A Secure, Efficient, and Transparent Network Architecture for Bitcoin

A. Pinar Ozisik[†], Gavin Andresen, George Bissias[†], Amir Houmansadr[†], Brian N. Levine[†]
[†]College of Information and Computer Sciences, UMass Amherst, MA

Abstract. *We contribute three complementary mechanisms that increase the security, efficiency, and transparency of blockchain systems. We evaluate the use of status report messages that, like canaries in a coal mine, allow peers to detect both malicious miners and eclipse attacks almost immediately. We outline a mechanism, using these reports, that allow blockchain users to quantify the risk of a double-spend attack within minutes, versus the several hours required by the current system.*

We also devise a novel method of interactive set reconciliation for efficient status reports and blocks. Our approach, called Graphene, couples a Bloom filter with an IBLT, and reduces traffic overhead by about 45%.

As an alternative for Bitcoin's inefficient and opaque peer-to-peer (p2p) architecture, we also introduce Canary that separates the network's data and control planes. Peers submit transactions directly to miners, who announce new blocks and transactions via distribution networks whose topology they manage. We show that Canary's tree-based topology reduces traffic overhead by about 30% compared to the current architecture. When Graphene is coupled with Canary, Bitcoin's traffic overhead is reduced by about 80%, while detecting eclipse attacks and increasing transparency.

1 Introduction

Blockchain-based currencies [20], such as Bitcoin, have seen widespread adoption despite several limitations not present in traditional financial systems, such as credit cards or cash. Transactions in these currencies allow merchants to accept coin in exchange for real goods; however, once a transaction has been submitted to the miners for validation, its status is opaque, at times for tens of minutes, until the transaction is confirmed by its presence in a new block. After a merchant releases goods, a customer can launch a *double-spend attack* [20] by publishing a longer branch of the blockchain that does not contain the original transaction, but one that moves the coin to a second account under her control. At that point, the customer has both the goods and her coin to spend again. Furthermore, well-known *eclipse attacks* [17] prevent a merchant from receiving fresh block and transaction data, increasing his vulnerability to other defrauding attacks. Currently, the only method by which a peer can detect eclipse attacks is by determining if he is receiving fresh block data from most miners.

Non-first authors are listed alphabetically.

Contributions. We contribute several complementary mechanisms to increase Bitcoin's security, efficiency, and transparency. Our proposed mechanisms work together or are separately deployable, and are applicable to other blockchain-based network protocols, such as Litecoin (https://litecoin.org) and Zerocash [23].

First, we propose a method for active mining pools to issue *status reports* every few minutes stating the block they are mining on top of. These reports of about 2–3 KB each, contain transaction IDs queued for a miner's next block. We then establish a mechanism using status reports for detecting eclipse attacks and quantifying the risk of falling victim to a double-spend attack within minutes, a determination that can take hours using the current network.

Second, we contribute an extremely efficient method of announcing new blocks and status reports called *Graphene*. Our blocks are a fraction of the size of related methods, such as Compact Blocks [7] and Xtreme Thinblocks [25]. For example, a 10 KB Compact Block can be encoded in 2.6 KB with Graphene. We use a novel interactive combination of Bloom filters [4] and IBLTs [14], providing a solution to the problem of set reconciliation that has applications beyond Bitcoin.

Finally, leveraging status reports, we introduce *Canary*, an alternative for Bitcoin's inefficient and opaque p2p architecture. Canary's architecture separates the network's data and control planes. Peers submit transactions directly to miners, who announce new blocks and transactions via distribution networks whose topology they manage. Canary's use of status reports allows for a structured topology that reduces traffic overhead by about 30% compared to the current network, yet also revealing malicious or uncooperative miners and full nodes. We evaluate performance empirically via our detailed network simulation that uses miner-managed trees of full nodes as a potential distribution network.

By coupling Graphene with Canary's tree-based topology, we show that traffic overhead is reduced by about 85% compared to today's network of Compact Blocks over a graph topology. Yet our approach has increased security as well.

We begin with an overview of the limitations of Bitcoin, and then we will describe our contributions. We assume a familiarity with blockchains and Bitcoin's operation; a detailed overview of Bitcoin's operation appears in Appendix A.

2 Limitations of Blockchains and Bitcoin's Architecture

Tschorsch et al. [26], Bonneau et al. [5], and Croman et al. [8] offer summaries of broad Bitcoin research issues. Among Bitcoin's limitations are its opaque status and network inefficiency. In this section, we review these issues and summarize related work.

Opaque block and transaction status. Bitcoin provides no method of querying the status of the network and its unconfirmed transactions. Therefore, it is never clear whether none, some, or all of the miners plan to include a submitted transaction in their next block. Similarly, when the blockchain is forked, there is no method to query the miners to determine which of them are working on each fork. Block updates can be infrequent at times: About 12% of new blocks are not found within 20 minutes, twice the target discovery period of 10 minutes, and

1% of blocks are not discovered within an hour. (Figure 10 in Appendix D shows historical delays.) Thus, even if a node does not see a new block for up to an hour, the delay does not necessarily imply that it is disconnected from the network. Furthermore, transactions do not always appear in the next block. About 80% of transactions pay at least 0.0001BTC in fees, but fail to receive initial confirmation in the next block about 22% of the time, and fail to receive initial confirmation within two blocks 15% of the time. (See Figure 11 in Appendix D.) We propose to address these problems via status reports that increase transparency and security. GHOST [24] is a cousin to our approach: it includes on-chain, orphaned blocks in a calculation of a subtree's proof-of-work; in our case, off-chain reports allow for a quantifiable security estimate by merchants.

Network inefficiency. There are a variety of inefficiencies in Bitcoin, including those due to low miner coordination [11,24]. At a lower level, Bitcoin's overhead is high due to its high-degree random graph for message propagation. The current network protocol causes every `inv` message to be sent along every edge in the p2p network, whenever a transaction or block is created. Since nodes have a network degree of at least 8, the resulting communication overhead is significant (as we show below). In reality, each peer needs to receive any particular `inv` for a transaction or block ID only once; we show status reports create such efficiency.

Bitcoin is also inefficient in disseminating block data. A block announcement must be validated using the transaction contents comprising the block. However, it is likely that the majority of the peers have already received these transactions, and they only need to discern them from those in their mempool. In principle, a block announcement needs to include only the IDs of those transactions, and accordingly, Corallo's *Compact Block* design [7] — which has been recently deployed — significantly reduces block size by including a transaction ID list at the cost of increasing coordination to 3 roundtrip times. *Xtreme Thinblocks* [25], an alternative protocol, works similarly to Compact Blocks but sends additional information. Specifically, if an `inv` is sent for a block that is not in the receiver's mempool, the receiver sends a Bloom filter of her IDpool along with the request for the missing block. As a result, Xtreme Thinblocks are larger than Compact Blocks but require 2 roundtrip times. Relatedly, the community has discussed in forums the use of IBLTs for reducing block announcements [2,22], but these schemes have not been fully evaluated. We propose a novel approach that couples a Bloom Filter and an IBLT, which we prove and demonstrate is smaller than these recent works, while requiring just 2 round trip times for coordination.

3 Eclipse Detection via Status Reports

In this section, we use status reports as a basis to outline a novel method to quantify the likelihood of eclipse, as well as double-spend attacks, and show that our approach is resilient to fraudulent miners. In Section 4, we will introduce an efficient method of relaying the reports, and in Section 5, we use our status reports to design an efficient, tree-based network topology for Bitcoin.

