

Carbyne: An Ultra-Lightweight DoS-Resilient Mempool for Bitcoin

Hina Binte Haq^a, Syed Taha Ali^a, Asad Salman^b, Patrick McCorry^c and Siamak F. Shahandashti^{d,*}

^aNational University of Sciences and Technology (NUST), Islamabad, Pakistan

^bX (formerly Twitter), USA

^cArbitrum, London, United Kingdom

^dUniversity of York, York, United Kingdom

ARTICLE INFO

Keywords:

Cryptocurrency

Mempool

Optimization

Denial-of-service attack

Bitcoin

ABSTRACT

The Bitcoin mempool plays an integral role in transaction processing and propagation through the network. Frequent transaction congestion events, as well as spam and dust attacks can clog the mempool, leading to dropped transactions, processing delays, and increased transaction fees. Moreover, increasing transaction loads on the network result in higher resource costs to operate full nodes, thereby restricting Bitcoin's network footprint and negatively impacting its overall health and performance. In this paper, we present Carbyne, a novel mempool optimization scheme, which uses counting bloom filter constructions to adapt to increased transaction flows, thereby making nodes resilient to congestion and spam and dust attacks. We implement Carbyne in C++ and benchmark its performance using a novel data set of Bitcoin mempool activity over a 90-day period. We dramatically reduced the mempool's memory consumption by up to two orders of magnitude (from 300 MB to 3 MB) while verifying and forwarding transactions with 99.9% fidelity and a slight increase in computational load. We simulate extensive spam attacks on Carbyne and demonstrate that mempool loads of 1 GB can be accommodated in as little as 10 MB. Carbyne does not necessitate a hard fork, it will help deploy high-functioning nodes on resource-constrained platforms, and it may also be adapted to other cryptocurrencies.

1. Introduction

Bitcoin and cryptocurrencies are achieving mainstream success. Bitcoin has a market capitalization of over \$806 billion, and the top 5 cryptocurrencies, collectively account for a valuation over \$ 1.4 trillion [25]. Bitcoin exceeded 130 million users in 2021 amid predictions it will grow to a billion users in the next four years [11]. Some 46 million Americans, 17% of the adult population, now own Bitcoins [98], and 36% of small-medium businesses accept it [30].

This popularity has motivated considerable research on the scalability of Bitcoin, to improve Bitcoin's transaction throughput (e.g. [101] [99]) and reduce its growing memory requirements (e.g. [75] [100]). Interestingly, these problems may also be interpreted as security challenges that typically manifest as degradation or denial of service attacks. However, this particular research domain has received little attention from a security perspective. Is it possible to reconcile these twin aims, scalability and security, to develop solutions to help Bitcoin grow in ways that also harden the Bitcoin network in terms of security and robustness?

In this paper, we consider the motivating example of growing transaction loads and network congestion. Bitcoin nodes use local memory (RAM) to store state information, primarily the unspent transaction output (UTXO) set and the Bitcoin transactions memory pool (mempool). The mempool plays an integral role in transaction processing: it logs unconfirmed transactions and assists in their propagation

through the network. Storing these transactions in RAM as opposed to disk is efficient, as disk latency slows transaction processing by up to two orders of magnitude [75].

The average number of transactions in the Bitcoin mempool, has more than doubled in 2021 [51]. Transaction congestion frequently results in significant increase in local RAM utilization which may negatively impact the overall performance of the network. If the queue of unconfirmed transactions exceeds a node's allocated mempool size limit, the node starts to drop or ignore transactions. This in turn causes a degradation in service in terms of prolonged processing delays and increased transaction fees [102] [109] [103]. For instance, a large spike in transaction volumes in October, 2020, led to some 145k transactions being stalled and median transaction fees reaching a three-year peak of \$11.66 [24]. There are frequent reports of mempool congestion on the Ethereum network, due in part to the popularity of decentralized finance applications [107]. Surge in transaction loads have led to extended network outages on Solana [67].

The mempool's susceptibility to congestion is also a highly potent attack vector which parties exploit by deliberately flooding the network with very low-cost transactions to clog the mempool. These activities, variously referred to as *stress tests*, *spam campaigns*, and *dust attacks*, may be considered effective Denial of Service (DoS) attacks. One spam campaign in October, 2015 grew the mempool to nearly 1 GB, with over 88,000 transactions, and reportedly caused 10% of nodes in the Bitcoin network to crash [74].

Moreover, sophisticated attacks have emerged which directly exploit mempool dynamics for profit. For instance, in March, 2020, when large sell-offs in crypto markets triggered collateral auctions on the Ethereum-based MakerDAO

A preprint of this paper is available on arXiv.

*Corresponding author

 siamak.shahandashti@york.ac.uk (S.F. Shahandashti)

ORCID(s):

platform, unknown actors flooded the network with low-cost transactions [108]. Many bidders could not adapt to the resulting spike in gas prices in time, thereby enabling the attackers to sweep the auctions with \$0 bids, and pocket \$8.3 million. In 2022, the Solana network was flooded by “complex-compound-instruction transactions”, preventing certain users from updating collateral positions in time, suffering liquidation, and abandoning Solana in protest. [66]

In response, researchers have devised mempool eviction strategies which screen incoming transactions for various features, including size, age, and fee, to identify and drop potential dust transactions [36] [42]. However, this approach has significant shortcomings: first, the solutions developed thus far have significant false positive rates, resulting in legitimate transactions being classified as spam, which some argue may constitute a DoS attack in its own right [74]. Second, attackers can discover filter heuristics and potentially craft spam transactions to evade them.

We propose Carbyne, a novel and flexible Bitcoin mempool optimization scheme to provide resilience against escalating transaction loads and spam and dust attacks. Carbyne uses space-efficient probabilistic data structures to store fingerprints of unconfirmed transactions. This reduces the mempool to a fraction of its size in RAM, while still verifying and forwarding incoming transactions with high fidelity.

Our key insight is that the two prime functions of the mempool, *transaction forwarding* and *transaction inventory*, can be dissociated, to prioritize one function over the other. A similar philosophy has commonly been employed to build lightweight clients, such as pruned nodes, simplified payment verification (SPV), EPBC [106], FlyClient [58], etc.

Moreover, Carbyne users can still choose to maintain a subset of transactions in RAM to mine blocks or to bootstrap new nodes on the network. Decoupling forwarding and inventory functions gives users the flexibility to control RAM usage and still contribute to the network in an accurate and efficient manner as per the resources available to them.

Thus, Carbyne also helps address the related but overlooked problem concerning Bitcoin’s growing local memory consumption. A sustained increase in transaction loads results in higher resource costs for users to operate Bitcoin nodes and, unlike miners, these parties are not incentivized for their contribution. This in turn restricts the footprint of the Bitcoin network, and may render it vulnerable to certain attacks. Our solution can help develop lightweight clients and alleviate these growing costs. Thus, Carbyne is not meant to supplant full nodes - but to lower entry costs for certain parties. Our specific contributions are:

- *We describe Carbyne, an optimized DoS-resilient mempool construction for Bitcoin* which uses multiple counting bloom filters to replicate the mempool’s core inventory functions. Carbyne reduces the mempool space requirements in local memory by up to two orders of magnitude while still processing unconfirmed transactions with very high fidelity.

- *We implement Carbyne in C++ and benchmark its performance* on a novel dataset of Bitcoin mempool activity logged over a 90-day period, comprising approximately 29 million distinct transactions – an independent contribution and useful for various applications apart from Carbyne (see Appendix A). Researchers can use this dataset and the accompanying scripts to efficiently reconstruct Bitcoin mempool state. We define various metrics to undertake a fine-grained comparison of Carbyne versus the Bitcoin Core mempool. We also provide multiple parameters to fine-tune the trade-off between memory usage, fidelity, and robustness when deploying Carbyne.
- *We simulate extensive congestion and spam attacks* to demonstrate that Carbyne easily copes with high transactions loads, up to a threefold increase over the maximum flows ever witnessed in the Bitcoin network. We further propose mechanisms to dynamically adapt to rising transaction rates with modest computational and memory overheads.

We achieve a dramatic reduction in memory consumption: we are able to fit approximately 300 MB worth of Bitcoin transactions in 3 MB while processing them with 99.915% accuracy. If 12 MB are allocated for Carbyne, accuracy increases to 99.997%. In our simulated congestion experiments, 600,000 transactions, which would ordinarily require over 1 GB, can be handled in under 10 MB.

Carbyne has some limitations: it does not resolve block congestion issues or directly reduce transaction fees. Carbyne does not detect or evict spam but it can easily be integrated with spam filtering mechanisms [36] [42]. The use of probabilistic data structures results in a minute false positive rate in the form of dropped transactions, which are significantly smaller than that reported in related work.

There are other considerable benefits: Carbyne can be deployed without necessitating a fork. Carbyne enables resource constrained systems (e.g. Raspberry Pi, smartphones [105]) to run high-performing functional mempools which validate and propagate incoming transactions, and can scale to handle congestion and spam attacks. Moreover, this application of bloom filters may hold value for other cryptocurrencies, such as Litecoin, Ethereum, Ripple, and Solana.

Our solution is orthogonal to other space-efficient optimizations for Bitcoin, including pruning, Segwit, and UTXO optimizations, and may therefore be combined with these strategies to aggregate their individual benefits.

We believe we are the first to rethink the structure of the mempool to combat congestion and spam attacks. Carbyne aligns with other ongoing attempts to harden the overall security and resilience of the Bitcoin network (such as MIT’s Bitcoin Software and Security Effort [104]). Our work also contributes to local memory optimization, which is a neglected domain in the research literature.

The paper is organized as follows: In §2, we present essential background material. We describe Carbyne in §4 and present our dataset in §6. This is followed by experimental

results, discussion and security analysis in §7. In §9, we simulate congestion and spam attacks. We conclude in §10.

2. Background

In this section we overview spam attacks, discuss prior work and present our threat model. We also describe the mempool and its structure.

2.1. Spam, Stress Tests, and Dust Attacks

The first major campaign, advertised as a “*stress test*”, occurred in July, 2015, with a daily average of 150,000 pending transactions. Another campaign in October, 2015 increased mempool size to nearly 1 GB, with more than 88,000 pending transactions. This incident reportedly knocked up to 10% of Bitcoin nodes offline, many of which were running on memory-constrained platforms like Raspberry Pi [74].

Numerous congestion and spam incidents have occurred since. In November, 2017, with the cancellation of the SegWit2x hard fork, many users started selling their coins, causing the mempool size to exceed 182,000 transactions and 125 MB in raw transaction size [102]. In December 2017, a DoS attack on Bitfinex inflated the mempool over 80,000 pending transactions and 135 MB in raw transaction size [109]. In November 2019 the mempool spiked to over 115,000 pending transactions and 90 MB in raw transaction size when Binance moved its coins in bulk [103]. Fig. 1 shows these spikes in transaction rates and fees.

Other sources of spam include mixing services that use dust as a promotional tool [110]. Dust attacks have also been used as a tactic to compromise privacy [74]. Games like Satoshi Dice also generate large numbers of low-value transactions that some categorize as ‘dust-like’ [63].

Other cryptocurrencies with a mempool-based architecture are also vulnerable to congestion and spam attacks. In 2019, over 200,000 Litecoin wallets received spam transactions as part of a publicity stunt by a mining pool [113]. The attack on the Ethereum-based MakerDAO platform specifically focused on spiking transaction fees so that bidders using automated scripts could not adapt in time to compete in collateral auctions [108]. In September 2021 the Solana network stalled for 17 hours as bots generated 400,000 transactions per second, overflowing the transaction processing queue, and causing some validator nodes to crash [119].

Our solution, in principle extends to other cryptocurrencies, including Litecoin, Ethereum, and Solana. We discuss these applications in §8.

2.2. Prior Work

Carbyne contributes to the literature on improving the security, resilience, and health of the Bitcoin network. It also aligns with research on scalability that improve Bitcoin’s transaction throughput (e.g. [120][101][17][35]) and ongoing efforts to develop lightweight clients (e.g. [41][58][78][121]).

The mempool has been largely overlooked in the research literature. Baqer et al. conducted a post-attack analysis of one of the first spam attacks on Bitcoin in 2015 [74]. They use clustering to classify approximately 23% of the

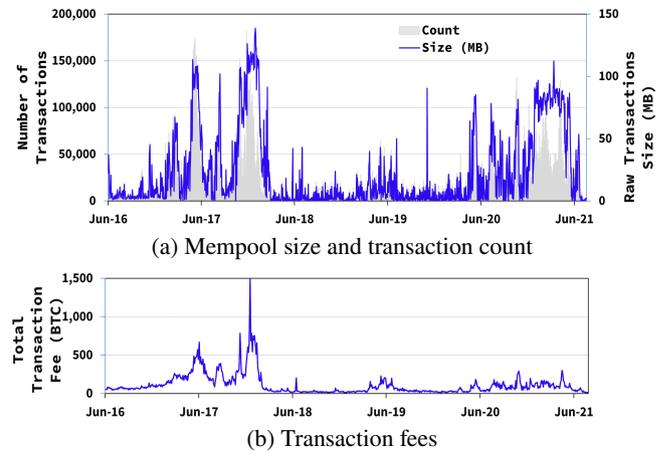


Figure 1: Bitcoin mempool & transaction fee trends [52]

observed transactions as spam. They explicitly identify the mempool as a vulnerable component and recommend evicting transactions to “relieve the pressure”. The authors suggest Bitcoin implements a $\text{mint} \times \text{fee}$ per output similar to Litecoin or a dynamic fees model to counter spam.

However, the authors also caution that filtering strategies can misclassify legitimate transactions as spam, and even a small 1–2% false positive rate amounts to a DoS attack in its own right. Furthermore, attackers can gain knowledge of filtering parameters and craft transactions to evade them.

Transaction eviction is the primary theme in papers that follow. Saad et al propose *Contra*, a solution which considers various metrics to identify and evict spam [36]. These consist of defining minimum thresholds for transaction age and fees, beyond which transactions are classified as dust and discarded. There is a tradeoff here: increasing the threshold values can result in false positives (when legitimate transactions are classified as spam) and decreasing it can yield false negatives (when spam transactions are categorized as legitimate). The authors undertake a modelling exercise and predict a maximum accuracy of 60% in identifying spam. This model has not yet been validated on real-world data.

The authors make two more noteworthy contributions: they describe how dust attacks may be extended by creating new dust transactions descended from earlier ones. They also recommend increasing block size dynamically during dust attacks to increase the probability that dust transactions are mined and thereby escalating the cost of the attack.

Wang et al. analyse Bitcoin transactions from 2009–2017 to construct *Anti-dust*, a classification model based on the Gaussian distribution [42]. Similar to *Contra*, the authors set a threshold for the amount of Bitcoins being transacted and any value below the threshold is forwarded to a dust transaction pool, making the mechanism susceptible to false positives and false negatives. If the mempool is not full transactions from the dust pool are moved into it. If the dust transaction pool becomes full, dust transactions are discarded. The authors undertake simulations where they generate 500 legitimate transactions and mix in different

amounts of dust transactions. When only genuine transactions are present, transaction validation time is estimated at 200 seconds which increases to 25,000 seconds in the presence of dust, and reduces to 215 seconds when Anti-dust is deployed.

