Kadcast: A Structured Approach to Broadcast in Blockchain Networks

Elias Rohrer
elias.rohrer@tu-berlin.de
Technical University of Berlin
Berlin, Germany

Florian Tschorsch
florian.tschorsch@tu-berlin.de
Technical University of Berlin
Berlin, Germany

ABSTRACT

In order to propagate transactions and blocks, today's blockchain systems rely on unstructured peer-to-peer overlay networks. In such networks, broadcast is known to be an inefficient operation in terms of message complexity and overhead. In addition to the impact on the system performance, inefficient or delayed block propagation may have severe consequences regarding security and fairness of the consensus layer. Therefore, we introduce Kadcast, a novel peer-to-peer protocol for block propagation in blockchain networks. Kadcast utilizes the well-known structured overlay topology of Kademlia to realize an efficient broadcast operation with tunable overhead. As our protocol is based on UDP, we incorporate forward error correction (FEC) to increase reliability while still maintaining its lightweight protocol architecture. To this end, we build a probabilistic model to analyze Kadcast's resilience to packet losses as well as random and adversarial node failures. Moreover, we evaluate Kadcast's block delivery performance, broadcast reliability, efficiency, and security based on advanced network simulations, which confirm the merits of the Kadcast protocol.

1 INTRODUCTION

Bitcoin [42] fundamentally challenged the role of banks by enabling decentralized money transfer on the Internet. It builds upon a peer-to-peer network to implement an electronic cash system, where nodes can interact directly without intermediaries. Following its genesis in 2008, a high number of blockchain networks emerged. In these systems, nodes may issue transactions by broadcasting them in the overlay network. Validator nodes collect and verify transactions and periodically consolidate them into blocks, which are appended to a replicated, immutable ledger—the blockchain. Blocks are broadcast in the network as well, which gives every node the capability to verify correctness locally. That is, nodes run a distributed agreement protocol to enable state replication.

Broadcast is accordingly *the* most commonly used network operation in blockchain networks. However, current implementations are typically based on unstructured overlay networks, which is not necessarily favorable for this kind of operation: while being relatively robust, broadcast in unstructured overlays suffers from high message overhead, as duplicates are introduced to the system. To reduce the load, many networks spread block messages only by gossiping to a subset of neighbors, which in turn might introduce additional propagation delays.

To date, the scalability of blockchain protocols is a huge concern, and the inefficiency of the utilized network protocols is a limiting factor for innovations striving for higher transaction rates, such as increased limits for block sizes, block rates, or changes that depart even further from the Nakamoto consensus [12, 20, 49]. Furthermore, it has been shown that the block propagation delay has severe effects on the consistency of blockchain networks, leading to higher rates of stale blocks and blockchain forks [27]. As this opens opportunities for fraud [15, 50], alleviating the network-layer deficiencies is not only a matter of *performance*, but also a pressing issue of *fairness* and *security*. While it has been shown that unsolicited block propagation has the largest impact on the stale block rate [27], it leads to flooding the network with block data, which the network architectures currently deployed in the blockchain landscape cannot handle. The emergence of third-party relay networks [25, 35] emphasize the need for an improved propagation mechanism. We however consider them orthogonal to our work, because they do not address the inherent shortcomings of blockchain networks.

In this paper, we present *Kadcast*, a new broadcast overlay for blockchain networks, based on the Kademlia [40] architecture. Kadcast allows for a more efficient broadcast operation with tunable redundancy and overhead. To this end, we exploit the structured overlay topology to delegate broadcast responsibilities for subtrees with decreasing height. By combining redundant execution of our algorithm with forward error correction (FEC), we achieve a high degree of reliability and resilience in the face of packet loss, as well as random and adversarial node failures. Kadcast is design-compatible with many open and decentralized blockchain systems. Our evaluation shows that Kadcast distributes blocks on average 30 % faster than the currently deployed blockchain protocols in a Bitcoin-like scenario, and even faster under an Ethereum-like parametrization with smaller block intervals and block sizes. Kadcast increases the efficiency of block propagation, making room to introduce additional features such as unsolicited block push. The simulations indicate that Kadcast achieves similar, often better, security results in terms of the stale block rate, in particular for the Ethereum-like case.