3.1 Quantifying Consensus of Each Block

Each block added to a blockchain is a vote towards consensus with regard to the block's ancestors. We propose an approach for peers to estimate quantitatively the amount of consensus associated with a block in terms of existing miners. Because of the possibility of latecomers, the Fischer, Lynch, and Paterson (FLP) impossibility result [13] prevents Nakamoto's algorithm from ever reaching consensus; the FLP result similarly restricts our algorithms to estimates as well.

ALGORITHM 1: EstimateMiningPower(block B, d days)

```
1: Let $P$ be the set of valid blocks published within $d$ days prior to $B$
2: Let $M$ be the set of miners attributable to blocks in $P$
3: Let $p$ be the count of blocks in $P$
4: for $m \in M$ do
5:     Let $p_m$ be the count of blocks from $P$ attributable to miner $m$
6:     $\pi_m = p_m / p$
7: end for
8: return $\pi$ (i.e., a vector of all miners)
```

Algorithms 1 and 2 specify the method for estimating mining power and quantifying consensus on block B. Let B be a block on the blockchain and M be the set of recent miners (i.e., mining pools). For each miner $m \in M$, its historic mining power π_m can be computed from the set of blocks they have discovered and claimed *prior* to when B was announced in a window of $d = 1$ day back; specifically $0 \leq \pi_m \leq 1$ is the proportion of blocks attributed to m out of the total number of (valid) blocks published during that time (including branches). Attribution can be verified by use of the coinbase private key, if necessary. Let $D(B)$ be the set of miners that have claimed blocks that are *descendants* from B on any branch. No duplicates appear in $D(B)$ since it is a set rather than a sequence. We define the *consensus* associated with B as $\phi(B) = \sum_m^{D(B)} \pi_m$, a value between 0 and 1. As the depth of B increases, so will the summed consensus.

ALGORITHM 2: EstimateConsensus(block B, d days)

```
1: Let $D(B)$ be the set of miners that issued valid blocks descendant from $B$
2: $\pi$ = EstimateMiningPower($B$, $d$)
3: $\phi = \sum_m^{D(B)} \pi_m$
4: return $\phi$
```

As a merchant computes this estimate, he can conservatively assume that a set of miners with power $q = 1 - \phi(B)$ is pursuing an alternate chain and a double-spend attack. An attacker with $q > 50\%$ of the mining power need not conduct an eclipse attack because she can rewrite the entire blockchain [20]. If $\phi(B) < 50\%$ then a merchant has no assurance that the majority of mining power is building on B, and should assume an eclipse attack is underway and not release goods to consumers. Detection of this scenario is our primary goal for this technique. Secondarily, given the depth z of the block, a merchant can also estimate the attacker's probability of a successful double-spend attack using Nakamoto's equation [20]: $1-\sum_{k=0}^{z}\frac{1}{k!}e^{(\frac{-zq}{1-q})}(\frac{zq}{1-q})^{k}\left(1-({}^{q}/_{1-q})^{z-k}\right)$.

As we discuss below in greater detail, using actual historical mining power with Alg. 2, we found that only after a block is 7 deep is $\phi(B) > 50\%$ typically, and only after a depth of 12 is $\phi(B)$ near 100% (See Fig. 2). Accordingly, next we develop and evaluate a method for miners to announce consensus earlier.

3.2 Confirming Mining Power with Status Reports

To drastically improve the speed at which a block reaches near 100% consensus, miners can periodically release *status reports*.[1] The reports are not added to the blockchain, and peers need no more than a recent history of them. Their purpose is to release fine-grained information from each active miner towards informing consensus; and in Section 5, we show how they can be leveraged to reduce traffic. Status reports consist of the list of transactions ID that a miner plans to include in its next block, the ID of the preceding block, and a nonce. Together, these values hash to a value v, which serves as a *partial proof-of-work* towards the network *target*. If the miner's report is not fraudulent, then the miner m's power π_m can be added to Alg. 2 for quantifying a block's consensus value.

ALGORITHM 3: CheckConsensus(block C, R_1, R_2, d days, threshold)

```
 1: Let $t_0$ denote arrival of $C$
 2: Let $t_1$ and $t_2$ denote the arrival of report $R_1$ (with nonce $v_1$) and $R_2$ (with nonce
    $v_2$), respectively, building on block $C$ from miner $m$
 3: $\pi$ = EstimateMiningPower($C$, $t$)
 4: Hashes for $R_1$ and $R_2$, respectively: $n_1 = (t_1 - t_0)H\pi_m$ and $n_2 = (t_2 - t_1)H\pi_m$
 5: $\text{Prob}(V_{\min} \geq v_1) = \left(1 - \frac{v_1-1}{2^{256}-1}\right)^{n_1}$ and $\text{Prob}(V_{\min} \geq v_2) = \left(1 - \frac{v_2-1}{2^{256}-1}\right)^{n_2}$
 6: if $\text{Prob}(V_{\min} \geq v_1) \geq$ threshold and $\text{Prob}(V_{\min} \geq v_2) \geq$ threshold then
 7:     Add $\pi_m$ as $m$'s mining power for estimating consensus of $C$
 8: else
 9:     Discard status reports
10: end if
```

A primary concern is, How do we detect fraudulent reports? The lowest hash value can be used to confirm quantitatively that the miner's historic mining power is being used, as follows. Each hash computed by a miner is a sample taken randomly from a discrete uniform distribution that ranges between $[0, 2^{256}-1]$. Using a miner's historical mining power and v computed from her status report, we can calculate the likelihood of her reporting a hash greater than or equal to v.

Let n be the number of samples drawn from a discrete random variable $V \sim \texttt{uniform}(0, 2^{256}-1)$. Let $V_{\min}$ be the first order statistic (i.e., the smallest value in the sample). It is well-known that for any discrete random variable, the complementary cumulative distribution function of the first order statistic is [6]:

[1] This broad idea has been discussed informally in forums by us and others (e.g., [3]), but to our knowledge, algorithms have not been formally defined or evaluated.

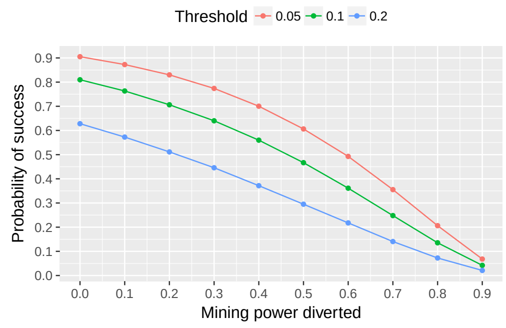


Fig. 1: The probability of a miner producing 2 status reports if diverting power. Lines represent different thresholds for Eq.1. Honest miners divert no power and will meet an Eq.1 threshold of 0.05 with 90% probability. A miner that diverts half her power will produce pairs of status reports with only a 60% probability for the same threshold.