The problems with this filtering approach identified earlier still persist: there is as yet no rigorous definition of spam or dust transactions. False positive rates using these methods are significantly higher than the 1–2% threshold Baqer et al. discussed. If multiple nodes were to deploy these filters, they may function as inadvertent blacklists. Attackers can modify spam transactions to evade filters. As yet, there is no dynamic filtering solution which adapts to network conditions in real-time to identify and filter spam. Implementing real-time filters and separate pools for spam transactions, will incur computation costs and increased local memory consumption which remain to be evaluated. Moreover, solutions such as adapting a fee-per-output policy or dynamic block sizes require hard forks which can be a contentious process.

In a broader sense, moreover, these solutions are spam-centric. They do not address the general problem of increasing transaction flows and mempool congestion in the network. Our work differs in that our primary focus is local memory consumption. We do not identify or filter spam. Instead, we make the mempool resilient to larger traffic flows. Carbyne is orthogonal to filtering and eviction strategies, and may easily be integrated with these if needed.

Previously, Neonpool [2] introduced a novel transaction pool design based on Bloom filter variants, offering a scalable solution for light clients. It enables resource-constrained devices, such as smartphones, browsers, and IoT platforms, to perform full-node functions effectively. In Carbyne, we extend this work by conducting a detailed study on Bitcoin, utilizing counting Bloom filters with datasets three times larger than those used previously. This analysis includes an in-depth examination of mempool architecture and an exploration of the trade-offs among error rates, memory utilization, reprocessing costs, security, and computational overhead. We also make our scheme resilient to DDoS attacks.

2.3. Threat Model

In our scenario, we assume users Alice, Bob, and others operate full nodes connected to the Bitcoin network. Their goal is to contribute to the health and footprint of the network by validating and circulating Bitcoin transactions. We assume the network encounters increasing transaction flows and is also prone to frequent congestion events and DoS attacks. Previous analyses mainly considered DoS attacks on mining pools [122], Bitcoin exchanges [123] or the blockchain [74]. In our case, our attacker Eve deliberately targets the mempool.

Eve's strategy is to flood the network with transactions and clog the mempool of individual nodes. Her potential goals include delaying transaction processing, increasing fees or negatively impacting user experience. Goals also

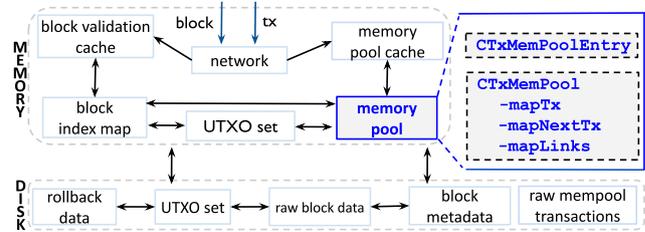


Figure 2: Memory and Disk Schematic

include publicity stunts or controversy (‘griefing’ the network). Eve may also use spam to cover for double-spends or sophisticated attacks like the MakerDAO attack detailed earlier.

We assume Eve can generate dust transactions at a rate higher than the throughput of the Bitcoin network and of considerable cumulative size, thereby creating a large backlog of unconfirmed transactions in the mempool. She may accomplish this using sybil accounts and automated scripts. However, Eve does not have any control over Bitcoin users or the ability to directly modify the functioning of their nodes.

Carbyne fundamentally rearchitects the mempool by storing transaction fingerprints instead of complete transactions. We are able to process large transaction flows with a dramatically reduced memory footprint (up to two orders of magnitude less) and very low false positive rates ($< 0.5\%$).

To clarify, Carbyne does not filter transactions or prevent spam and dust attacks. It does not resolve transaction congestion in blocks or directly affect transaction fees. Carbyne does not protect against attacks leveraging spam to compromise user privacy [124]. However, since Carbyne optimizes mempool storage, it effectively copes with congestion and provides strong resilience against dust attacks. Our experiments, using real-world data, indicate that Carbyne can handle up to three times the maximum traffic loads documented on the Bitcoin network in just 10 MB.

Carbyne uses probabilistic data structures which yield false positives or false negatives, resulting in a minute amount of transactions being dropped or reprocessed respectively. However, the effect on transaction propagation in the network is not significant: each node initializes its filters independently using random seeds, which makes it highly unlikely that large numbers of nodes drop the same transactions. With a false positive probability of p , the probability that the same transaction is categorized as a false positive by n nodes is p^n . As p is less than 10^{-3} , the probability of more than two nodes categorizing it as a false positive is infinitesimally small. The same argument applies to false negatives.

2.4. Bitcoin Mempool and Structure

We briefly describe the Bitcoin mempool and its structure. The mempool is an in-memory repository for unconfirmed transactions and is essential to Bitcoin’s transaction propagation mechanism. When a node receives a valid new transaction, it stores it in the mempool while it is being

processed by the network. Nodes use this real-time index of transactions to ensure they broadcast incoming transactions over the network only when first received. This minimizes traffic overhead and prevents infinite loops in the network.

The size of the mempool depends on the number of transactions it contains and their individual size, as determined by the transaction content, including the number of inputs and outputs and the length of the scripts. Fig. 1a shows raw transaction data size. Raw transactions of almost 100 MB take up to 300 Mb of memory [90]. Bitcoin's mempool size has varied dramatically over the years, ranging from the order of a few megabytes to over several hundred megabytes, and potentially impacting live transaction fees (Fig. 1b).

Fig. 2 depicts node storage components distributed over hard disk and RAM. Major components in RAM are the *mempool* and the *UTXO set* which stores unspent transaction outputs to validate incoming transactions. Other components include *memory pool cache* which temporarily stores new transactions pending validation before being accepted to the mempool; *block validation cache* which stores signature verification results of unconfirmed transactions to avoid repeating the computationally expensive verification operations when blocks arrive; *block index map* which optimizes locating specific block data on disk; and *network connections* data about peers, threads, etc.

Major components of disk storage are *raw block data* (blocks/blk*.dat) i.e. block data that make up the blockchain; *block metadata* (blocks/index/*) which optimize the process of locating specific block data; and *rollback data* (blocks/rev*.dat) which caters to block reorganization events. These latter databases are redundant as they can be reconstituted from the *raw block data*, but they are maintained on disk to speed up and optimize block validation and other operations [125].

Some local memory storage components are also maintained on disk: the entire *UTXO set* (chainstate/*) (over 79 million inputs, 4 GB in size) stored on disk and partially mirrored in RAM [100]. Similarly, the mempool is maintained in RAM, but since Bitcoin Core version 0.14.0, *raw mempool transactions* are saved to disk at node shutdown, so the Bitcoin mempool state persists across restarts [116].

Storage within the mempool is governed by two main classes: 1. `CTxMemPoolEntry` has a bookkeeping role and stores transactions data including size, fee, fee delta, entry height, coinbase status, and information about ancestor and descendant transactions [86]; 2. `CTxMemPool` is particularly relevant to our study and comprises 3 main structures: `mapTx` (`boost::multi_index`) [7] sorts the mempool on five criteria: transaction hash, witness-transaction hash, descendant and ancestor fee rate, and time; `mapNextTx` (`std::map`) [86] tracks the transaction inputs; and `mapLinks` (`std::map`) [86] tracks in-mempool ancestor and descendant transactions.

`CTxMemPool` and `CTxMemPoolEntry` introduce upto 3x memory overhead in terms of pointers, indexes, and metadata [90]. The minimum recommended storage to set up a Bitcoin full-node is 2 GB RAM and 350 GB hard disk

space [114]. The mempool is allocated 300 MB by default [132]. Users can define a custom mempool acceptance policy. In a low memory environment, it can be reduced (`-maxmempool`) or disabled entirely (`-blocksonly`).

3. Building Blocks: Counting Bloom Filters

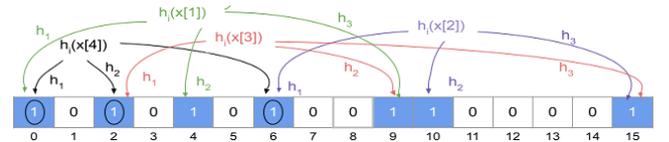
Bloom filters are memory-efficient probabilistic data structures used to test for set membership [22]. A Bloom filter is essentially a bit vector v , of a predefined size of m bits, initially set to zero. To insert an element x , belonging to a set $S = \{x[1], x[2], \dots, x[n]\}$ of n elements, we hash each x using k independent hash functions with uniformly distributed outputs over the range $\{1, 2, \dots, m\}$, to get $h_1(x)$, $h_2(x)$, , $h_k(x)$, and set the corresponding indices in the bit vector to 1, i.e. $v[h_1(x)] = v[h_2(x)] = \dots = v[h_k(x)] = 1$. In Fig. 3a we insert three elements $x[1]$, $x[2]$, $x[3]$ in the filter, which map to indices $\{2, 9, 15\}$, $\{6, 10, 15\}$ & $\{0, 4, 9\}$.

To query for set membership, an element y is hashed k times and the corresponding indexes are then checked. If any of the indexes is 0, we can be certain that y is not in the set, a *true negative*. If all relevant indexes are set to 1, y may be in the set. We illustrate this with two examples: If we query the filter regarding $x[1]$ which maps to indices $\{2, 9, 15\}$, the element was in the set and the bloom filter reported it so, referred to as a *true positive*. If we query the filter regarding $x[4]$, which maps to indexes $\{0, 2, 6\}$, we observe that, even though $x[4]$ was not inserted into the filter, the combination of earlier insertions has set those same bits to 1, resulting in a *false positive*. The probability of false positives, p , or *false positive rate (FPR)* [21] is:

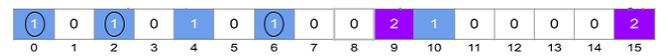
$$p \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \quad (1)$$

FPR highlights the trade off between space and accuracy. Filter size m can be provisioned as per set size n :

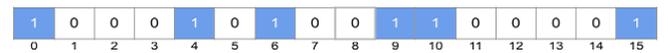
$$m \approx n \cdot \frac{-\ln(p)}{(\ln(2))^2} \quad (2)$$



(a) Insertion and queries in a bloom filter



(b) Insertion and queries in a counting bloom filter



(c) Deletion in a counting bloom filter

Figure 3: Bloom filters example ($m = 16, k = 3, n = 3$)

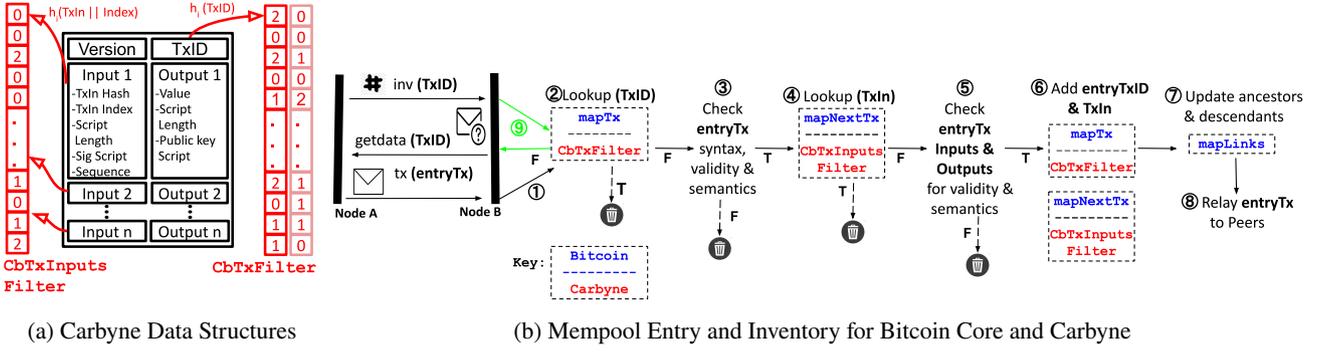


Figure 4: Entry and Inventory Processes

Optimum value of number of hash functions, k is given by:

$$k \approx \frac{m}{n} \cdot \ln(2) \quad (3)$$

Counting bloom filters extend bloom filter indices from a single bit to a multi-bit counter (or *bucket*), enabling delete operations. Insert and delete operations increment and decrement the corresponding counters by 1, respectively. For instance, in Fig. 3b, the counting bloom filter contains elements $x[1], x[2], x[3]$. When $x[3]$ is deleted, index $\{2, 9, 15\}$ are decremented to $\{0, 1, 1\}$, as shown in Fig. 3c.

Counting bloom filters also produce *false negatives*. For instance in Fig. 3c, a query for $x[4]$ will result in a false positive. Deleting $x[4]$ will result in indexes $\{0, 2, 6\}$ being decremented. Hence, a subsequent query for $x[1]$ or $x[2]$ will result in a *false negative*. Thus incorrect deletion of the false positive items lead to false negatives. The occurrence of false negatives also increases the false positive rate with time as an item is not deleted, although it should be [46].

4. Proposed Solution: Carbyne

We now describe the design of Carbyne. Carbyne stores fingerprints of transactions instead of retaining full transactions, whilst preserving essential transaction verification and forwarding functionality.

Counting bloom filters are commonly used to cache unbounded real-time data streams (for a primer on counting bloom filters, see §3). However, we face considerable additional challenges as we seek to preserve key functions of the mempool: devising a mechanism to expire transactions based on age; devising a mechanism for the bulk expiry of transaction inputs; tracking and preventing double-spends; and ensuring resilience against DoS attacks.

To address these requirements, we propose a novel design that comprises two counting bloom filter constructions, as shown in Fig. 4a: the first filter, $CbTxFilter$ maps valid incoming transactions using the unique transaction hash $TxID$. Its counterpart in Bitcoin Core is $mapTx$. The second filter, $CbTxInputsFilter$ checks for duplicate inputs using the tuple $\langle TxIn, Index \rangle$, i.e. the $TxID$ of the previous transaction and index of the input within the specified transaction. Its counterpart in Bitcoin Core is $mapNextTx$. Transactions are

removed by deleting the $TxID$ from $CbTxFilter$ and its inputs from $CbTxInputsFilter$.

4.1. Entry

Fig. 4b details the mempool entry process and associated structures for Bitcoin Core [115] and Carbyne, depicted in blue and red respectively.

① A transaction, $entryTx$, is received over the network via an inventory message. ② In Bitcoin Core, the $TxID$ of the transaction $entryTx$ is used to query $mapTx$ to check if the transaction already exists in the mempool. In Carbyne, $TxID$ is used to query $CbTxFilter$. An already-received transaction is dropped. ③ If the transaction is new, it undergoes syntax and semantics checks. These checks are identical for both Bitcoin Core and Carbyne and invalid transactions are rejected in either case.

④ Next inputs, $TxIn$, of $entryTx$ are scanned for double-spends. In-mempool inputs are verified from $mapNextTx$ in Bitcoin Core and using $CbTxInputsFilter$ in Carbyne. This step was a particular challenge for Carbyne and necessitated the deployment of a separate dedicated filter. If any of the inputs already exist in the mempool, the transaction is dropped. ⑤ Transaction inputs are also validated using the UTXO set. This check is also identical for Bitcoin Core and Carbyne and transactions with invalid or spent UTXOs are not permitted to enter the mempool. If any transaction input (referred to as *parent* or *ancestor*) is missing, $entryTx$ is added to the orphan pool. It resides in the orphan pool until its ancestor is received.