Our key contributions can be summarized as follows: (1) We design an efficient broadcast protocol for blockchain networks, which exploits Kademlia's structured architecture. (2) We introduce parallelization and FEC to improve the reliability and resilience of our algorithm in a completely adjustable and predictable way. (3) We discuss attack vectors, provide mitigation strategies against Sybil and Eclipse attacks, and analyze the network's resilience to adversarial nodes obstructing block delivery. (4) We conducted a comprehensive simulation study and evaluated the performance, reliability, efficiency, and security of Kadcast in comparison to "VanillaCast", a paradigmatic blockchain protocol, which generalizes the currently prevalent networking layer of blockchains. To this end, we developed a new simulation framework for blockchain

networks, bns, which focuses on networking aspects, and which we make accessible to the research community.

The remainder of this paper is structured as follows. First, we describe primitives of information dissemination currently found in the blockchain landscape and introduce the *VanillaCoin* model in Section 2. In Section 3, we introduce Kadcast, including its overlay construction, block propagation algorithm, and adjustable reliability factors. Subsequently, we analyze Kadcast's security and discuss various threats and mitigation strategies in Section 4. We present our simulation model and evaluation results in Section 5. In Section 6, we discuss related work and emphasize the novelty of our approach, before we conclude the paper in Section 7.

2 BLOCKCHAIN NETWORKS

In this section, we portray the workings of blockchain systems in a generalized form. To this end, we first introduce a straw man blockchain design and then discuss the information dissemination method most prevalent in current blockchain networks.

2.1 VanillaCoin: A Paradigmatic Blockchain

For the sake of clarity, we introduce a prototypical blockchain protocol, *VanillaCoin*, that mirrors the quintessential operations of current blockchain systems. This allows us to describe how blockchain networks generally function, while abstracting from the particularities of specific implementations.

The central purpose of VanillaCoin is to keep track of the current state of accounts, which are bound to cryptographic key pairs. Entities that are in possession of this key material function as *account owners* and are able to issue publicly verifiable *transactions*, i.e., transitions in the account state. Note that this is independent of the nature of the managed state: it can solely consist of a simple (financial) account balance, as for example in Bitcoin, or hold more complex data, as in Ethereum. Moreover, we do not specify how the account owners come to possession of the required key materials, as the VanillaCoin model aims to capture open blockchain networks as well as networks with restricted access, i.e., so-called private blockchains. That is, the owners might be able to simply create new accounts by generating a new key pair, or they would have to be approved by some sort of registry or public key infrastructure.

Transactions, issued by account owners, are broadcast in the network of VanillaCoin nodes, whereby they also reach *validator* nodes. Elected validators (a.k.a., "round leaders") collect and verify new transactions, decide on an authoritative transition ordering, and finally batch them into blocks, effectively issuing a new global state transition and initializing a new round.[1] Again, depending on the system specifications, leaders may be elected due to some kind of byzantine fault tolerant consensus mechanism, e.g., Nakamoto-style Proof-of-Work [42], Proof-of-Stake [33], or PBFT [7]. After their issuance, blocks are broadcast to all nodes in the blockchain network, which append them to the local state of their ledger. The nodes are able to independently verify the validity of incoming state transition based on the VanillaCoin consensus rules and the previous blockchain state. Such protocols have been proven to be consistent in the partial synchronous network model [26, 33, 34, 45], i.e., under the assumption that blocks are delivered to all nodes in a timely fashion (with regard to an upper bound).

The resulting VanillaCoin blocks are of a certain *block size* and are issued at a rate roughly following a certain *block interval*. These parameters are especially interesting from a networking perspective, since they not only determine the transaction throughput of the system, but also the bandwidth limit the networking layer has to be able to handle, which is known to be a bottleneck in scaling blockchain systems [12].