$\text{Prob}(V_{\min} \geq v) = (1 - V(v-1))^n$, where the cumulative distribution function $V(v)$ is the probability that the random variable V takes a value less than or equal to v. In our case, we know that $V(v) = \frac{v}{2^{256}-1}$; therefore,

$$\text{Prob}(V_{\min} \geq v) = \left(1 - \frac{v-1}{2^{256}-1}\right)^n . \tag{1}$$

Accordingly, we have the method outlined by Algorithm 3. At time t_0, a peer receives a valid announcement for block A, and at time t_1, a status report is received from miner m, who claims to be building on A. (Whether m mined A is not relevant.) We know the total hash rate H for the Bitcoin network, which can be computed from the network *difficulty*.[2] The peer estimates the number of hashes that m computed since the announcement of A as $n = (t_1 - t_0)H\pi_m$ hashes, and then calculates $\text{Prob}(V_{\min} \geq v)$. At time t_2, the peer receives a second status report from m, with a new nonce, and performs the same set of calculations; this time, $n = (t_2 - t_0)H\pi_m$. If both probabilities are above a set threshold, the peer believes in his estimate of the mining power π_m. To increase the level of difficulty for the attacker, peers do not accept status reports before two minutes since a block announcement or previous status report from the same miner. Additionally, peers calculate an average of status reports accepted during the last w rounds, and ignore all reports from miners that frequently miss the threshold. For a threshold of 0.05 for Eq. 1, an honest miner will be able to submit a pair of reports with probability $0.95^2 = 0.90$. If honest miners quickly drop their mining power, they will fail to produce acceptable status reports; if honest miners quickly increase their power, they will produce reports, but the algorithm will use their historic power. Neither case increases risk for the merchant. Evidence that mining power changes over long time scales is encoded in the blockchain. Unfortunately, there is no available data that quantifies whether these changes take place gradually or over short time scales.

Thwarting fraudulent miners. Fig. 1 shows the results of a Monte Carlo simulation that calculates the probability of a malicious miner subverting our estimate. Over 100,000 trials, we calculated the probability of a peer incorrectly

[2] A finer estimate of hashrate can be computed from a moving window of blocks, instead, if desired; see https://bitcoinwisdom.com/bitcoin/difficulty as an example.

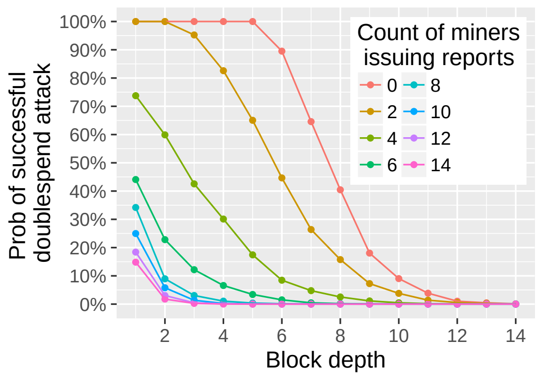


Fig. 2: The probability of a successful double-spend attack, given a sequence of n blocks and largest c mining pools' issuing status reports. Without status reports, a block must be $z = 11$ deep before double-spend attack success drops below 5%. Status reports from the top $c = 10$ miners drops the probability to about 5% after $z = 2$ blocks.

verifying their estimate of mining power, given 2 status reports by a fraudulent miner who uses only partial mining power on the main chain. The probability of producing a pair of consecutive status reports decreases quickly as a miner diverts more mining power on an alternative chain. A miner that diverts half his power will have only a 60% chance of producing two status reports when the threshold is set to 0.05; as noted above, an average below 90% over w rounds is quickly detectable. As a secondary mechanism, in the long-term, the estimate of the mining power from Alg. 1 will also decrease if the miner consistently uses less power on the main chain.

Effectiveness of gradual deployment. In Fig. 2, we estimate the consensus of blocks that are z deep, using a Monte Carlo simulation based on the real blockchain. The simulation is based on the mining power of all mining pools that claimed blocks in 2016. Each trial of the simulation selected a sequence of z mining pools with replacement, representing a block and a set of $z - 1$ descendants (as if there was no branching). The mining pools were selected at random, weighted by their observed mining power. We also varied the number of honest miners $c = 0 \ldots 14$ issuing status reports, using the most powerful miners first. Using Alg. 2, we summed the represented mining power. For miners issuing status reports, we followed Alg. 3, using a threshold of 0.05, and discarded 10% of reports. We ran 40k trials for each combination of z and c. Though not shown in the plot, the simulation results show that without any status reports, on average it takes 6 blocks before $\phi(B) > 50\%$, but mining power reaches 50% immediately if the top 8 miners release status reports. The y-axis of the plot shows the median probability of a successful double-spend attack, using Nakamoto's equations [20].

As the figure shows, without status reports ($c = 0$), a block must be $z \geq 8$ deep before the probability of a successful double-spend attack is less than 50%, and $z \geq 12$ blocks deep before attacker success is less than 1%. In contrast, on average if $c = 10$ miners start sending out status reports, the probability of a successful double-spend attack from existing mining power drops to 5% for $z = 2$.

We expect these algorithms to work better with Ethereum [10] because it incentivizes miners to release blocks that may become stale. The stale blocks help estimate π_m correctly. We plan to evaluate this case in future work.

Size of status reports and transaction status. A micro version of reports would need to announce only the 80-byte block header; that is, a Merkle Root is sufficient to verify a proof of work, rather than the full list of transactions. Micro status reports could easily fit in a tweet by each miner, for example. However, it is also useful to announce the list of transactions that were to be included by the miner, which increases transparency. These full reports can be made lightweight by using any efficient block announcement mechanism. For example, using Compact Blocks [7], this list can be compressed to about 18 KB. Through the techniques we propose in Section 4, our status reports are about 2–3 KB. Thus, if the 15 largest mining pools each release full status reports twice every 10 minutes, the extra traffic on the network will be roughly 125 B/s.

4 Graphene: Efficient Block Announcements and Status Reports

In this section, we detail *Graphene*, which is our efficient method for announcing blocks and releasing status reports. The goal of Graphene is for a receiver to learn the set of specific transaction IDs that are contained in a (pending or confirmed) block containing n transactions. Unlike other approaches, Graphene never sends an explicit list of transaction IDs, instead it sends a small Bloom filter and a very small IBLT. (The operation of IBLTs is reviewed in Appendix B.)

The intuition behind Graphene is as follows. The sender creates an IBLT $\mathcal{I}$ from the set of transaction IDs in the block. To help the receiver create the same IBLT (or similar), he also creates a Bloom filter $\mathcal{S}$ of the transaction IDs in the block. The receiver uses $\mathcal{S}$ to filter out transaction IDs from her pool of received transaction IDs (which we call the IDpool) and creates her own IBLT $\mathcal{I}'$. She then attempts to use $\mathcal{I}'$ to *decode* $\mathcal{I}$, which, if successful, will yield the transaction IDs comprising the block. The number of transactions that falsely appear to be in $\mathcal{S}$, and therefore are wrongly added to $\mathcal{I}'$, is determined by a parameter controlled by the sender. Using this parameter, he can create $\mathcal{I}$ such that it will decode with very high probability.

PROTOCOL 1: Graphene

```
1: Sender:    Sends $inv$ for a block or status report.
2: Receiver:  Requests unknown block; includes count of trans. in her IDpool, $m$.
3: Sender:    Sends Bloom filter $\mathcal{S}$ and IBLT $\mathcal{I}$ (each created from the set of $n$
   transaction IDs in the block) and essential Bitcoin header fields. The FPR of the
   filter is $f = \frac{a}{m-n}$, where $a = n/(c\tau)$.
4: Receiver:  Creates IBLT $\mathcal{I}'$ from the transaction IDs that pass through $\mathcal{S}$. She
   decodes the *subtraction* [9] of the two blocks, $\mathcal{I} \triangle \mathcal{I}'$.
```