⑥ If $entryTx$ and its inputs are successfully verified, in Bitcoin Core it is added to $mapTx$, and each of its inputs to $mapNextTx$. In Carbyne, the $TxID$ is added to $CbTxFilter$ and each of its inputs, $\langle TxIn, Index \rangle$, is added to $CbTxInputsFilter$. ⑦ In Bitcoin Core, the ancestor-descendant transaction chains are also updated in $mapLinks$. This information is required by miners to identify and prioritize transactions to be mined in the next block. Carbyne users have the flexibility to store a subset of complete transactions as per their resources, for purposes of mining or to bootstrap new nodes (see detailed discussion in Appendix B).

⑧ The transaction hash $TxID$ is then broadcast to the node's peers with an inv message and full transactions are forwarded on request. After a short interval to propagate the

transaction, Carbyne discards it. The length of the interval that full transactions are stored can vary and may be configured (the trade-off is quantified in Appendix C).

⑨ Nodes typically receive `inv` messages announcing a transaction multiple times. To check if a transaction already exists in the mempool, `mapTx` in Bitcoin Core and `CbTxFilter` in Carbyne is queried using `TxID`. A fresh transaction is requested via the `getdata` message [115].

4.2. Exit

Transactions exit the mempool for six reasons: 1. inclusion in a block, 2. the transaction, or one of its unconfirmed ancestors, conflict with a transaction included in a block, 3. replacement by a newer version paying more fee, 4. eviction due to mempool size limitation, 5. a chain reorganization event at the node, and 6. expiry due to age.

Fig. 5 depicts the exit process for Bitcoin Core and Carbyne: ① When a block arrives over the network, it includes transactions that have been confirmed and should be removed from the mempool. ② In Bitcoin Core, `mapTx`, and in Carbyne `CbTxFilter`, are queried to confirm that the transaction exists in the mempool. ③ In Bitcoin Core, the transaction is removed from `mapTx`, `mapNextTx`, and `mapLinks`, and in Carbyne, it is removed from `CbTxFilter` only.

④ If the transaction needs to be removed for reasons other than inclusion in a block, Bitcoin Core will remove the transaction as well as its descendants from `mapTx`, `mapNextTx`, and `mapLinks`. In Carbyne, however, these transactions accumulate and are removed by clearing the filter at periodic intervals. For this purpose, we cascade an additional filter to `CbTxFilter` (as shown in Fig. 4a) to work in rotation. This configuration enables us to separate mempool transactions based on age. This process is detailed in §7.

`CbTxInputsFilter` is also cleared after a predefined interval, to prevent overflow and degradation in performance. Batch deletion relieves us of the need to store individual mappings of transactions and their inputs, and also saves time over deleting each transaction individually.

Replace-by-Fee (RBF) is an opt-in node policy enabling to replace a transaction with a new version by paying a higher fee [44]. Carbyne does not specifically cater to RBF, but it allows nodes to process and circulate new versions of prior transactions regardless of fee (described in §7.2.2). We discuss potential strategies to enable RBF in Appendix D.

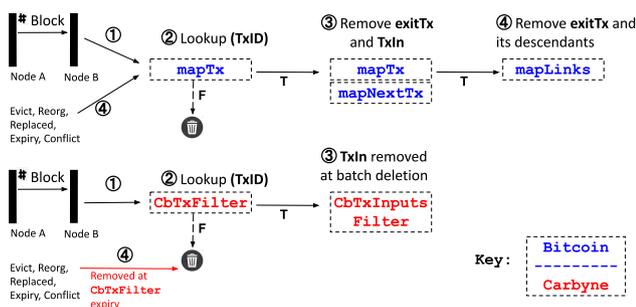


Figure 5: Mempool Exit for Bitcoin Core and Carbyne

5. Methodology

We follow a systematic approach to evaluate Carbyne's performance against the traditional Bitcoin Core mempool. The methodology consists of six key steps:

Data Collection: We deployed an instrumented version of Bitcoin Core to collect data over a 90-day period, capturing real-time network activity. Our dataset includes mempool activity and statistics along with network activity. The details of the dataset are provided in §6. This dataset enables a comprehensive reconstruction of the Bitcoin mempool state, serving as the foundation for our evaluation.

Design and Implementation: Carbyne is implemented in C++, the same language as Bitcoin Core, the most widely used Bitcoin client. C++ provides a robust standard library optimized for performance, efficient memory management, and low-overhead data handling. We implement `CbTxFilter` and `CbTxInputsFilter`, both counting Bloom filters using the Berkeley libbf library [88], replacing Bitcoin Core's `mapTx` and `mapNextTx` structures.

Experimental Setup: We conduct our evaluation, running a simulated Bitcoin Core mempool and Carbyne in parallel while processing the recorded dataset. Bitcoin Core's default mempool serves as the ground truth, while Carbyne represents the experimental condition. Each transaction event, entry, exit, and inventory, is replayed in both environments under identical workloads, allowing a direct performance evaluation.

Performance Evaluation: We evaluate the system based on false positive and negative rates, memory, and computational efficiency. The results are presented through graphs, statistical analysis, and comparative tables to provide a clear and quantitative performance comparison.

Discussion and Analysis: We analyze the system's behavior, linking theoretical expectations to experimental results. This includes an examination of false positives and negatives, a challenge addressed through careful parameter tuning. Additionally, we evaluate the security of our approach, particularly in adversarial conditions, to assess its robustness against potential attacks.

Stress Testing for Dos-Resilience: To assess Carbyne's resilience against Denial-of-Service (DoS) attacks, we conduct controlled stress tests designed to evaluate its ability to handle extreme transaction loads while maintaining efficiency and accuracy. We simulate extreme congestion with 600,000 transactions—over three times Bitcoin's peak observed volume. To quantify the impact of congestion on filter accuracy and memory usage, we examine two mitigation strategies in §9.

Our dataset does not specifically include dust or spam transactions, as Carbyne is not a spam filtering mechanism. Instead, it optimizes transaction processing and enhances network capacity without differentiating between legitimate and spam transactions. Since our approach treats all transactions identically, incorporating spam transactions into the dataset would not alter the results. However, Carbyne is complementary to existing spam mitigation techniques and can operate alongside them.

6. The MempoolState Dataset

To test Carbyne we create an instrumented customized version of Bitcoin Core version 0.11. We used it to set up a full Bitcoin node, running on Intel Core i7-8700 CPU @3.2 GHz \times 12, 16 GB RAM, 2 TB HDD, running Ubuntu 18.04. We assume that our observation of the network is largely consistent with the rest of the network. Research confirms high similarity in mempools of full nodes [13] and this approach has been widely used in the literature (e.g. [12] [74]). Our dataset is longitudinal, spanning 90 days from 1 January, 2021 to 31 March, 2021 (2160 hours). It comprises:

1. *Mempool Activity*: We modify Bitcoin Core, specifically `src/txmempool.cpp`, to capture all mempool entries and exits in JSON format. We log \sim 29 million unique transactions with \sim 88 million inputs. This dataset is novel and enables researchers to efficiently reconstruct the Bitcoin mempool state for various applications apart from Carbyne.
2. *Network Inventory*: includes the `inv` messages received over the network. We log \sim 89 million `inv` messages via the `-network flag` in the Bitcoin configuration file. We store them together with timestamps in CSV format.
3. *Mempool Statistics*: includes the raw transaction size, resulting memory usage, and transaction count in the mempool, obtained using the Bitcoin daemon's JSON-RPC interface, specifically the `getmempoolinfo` method. We invoked the `getmempoolinfo` method at 10-minute intervals and stored the data as JSON.

7. Experiments and Results

In this section we describe our experiments. We undertake a detailed assessment of Carbyne versus the Bitcoin Core mempool for a range of performance metrics, including accuracy, memory consumption, and processing time.

7.1. Implementation and Methodology

We use the *Mempool Activity* and *Network Inventory* data set components to recreate transaction flow at a Bitcoin node over a 90-day period. Each of *entry*, *exit*, and *inv* is treated as a distinct event that changes the mempool state. We replay these events and process them in parallel, using Carbyne and a simulated version of the Bitcoin Core mempool. The latter serves as ground truth, enabling us to document precisely how Carbyne processes each event compared to the how Bitcoin Core originally handled it.

We use C++ to implement Carbyne and simulate the Bitcoin Core mempool. We use the Berkeley libbf library [88] to implement counting bloom filters. This library uses the `H3exp` class of hashing functions designed by Carter and Wegman [56]. The Carbyne implementation consists of probabilistic data structures `CbTxFilter` and `CbTxInputsFilter`. We simulate Bitcoin Core mempool's key structures, `mapTx`,

`mapNextTx` and `mapLinks`. The `CbTxFilter` in Carbyne and the `mapTx` structure in Bitcoin Core are independently queried at every *entry*, *inv* and *exit* event in our dataset to check if the relevant transaction exists in the mempool or not. Due to its probabilistic nature, Carbyne's `CbTxFilter` will sometimes deviate from the ground truth and yield false positives and negatives. We define metrics for these next.

7.1.1. Performance Metrics

The responses to mempool queries can be categorized into a confusion matrix: *True Positives (TP)*: Bitcoin Core mempool and Carbyne both indicate that the queried transaction is present; *True Negatives (TN)*: Bitcoin Core mempool and Carbyne both indicate that the transaction is not in the pool; *False Positives (FP)*: Bitcoin Core mempool indicates that the transaction is not in the pool, but Carbyne erroneously records it as present; and *False Negatives (FN)*: Bitcoin Core mempool indicates that the transaction is present, but Carbyne erroneously reports it as absent.

At the filter level, the outcomes as per event are as follows: For an *entry* event, when the mempool is queried on adding a received transaction: TP_{entry} : the transaction already exists in the pool and will be discarded; TN_{entry} : the transaction is new and will be added to the pool; FP_{entry} : the transaction is new and should be added to the pool but will erroneously be discarded; FN_{entry} : the transaction already exists in the pool, but will erroneously be added again.

For an *inv* event, when the mempool is on a transaction being available at a node: TP_{inv} : the transaction already exists in the pool and the full transaction will not be requested; TN_{inv} : the transaction is new and the full transaction will be requested to add to the pool; FP_{inv} : the transaction is new but will erroneously not be requested to add to the pool; FN_{inv} : the transaction already exists in the pool but the full transaction will erroneously be requested to add to the pool.

At *exit*, when the mempool is queried to remove a transaction: TP_{exit} : the transaction exists in the pool and will be removed; TN_{exit} : the transaction does not exist in the pool and cannot be removed; FP_{exit} : the transaction does not exist in the pool but is erroneously 'removed' by decrementing the filter counters; FN_{exit} : the transaction exists in the pool but is erroneously not removed.

Each filter-level outcome has different consequences for Carbyne performance. False positives at all three events lead to transactions not being processed and hence a reduction in the overall accuracy of the system. Therefore, an overall false positive rate is a vital metric to consider. Specifically, any false positives at inventory and entry result in transactions being discarded and the rate of such discarding needs to be kept down. Looking at false negatives, any such outcome at inventory (or equivalently entry) will cause unnecessary reprocessing of transactions. False negatives at exit will not inflict any immediate cost to the system, but will eventually cause more false positives and hence their effect can be captured by the overall false positive rate. Based on these insights, we define performance metrics for Carbyne:

CbTx Filter size	Buckets (m)	Hash Func. (k)	False Positive Rate			Discarded Transactions		Reprocessed Transactions	
			Theoretical	No Expiry	Expiry	No Expiry Num/(%)	Expiry Num/(%)	No Expiry Num/(%)	Expiry Num/(%)
600 KB	2.4M	8	1.2E-03	1.5E-02	3.7E-03	1,646,878(1.402)	497,795(0.339)	82,028(0.1964)	30,034(0.0339)
800 KB	3.2M	11	4.6E-04	9.3E-03	1.4E-03	1,050,915(0.893)	195,073(0.133)	58,063(0.0655)	10,186(0.0115)
1 MB	4M	14	6.7E-05	8.1E-03	8.1E-04	911,695(0.774)	109,338(0.074)	52,650(0.0594)	5,822(0.0066)
2 MB	8M	28	5.0E-09	3.8E-03	1.3E-04	427,242(0.362)	17,441(0.012)	30,712(0.0346)	803(0.0028)
3 MB	12M	42	< 1E-10	2.3E-03	4.9E-05	263,263(0.223)	7,096(0.005)	22,608(0.0255)	298(0.0003)
4 MB	16M	55	< 1E-10	1.4E-03	2.8E-05	163,923(0.139)	4,273(0.003)	15,600(0.0176)	77(0.0001)

Table 1

Performance Metrics for CbTxFilter of various sizes dimensioned for $n = 200,000$ transactions

- *False Positive Rate (FPR)* is a measure of accuracy, defined as the ratio of the false positives to the total number of queries (*entry*, *inv* and *exit*).

$$FPR = \frac{FP_{\text{entry}} + FP_{\text{inv}} + FP_{\text{exit}}}{\text{Queries}_{\text{entry}} + \text{Queries}_{\text{inv}} + \text{Queries}_{\text{exit}}}$$

- *Discarded Transactions* is a measure of the proportion of new transactions at *entry* and *inventory* that were erroneously discarded due to false positives.

$$\text{DiscardedTxs} = \frac{FP_{\text{inv}} + FP_{\text{entry}}}{\text{Queries}_{\text{inv}} + \text{Queries}_{\text{entry}}}$$

- *Reprocessed Transactions* is a measure of transactions that were processed twice due to false negatives at *inventory*.

$$\text{ReprocessedTxs} = \frac{FN_{\text{inv}}}{\text{Queries}_{\text{inv}}}$$

We note here that there is precedent for erroneous transaction handling in the Bitcoin ecosystem, especially for resource-constrained clients. For instance, false positives in SPV clients occasionally result in forwarding of erroneous transactions. However, circulating these transactions does not equate to actual double-spends, since nodes in the network, including Carbyne nodes, will still screen transactions in all incoming blocks to ensure there is no double-spending.

7.1.2. Dimensioning CbTxFilter

The highest transaction volumes observed to date have edged close to the 200k transactions mark, as depicted in Fig. 1a. We therefore choose a maximum transaction load of 200k as a starting point to dimension CbTxFilter and a false positive rate of 10^{-3} (1 in 1000). This is an order of magnitude less than the 1% failure rate Baqer et al. suggest may be ‘disruptive’ and ‘a denial of service in itself’ [74]. Using Eq. 2 and Eq. 3, we derive a starting bloom filter size of 600 KB consisting of 2.4M buckets and 8 hash functions.

We also test other bloom filter sizes, listed in Table 1. For comparative purposes, we focus on two other filter candidates for our experiments, sized at 2 MB (medium) and 4 MB (large), with 8M and 16M buckets each, and provisioned with 14 and 28 hash functions respectively. Fan

et al. recommend a bucket width of 4 at which probability of overflow in counting bloom filters is infinitesimally small [117]. However, in our experiments, counters in the filters did not exceed 2. We therefore allocate a bucket width of 2 bits.