To summarize, the VanillaCoin protocol combines the concept of distributed state machine replication with a consensus mechanism to implement a byzantine fault tolerant agreement protocol.

[1] In [20], Eyal et al. observe that the two described operations of *leader election* and *transaction serialization* do not necessarily have to be fulfilled by the same entity. However, as they often are, we assume this is the case for our VanillaCoin model.

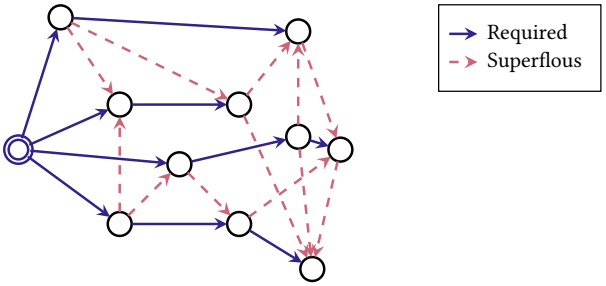

Figure 1: Example broadcast in an unstructured overlay.

2.2 Information Dissemination in Blockchain Networks

As we have seen, VanillaCoin mimics the typical blockchain operations, and as these systems, VanillaCoin heavily relies on the networking layer for transaction and block propagation. In particular, it requires an efficient broadcast operation, since most data items are typically transferred to all participants in the network. In the following, we give a blueprint of VanillaCoin's networking layer, *VanillaCast*, which again abstracts from the specifics of each individual blockchain system. However, more detailed information on the Bitcoin [42] and Ethereum [18] networking layers can be found in Appendix A and Appendix B, respectively.

When joining the VanillaCast network, nodes retrieve the addresses of a number of other participants by the means of an adequate bootstrapping mechanism. Then, it establishes TCP connections to a random subset of nodes $R \subset N$, which are henceforth its *neighbors* in the peer-to-peer network. This networking paradigm is known as an *unstructured* overlay network. As nodes are only able to communicate to the rest of the network via the neighbors, messages are passed hop-by-hop in a store-and-forward manner. In particular, upon arrival of new transactions and blocks, each VanillaCoin node first stores them in local memory, verifies their validity based on its current blockchain state, and then forwards them to adjacent nodes in the network.

To ensure timely message delivery via the shortest path, the messages are forwarded to all neighbors, which then follow the same protocol. However, while this propagation method indeed covers the shortest path, it actually covers all paths in the network. As Figure 1 illustrates, this introduces a large amount of superflous

messages to the network. Therefore, this kind of broadcast operation exhibits a high message complexity ($O(N \cdot R)$), which has been shown to have severe consequences on network scalability in the past [8]. Therefore, VanillaCast tries to reduce the net accruing traffic by introducing a request-response scheme in which every node first advertises larger data items to neighbor nodes, whose transmission then may subsequently be requested. However, this again adds at least one round-trip time (RTT) per hop to the message propagation delay. In the case of blockchain networks, such delayed block propagation however has been shown to be unfavorable compared to *unsolicited* block propagation [27]. However, as unsolicited block relay would lead to blindly flooding the network, it is currently not considered option for blockchain systems based on unstructured overlay networks, such as VanillaCoin.

3 THE KADCAST PROTOCOL

In contrast to the prevalence of VanillaCoin-like approaches to networking in blockchain systems, we believe that a thorough exploration of the design space is necessary to facilitate higher networking performance. In particular the unpredictable nature of information dissemination in unstructured peer-to-peer networks is an issue when it comes to find new solutions that are tailored to the problem at hand. We therefore in the following introduce Kadcast, a new *structured* approach to information propagation in blockchain networks, whose tunable characteristics create a more predictable environment for optimized block transmission.

While Kadcast can be used for other broadcast operations, such as transaction propagation, such an application may have additional requirements, e.g., privacy-wise. It is however considered out-of-scope of this initial discussion. In the following, we therefore mainly focus on block propagation. Kadcast is based on Kademlia [40], a DHT design that is typically used for efficient lookup procedures. Kadcast, however, makes use of Kademlia's overlay structure to enable an efficient broadcast. In the following, we describe the overlay construction and the broadcast algorithm as the two main building blocks of our approach. Moreover, we introduce means to improve the performance, reliability, and resilience of Kadcast.