Protocol details are as follows. A Bloom filter is an array of x bits representing y items. Initially, the x bits are cleared. Whenever an item is added to the filter, k bits, selected using k hash functions, in the bit-array are set. The number of bits required by the filter is $x = y\frac{-\ln(f)}{\ln^2(2)}$, where f is the intended false positive rate. For Graphene, we set $f = \frac{a}{m-n}$, where a is the expected difference between $\mathcal{I}$ and $\mathcal{I}'$. Since the Bloom filter contains n entries, and we need to convert to bytes, its size is $\frac{-\ln(\frac{a}{m-n})}{\ln^2(2)}\frac{1}{8}$. It is also the case that a is the primary parameter of the IBLT size. IBLT $\mathcal{I}$ can be decoded by IBLT $\mathcal{I}'$ with very high probability if the number of cells in $\mathcal{I}$ is d-times the expected symmetric difference between the list of entries in $\mathcal{I}$ and the list of entries in $\mathcal{I}'$. In our case, the expected difference is a, and we set $d = 1.5$ (see Eppstein et al. [9], which explores settings of d). Each cell in an IBLT has a *count*, a *hash* value, and a stored *value*. (It can also have a key, but we have no need for a key). For us, the count field is 2 bytes, the hash value is 4 bytes, and the value is the last 5 bytes of the transaction ID (which is sufficient to prevent collisions). In sum, the size of the IBLT with a symmetric difference of a entries is $1.5(2+4+5)a = 16.5a$ bytes. Thus the total cost in bytes, T, for the Bloom filter and IBLT are given by $T(a) = n\frac{-\ln(f)}{c} + a\tau = n\frac{-\ln(\frac{a}{m-\mu})}{c} + a\tau$, where all Bloom filter constants are grouped together as $c = 8\ln^2(2)$, and we let the overhead on IBLT entries be the constant $\tau = 16.5$.

To set the Bloom filter as small as possible, we must ensure that the false positive rate of the filter is as high as permitted. We enforce a rule in Graphene that all `inv` messages are sent ahead of a block announcement, and thus, can assume that the receiver already has all of the transactions in the block in her IDpool (they need not be in her mempool). Thus, we know that $\mu = n$; i.e., we allow for a of $m - n$ transactions to become false positives, since all transactions in the block are already guaranteed to pass through the filter. It follows that

$$T(a) = n\frac{-\ln(\frac{a}{m-n})}{c} + a\tau. \quad (2)$$

Taking the derivative w.r.t. a, Eq. 2 is minimized when when $a = n/(c\tau)$. (Refer to Eq. 3 in Appendix C for a more detailed discussion on how to calculate a.)

Due to the randomized nature of an IBLT, there is a non-zero chance that it will fail to decode. In that case, the sender resends the IBLT with double the number of cells (which is still very small). In our simulations, presented in the next section, this doubling was sufficient for the incredibly few IBLTs that failed.

4.1 Comparison to Compact Blocks

PROTOCOL 2: CompactBlocks

```
1: Sender:    Sends inv for a block or status report that has $n$ transactions.
2: Receiver:  If block is not in mempool, requests compact block.
3: Sender:    Sends the block header information, all transaction IDs in the block and
   any full transactions he predicts the sender hasn't received yet.
4: Receiver:  Recreates the block with additional information, and requests missing
   transactions if there exist any.
```

Compact Blocks [7] is to our knowledge the best-performing related work. It has several modes of operation. We examined the *Low Bandwidth Relaying* mode due to its bandwidth efficiency, which operates as follows. After fully validating a new block, the sender sends an *inv*, for which the receiver sends a *getdata* message if she doesn't have the block. The sender then sends a *compact block* that contains block header information, all transaction IDs (shortened to 5 bytes) in the block, and any transactions that he predicts the receiver does not have (e.g., the coinbase). If the receiver still has missing transactions, she requests them via an *inv* message. Protocol 2 outlines this mode of Compact Blocks.

The main difference between Graphene and Compact Blocks is that instead of sending a Bloom filter and an IBLT, the sender sends block header information and all shortened transaction IDs to the receiver. For a block of n transactions, Compact Blocks costs $5nB$ bytes. For both protocols, the receiver needs the *inv* messages for the set of transactions in the block before the sender can send it. Therefore, we expect the size of the IDpool of the receiver, m, to be constrained such that $m \geq n$. Assuming that $m > 0$ and $n > 0$, the following inequality must hold for Graphene to outperform Compact Blocks, $n\frac{-\ln(\frac{a}{m-n})}{c} + a\tau < 5n$; after algebraic simplification, our scheme is strictly smaller whenever $n \geq {}^{m}/_{1,287,670}$.

It does not seem likely that blockchain systems would in practice have a set of unconfirmed transactions held by peers that is 1,287,670 times larger than the block size. (Already over 22 billion transactions for the current block size.) In the next section, we provide further empirical results; and a detailed example of how to calculate the size of each scheme appears in Appendix C. Finally, we note that Xtreme Thinblocks [25] are strictly larger than Compact Blocks since they contain all IDs and a Bloom filter, and so Graphene performs strictly better than Xtreme Thinblocks as well.

Ordered blocks. Graphene does not specify an order for transactions in the blocks, and instead assumes that transactions are sorted by ID. Bitcoin requires transactions depending on another transaction in the same block to appear later, but a canonical ordering is easy to specify. If a miner would like to order transactions with some proprietary method (e.g., [15]), that ordering would be sent alongside the IBLT. For a block of n items, in the worst case the list will be $n\log_2(n)$ bits long. Even with this extra data, our approach is must more efficient than Compact Blocks.

5 Canary: Improved Network Efficiency and Security

In this section, we show how to leverage status reports and Graphene to rearchitect the p2p network underlying Bitcoin. The current p2p network is designed to prioritize decentralized control. Status reports can allow peers in Bitcoin and other blockchains to instead prioritize security, resiliency, and performance with *managed networks*. We take inspiration from networking technologies such as software defined networks (SDNs) [19] in which a centralized network controller dictates the behavior of a local network. Separating the control and data planes is a paradigm that predates SDNs going back to the 1970s [1,12,16,18]. *We are specifically not advocating for the centralized control of the Bitcoin network, but rather for the creation of a set of network overlays, each managed by an independent entity.*

The existing p2p network is not Bitcoin's only network: miners also join the Bitcoin Relay Network (BRN), a separate network for fast and reliable dissemination of transaction and block data. The BRN is nearly a clique topologically, and it is not realistic for the majority of peers to join the BRN. With deployment of status reports by mining pools, peers are free to pursue an alternative to the existing unreliable and ungoverned p2p network. In our approach, miners remain in the BRN, but they additionally offer managed network services to interested clients for a fee. Miners forward all received transactions to the BRN. In fact, the service can be provided by any entity that joins the BRN as a full node.

The advantage of this approach is that (like an SDN) the miners can arrange their dependent full node clients in any topology that suits them. We imagine most will charge a small fee to support a star topology based on redundant and resilient cloud services. However, it is possible for the miner to arrange clients in a tree that scales well with the number of clients; status reports can be used by clients to detect failure, and comparison of client feedback can be used by the miner to identify malicious nodes. Full nodes have the option of joining a secondary managed network, allowing for the detection of any missing status reports more quickly than a timeout would. In this way, the reliability or maliciousness of a provider can be determined. Providers of this service would advertise availability like any other Internet-based business. Meanwhile, the original p2p network would be available to peers that prefer its paradigm.

Receiving updates from miners. Any node connected to the BRN can be the root of a distribution network that propagates new transactions, status reports, and blocks. For simplicity, we assume the root is a miner. Status reports are propagated along the BRN, and the roots of a distribution network forward all status reports that they receive. It is sufficient for a full node to join a single distribution network and watch if status reports are received from a sufficient number of miners on schedule; full nodes that wish to more quickly recover from being eclipsed can listen for status reports on a second distribution network.