7.2. Empirical Results and Discussion

We replay transaction events in the *MempoolState* dataset for the three month period through all filters. Table 1 shows the average false positive rate is highest for the 600 kB filter at 1.46×10^{-2} , reducing marginally to 8.07×10^{-3} for a 1 MB filter, and further to 1.43×10^{-3} for the 4 MB filter. For the 600 kB, 1 MB, and 4 MB filters we also observe 1.402%, 0.774%, and 0.139% of transactions being erroneously *discarded* due to false positives and 0.1964%, 0.0594%, and 0.0176% of transactions being *reprocessed* due to false negatives. These trends are as expected with false positives and negatives decreasing with increasing filter size.

Fig. 6 depicts in real-time the number of transactions in these filters along with the false positive rate for the entire three month period. We also plot the corresponding number of transactions in the Bitcoin Core mempool, the ground truth in our evaluation. In all three cases, we observe the number of transactions closely tracks the pattern in the Bitcoin Core mempool, with an increasing offset.

This offset is due to two main sources: first, transactions that should have been removed from the mempool because of age or were replaced or in conflict with other transactions. The Bitcoin Core mempool ‘expires’ old transactions after a two-week (default) period and may evict certain transactions if they are in-conflict with any in-mempool transactions. Counting bloom filters have no inherent mechanism to track age or conflicts. Second, false negatives result in some transactions being erroneously added to the mempool at *entry* events and some, due to be removed, to persist at *exit* events. We term these erroneous artifacts *debris*, which accumulates over time and corresponds to growing deterioration in performance.

We also observe in Fig. 6a–6c, a marked increase in the false positive rate in all three filters, after the number of transactions reach 200,000, i.e. the size the filters were originally dimensioned for. For the 600 kB and 1 MB filters, transaction numbers for Carbyne and Bitcoin Core do not diverge indefinitely but stabilize around the 300,000

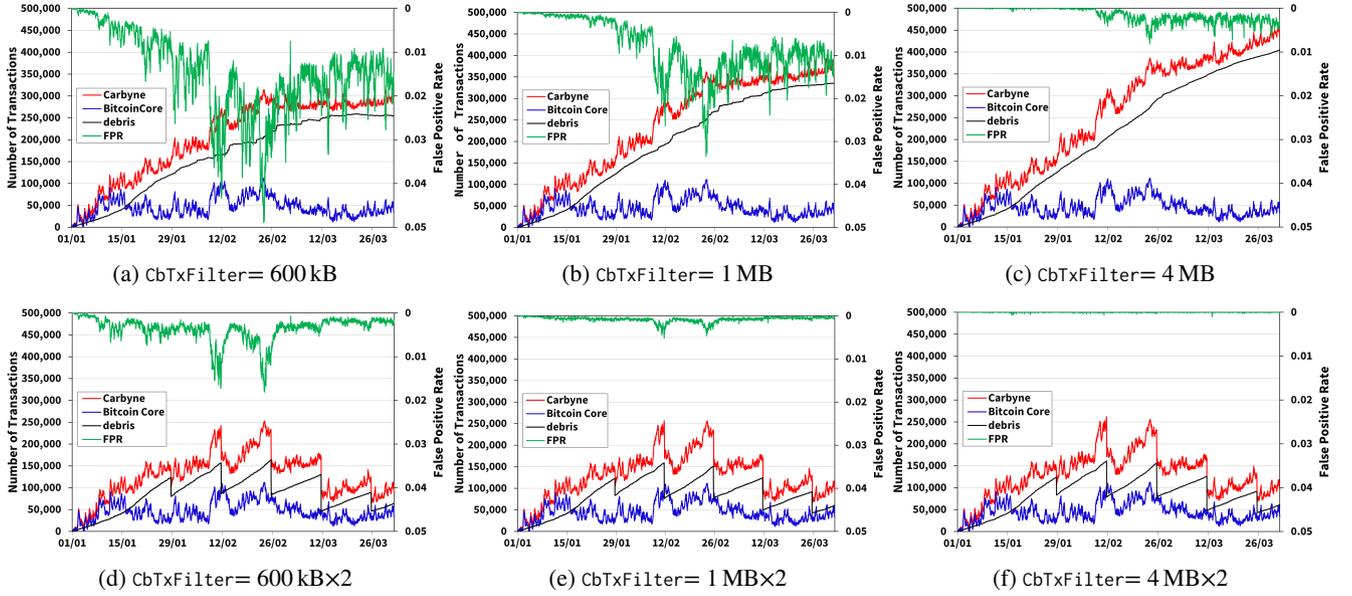


Figure 6: Number of Transactions in Bitcoin vs Carbyne, Debris Transactions and False Positive Rate

mark. Our investigation reveals this is due to the high false positive rate, which has the interesting effect of evicting large numbers of transactions from the filter at *exit* events (by decrementing the counters). This, in turn, significantly reduces the transactions count and the false positive rate.

This is similar to expiry in temporal counting bloom filters, where counters are periodically decremented by a decay factor to prevent saturation of the filters for continuous streams of item insertions [118]. This effect does not occur in the 4 MB filter, because the false positive rate is too low to evict large numbers of transactions, and the offset between Carbyne and Bitcoin Core continues to grow.

Empirical false positive rates are also significantly higher than our theoretical target of 10^{-3} for all filters, almost by an order of magnitude (e.g. 6.7×10^{-5} vs. 8.1×10^{-4} for the 1 MB filter). We theorize as to the causes: first, multiple works have reported false positive rates in real deployments are higher than theoretically computed [20] [19]. Researchers contend that this is because theoretical calculations assume that “each hash transformation is perfect” [20] and that transactions “are independent and uniformly distributed over all records” whereas real activity tends to be “clumped” [19]. In this context, Bose et al. prove that Eq. 1 actually gives us a lower bound on the false positive rate [21].

Second, repeated insertion and deletion from an unbounded set over time, leads to increased false positives and false negatives. Moreover, debris rapidly causes transactions in CbTxFilter to exceed the 200,000 mark for which it was provisioned. This results in a complex dynamic and interdependent effects that compound over time. For instance, Guo et al. note in their study that false negatives at *exit* will cause items that should be removed to persist in the filter, in turn resulting in yet more false positives [112]. We can confirm that we observe this specific effect in our own experiments.

7.2.1. Transaction Expiry

There is no inherent mechanism to resolve the issue of debris in counting bloom filters. We propose a solution to periodically ‘clean up’ the filters. This is done by employing two identical counting bloom filters, a primary and a secondary, working together in rotation, which switch status after a predefined interval. This configuration enables us to clearly separate transactions on the basis of age.

All queries are directed to the primary filter first. If a transaction is not present there, then the secondary filter is queried. Transactions are removed from the filter that first reports them to be present. However, insertions only occur into the primary filter. After every predefined interval, the secondary filter is reset, i.e. all counters are decremented to zero, and status of the two filters is switched again. This cycle repeats after every interval, and effectively simulates Bitcoin’s transaction expiry mechanism, where transactions are automatically removed from the mempool after a default period of 14 days.

We implement this mechanism with an expiry period of 14 days. Results are in Table 2 and Figs. 6d–6f. The filters first switch roles on 15 January, followed by the first expiry event on 29 January, and then again every 14 days, corresponding to sharp drops in transaction numbers in the filters. The Carbyne mempool now more closely tracks the contours of the Bitcoin Core transactions pattern. Average false positive rates for our three filters of interest are reduced by at least one order of magnitude, to the point that performance of filters with expiry is comparable to that of larger filters. For instance, the 1 MB filter, with expiry demonstrates an average false positive rate of 8.1×10^{-4} , which is less than 1.4×10^{-3} resulting from the 4 MB filter without expiry. The number of discarded and reprocessed transactions are also dramatically reduced.

Expiry Time (days)	False Positive Rate			Discarded Transactions			Reprocessed Transactions		
	600 kB	1 MB	4 MB	600 kB Num/(%)	1 MB Num/(%)	4 MB Num/(%)	600 kB Num/(%)	1 MB Num/(%)	4 MB Num/(%)
14–28	3.7E-03	8.1E-04	2.8E-05	497,795(0.339)	109,338(0.074)	4,273(0.003)	30,034(0.034)	5,822(0.007)	77(< 0.001)
7–14	2.1E-03	4.6E-04	2.2E-05	287,670(0.196)	62,271(0.042)	3,519(0.002)	34,549(0.039)	18,169(0.021)	16,562(0.019)
1–2	6.2E-04	1.5E-04	1.6E-05	85,589(0.058)	20,814(0.014)	2,654(0.002)	148,128(0.167)	126,017(0.142)	146,265(0.165)
$\frac{1}{2}$ –1	4.4E-04	1.2E-04	1.5E-05	60,852(0.042)	16,258(0.011)	2,537(0.002)	231,543(0.261)	191,210(0.216)	231,428(0.261)
$\frac{1}{4}$ – $\frac{1}{2}$	3.1E-04	8.3E-05	1.4E-05	43,085(0.030)	11,790(0.008)	2,400(0.002)	294,164(0.332)	262,567(0.296)	294,530(0.280)

Table 2

False positive rates for different Expiry times and CbTxFilter sizes, dimensioned for n=200,000 transactions

The expiry period here is not strictly 14 days but in the range of 14–28 days, since a filter is cleared every 28 days. The false positive rate can be further improved by reducing the expiry period. A case could be made for this given that, since 2016, the median transactions confirmation time for Bitcoin has not exceeded 30 mins [49] and the average confirmation time of a transaction is around 104 mins [50].

This means that even 6–12 hours is a plausible value for expiry. As shown in Table 2, when the expiry period is reduced from 14–28 days to $\frac{1}{4}$ – $\frac{1}{2}$ day, there is an order of magnitude decrease in the false positive rate for the 600 kB and 1 MB filters which reduced from 3.7×10^{-3} to 3.1×10^{-4} and 8.1×10^{-4} to 8.3×10^{-5} , respectively. For the 4 MB filter the decrease is only marginal. Similarly, the discarded transactions percentage for the 600 kB and 1 MB filters reduced from 0.339% to 0.030% and from 0.074% to 0.008%, respectively. However shorter expiry intervals result in increased reprocessing costs: if an old transaction is received again after the filter has been cleared, Carbyne will consider it a new transaction and process it again. For a 600 kB filter, reprocessed transactions increase from 0.034% to 0.332%, for a 1 MB filter from 0.007% to 0.296%, and for a 4 MB filter from less than 1×10^{-5} % to 0.280%.

7.2.2. CbTxInputsFilter Dynamics

CbTxInputsFilter scans inputs of incoming transactions to prevent double spends. The implications are:

- TP_{inputs} : a transaction bearing that input was recently added to CbTxInputsFilter and the new transaction should be discarded;
- TN_{inputs} : a transaction bearing that input does not exist in CbTxInputsFilter and the new transaction should be added;
- FP_{inputs} : a transaction bearing that input does not exist in CbTxInputsFilter but the new transaction was erroneously discarded;
- FN_{inputs} : a transaction bearing that input already exists in CbTxInputsFilter, but the new transaction was erroneously added.

In our dataset incoming transactions average 40,000 inputs hourly and peak at 191,947 inputs. We therefore dimension CbTxInputsFilter for 200,000 transactions, similar to CbTxFilter. We reset CbTxInputsFilter hourly. This relieves the requirement of tracking individual inputs to transactions. With hourly reset and CbTxInputsFilter of 600 kB, 1 MB

CbTx Inputs Filter	Buckets m	False Positives	
		1 hour Num/(FPR)	3 hour Num/(FPR)
600KB	2.4M	69,644(7.9E-04)	446,476(5.1E-03)
800 KB	3.2M	26,665(3.0E-04)	152,865(1.7E-03)
1 MB	4M	15,258(1.7E-04)	91,480(1.0E-03)
2 MB	8M	2,249(2.6E-05)	20,554(2.3E-04)
3 MB	12M	781(8.9E-06)	10,312(1.2E-04)
4 MB	16M	411(4.7E-06)	6,128(7.0E-05)

Table 3

CbTxInputsFilter, n=200,000

and 4 MB the false positive rate is 7.9×10^{-4} , 1.7×10^{-4} and 4.7×10^{-6} respectively as shown in Table 3.

As we do not individually delete transactions from CbTx InputsFilter, no false negatives occur due to incorrect deletion of false positive items. False negatives occur due to expiry of transaction inputs. Thus an attacker can resend a transaction with the same input after the expiry interval. With regards to transactions that have conflicting inputs there is no hard and fast policy as to which transaction nodes accept. Typically, nodes accept and forward the first seen transaction and the first seen can differ for nodes. Moreover, Core clients with disabled mempool forward all valid incoming transactions. It is the job of miners to not add conflicting transactions to a block and the network nodes to screen incoming blocks. Since Carbyne maintains complete UTXO information, nodes will reject blocks that include double-spend transactions.

For a 600 kB CbTxFilter (14-28 day expiry) discarded transactions stand at 0.387%, i.e. 99.613% of transactions are accurately processed. With a 600 kB CbTxInputsFilter, an additional 69,644 transactions are discarded due to false positives. For a 1 MB CbTxInputsFilter, 15,258 transactions are discarded. If we factor this in for a 1 MB CbTxFilter (14-28 day expiry), the discarded transactions stand at 0.0849%, i.e. 99.915% of transactions are accurately processed. For a 4 MB CbTxInputsFilter (14-28 day expiry), 411 transactions are discarded due to false positives. Thus for a 4 MB CbTxFilter discarded transactions stand at 0.003%, that is 99.997% transactions are accurately processed.

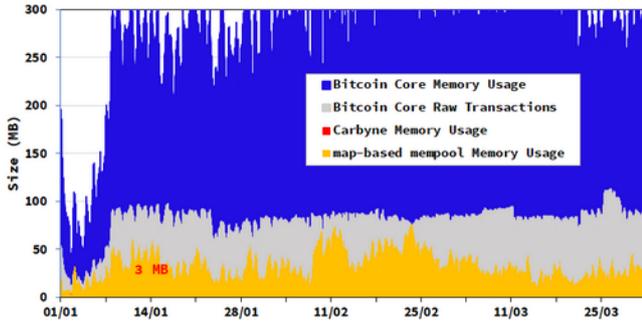


Figure 7: Raw transaction size and memory footprint

7.2.3. Memory and Computation Footprint

We compare here the memory footprint and computational overhead of Carbyne and Bitcoin Core. Fig. 7 depicts the size of the raw transactions versus the actual size of the mempool over the three month period of our experiments. This information is derived from the Mempool Statistics component of our dataset, described in §6. As depicted, raw transaction data of 100 MB typically has a footprint of 300 MB or more in the mempool due to data structure overheads to optimize functionality (as noted in §2). The graph clips at 300 MB, i.e. the maximum size allocation of our node and the default allocation for Bitcoin Core.

As baseline, we consider a straightforward mempool optimization scheme, which uses deterministic data structures but only keeps transaction hash in memory and with fewer indices. Rudimentary calculations indicate that the storage size for this scenario would be significantly more than Carbyne. For complete transaction validation, we would need to store the following indices: transaction ID (32 bytes), transaction input hash (32 bytes) and index (4 bytes) - amounting to $(32+36n)$ bytes for each raw transaction where n is the number of transaction inputs. Using standard data structures, such as maps) would incur a memory overhead almost three times the size of the raw data. Overall, we get savings of up to one order of magnitude.

