed: April 10, 2021.

[47] T. Neudecker, "Characterization of the Bitcoin Peer-to-Peer Network (2015-2018)." http://dsn.tm.kit.edu/bitcoin/publications/bitcoin_network_characterization.pdf, 2019.

[48] H. Kalodner, M. Möser, K. Lee, S. Goldfeder, M. Plattner, A. Chator, and A. Narayanan, "BlockSci: Design and applications of a blockchain analysis platform," in *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2721–2738, USENIX Association, Aug. 2020.

[49] E. Rohrer and F. Tschorsch, "Kadcast: A Structured Approach to Broadcast in Blockchain Networks," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, AFT '19, (New York, NY, USA), p. 199–213, Association for Computing Machinery, 2019.

[50] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," in *IEEE P2P 2013 Proceedings*, pp. 1–10, 2013.

[51] 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*, CCS '19, (New York, NY, USA), p. 817–831, Association for Computing Machinery, 2019.

[52] M. Zhang, Y. Cheng, X. Deng, B. Wang, J. Xie, Y. Yang, and J. Zhang, "Accelerating transactions relay in blockchain networks via reputation," in *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pp. 1–10, 2021.

[53] naumenkogs, "Erlay: bandwidth-efficient transaction relay protocol." https://github.com/bitcoin/bitcoin/pull/21515. Online; Accessed: December 29, 2021.

[54] naumenkogs, "p2p: Erlay support signaling." https://github.com/bitcoin/bitcoin/pull/23443. Online; Accessed: December 29, 2021.

[55] M. A. Imtiaz, *Measuring and Improving the Performance of the Bitcoin Network*. PhD thesis, Boston University, Boston MA, USA, 2022. https://open.bu.edu/handle/2144/43707.

[56] M. A. Imtiaz, "bitcoin-releases." https://github.com/nislab/bitcoin-releases/tree/wip, 2021.

[57] an4s, "Unreachable code when checking for valid Merkle root in compact block processing." https://gitlab.com/bitcoinunlimited/BCHUnlimited/-/issues/2226. Online; Accessed: May 21, 2021.

[58] M. A. Imtiaz, "bitcoin-logs." https://github.com/nislab/bitcoin-logs/tree/wip, 2021.

[59] D. Zwillinger, *CRC standard probability and statistics tables and formulae*. Boca Raton: Chapman & Hall/CRC, 2000.

[60] "scipy.stats.spearmanr." https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.spearmanr.html. Online; Accessed: Oct 11, 2021.

[61] F. Wilcoxon, *Individual Comparisons by Ranking Methods*, pp. 196–202. New York, NY: Springer New York, 1992.

[62] W. W. LaMorte, "PH717 Module 9 - Correlation and Regression: Evaluating Association Between Two Continuous Variables." https://sphweb.bumc.bu.edu/otlt/MPH-Modules/PH717-QuantCore/PH717-Module9-Correlation-Regression/PH717-Module9-Correlation-Regression4.html. Online; Accessed: May 14, 2021.

[63] B. H. Bloom, "Space/Time Trade-Offs in Hash Coding with Allowable Errors," *Commun. ACM*, vol. 13, p. 422–426, July 1970.

[64] Li Fan, Pei Cao, J. Almeida, and A. Z. Broder, "Summary cache: a scalable wide-area Web cache sharing protocol," *IEEE/ACM Transactions on Networking*, vol. 8, no. 3, pp. 281–293, 2000.

[65] S. Agarwal and A. Trachtenberg, "Approximating the number of differences between remote sets," in *2006 IEEE Information Theory Workshop - ITW '06 Punta del Este*, pp. 217–221, 2006.

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

[67] D. Eppstein and M. T. Goodrich, "Straggler Identification in Round-Trip Data Streams via Newton's Identities and Invertible Bloom Filters," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 2, pp. 297–306, 2011.

[68] D. Eppstein, M. T. Goodrich, F. Uyeda, and G. Varghese, "What's the Difference? Efficient Set Reconciliation without Prior Context," in *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, (New York, NY, USA), p. 218–229, Association for Computing Machinery, 2011.

Muhammad Anas Imtiaz works as a Senior Software Engineer in the Financial Analytics department at Bloomberg LP. He received his Ph.D. and MS in Computer Engineering from Boston University in 2022. His thesis investigated the presence and effects of churn in the Bitcoin network, and possible improvements in the Bitcoin protocol to help mitigate such issues as well as empirical comparisons between the performance of block relay protocols in Bitcoin Unlimited network. 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 in 2020 and at IEEE TNSM and IEEE Access in 2022, as a TPC member for the 3rd International Congress on Blockchain and Applications in 2021, and as a Shadow PC at the ACM IMC 2019 conference. Anas has previously 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 in 2002, an Early Career Principal Investigator (ECPI) award from the U.S. Department of Energy in 2004, and BU ECE Faculty Teaching Awards in 2010 and 2020. He co-authored papers that received the best paper awards at the WiOpt Symposium in 2010, the IEEE Conference on Communications and Network Security (CNS) in 2016, and the IEEE International Conference on Blockchain and Cryptocurrency (ICBC) in 2020. He has served on the Editorial Boards of the IEEE Transactions on Information Forensics and Security, the IEEE/ACM Transactions on Networking, and the IEEE Open Journal of the Communications Society. His research interests are in cybersecurity, wireless networking, blockchain and cryptocurrency, 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 with Red Hat, TripAdvisor, MIT Lincoln Lab, HP Labs, and the Johns Hopkins Center for Talented Youth. He has been awarded an Institute for Health System Innovation & Policy fellowship (2020), 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 (side-channels, page cache), Distributed systems (data reconciliation, cryptocurrencies, automated exposure notification) and theory (hashing, graph algorithms, rateless codes).