A trivial structure for a miner distribution network is a star structure in which each constituent full node has a direct connection to the owner miner. However, one-to-many connections do not scale well. Therefore, we suggest that full nodes be organized in a spanning tree with the miner at the root. The limitation of trees is that they offer the chance for full nodes to act maliciously and stop forwarding information to descendants. To address this problem, we have full nodes pay the root of the tree a membership fee to join the tree, which is in-line with Bitcoin's fee-based design.

Full nodes joining a distribution tree will be placed by the owner (at the root) in a chosen location on the tree depending on factors such as the joining fee, full node's reputation/reliability, etc. For instance, an unknown, recently joined full node is likely to be placed as a leaf, whereas the root can move well-behaving nodes higher in the tree over time, i.e., closer to the root. This incentivizes the member full nodes to behave well on the tree in order to be placed closer to the root, and therefore benefit from more reliable and faster message reception.

When status reports are not received (or are received malformed), a full node can report directly to the root, and through comparison of reports, faulty nodes in the middle of the tree can be easily discovered. Full nodes should also check that their submitted transactions appear in the status reports of miners other than the one they submitted it to. The status reports need only confirm that at least 50% of the mining power is reachable, rather than a low probability of double spend attack.

We expect that the set of available distribution networks will be small and with low churn, just as the set of mining pools has low churn (see Fig. 9 in Appendix D). We also expect that the miners will charge membership fees and that standard solutions for DoS-protection for Web services will be applied (e.g., cloud services) as they are more likely in practice than the tree-based solution we propose here; full nodes that do not wish to pay such membership fees could remain on the existing p2p network.

Submitting transactions. Full nodes submit their transactions to the Bitcoin network by sending them to one or more miners. In contrast to transaction flooding in the current Bitcoin network, our mechanism is more scalable and bandwidth efficient; however, it is potentially susceptible to adversarial flooding attacks. Many DoS-resistant mechanisms exist and are used by commercial web-based services. In the spirit of keeping mining open to all, we suggest each miner to run a Tor hidden service [21] by setting up an *introduction point.*

Traffic reduction. All transactions and block announcements are preceded by `inv` messages, both in today's p2p network as well as Canary's, and they represent a significant fraction of the network's overhead. In both networks, `inv` messages cross each (undirected) edge once. Since each peer in today's network maintains connections to at least 8 peers, `inv` messages cross four times as many edges as the number of peers. In Canary's tree-based topology, however, the number of edges is equal to the number of peers (assuming each peer is on only one tree), hence reducing `inv` traffic.

6 Canary and Graphene Evaluation

Our evaluation addresses the following questions in service of the dual design goals of reducing traffic while detecting eclipse attacks:

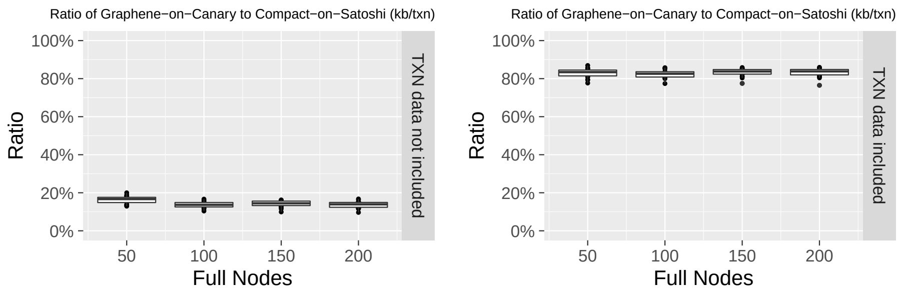


Fig. 3: **Main evaluation result:** A comparison of Graphene on Canary's tree topology to Compact Blocks on the Satoshi graph. The plot shows boxplots of ratios of traffic costs. (Right) Excluding transaction content, our approach reduces Bitcoin traffic overhead to 20%(i.e., reduces it by 80%). (Left) When including the cost of transaction content (set at 500B each), our approach reduces Bitcoin's total traffic by 20% of the current cost.

1. What reduction in traffic overhead results from using Graphene and Canary compared to using Compact Blocks on the original Satoshi-based graph topology? (Answer: traffic is reduced by 80%.)

Secondarily, we isolate each component by asking:

2. What is the reduction in traffic from using Graphene for block announcements compared to Compact Blocks? (Answer: overhead is reduced by 45–80%.)
3. What is the reduction in traffic from using Canary's tree-based graph compared to the Satoshi-based graph? (Answer: overhead is reduced by 30–70%.)

Simulator assumptions. Our evaluations are based on a detailed, custom simulator of Bitcoin using a Python-based discrete event simulator package. Our simulation models the propagation of messages across network links (ignoring effects from variable network bandwidth, TCP, etc.). Nodes accurately model any part of Bitcoin's operation necessary for evaluating our metrics, including maintaining a mempool, the blockchain and its forks, and using realistic signaling.

For Graphene and Compact Blocks, our simulator creates and decodes real Bloom filters and IBLTs, rather than merely estimating whether they might decode or return any false positives. If these data structures fail due to random chance, the nodes recover within the simulation. Because our simulation models detailed signaling and is written in a high-level language, our evaluations are based on a modest number of peers. Since our goal is a comparison between two choices, we expect that our results are representative of larger-scale scenarios.

A challenging parameter to set is the number of transactions per second offered to the network by peers. Our approach is to create kernel density estimates (KDEs) from the transaction generation patterns of real world peers. To that end, we gathered data for all Bitcoin transactions between October 27, 2015 and January 7, 2016 from http://blockchain.info. Each transaction in the dataset is labeled with an IP associated with the peer believed to have generated it, as well as the time it was released to the network. For each peer, we normalized the release times by the time of the day in which they were released. We then constructed the KDE for each peer using these normalized transactions times and gaussian kernels with one hour bandwidth. The KDE for a given peer represents a probability distribution from which we can draw transactions over the course of a simulated day. For each peer in the simulator, we randomly select one of the KDEs corresponding to a real world peer. Because these distributions have been generated from real data, they are a good approximation of the activity of real peers over the average one-day interval. On the other hand, this approach is not able to model days of the week or seasonal phenomena in transaction creation times.

6.1 Evaluation

Each simulation is configured to use the following parameters: *(i)* Topology: *Satoshi*'s original high-degree graph topology; or our *Canary* protocol's binary tree topology. *(ii)* Block Protocol: *Compact Blocks*; or our *Graphene* protocol. *(iii)* Block capacity: 2000 transactions. *(iv)* Full nodes: 50, 100, 150, or 200 peers. In all, we ran 16 combinations of parameters, and we ran each combination with at least 20 different seeds (and up to 30 seeds in some cases as time allowed); all told, we completed 376 simulations. The seeds determined the number of transactions per second (by sampling our KDE, as described above), and the interarrival of transactions and blocks. We set status reports to be generated every 2.5 minutes by each miner. In all simulations, we used 6 miner nodes, representing 6 mining pools. Each simulation was equivalent to 120 minutes of running Bitcoin; in sum, we simulated about 30 days of Bitcoin's operation.

Our main results are shown in Fig. 3, which compares Graphene on Canary's tree topology against Compact Blocks on the Satoshi graph, as a function of the number of nodes in the network. Since each run is a different number of KBs, we compare the ratio of an exact set of parameters (including the seed), varying only the protocol and topology combination. Boxplots[3] show the distribution of results across all trials. Fig. 3(left) shows that our approach of using Canary and Graphene reduces *traffic overhead* by a median value of about 80% compared to Compact Blocks over the Satoshi graph (i.e., the current method sends 6.6× more bytes than our approach). Most traffic in Bitcoin is due to the transactions themselves, and Fig. 3(right) shows that using Canary and Graphene reduces *total traffic* by 20% compared to using Compact Blocks over the Satoshi graph. As the number of full nodes increases along the x-axis, the ratio of total traffic in the network between the two protocol-topology combinations remains steady, suggesting that our results are representative of larger networks.