This approach reduces memory consumption from an optimization perspective, but the disadvantages are still significant. It is less resilient to congestion events and spam attacks. This node does not store full transactions, it is equally limited as Carbyne from an inventory perspective, i.e., it cannot bootstrap mempools of other nodes. The only benefit is that it does not suffer from false positives.

Carbyne results thus far are immensely promising: to process an equivalent volume of transactions with 99.9151% accuracy, Carbyne requires only 3 MB of memory, corresponding to `CbTxFilter` sizes of $1\text{MB} \times 2$ (with expiry) and a `CbTxInputsFilter` size of 1 MB. Carbyne represents a two-order-of-magnitude reduction compared to Bitcoin Core, with memory usage dropping from 300 MB to just 3 MB. Notably, this value remains constant regardless of transaction volume. Given this significant efficiency gain, we characterize Carbyne as "ultra-light." This improvement is quantitatively illustrated in Fig. 7.

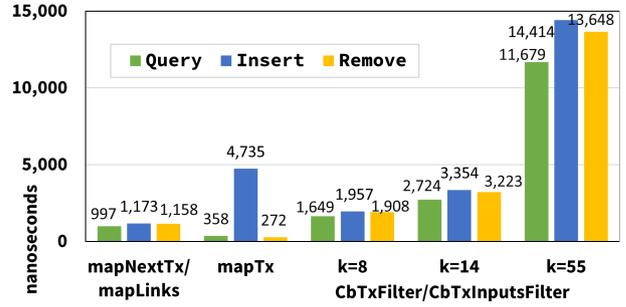


Figure 8: Computation Time

We have identified various parameters users can tweak to navigate the tradeoff between accuracy, memory footprint, and computation overhead as per their own requirements. Filters can be dimensioned based on expected transaction flows and expiry can be tweaked, enabling smaller filters to achieve accuracy comparable to much larger ones.

We next calculate the computational overheads. Bitcoin Core components `mapTx`, `mapNextTx` and `mapLinks` undertake query, insertion and deletion operations in $O(\log n)$ time, where n is the number of stored transactions. Counting bloom filters in Carbyne operate in constant time, $O(k)$, for query, insertion and deletion operations, where k is the number of hash functions.

We compare their timings, depicted in Fig. 8, using an Intel Core i7 system with 8700 CPU @3.2GHZ $\times 12$ and 32 GB RAM, running Ubuntu 18.04 with GCC 5.4.0. We replicate Bitcoin Core structures `mapTx` (Boost multi-index), `mapNextTx` and `mapLinks` (C++ STL maps). We use `libbf` to instantiate counting bloom filters and perform query, insert and delete operations, averaging over 1 million iterations.

- **CbTxFilter vs. mapTx:** Query, insert and delete operations in `mapTx` take 358 ns, 4,735 ns, and 272 ns, respectively, whereas `CbTxFilter` (for $k = 14$) requires 2,724 ns, 3,354 ns, and 3,223 ns, respectively. For a complete transaction lifecycle comprising query, insert and removal, `mapTx` requires 5,365 ns compared to 9,301 ns for `CbTxFilter`.
- **CbTxInputFilter vs. mapNextTx:** Query, insert and delete operations for `mapLinks` and `mapNextTx` require 997 ns, 1,173 ns, and 1,158 ns, respectively, whereas `CbTxInputsFilter` (for $k = 14$) takes 2,724 ns and 3,354 ns for query and insert operations. There is no delete operation for this filter as it is periodically reset (as described in §4). Over a single transaction input life-cycle, `mapNextTx` requires 3,328 ns compared to 6,078 ns for `CbTxInputsFilter`.
- **mapLinks:** Maintaining unconfirmed transaction chains in `mapLinks` is memory and computation intensive. For each entry, ancestor-descendent information is kept in the mempool to prioritize transactions for blocks. But with every entry or exit in the mempool, all ancestors and descendants need to be recursively updated. A single series of operations requires 3,328 ns. To cap these costs, also be a potential DDoS vector, Bitcoin Core

0.12 introduced a default policy limiting unconfirmed chains to 25 transactions and total size of 101 kB. Carbyne does not store these mappings as explained in §4 and does not incur these costs.

For a complete transaction lifecycle—including query, insertion, and removal—`mapTx` in Bitcoin Core requires 5,365 ns, whereas `CbTxFilter` incurs a slightly higher cost of 9,301 ns. Considering the transaction lifecycle, `mapNextTx` in Bitcoin Core requires 3,328 ns, while `CbTxInputsFilter` in Carbyne demands 6,078 ns. Managing unconfirmed transaction chains within `mapLinks` over a single sequence of operations requires 3,328 ns in Bitcoin Core. Typical Bitcoin transactions have around 2–3 inputs. Assuming the addition of a 2 input transaction necessitates two sets of query, insertion and removal operations, the total cost in `mapLinks` amounts to 6,656 ns. In contrast, Carbyne eliminates this overhead entirely. Summing these computational costs, the total processing time per transaction lifecycle amounts to 15,349 ns for Bitcoin Core and 15,379 ns for Carbyne. These figures indicate that Carbyne and Bitcoin Core exhibit comparable computational loads.

From a practical standpoint, Carbyne is demonstrably capable of supporting real-time transactions. Given that each transaction requires 15,379 ns, the system can theoretically sustain a transaction throughput of approximately 65,000 transactions per second (tps). This significantly exceeds Bitcoin’s current throughput of 3–7 tps, highlighting Carbyne’s scalability potential. If Bitcoin’s throughput increases, Carbyne can seamlessly scale to accommodate loads to the order of thousands of transactions per second.

As we noted, there are no financial incentives in the Bitcoin protocol for full nodes (as compared to miners), and the resource costs for this exercise are constantly increasing. On our part, we assume that users operate nodes primarily to contribute to the Bitcoin network out of a sense of community or altruism – similar to how people operate nodes for the Tor network.

In this sense, we believe it is useful to provide such users, via our solution, with control over the local memory resources they allocate for this task to make it cost-efficient. This is conceptually similar to Bitcoin Core’s inbuilt ‘pruned node’ option which aims to reduce hard disk requirements. i.e., pruned nodes do not store the entire blockchain on disk but still contribute to the footprint and health of the network by validating and forwarding transactions.

Carbyne, as a lightweight solution, lowers the entry cost for participation in the Bitcoin network. Similar to pruned nodes, Carbyne should be viewed as a solution not to supplant mining nodes but to reduce costs for certain users while enhancing network health by validating transactions and improving congestion resilience. In summary, Carbyne minimizes memory costs while preserving computational efficiency, making it a practical, cost-effective solution for Bitcoin node operators.

7.3. Security Analysis in an Adversarial Setting

Here, we consider if an adversary can exploit Carbyne’s design or properties in unintended ways. We specifically discuss two situations: can an adversary craft transactions to evade or trigger filters at individual Carbyne nodes? If so, can the adversary trigger censorship of select transactions across large numbers of Carbyne nodes in the network?

To recap the fundamental point in §2.3 each node initializes its filters independently using random 128-bit seeds that are kept secret. This makes it highly unlikely that large numbers of nodes drop the same transactions. The seeds we are using to initialize the filters are 128 bits long and kept secret by the nodes (as explicitly recommended in the literature [1]). This means that individual nodes are effectively using independent hash functions to construct their filters.

To trigger a false positive in a certain Carbyne node, an adversary would therefore have to access two pieces of information: 1) the seeds used to initialize that node’s most recent filter, and 2) the current state of that filter (i.e. which transactions it has already logged). We assume the first item would be difficult for an attacker to procure. A node can even change its random seeds every time a new `CbTxFilter` is generated (every two weeks in our basic scenario).

Seeding `CbTxFilter` with secret random seeds effectively reduces the adversary’s capability from mounting a chosen transaction attack to a random transaction attack: although the adversary controls the transactions they broadcast, without knowledge of the seeds, the output of the hash function and hence the marked indices in the filters will be effectively unpredictable for the adversary. Hence, the adversary cannot do much better than broadcasting random transactions. He will not be able to effectively craft specific transactions that lead to false positives with higher probability.

Moreover, since individual nodes use random seeds, even if an adversary somehow crafts a transaction that induces a false positive in a targeted node, the probability of inducing a false positive in another node using the same transaction will be very very low (as described in §2.3). Adversarial attempts targeted at individual nodes will, therefore, not scale to Carbyne nodes network-wide. We do not believe there is any low-cost means for an attacker to censor specific transactions across all Carbyne nodes on the network.

8. Carbyne for Other Cryptocurrencies

The Carbyne approach, in principle, can be extended to other cryptocurrencies. However the design of such a scheme has to be tweaked considerably according to the specific protocols. Adapting Carbyne to Bitcoin forks such as Bitcoin Cash, Bitcoin Gold, Litecoin and Dogecoin is straightforward, as they are all based on the UTXO model. For these currencies only certain parameters such as transaction expiry time, bloom filter size, etc. need to be tweaked according to the traffic on the network and the block time.

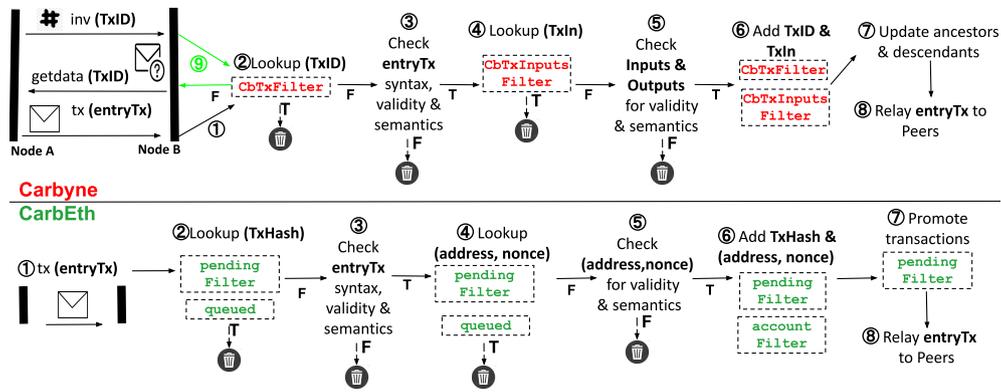


Figure 9: Carbyne vs CarbEth

8.1. CarbEth (Carbyne for Ethereum)

An accounts-based cryptocurrency like Ethereum will necessitate considerable changes in addition to the changes identified above. We propose an equivalent scheme for Ethereum, CarbEth (Carbyne for Ethereum). However, the empirical analysis regarding the various parameters users can tweak to navigate the tradeoff between accuracy, memory footprint, and computation overhead will have to be determined based on a dataset specific to the Ethereum network. This investigation warrants a separate and dedicated examination. We will address this in our future research.

8.2. Ethereum txpool vs Bitcoin mempool

The transaction pool in Ethereum, namely the Txpool comprises three primary map structures: **pending** holds executable transaction data; **queued** manages non-executable transaction data and **beats** keeps a record of the most recent heartbeat from each known account.

As outlined in §2.4, the Bitcoin mempool is composed of two primary structures: the CTxMemPoolEntry, which plays bookkeeping role, and the CTxMemPool, which encompasses three key data structures: mapTx indexes transactions on five criteria: transaction hash, witness-transaction hash, descendant and ancestor fee rates, and timestamp; mapNextTx tracks transaction inputs; mapLinks monitors in-mempool ancestor and descendant transactions.

8.3. Primary components of CarbEth & Carbyne

CarbEth is composed of two components: The first component, a counting bloom filter named pendingFilter, utilizes the transaction hash (<txHash>) to map and track valid incoming transactions. In Carbyne, the first component, a counting bloom filter named CbTxFilter maps valid incoming transactions using the unique transaction hash <TxID>.

In CarbEth, the second component a bloom filter with key-value storage, named accountFilter ensures that double-spend transactions are identified. It does this by storing key-value pair <address, nonce>, which represent the sender’s address and the nonce value of the most recent transaction added to pendingFilter. In Carbyne, the second component, a counting bloom filter, CbTxInputsFilter checks for duplicate inputs using the tuple <TxIn, Index>.

Entry ① In Ethereum the process begins with the arrival of a complete transaction, referred to as entryTx, over the network through a Transaction message. In Bitcoin, a transaction announcement which only includes the transaction hash is made via the inv message and a full transaction is relayed only on request. This means that the Bitcoin node has to hold on to transactions, in anticipation of a complete transaction request (quantitative analysis offered in Appendix C). Ethereum can forego this. ② In CarbEth the transaction’s txHash is used to query the pendingFilter and queued data structures to determine if the transaction already exists in the txpool. In Bitcoin Core, the transaction hash of entryTx, the TxID is used to query CbTxFilter to check the same. ③ If the transaction is new, it undergoes syntax and semantics checks in both CarbEth and Carbyne.

④ Next, in CarbEth, the <address, nonce> of entryTx is scanned in the accountFilter to detect any potential double spends. If a transaction exists in pendingFilter that has the same <account, address> as the incoming transaction, it is flagged as a potential double-spends and dropped. In Carbyne, the inputs, TxIn, of entryTx are scanned for double-spends using CbTxInputsFilter in Carbyne.

⑤ In CarbEth transactions are also validated against the State Trie using the key-value pair <address, nonce>. In Carbyne, transaction inputs are also validated using the UTXO set. If any transaction input (referred to as *parent* or *ancestor*) is missing, entryTx is added to the orphan pool.

⑥ Upon successful verification, in CarbEth, the txHash is added to pendingFilter, and the key-value pair <address, nonce> is added to accountFilter if the nonce is in order, or to queued if the nonce is out of order. In Carbyne If the entryTx and its inputs are successfully verified, the TxID is added to CbTxFilter and each of its inputs, <TxIn, Index>, is added to CbTxInputsFilter. ⑦ In CarbEth the node then attempts to promote any eligible transactions from queued to pending. In Bitcoin Core, the ancestor-descendant transaction chains are updated and transactions are unorphaned. ⑧ In CarbEth the complete transaction is then relayed to a small, random fraction of connected peers via a Transaction message. The transaction hash TxID is then broadcast to the node’s peers

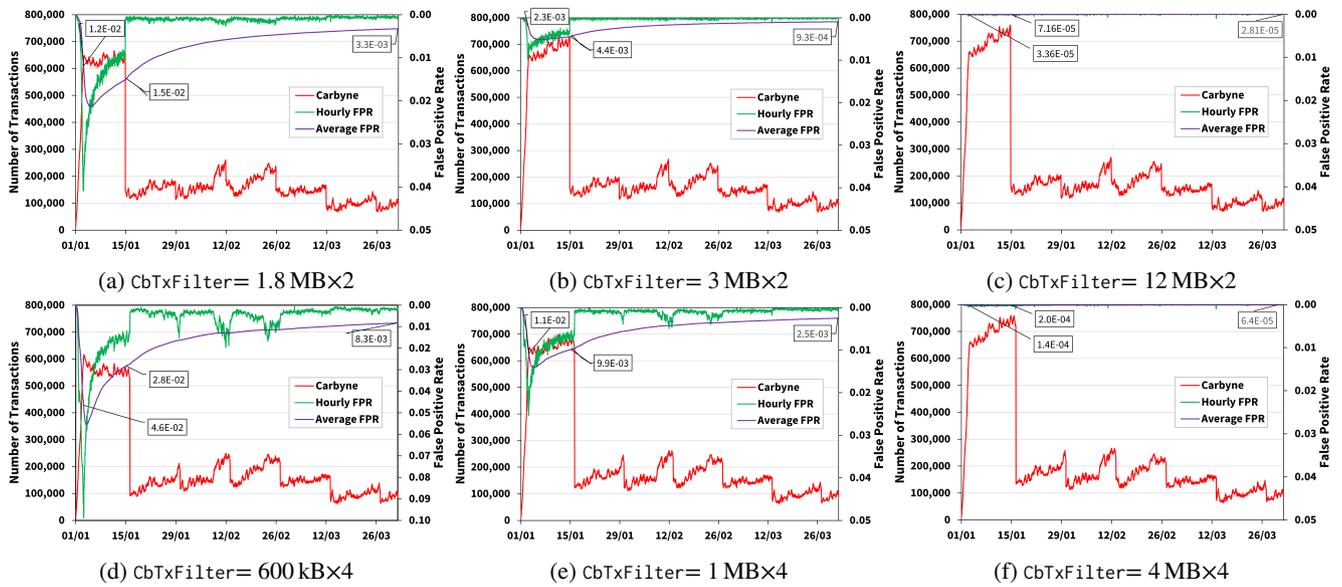


Figure 10: Dimensioning for larger transaction volume and dynamic adaptation for congestion and DDoS

with an `inv` message and full transactions are forwarded on request. This entry process is summarized in Fig. 9.

Exit transactions are removed from the txpool for several reasons: inclusion in a block, replacement by higher fee versions, reaching size limits, removal during chain reorganization, transaction age expiry. Additionally in Ethereum transactions may be removed for running out of gas during execution, and demotion to “queued” (non-executable) status. To remove transactions in CarbEth, the txHash is deleted from pendingFilter, and the corresponding <address, nonce> entry in accountFilter is updated. In Carbyne, the transaction is removed from the CbTxFilter and the CbTxInputsFilter is cleared periodically.

Nonetheless, it is imperative to empirically validate the suggested approach using Ethereum-specific data. We require accurate measurements and quantification of outcomes, particularly concerning the rates of false positives and negatives and to understand various parameters through evidence-based analysis.

9. Stress Testing Carbyne

We simulate a stress test to evaluate Carbyne’s performance under high transaction flows, congestion or spam attacks. We consider a transaction load of over 600,000 transactions, more than 3x the peak amount considered in §7. This load is generated by artificially halting exit of transactions for 55 hours to let entry transactions accumulate. Once 600,000 transactions accumulate, we resume transaction exits at the natural rate.

Our dataset does not include dust or spam transactions as Carbyne, which is not a spam filtering technique, handles both genuine and dust or spam transactions identically. It

improves transaction flow and enhances the network’s capacity to handle a greater volume of transactions. Since our approach does not differentiate between benign and spam transactions, incorporating spam into the dataset would yield the same results. That said, our scheme is orthogonal to existing spam mitigation mechanisms and can complement them without altering its core functionality.

By demonstrating that Carbyne efficiently processes even extreme transaction volumes, we provide empirical evidence validating its DoS-resilient properties. We consider two strategies:

Dimensioning for Large Transaction Volumes. We explore a preemptive strategy to sustain a transaction volume of 600k transactions. We consider CbTxFilter with expiry of sizes of 1.8 MB×2, 3 MB×2, and 12 MB×2, three times the size of filters considered in §7. CbTxInputsFilter similarly expands to 1.8 MB, 3 MB and 12 MB respectively. The secondary filter activates when transactions exceed 600,000.

Figs. 10a–10c depict how preemptive dimensioning performs. The first filter stores the first 600,000 transactions. However when the volume exceeds 600,000 the second filter is activated, marked by a pronounced decrease in the false positive rate in Figs. 10a–10c. The first filter expires after 14 days, and the number of transactions in the mempool returns to normal levels. Figs. 10a–10c shows the average false positive rate at three key points: when transactions cross 600,000, when the first filter expires, and at the end of 90 days. We see that 1.8 MB, 3 MB experience degradation but quickly recovers from the congestion event, however the 12 MB filter is unperturbed throughout. As expected, the FPR is significant for the smaller filters and decreases for larger filters. The 1.8 MB, 3 MB and 12 MB filters process transactions with 99.698%, 99.914% and 99.997% accuracy.

Dynamic Adaptation for Congestion. Our next strategy considers a more dynamic strategy. We initialize a counter to track total number of transactions in `CbTxFilter`, and as soon as filter capacity of 200,000 is exceeded, additional bloom filters are generated recursively as per demand [46] [97]. The user can tweak the filter capacity parameter as per resources available at the node. The filters can then be expired in the order of their age at defined intervals. New filters can be generated repeatedly if congestion persists.

Figs. 10d–10f show the results for filter sizes 600 kB, 1 MB and 4 MB filter sizes. In all three case, additional filters are generated at the 200k, 400k, and 600k transaction marks. This means that four filters are functional at the peak of the congestion event (consuming 2.4 MB, 4 MB and 16 MB respectively). Each filter expires 14 days after it was generated. When transaction volume normalizes, additional filters are no longer needed. We also highlight the average FPR at three key points: when transactions cross 600,000, when the first filter expires, and at the end of 90 days.

As expected, the false positive rate decreases with increasing filter sizes for the duration of the congestion period and otherwise: the 600 kB, 1 MB and 4 MB filters process transactions with 99.222%, 99.766% and 99.994% accuracy over the 90 day period. We see that the 600 kB and 1 MB filters experience significant performance degradation but quickly recover from the congestion event, whereas the 4 MB filter does not deviate much. Even in the worst case scenario, for the 600 kB filter false positive rates over the 55 hour period of congestion, the cumulative false positive rate is 4.6×10^{-2} . Medium size filters of size 1 MB can process this load with a false positive rate of approximately 1.1×10^{-3} .

Both approaches perform well in combating increased spam events and have their pros and cons. The first strategy of preempting congestion performs slightly better in terms of false positives but has a larger memory footprint. The second approach dynamically adjusts to congestion and has a considerably smaller memory footprint. Overall, though a transaction load of 600,000, over three times the historic maximum witnessed on the network, is estimated to push mempool space over the 1 GB mark, but Carbyne processes it within a few tens of MBs with extremely high accuracy.

10. Conclusion and Future Work

In this paper, we propose Carbyne, an ultra-lightweight counting bloom filter-based scheme to improve the performance and resilience of the Bitcoin network to increased transaction flows, network congestion and spam.

Carbyne reduces the footprint of the mempool by two orders of magnitude while preserving key functions for processing and forwarding transactions. We devise a mechanism to expire transactions based on age, tracking and limiting double-spends, and explore potential strategies to enable Replace-by-Fee transactions. We also demonstrate strategies to cope with congestion and spam. Implementing Carbyne does not necessitate forking the network.

Carbyne supports typical real-world transaction volumes of 300 MB in as little as 3 MB of memory, with more than 99.9% accuracy in processing and forwarding transactions. In simulated stress tests, Carbyne is demonstrated to cope with congestion and spam attacks with a total footprint of around 9 MB as opposed to around 1 GB in Bitcoin Core, and with very high fidelity. Our experimental results are obtained using Bitcoin transaction data over 90 days. This dataset is a distinct contribution, of independent interest to researchers.

In future work, we intend to further explore and fine tune trade offs in memory, accuracy, and computation. We hope to develop a functional prototype for live deployment. We are also currently adapting Carbyne to Ethereum.

We hope this effort motivates research on the mempool and its intricacies and contributes to the security and robustness of the Bitcoin ecosystem.

References

- [1] R. Bayer, "Symmetric binary B-trees: Data structure and maintenance algorithms," *Acta Informatica*, vol. 1, no. 4, pp. 290–306, 1972.
- [2] H. B. Haq, S. T. Ali, A. Salman, P. McCorry, and S. F. Shahandashti, "Neonpool: Reimagining Cryptocurrency Transaction Pools for Lightweight Clients and IoT Devices," *arXiv preprint*, vol. 2412, p. 16217, 2024. [Online]. Available: <https://arxiv.org/pdf/2412.16217>
- [3] R. Patgiri, S. Nayak, and S. K. Borgohain, "Hunting the pertinency of bloom filter in computer networking and beyond: A survey," *Journal of Computer Networks and Communications*, vol. 2019, 2019.
- [4] J. Larisch, D. Choffnes, D. Levin, B. M. Maggs, A. Mislove, and C. Wilson, "CRLite: A scalable system for pushing all TLS revocations to all browsers," in *Proc. IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 539–556.
- [5] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor: Randomized aggregatable privacy-preserving ordinal response," in *Proc. ACM SIGSAC Conference on Computer and Communications Security*, 2014, pp. 1054–1067.
- [6] S. Nayak, R. Patgiri, and A. Borah, "A survey on the roles of Bloom Filter in implementation of the Named Data Networking," *Computer Networks*, vol. 196, p. 108232, 2021.
- [7] Boost Multi-Index, "Boost.MultiIndex Documentation - Performance," Oct. 2004. [Online]. Available: https://cs.brown.edu/~jwicks/boost/libs/multi_index/doc/performance.html
- [8] J. Waldo, "A hitchhiker's guide to the blockchain universe," *Communications of the ACM*, vol. 62, no. 3, pp. 38–42, 2019.
- [9] M. Pisa, "Reassessing expectations for blockchain and development," *Innovations: Technology, Governance, Globalization*, vol. 12, no. 1–2, pp. 80–88, 2018.
- [10] T. Gayvoronskaya and C. Meinel, "Projects and Application Areas of Blockchain Technology," in *Blockchain*, Springer, 2021, pp. 79–96.
- [11] Nasdaq, "Can Bitcoin Grow Faster Than the Internet," Jul. 2021. [Online]. Available: <https://www.nasdaq.com/articles/can-bitcoin-grow-faster-than-the-internet-2021-05-07>

- [12] S. T. Ali, P. McCorry, P. H.-J. Lee, and F. Hao, "ZombieCoin 2.0: managing next-generation botnets using Bitcoin," *International Journal of Information Security*, vol. 17, no. 4, pp. 411–422, 2018.
- [13] D.-Y. Kim, E. Meryam, and H. Ju, "Examining Bitcoin mempools resemblance using Jaccard similarity index," in *Proc. Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2020, pp. 287–290.
- [14] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *Proc. IEEE P2P*, 2013, pp. 1–10.
- [15] K. Joshi, V. Fernando, and S. Misailovic, "Statistical algorithmic profiling for randomized approximate programs," in *Proc. IEEE/ACM International Conference on Software Engineering (ICSE)*, 2019, pp. 608–618.
- [16] G. Yu, X. Wang, K. Yu, W. Ni, J. A. Zhang, and R. P. Liu, "Survey: Sharding in blockchains," *IEEE Access*, vol. 8, pp. 14155–14181, 2020.
- [17] G. Wang, Z. J. Shi, M. Nixon, and S. Han, "SoK: Sharding on blockchain," in *Proc. ACM Conference on Advances in Financial Technologies*, 2019, pp. 41–61.
- [18] S. Park, S. Im, Y. Seol, and J. Paek, "Nodes in the bitcoin network: Comparative measurement study and survey," *IEEE Access*, vol. 7, pp. 57009–57022, 2019.
- [19] L. L. Gremillion, "Designing a Bloom filter for differential file access," *Communications of the ACM*, vol. 25, no. 9, pp. 600–604, 1982.
- [20] J. K. Mullin, "A second look at Bloom filters," *Communications of the ACM*, vol. 26, no. 8, pp. 570–571, 1983.
- [21] P. Bose, H. Guo, E. Kranakis, A. Maheshwari, P. Morin, J. Morrison, M. Smid, and Y. Tang, "On the false-positive rate of Bloom filters," *Information Processing Letters*, vol. 108, no. 4, pp. 210–213, 2008.
- [22] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.
- [23] J. Thomas, "Bitcoin Transaction Volume Primed to Overtake PayPal in 2020," *BeInCrypto*, Feb. 2020. [Online]. Available: <https://beincrypto.com/bitcoin-transaction-volume-primed-to-overtake-paypal-in-2020>
- [24] "Your BTC transaction is stuck in the mempool? Here's what you can do," *CoinGate*, Nov. 2020. [Online]. Available: <http://blog.coingate.com/2020/11/btc-mempool-stuck/>
- [25] "Cryptocurrency Market Capitalizations," *CoinMarketCap*, Jan. 2022. [Online]. Available: <https://coinmarketcap.com>
- [26] "Home," *Ethereum.org*, Jan. 2022. [Online]. Available: <http://ethereum.org>
- [27] "Instantly Move Money to All Corners of the World," *Ripple*, Apr. 2020. [Online]. Available: <https://ripple.com>
- [28] G. Hileman and M. Rauchs, "Global cryptocurrency benchmarking study," *Cambridge Centre for Alternative Finance*, vol. 33, 2017.
- [29] B. Bambrough, "As Bitcoin Nudges \$8,000, Survey Reveals The 'Rapid' Pace Of Crypto Adoption," *Forbes*, May 2019. [Online]. Available: <https://www.forbes.com/sites/billybambrough/2019/05/13/as-bitcoin-nears-8000-survey-reveals-the-rapid-pace-of-crypto-adoption>
- [30] "HSB Survey Finds One-Third of Small Businesses Accept Cryptocurrency," Apr. 2020. [Online]. Available: <https://www.businesswire.com/news/home/20200115005482/en/HSB-Survey-Finds-One-Third-Small-Businesses-Accept>
- [31] R. Schultze-Kraft, "How Many Entities Hold Bitcoin?," *Glassnode*, Jan. 2020. [Online]. Available: <https://medium.com/glassnode-insights/how-many-entities-hold-bitcoin-e945ecc5d0a1>
- [32] O. Faridi, "700% Increase in Bitcoin Adoption Worldwide, Kaspersky's Survey Reveals," *CryptoGlobe*, Feb. 2019. [Online]. Available: <https://www.cryptoglobe.com/latest/2019/02/700-increase-in-bitcoin-adoption-worldwide-kaspersky-s-survey-reveals>
- [33] M. Saad, L. Njilla, C. Kamhoua, J. Kim, D. Nyang, and A. Mohaisen, "Mempool Optimization for Defending Against DDoS Attacks in PoW-based Blockchain Systems," in *Proc. IEEE ICBC*, 2019, pp. 285–292.
- [34] A. P. Ozisik et al., "Graphene: Efficient Interactive Set Reconciliation Applied to Blockchain Propagation," in *Proc. ACM SIGCOMM*, 2019, pp. 303–317.
- [35] J. Poon and T. Dryja, "The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments," 2016. [Online]. Available: <https://lightning.network/lightning-network-paper.pdf>
- [36] M. Saad, J. Kim, D. Nyang, and D. Mohaisen, "Contra-*: Mechanisms for Countering Spam Attacks on Blockchain Memory Pools," *arXiv preprint arXiv:2005.04842*, 2020.
- [37] L. Yang et al., "Prism: Scaling Bitcoin by 10,000x," *arXiv preprint arXiv:1909.11261*, 2019.
- [38] I. Eyal et al., "Bitcoin-NG: A scalable blockchain protocol," in *Proc. USENIX NSDI*, 2016, pp. 45–59.
- [39] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey," *Internet Mathematics*, vol. 1, no. 4, pp. 485–509, 2004.
- [40] E. Georgiadis, "How Many Transactions per Second Can Bitcoin Really Handle? Theoretically," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 416, 2019.
- [41] Zamyatin, Alexei and Avarikioti, Zeta and Perez, Daniel and Knottenbelt, William J. TxChain: Efficient Cryptocurrency Light Clients via Contingent Transaction Aggregation. *IACR Cryptol. ePrint Arch.*, 2020, 580.
- [42] Wang, Yunpeng and Yang, Jin and Li, Tao and Zhu, Fangdong and Zhou, Xiaojun. Anti-Dust: A Method for Identifying and Preventing Blockchain's Dust Attacks. In *Proc. of 2018 Int. Conf. on Information Systems and Computer-Aided Education (ICISCAE)*, IEEE, 2018, pp. 274–280.
- [43] Bitcoin wiki - Network. Available: <https://en.bitcoin.it/wiki/Network>, Jan. 2022.
- [44] Transaction Replacement. Available: https://en.bitcoin.it/wiki/Transaction_replacement, Jan. 2022.
- [45] Bitcoin Core, Compact Blocks FAQ. Available: <https://bitcoincore.org/en/2016/06/07/compact-blocks-faq/>.
- [46] Guo, Deke and Wu, Jie and Chen, Honghui and Yuan, Ye and Luo, Xueshan. The dynamic bloom filters. *IEEE Trans. on Knowledge and Data Engineering*, vol. 22, no. 1, pp. 120–133, 2009.
- [47] Yoon, MyungKeun. Aging bloom filter with two active buffers for dynamic sets. *IEEE Trans. on Knowledge and Data Engineering*, vol. 22, no. 1, pp. 134–138, 2009.