We also evaluated the sum number of bytes per message type for two example seeds across four protocol-topology combinations. First, we found that in all cases, changing from Satoshi graphs to trees drastically reduces the number of `inv` messages. Second, we saw that the amount of data used by Compact Blocks is much greater than Graphene's use of a Bloom filter of the block and an IBLT of the transaction list. (See Fig. 7 in Appendix C.) We also grouped our larger set of results according to transactions-per-second, and found that Compact Blocks

[3] See https://en.wikipedia.org/wiki/Box_plot.

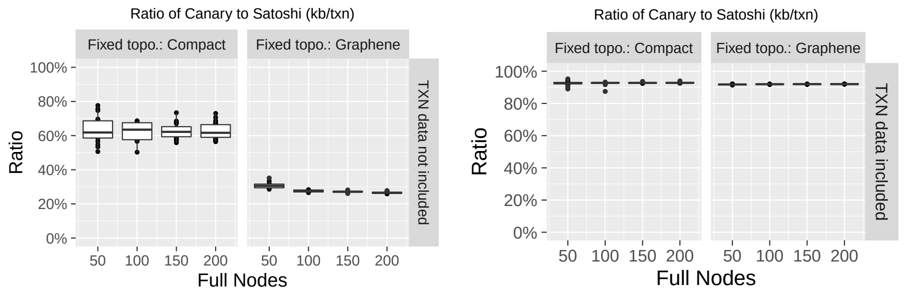


Fig. 4: **Varying Topology**. When Graphene is used, Canary reduces traffic overhead by 70% compared to Satoshi (or by 10% for total traffic, which includes transaction data). When Compact Blocks are used, Canary reduces traffic by more than 30% compared to Satoshi graphs (or by 10% for total traffic).

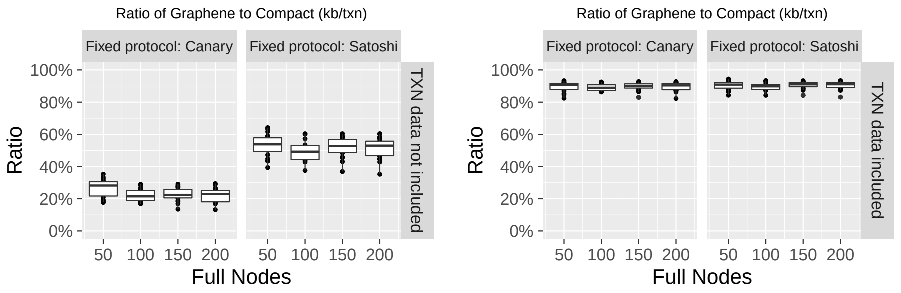


Fig. 5: **Varying Protocol**: On Canary's tree-based topology, Graphene reduces traffic overhead by 80% compared to Compact Blocks (or by 15% for total traffic, which includes transaction data). When using a Satoshi-style graph, the cost of Graphene reduces traffic by 45% compared to Compact Blocks (or by 15% for total traffic).

on the Satoshi graph generates a wide range of bytes-per-transaction, even at the lowest transactions-per-second rate. In contrast, Graphene on Canary is both more efficient and stable as load changes. (Details appear in Fig. 6, Appendix C.)

Canary. To isolate the efficiency gains of Canary, we compared the two protocols on the Satoshi graph and Canary tree topologies. Fig. 4(left) shows that both protocols benefit from using a tree topology. Graphene benefits more from Canary, which reduces traffic overhead by 70%. When transaction data is also included as part of the total traffic, Graphene reduces overhead by about 10%. Fig. 7 in Appendix D details the reduction for two sample seeds, and shows that Canary reduces traffic by cutting down `inv` messages.

Graphene. When we evaluate the total bandwidth ratio of Graphene to Compact Blocks, we can see that both Canary and Satoshi topologies benefit from using Graphene. Fig. 5(left) shows that Graphene benefits with a reduction of traffic by 80%, while gains on Satoshi graphs are less than 55% due to increased `inv` traffic. Again, for these detailed gains, refer to Fig. 7 in Appendix D.

7 Conclusion

First, we have presented status report messages that allow peers to detect eclipse attacks almost immediately. We have also described how these reports can estimate the risk of a double-spend attack. Second, we presented Graphene, a protocol for efficiently announcing new blocks and status reports. Third, we proposed Canary, a new tree-based topology for Bitcoin, which separates the network's data and control planes. Via a detailed network simulation, we have demonstrated that the combination of Canary and Graphene reduces network traffic by 80%, compared to the state of the art use of Compact Blocks on the Satoshi graph. Our proposed mechanisms work together or are incrementally deployable, and provide an efficient method for increasing transparency and detecting eclipse attacks, not only for Bitcoin but also for other blockchain-based network protocols.

References

1. Systems Network Architecture General Information. No. GA27-3102-0, IBM, http://docwiki.cisco.com/wiki/IBM_Systems_Network_Architecture_Protocols, first edn. (Jan 1975)
2. Andresen, G.: O(1) Block Propagation. https://gist.github.com/gavinandresen/e20c3b5a1d4b97f79ac2 (August 2014)
3. Bishop, B.: bitcoin-dev mailling list: Weak block thoughts... https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2015-September/011158.html (Sep 2015)
4. Bloom, B.H.: Space/Time Trade-offs in Hash Coding with Allowable Errors. Commun. ACM 13(7), 422–426 (Jul 1970)
5. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J., Felten, E.: Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In: IEEE S&P. pp. 104–121 (May 2015), http://doi.org/10.1109/SP.2015.14
6. Casella, G., Berger, R.L.: Statistical inference. Duxbury advanced series, Brooks Cole, Pacific Grove (Calif.) (2002), http://opac.inria.fr/record=b1134456
7. Corallo, M.: Bip152: Compact block relay. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki (April 2016)
8. Croman, K., et al.: On Scaling Decentralized Blockchains . In: Workshop on Bitcoin and Blockchain Research (Feb 2016)
9. Eppstein, D., Goodrich, M.T., Uyeda, F., Varghese, G.: What's the Difference?: Efficient Set Reconciliation Without Prior Context. In: ACM SIGCOMM Conference. pp. 218–229 (2011), http://doi.org/10.1145/2018436.2018462
10. Ethereum Homestead Documentation. http://ethdocs.org/en/latest/
11. Eyal, I., Gencer, A.E., Sirer, E.G., van Renesse, R.: Bitcoin-NG: A Scalable Blockchain Protocol. In: USENIX NSDI (2016)
12. Feamster, N., Balakrishnan, H., Rexford, J., Shaikh, A., van der Merwe, J.: The case for separating routing from routers. In: Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture. pp. 5–12. FDNA '04, ACM, New York, NY, USA (2004), http://doi.acm.org/10.1145/1016707.1016709
13. Fischer, M., Lynch, N., Paterson, M.: Impossibility of distributed consensus with one faulty process. JACM 32(2), 374–382 (1985)

14. Goodrich, M., Mitzenmacher, M.: Invertible bloom lookup tables. In: Conf. on Comm., Control, and Computing. pp. 792–799 (Sept 2011), http://doi.org/10.1109/Allerton.2011.6120248
15. Hanke, T.: A Speedup for Bitcoin Mining. http://arxiv.org/pdf/1604.00575.pdf (Rev. 5) (March 31 2016)
16. Haskin, D.L.: A BGP/IDRP Route Server alternative to a full mesh routing. RFC 1863 (Oct 1995), http://rfc-editor.org/rfc/rfc1863.txt
17. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse Attacks on Bitcoin's Peer-to-peer Network. In: USENIX Security (2015)
18. Keshav, S., Paul, S.: Centralized multicast. In: Proceedings of the Seventh Annual International Conference on Network Protocols. pp. 59–. ICNP '99, IEEE Computer Society, Washington, DC, USA (1999), http://dl.acm.org/citation.cfm?id=850936.852461
19. Kim, H., Feamster, N.: Improving network management with software defined networking. IEEE Communications Magazine 51(2), 114–119 (February 2013)
20. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. https://bitcoin.org/bitcoin.pdf (May 2009)
21. Overlier, L., Syverson, P.: Locating hidden servers. In: IEEE S&P. pp. 15–pp. IEEE (2006)
22. Russel, R.: Playing with invertible bloom lookup tables and bitcoin transactions. http://rustyrussell.github.io/pettycoin/2014/11/05/Playing-with-invertible-bloom-lookup-tables-and-bitcoin-transactions.html (Nov 2014)
23. Sasson, E.B., Chiesa, A., Garman, C., Green, M., Miers, I., Tromer, E., Virza, M.: Zerocash: Decentralized anonymous payments from bitcoin. In: IEEE S&P. pp. 459–474 (2014), http://dx.doi.org/10.1109/SP.2014.36
24. Sompolinsky, Y., Zohar, A.: Secure high-rate transaction processing in Bitcoin. Financial Cryptography and Data Security (2015), http://doi.org/10.1007/978-3-662-47854-7_32
25. Tschipper, P.: BUIP010 Xtreme Thinblocks. https://bitco.in/forum/threads/buip010-passed-xtreme-thinblocks.774/ (Jan 2016)
26. Tschorsch, F., Scheuermann, B.: Bitcoin and beyond: A technical survey on decentralized digital currencies. IEEE Communications Surveys Tutorials PP(99), 1–1 (2016)

A Introduction to Blockchains and Bitcoin

Account balances. A Bitcoin is a unit of currency, which is fungible, divisible (up to eight decimal places), and recombinable. It is measured as a balance across multiple accounts, which are themselves manifested in *addresses*.[4] Each address comprises a stored asymmetric cryptographic key and an associated balance of Bitcoin. The public portions of an address are the public key and the balance of coin. When an address is involved in a *transaction* with one or more other addresses, Bitcoins are transferred among them.

[4] Internally, Bitcoins exist only as "unspent transaction outputs" (UTXO), but users of the system think of them as balances in addresses, and that view does not affect the results of this paper.

Roles. Users wishing to exchange coins broadcast the details of their transactions over Bitcoin's p2p network, signed with their private keys. A set of *miners* on the p2p network verify that each transaction is signed correctly and does not conflict with another transaction. Miners independently agglomerate a set of valid transactions into a *block* and attempt to solve a predefined proof-of-work (POW) problem involving this block and a chain of prior valid blocks. In Bitcoin, the POW computation is dynamically calibrated to take approximately ten minutes per block. The first miner to solve the problem broadcasts his solution to the network, adding it to the ever-growing *blockchain*; the miners then start over, with the appended blockchain and the set of transactions that were not added as part of the previous block. When transactions appear in a block, they are considered *confirmed*, and each subsequent block provides additional confirmation. The miners' incentive for discovering a block is a reward of coins, called the *coinbase*, consisting of a predetermined *block reward* (currently 12.5 BTC) and fees from transactions included in the block.

Full nodes are peers in the network that do not mine, but do generate, validate, and propagate transactions and blocks to other nodes including miners. Consumers (i.e., those who purchase goods or services) typically have no need to process and validate all transactions, so they can instead operate *simple payment verification* (SPV) nodes that process, store, and transmit data involving only addresses-of-interest, which are typically addresses they control, make payments to, or receive payments from. SPV nodes rely on full nodes to relay transactions-of-interest.

Bitcoin transaction consistency. The main goal of the Bitcoin p2p network is to provide a consistent view of blocks and unconfirmed transactions across all network peers. Each peer maintains a local snapshot of the transactions in a memory pool dubbed the *mempool*. Blocks consist of a list of transactions that have already (almost always) been broadcast to miners and full nodes in the network.

To announce a new block, a miner lists all transactions contained in the new block along with a header that provides an easily verifiable POW solution. When a full node or miner receives a new block, it validates each transaction in the block and the proof of work.

Due to propagation delays in the network, it is possible for the miners to receive competing (but valid) block announcements, which bifurcates the chain, until one of the two forks is appended to first. It is also possible and valid for a miner to receive a set of blocks that retroactively rewrites many blocks; doing so is a demonstration of computational work that miners accept despite the age or depth[5] of a rewritten block.

Topology and flooding. Bitcoin propagates new transaction and block announcements by flooding throughout a p2p random graph of full nodes and miners. Each peer in the graph requests direct connections to 8 other peers, and accepts requests for connections from up to 117 other peers. A peer will offer a

[5] The *depth* of a block refers to the number of blocks that follow it; the *height* of a block is the number of blocks that precede it.

newly created transaction to each neighbor via an `inv` message, which reports the hash of the transaction content as its ID. If a peer does not already possess the transaction, it will request it using a `getdata` message. Blocks are handled similarly: `inv` messages describe a block by its ID, which is created from the hash of the block's contents. Upon receiving the `inv`, peers will request the block if they do not already have it. Hence, in today's topology, `inv` messages cross every edge in the random graph once, while the actual transaction and block data typically propagate along only a spanning tree of the graph (more edges will be traversed if there are propagation delays). For convenience, in this paper, we refer to the set of (unconfirmed) transaction IDs that a peer knows about as the *IDpool*. Actual transaction contents are placed in the mempool.

B Overview of IBLTs

An IBLT [14] is an efficient data structure that supports insertion and retrieval of key-value pairs. Like Bloom filters, an IBLT is an efficient data structure that allows two parties to determine, with high probability, which values from a set they share in common. Unlike Bloom filters, IBLTs allow for recovery of any missing values, which are assumed to be of fixed size and encoded as binary strings. Key-value pairs can be inserted, retrieved and deleted like an ordinary hash table. An IBLT consists of m entries, each storing a `count`, a `keySum`, and a `valueSum`, all initialized to zero. A new value v is inserted by determining its entry $i = h(v)$ based on the hash of its value such that $i < m$. At entry i, all three fields are incremented or xored. In particular, standard addition is used for the `count` field, but `xor` is used to add to the `keySum` and `valueSum` fields. An item can be deleted similarly: at the correct entry, `count` is subtracted by 1, and the `valueSum` and `keySum` fields are `xor`'ed. When `count` $\equiv 1$ the `valueSum` field contains the actual value of the sole item remaining in the cell. (The purpose of the `keySum` field is to support a `GET()` operation for a given key: that is, if `count` $\equiv 1$ and `keySum` $\equiv h(v)$, then `valueSum` $\equiv v$.) IBLTs use $k > 1$ hash functions to store each value in k entries, which we collectively call a value's *entry set*. If table space is sufficient, then with high probability for at least one of the k entries, `count` $\equiv 1$, and so `keySum` and `valueSum` fields are recoverable [14] .