- [48] Naor, Moni and Yogev, Eylon. Sliding bloom filters. In *Proc. of Int. Symp. on Algorithms and Computation*, Springer, 2013, pp. 513–523.
- [49] Blockchain Charts - Median Confirmation Time. Available: <https://www.blockchain.com/charts/median-confirmation-time>, Jan. 2022.
- [50] Blockchain Charts - Average Transaction Confirmation Time. Available: <https://www.blockchain.com/charts/avg-confirmation-time>, Jan. 2022.
- [51] Mempool Transaction Count. Available: <https://www.blockchain.com/charts/mempool-count>, Jan. 2022.
- [52] Mempool Transaction Size. Available: <https://www.blockchain.com/charts/mempool-size>, Jan. 2022.
- [53] Saad, Muhammad and Spaulding, Jeffrey and Njilla, Laurent and Kamhoua, Charles and Shetty, Sachin and Nyang, Dae Hun and Mohaisen, David. Exploring the Attack Surface of Blockchain: A Comprehensive Survey. *IEEE Commun. Surveys & Tutorials*, 2020.
- [54] Caffyn, Grace. Bitcoin Network Stress Test Could Occur Next Week. Available: <https://www.coindesk.com/bitcoin-network-stress-test-could-occur-next-week>, Sept. 2015.
- [55] Caffyn, Grace. Bitcoin Node Numbers Fall After Spam Transaction ‘Attack’. Available: <https://www.coindesk.com/bitcoin-node-numbers-fall-after-spam-transaction-attack>, Oct. 2015.
- [56] Carter, J. L. and Wegman, M. N. Universal classes of hash functions. *J. Comput. and Syst. Sci.*, vol. 18, no. 2, pp. 143–154, 1979.
- [57] Matzutt, Roman and Kalde, Benedikt and Pennekamp, Jan and Drichel, Arthur and Henze, Martin and Wehrle, Klaus. How to Securely Prune Bitcoin’s Blockchain. In *Proc. of 2020 IFIP Networking Conf.*, IEEE, 2020, pp. 298–306.
- [58] Bünz, Benedikt and Kiffer, Lucianna and Luu, Loi and Zaman, Mahdi. Flyclient: Super-light clients for cryptocurrencies. In *Proc. of 2020 IEEE Symp. on Security and Privacy (SP)*, IEEE, 2020, pp. 928–946.
- [59] Gervais, Arthur and Capkun, Srdjan and Karame, Ghassan O and Gruber, Damian. On the privacy provisions of bloom filters in lightweight bitcoin clients. In *Proc. of the 30th Annual Computer Security Applications Conf.*, 2014, pp. 326–335.
- [60] Mišić, Jelena and Mišić, Vojislav B and Chang, Xiaolin. On the benefits of compact blocks in Bitcoin. In *Proc. of ICC 2020 IEEE Int. Conf. on Commun. (ICC)*, 2020, pp. 1–6.
- [61] Franzoni, Federico and Daza, Vanesa. SoK: Network-Level Attacks on the Bitcoin P2P Network. *IEEE Access*, vol. 10, pp. 94924–94962, 2022.
- [62] Apostolaki, Maria and Zohar, Aviv and Vanbever, Laurent. Hijacking bitcoin: Routing attacks on cryptocurrencies. In *Proc. of 2017 IEEE Symp. on Security and Privacy (SP)*, IEEE, 2017, pp. 375–392.
- [63] Bruce, J. D. The Mini-Blockchain Scheme. *White Paper*, 2014.
- [64] Wood, Gavin. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.
- [65] Hearn, M. and Corallo, M. BIP37: Connection Bloom filtering. Available: <https://github.com/bitcoin/bips/blob/master/bip-0037.mediawiki>, 2012.
- [66] Haqshanas, Ruholamin. Solana’s Network Congestion Prompts Liquidations, Drives Away Users. Available: <https://cryptonews.com/news/solanas-network-congestion-prompts-liquidations-drives-away-users.htm>, Dec. 2022.
- [67] Binance Is Facing Issues with Solana Withdrawals. Available: <https://www.financemagnates.com/cryptocurrency/news/binance-is-facing-issues-with-solana-withdrawals>, Dec. 2022.
- [68] Bitcoin Network Monitor - DSN Research Group, KASTEL @ KIT. Available: <https://www.dsn.kastel.kit.edu/bitcoin/index.html>, Apr. 2022.
- [69] RapidJSON Documentation. Available: <https://rapidjson.org>, 2015.
- [70] Frey, D., Makkes, M. X., Roman, P.-L., Taïani, F., and Voulgaris, S. Dietcoin: Hardening bitcoin transaction verification process for mobile devices. *Proceedings of the VLDB Endowment (PVLDB)*, 12(12):1946–1949, 2019.
- [71] Kattis, A. and Bonneau, J. Proof of necessary work: Succinct state verification with fairness guarantees. *Cryptology ePrint Archive*, 2020.
- [72] Rottenstreich, O. Sketches for blockchains. In *2021 International Conference on COMMunication Systems & NETWORKS (COMSNETS)*, pages 254–262. IEEE, 2021.
- [73] 250+ companies and stores that accept cryptocurrency. Bit Pay, 2023. Available at: <https://bitpay.com/directory>.
- [74] Baqer, K., Huang, D. Y., McCoy, D., and Weaver, N. Stressing out: Bitcoin “stress testing”. In *International Conference on Financial Cryptography and Data Security*, pages 3–18. Springer, 2016.
- [75] Jiang, S., Li, J., Gong, S., Yan, J., Yan, G., Sun, Y., and Li, X. BZIP: A Compact Data Memory System for UTXO-based Blockchains. In *2019 IEEE International Conference on Embedded Software and Systems (ICESSE)*, pages 1–8. IEEE, 2019.
- [76] Nakamoto, S. Bitcoin P2P e-cash paper. *The Cryptography Mailing List*, 2008.
- [77] "Transactions Bitcoin, Raw Transactions format." Available at: <https://developer.bitcoin.org/reference/transactions.html>
- [78] Kiayias, A., Miller, A., and Zindros, D. Non-interactive proofs of proof-of-work. In *International Conference on Financial Cryptography and Data Security*, pages 505–522. Springer, 2020.
- [79] Naor, M. and Yogev, E. Bloom filters in adversarial environments. *ACM Transactions on Algorithms (TALG)*, 15(3):1–30, 2019.
- [80] Han, Y., Li, C., Li, P., Wu, M., Zhou, D., and Long, F. Shrec: Bandwidth-efficient transaction relay in high-throughput blockchain systems. In *Proceedings of the 11th ACM Symposium on Cloud Computing*, 2020.
- [81] Naumenko, G., Maxwell, G., Wuille, P., Fedorova, A., and Beschastnikh, I. Erelay: Efficient transaction relay for bitcoin. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 817–831. 2019.
- [82] "Blockchain.com | Charts - Mempool Size (Bytes)," December 2024. Available at: <https://www.blockchain.com/explore/r/charts/mempool-size>.

- [83] P. Chatzigiannis, F. Baldimtsi, and K. Chalkias, "SoK: Blockchain light clients," in **Financial Cryptography and Data Security**, Springer, 2022, pp. 615–641.
- [84] Eduardo et al., "Fighting under-price DoS attack in Ethereum with machine learning techniques," **ACM SIGMETRICS Performance Evaluation Review**, vol. 48, no. 4, pp. 24–27, 2021.
- [85] K. Li, Y. Wang, and Y. Tang, "Deter: Denial of Ethereum txpool services," in **Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security**, ACM, 2021, pp. 1645–1667.
- [86] Bitcoin source code. GitHub, 2021. Available at: <https://github.com/bitcoin/bitcoin/blob/master/src/txmempool.h>.
- [87] Ethereum, go-ethereum. GitHub, 2023. Available at: <https://github.com/ethereum/go-ethereum/blob/master/light/txpool.go>.
- [88] libbf Bloom filters for C++11. Available at: <http://mavam.github.io/libbf>.
- [89] Jochen Hoenicke, "Johoe's Bitcoin Mempool Size Statistics," Available at: <https://test.jochen-hoenicke.de/queue/#BTC,all,weight>.
- [90] "The 300 MB default maxmempool Problem," December 2017. Available at: <https://b10c.me/blog/001-the-300mb-default-maxmempool-problem/>.
- [91] "Glassnode Studio - On-Chain Market Intelligence," December 2024. Available at: <https://studio.glassnode.com/charts/transactions.TxTypesBreakdownRelative?a=ETH&category=&ema=0&mAvg=7&mMedian=0&pScl=log&s=1667924083&u=1675700083&zoom=90>.
- [92] Etherscan.io, "Daily Pending Transactions | Etherscan," Ethereum (ETH) Blockchain Explorer, December 2024. Available at: <https://etherscan.io/dashboards/daily-pending-tx>.
- [93] Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., and Saxena, P. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30, 2016.
- [94] Kim, J.-Y., Lee, J., Koo, Y., Park, S., and Moon, S.-M. Ethanos: Efficient bootstrapping for full nodes on account-based blockchain. In *Proceedings of the Sixteenth European Conference on Computer Systems*, pages 99–113, 2021.
- [95] Ethereum nodes and clients. Ethereum, 2023. Available at: <https://ethereum.org/en/developers/docs/nodes-and-clients>.
- [96] MSVC's implementation of the C++ Standard Library. GitHub, 2023. Available at: <https://github.com/microsoft/STL>.
- [97] Beyer, K. S., Rajagopalan, S., and Zubiri, A. System and method for generating and using a dynamic bloom filter. Google Patents, US Patent 7,937,428, May 2011.
- [98] Nasdaq, "About 46 Million Americans Now Own Bitcoin," July 2021. [Online]. Available: <https://www.nasdaq.com/articles/about-46-million-americans-now-own-bitcoin-2021-05-14>.
- [99] C. Decker, R. Russell, and O. Osuntokun, "Eltoo: A simple layer 2 protocol for Bitcoin," June 2018. [Online]. Available: <https://blockstream.com/eltoo.pdf>.
- [100] T. Dryja, "Utreexo: A Dynamic Hash-Based Accumulator Optimized for the Bitcoin UTXO Set," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 611, 2019.
- [101] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 51–68.
- [102] F. Memoria, "\$700 Million Stuck in 115,000 Unconfirmed Bitcoin Transactions," November 2017. [Online]. Available: <https://www.ccn.com/700-million-stuck-115000-unconfirmed-bitcoin-transactions/>.
- [103] A. Zmudzinski, "Bitcoin's Mempool Saw an Anomalous Number of Big Transactions on Friday," November 2019. [Online]. Available: <https://coingecko.com/news/bitcoin-mempool-saw-an-anomalous-number-of-big-transactions>.
- [104] MIT Digital Currency Initiative, "Bitcoin's (un)common good," MIT Media Lab, February 2021. [Online]. Available: <https://dci.mit.edu/research/2021/2/25/dci-bitcoin-security-effort>.
- [105] B. Pirus, "New HTC Exodus Able To Run Full Bitcoin Node," October 2019. [Online]. Available: <https://www.forbes.com/sites/benjaminpirus/2019/10/19/new-htc-exodus-able-to-run-full-bitcoin-node/#19660abe27d2>.
- [106] L. Xu, L. Chen, Z. Gao, S. Xu, and W. Shi, "EPBC: Efficient public blockchain client for lightweight users," in *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, 2017, pp. 1–6.
- [107] S. O'Neal, "Ethereum Topped Bitcoin in Network Daily Fees Over Weekend," June 2020. [Online]. Available: <https://coingecko.com/news/ethereum-topped-bitcoin-in-network-daily-fees-over-weekend>.
- [108] B. Dale, "Mempool Manipulation Enabled Theft of \$8M in MakerDAO Collateral on Black Thursday: Report - CoinDesk," *CoinDesk*, July 2020. [Online]. Available: <https://www.coindesk.com/mempool-manipulation-enabled-theft-of-8m-in-makerdao-collateral-on-black-thursday-report>.
- [109] K. Sedgwick, "200,000 Unconfirmed Transactions Pile Up in Another Crazy Day for Bitcoin," December 2019. [Online]. Available: <https://news.bitcoin.com/200000-unconfirmed-transactions-pile-up-another-crazy-day-bitcoin/>.
- [110] C. Harper, "Dust Attacks Make a Mess in Bitcoin Wallets, but There Could Be a Fix," August 2020. [Online]. Available: <https://www.nasdaq.com/articles/dust-attacks-make-a-mess-in-bitcoin-wallets-here-could-be-a-fix-2020-08-18>.
- [111] Solana, "9-14 Network Outage Initial Overview," *Solana | News*, September 2021. [Online]. Available: <https://solana.com/news/9-14-network-outage-initial-overview>.
- [112] D. Guo, Y. Liu, X. Li, and P. Yang, "False negative problem of counting Bloom filter," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 5, pp. 651–664, 2010.
- [113] Litecoin, "Litecoin - Open source P2P digital currency," July 2019. [Online]. Available: <https://litecoin.org>.
- [114] Bitcoin.org, "Running a full-node. Support the Bitcoin network by running your own full node," January 2022. [Online]. Available: <https://bitcoin.org/en/full-node#minimum-requirements>.
- [115] Bitcoin Wiki, "Bitcoin Wiki - Protocol Rules," January 2022. [Online]. Available: https://en.bitcoin.it/wiki/Protocol_rules.