Suppose that two peers each have a list of values, V and V', respectively, such that the difference is expected to be small. The first peer constructs an IBLT L (with m entries) from V. The second peer constructs V' from L' (also having m entries). Eppstein et al. [9] showed that a cell-by-cell difference operator can be used to efficiently compute the symmetric difference $L \triangle L'$. For each pair of fields (f, f'), at each entry in L and L', we compute either $f \oplus f'$ or $f - f'$ depending on the field type. When $|$`count`$| \equiv 1$ at any entry, the corresponding value can be recovered. If `count` $\equiv 1$, then the value belongs to $L \setminus L'$. And if `count` $\equiv -1$, then the value belongs to $L' \setminus L$. Peers proceed by removing all values corresponding to these unit counts—not only from the recoverable entry, but also from all entries in the value's entry set. This process will generally produce new recoverable entries, and continues until there remain no recoverable entries.

In general it is possible to place an arbitrary number of values t into an IBLT, but only a fixed number τ can actually be recovered and only after the other $t - \tau$ values have first been removed (using the procedure above for example). The *capacity* of an IBLT refers to this value τ and is proportional to the number of entries m it comprises.

C Supplementary Material for Graphene and Canary

Setting a, the expected symmetric difference. Actual implementations of Bloom filters and IBLTs involve several (non-continuous) ceiling functions:

$$T(a) = \left(\lceil \ln(\frac{m-n}{a}) \rceil \left\lceil \frac{n \ln(\frac{m-n}{a})}{\lceil \ln(\frac{m-n}{a}) \rceil \ln^2(2)} \right\rceil \right) \frac{1}{8} + \lceil a \rceil \tau \tag{3}$$

The optimal value of Eq. 3 can be found by brute force using a simple loop. We compared the value of a picked by using $a = n/(c\tau)$ to the cost for that a using Eq. 3 for valid combinations of $50 \leq n \leq 2000$ and $50 \leq m \leq 10000$. We found that it is always within 37% of the cost of the optimal value, with a median difference of 16%. In practice, a for-loop brute-force search for the lowest value of a is almost no cost to perform, and we do so in our simulations.

Example. A receiver with an IDpool of $m = 4000$ transactions makes a request for a new block that has $n = 2000$ transactions. The value of a that minimizes the cost is $a = n/(c\tau) = 31.5$. The sender creates a Bloom filter $\mathcal{S}$ with $f = \frac{a}{m-n} = 31.5/2000 = 0.01577$, with total size of $2000 \times \frac{-ln(0.01577)}{c} = 2.1$KB. The sender also creates an IBLT with a cells, totaling $16.5a = 521B$. In sum, a total of $2160B + 521B = 2.6KB$ bytes are sent. The receiver creates an IBLT of the same size, and using the technique introduced in Eppstein et al. [9], the receiver subtracts one IBLT from the other before decoding. In comparison, for a block of n transactions, Compact Blocks costs $n * 5B = 5nB$. Therefore, Compact Blocks costs $2000 * 5B = 10KB$, over 3 times the cost of Graphene. If Graphene was to impose an ordering, the additional cost for $n = 2000$ transactions would be $n \log_2(n)$ bits $= 2000 * log_2(2000)$ bits $= 2.74$ KB. This increases the cost of Graphene to 5.34 KB, still almost half of Compact Blocks.

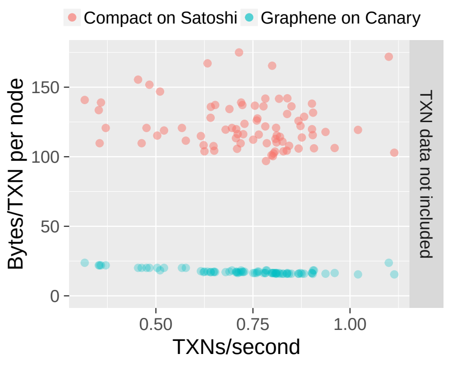
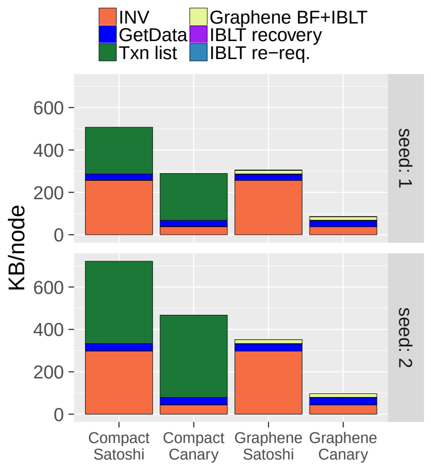


Fig. 6: Traffic sent by Graphene over the tree topology and Compact Blocks over the Satoshi topology, where each trial's transaction rate is the independent variable. Transparency reveals some over-plotting in this scatterplot.



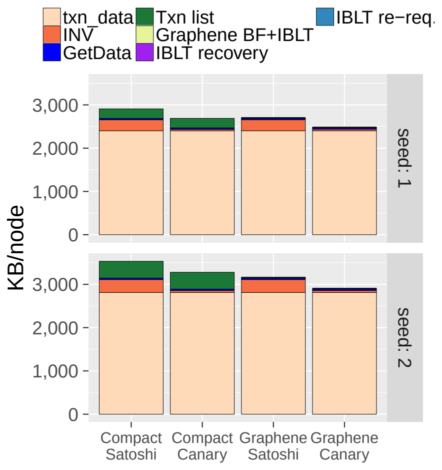


Fig. 7: A comparison of traffic, broken down by message type, for two specific seeds for Graphene and Compact Blocks, with each run on both tree and Satoshi-based topologies. Note that traffic does not include transaction data.

Fig. 8: Traffic by message type, for two specific seeds. Figure 7 shows the same plot without transaction data included.

D Supplementary Plots

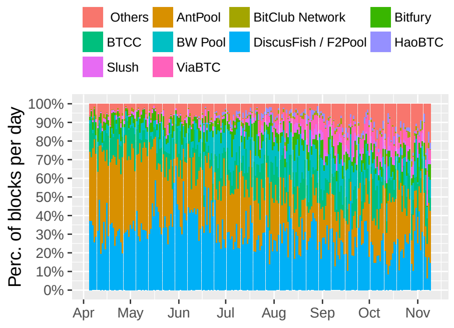


Fig. 9: Eight mining pools are responsible for 89% of all blocks, and a similar trend has held for years. Data from 2016 (source: blocktrail.com).

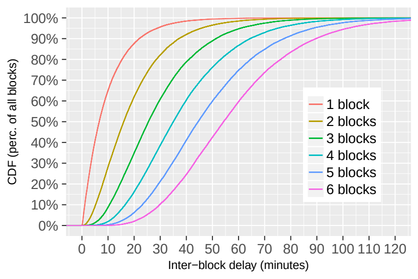


Fig. 10: Inter-block delay. 32,213 blocks total (Jan–Aug 2016). Data is based on logs of Bitcoin network activity collected and released by blockchain.info and blocktrail.com.

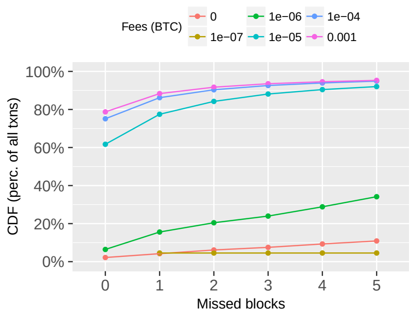


Fig. 11: Number of blocks missed by transactions, grouped logarithmically by fee. About 13% Bitcoin transactions, all backed with fees of at least 0.0001BTC, are not put in the block directly following their announcement (Jan–Aug 2016).