- [116] Bitcoin Core, "Bitcoin Core version 0.14.0 released," March 2017. [Online]. Available: <https://bitcoin.org/en/release/v0.14.0>.
- [117] L. Fan, P. 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.
- [118] Y. Zhao and J. Wu, "The design and evaluation of an information sharing system for human networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 3, pp. 796–805, 2013.
- [119] Solana, "Solana Status on Twitter," *Twitter*, September 2021. [Online]. Available: <https://twitter.com/solanastatus/status/1437856639441424395?lang=en>.
- [120] Segwit, "Segregated Witness - Bitcoin Wiki," January 2022. [Online]. Available: https://en.bitcoin.it/wiki/Segregated_Witness.
- [121] S. Cao, S. Kadhe, and K. Ramchandran, "CoVer: Collaborative Light-Node-Only Verification and Data Availability for Blockchains," in *2020 IEEE International Conference on Blockchain (Blockchain)*, 2020, pp. 45–52.
- [122] D. Vandervort, "Challenges and opportunities associated with a bitcoin-based transaction rating system," in *International Conference on Financial Cryptography and Data Security*, 2014, pp. 33–42.
- [123] C. Wueest, "The continued rise of DDoS Attacks," *White Paper: Security Response, Symantec Corporation*, 2014.
- [124] C. Rahalkar and A. Virgaonkar, "Summarizing and Analyzing the Privacy-Preserving Techniques in Bitcoin and other Cryptocurrencies," *arXiv preprint arXiv:2109.07634*, 2021.
- [125] Bitcoin Core, "Bitcoin Core 0.11 (ch 2): Data Storage - Bitcoin Wiki," June 2020. [Online]. Available: [https://en.bitcoin.it/wiki/Bitcoin_Core_0.11_\(ch_2\):_Data_Storage](https://en.bitcoin.it/wiki/Bitcoin_Core_0.11_(ch_2):_Data_Storage).
- [126] Raw Transactions | Ethereum.org. Ethereum. Available: <https://ethereum.org/en/developers/docs/transactions>
- [127] Ethereum Transactions Message. devp2p. GitHub. Available: <https://github.com/ethereum/devp2p/blob/master/caps/eth.md#transactions-0x02>
- [128] H. B. Haq, T. Ahmad, A. Buriro, and S. Ullah, *Neonpool: Reimagining Cryptocurrency Transaction Pools for Lightweight Clients and IoT Devices*, 2024. [Online]. Available: https://drive.google.com/drive/folders/1KkjPxnI7NvWYq1Z3j1rcCGhYxwUzbXEJ?usp=drive_link
- [129] Chepurnoy, Alexander, Charalampos Papamanthou, Shraavan Srinivasan, and Yupeng Zhang. "Edrax: A cryptocurrency with stateless transaction validation." *Cryptology ePrint Archive* (2018).
- [130] "P2P Network—Bitcoin, Inventory Messages." April 2021. Available at: <https://developer.bitcoin.org/reference/p2p-networking.html>.
- [131] Bianchi, Giuseppe, Nico d'Heureuse, and Saverio Niccolini. "On-demand time-decaying bloom filters for telemarketer detection." *ACM SIGCOMM Computer Communication Review* 41, no. 5 (2011): 5-12. ACM New York, NY, USA.
- [132] Bitcoin Network Guide. "P2P Network Guide - Bitcoin." May 2020. Available at: <https://bitcoin.org/en/p2p-network-guide>.

Content	Data Points	Size
Mempool Activity	29 million	40 GB
Network Inventory	89 million	10 GB
Mempool Statistics	12962 readings	4 MB

Table 4
MempoolState Dataset (Jan 01, 2021 – 31 Mar 2021)

A. *MempoolState* Dataset

To date, data-driven research in cryptocurrencies has mostly focused on price fluctuations and economic trends, analyzing cryptocurrency transaction graphs, and mapping network topology. To the best of our knowledge there is no publicly released dataset which studies the network state of cryptocurrency networks. Recording live network state allows us to reconstruct network state at the client and replay network activity for simulation and modeling purposes. This has several useful applications which include the following: *Mining strategies* to maximize the profit earned through transaction fee; *Retroactively study network state* through the dataset spanning a 90-day period, with network message entries; *Time-based snapshots* can be recreated to visualize network view for a particular node; *Comparison among different nodes* can be made, for instance between the mempool state of different nodes; *Identification of anomalies* can be undertaken for debugging and flagging suspicious activity; *Forensic Investigation* can be undertaken by researchers in the wake of incidents (like the 'dust' attack of 2015 and onwards); *Client-side optimizations* and security defences can be developed and tested; *Network-Wide Impact* of community policies like pay-per-fee can be studied.

Our *MempoolState* Dataset comprises three datasets, with dimensions listed in Table 4, and details as follows:

- **Mempool Activity** includes all transaction entries and exits in the mempool in JSON format. Code written in C++ was used to modify Bitcoin Core, specifically `src/txmempool.cpp`, to capture the data. For our 90-day dataset, we log ~29 million unique transactions with ~88 million inputs. The resulting dataset is sized at 40 GB.
- **Network Inventory** includes the `inv` messages received over the network. We log ~89 million `inv` messages via the `-network` flag in the Bitcoin configuration file. The logs generated by Bitcoin included other network messages as well, such as `getdata` and `tx`. To prevent the size of the logs from becoming intractable we maintained daily log files using *Logrotate* in Linux. We wrote C++ code to scan these files and extract only those `inv` messages relevant to our application. The resulting `inv` messages along with timestamp are stored in CSV format. The resulting dataset is sized at 10 GB.
- **Mempool Statistics** includes details for the raw transaction size, resulting memory usage and transaction count in the mempool. We used a Python script to invoke the Bitcoin daemon's JSON-RPC `getmempoolinfo` method at 10 minute intervals and store the output as JSON. We use another Python script to scrape the JSON data from these files into a single CSV file for ease of plotting.

The *MempoolState* Dataset is accompanied by the following tools we have developed:

- **Parallel Simulations for Carbyne and Bitcoin** This code tool simulates the Bitcoin Core and Carbyne mempools in parallel to evaluate their performance. Carbyne simulation

code is written in C++. It uses the library libbf [88] for counting Bloom filter and the library RapidJSON [69] to support JSON operations in C++. This code requires the **Mempool Activity** and **Network Activity** datasets as inputs. It logs performance at hourly intervals and generates hourly as well as cumulative stats in CSV format. It also generates forensics data to help us analyze the false positive and false negatives in detail in CSV format.

- **Computation Time.** This code benchmarks the querying, insertion and deletion times for `mapTx`, `mapLinks` and `mapNextTx` as well as `CbTxFilter` and `CbTxInputs Filter`. It uses C++ STL `std::chrono` to perform the computation time analysis.
- **Snapshot.** This code written in C++ simulates the Bitcoin Core mempool. It recreates the state of the Bitcoin Core mempool for any given time instant. It can be used for application scenarios described above.
- **mapRelay** This code written in C++ using the network logs recreates the `mapRelay` to report the average number of transactions in `mapRelay` over an hour.
- **Reconstruct** This code, written in C++ is used to check how multiple Carbyne nodes may be used to bootstrap the mempool of a newly connected node. We run experiments where Carbyne nodes store 5%, 10%, 20%, 25%, 33% and 50% of transactions on a random basis. Similarly, we vary the number of Carbyne nodes that a new node that joins the network connects to 4,8,12.
- **Carbyne.** This code, written in C++, runs the Carbyne mempool. It can be used as a starting point to prototype Carbyne or integrate it into another client.

B. Storing Full Transactions

By optimizing the mempool, Carbyne users may still choose to store subsets of full transactions in RAM for various purposes, depending on the resources available to them.

For instance, users may store full transactions for mining. The Bitcoin Core protocol limits blocks to 1 MB in size. Carbyne nodes may maintain a live pool of 1 MB of current transactions and prioritize them on the basis of fee per size, value, age of inputs, number of ancestors/ descendants etc. and propose blocks.

A subset of transactions may be saved in RAM to bootstrap newly connecting nodes. If multiple Carbyne nodes on the network each randomly store a subset of full transactions, they can collectively bootstrap mempool of new nodes.

We undertake a preliminary analysis using our *MempoolState* dataset. We run experiments where Carbyne nodes store 5%, 10%, 20%, 25%, 33% and 50% of total transactions they receive over a 90 day period. We also vary the number of Carbyne nodes that new nodes may connect to. Our results show: if a new node connects to 4,8 and 12 Carbyne nodes each maintaining 10% of randomly selected current transactions, the new node can recover 33%, 56% and 70% mempool transactions respectively as shown in Fig. 11.

C. Transaction Retention

When an `entryTx` and its inputs are successfully verified, the transaction is added to Carbyne mempool as described in §4. The transaction hash `TxID` is then broadcast to the node's peers with an `inv` message and full transactions are forwarded on request. In Bitcoin's diffusion protocol (a variation on random flooding) peers

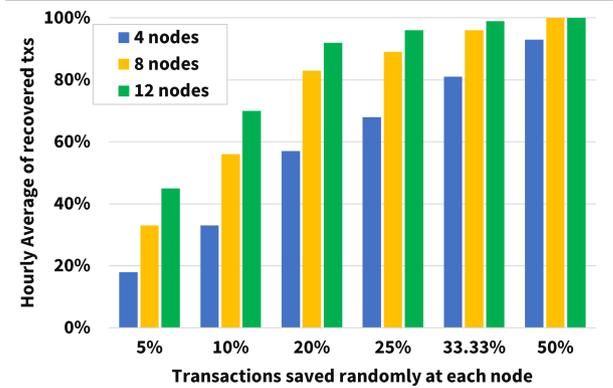
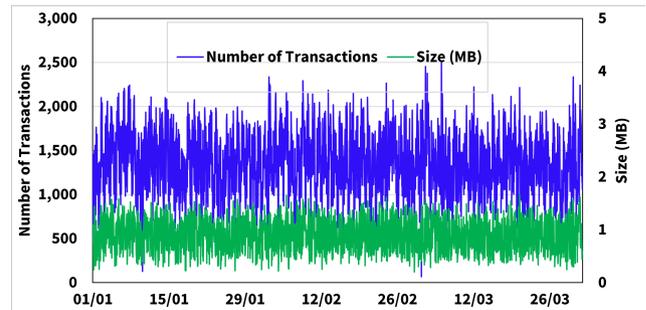
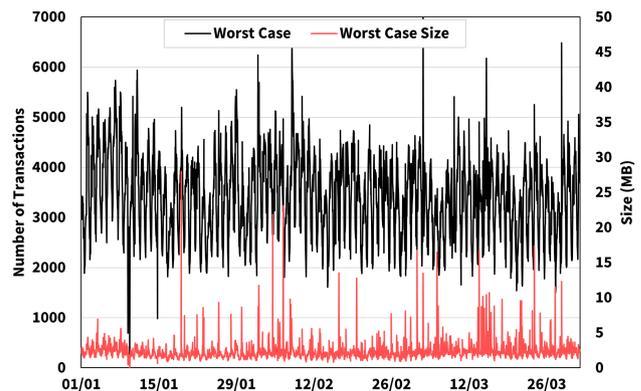


Figure 11: Recovery Rate



(a) mapRelay (Bitcoin Core)



(b) mapRelay (Worst Case)

Figure 12: mapRelay

inject a random delay before announcing a received transaction to its peers, to mitigate timing attacks and in-flight collisions.

Bitcoin segregates the cache of transactions for mempool and relay in separate containers. The data structure `mapRelay` is responsible for keeping transactions until they are relayed to peers. This memory consumption comes under the umbrella of networks and connections and is in addition to the memory incurred by the mempool transactions. The length of the interval that full transactions are stored can vary: it is stored until it is relayed to all peers, or a 15 minute default expiry can also be configured by the user. This structure is part of Bitcoin Core's network and

connection handling mechanism, it is independent of the Bitcoin mempool, and our solution retains it as it is.

We undertake experiments to quantify the trade-off between retention time and memory consumption of `mapRelay`. Our node ran on default settings with 8 outbound peers and upto 125 inbound peers. Firstly, we estimate the number of transactions in `mapRelay` at any given time using the network logs collected over the 90 day period. We observe that the number of these transactions average at 1000, with a cap at around 2500 occupying no more than 2 MB at any given instant as shown in Fig. 12a. Secondly, we estimate the worst case scenario where each transaction stays in `mapRelay` for 15 mins. This comes out to be 3,500 transactions on average and no more than 7000 at maximum, as shown in Fig. 12b. However as empirical data shows it takes almost 13 seconds on average for a transaction to propagate 90% Bitcoin nodes [68]. Hence we do not anticipate the memory consumption of `mapRelay` to be substantial.

transaction as the transaction inputs, outputs and fee all need to be stored.

We defer the detailed empirical analysis of these strategies as future work.

D. Replace-by-Fee

Replace-By-Fee (RBF) is an opt-in node policy that allows an unconfirmed transaction in a mempool to be replaced with a different transaction that spends at least one of the same inputs and pays a higher fee. RBF has multiple variants. Here we briefly discuss potential strategies for each.

- **Full RBF** unconditionally allows a transaction to replace older ones so long as it pays a sufficient fee. We can already address this within Carbyne in a sense by incrementing the threshold value of the `CbTxInputsFilter` counter to 2. The intuition here is that if a transaction input already exists in `CbTxInputsFilter`, then its respective counters will each be 1 or more. When the corresponding RBF transaction arrives, it is accepted and the counters are incremented to 2, after which more RBF transactions for this case will be dropped. A user can choose a custom threshold value. Moreover, at this point, a variable could also track the moving average of RBF transactions to detect potential misuse of the RBF provision, and halt processing these transactions.
- **Opt-in RBF** allows the replacement when the transactions being replaced have explicitly signalled they allow replacement via the "sequence" field. Dual-load bloom filter could be deployed that saves the transaction inputs, along with the original fee. While Bloom filters usually hold a single type of information, which is either the membership in a given set or the return values of elements, the proposed DLBF holds both the membership and the return values in a single Bloom filter. If a node has Opt-in RBF then it can initialize two separate bloom filters for transaction inputs, one which holds inputs of replaceable transactions and the other which holds inputs of transactions that can not be replaced. A transaction should only be replaced if the inputs are in the filter that signals RBF.
- **Delayed RBF** is a variant which allows transactions to be replaced unconditionally, but only after a given number of blocks have been mined since the replaced transactions were first seen by the node. A Dual-load bloom filter could be deployed that saves the transaction inputs, along with the block number when it is safe to replace the transaction.
- **First-seen-safe RBF** only allows the replacement if an additional criteria is met: the replacement transaction must pay all the same outputs as the transactions being replaced. If a node employs this policy then it is best to save the complete