3.1 Overlay Construction

Kademlia is an UDP-based peer-to-peer protocol in which nodes form a structured overlay network. Nodes in the network are addressed by unique L-bit binary node identifiers, in the following denoted as `ID`, which are generated upon joining the network. The `ID` determines a node's position in a binary routing tree that builds the foundation of Kademlia's structured peer-to-peer overlay. An example of such a tree for a 4-bit address space is shown in the upper part of Figure 2. Please note that this tree is never actually constructed and serves as a mental model only. Peers, however, still use their local state to traverse the network structure efficiently, yielding a message complexity of $O(\log N)$. Therefore, nodes maintain routing state and organize known nodes in so-called *k-buckets*, storing triplets (`ip_addr`, `port`, `ID`). Each bucket is a list of the k least recently seen nodes that have a certain *distance*, in relation to the node identifier `ID`. The factor k is a system-wide parameter which determines the routing state and the lookup complexity.

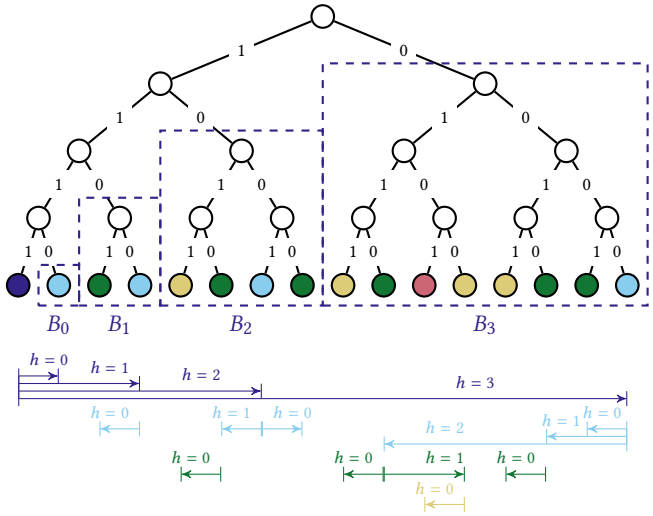

Figure 2: Example broadcast initiated by node 1111 ($\beta = 1$). Colors indicate node distances in the spanning tree, relative to the initiator.

Characteristically, Kademlia's notion of distance is based on the non-euclidean XOR-metric, calculated by applying the $\oplus$-operation on two node identifiers and interpreting the result as an integer number, i.e.,

$$d(x, y) = (x \oplus y)_{10}.$$

This means, that for node identifiers of length L, a node $\texttt{ID}_0$ holds buckets B_i, $i = 0 \ldots L-1$, whereby bucket B_i holds the node information of k nodes with $\texttt{ID}_j$ so that $2^i \leq d(\texttt{ID}_0, \texttt{ID}_j) < 2^{i+1}$. It follows that the node space covered by each bucket is exponential with i. This can be illustrated by that fact that, since the XOR-metric is unidirectional, the bucket B_0 only holds one specific node of distance one, while B_{L-1} covers a possible node space of 2^{L-1} nodes. The buckets can be thought of holding up to k nodes belonging to a series of subtrees with identifiers whose binary prefixes do not match the nodes' prefix, i.e., also not containing the node itself. For example, given the fully populated tree shown in Figure 2, the 4 buckets of node $\texttt{ID}_0 = 1111$ would hold nodes from the ranges `1110`, `110*`, `10**`, and `0***`, respectively. If a node wants to add a new entry to a given bucket that already holds k entries, it employs a least recently used (LRU) drop policy. Before dropping an entry from the list, the peer will send a `PING` message (see Figure 3) to see whether the respective node is still reachable. Only if the node is not reachable anymore, it will be dropped. This way, the protocol favors older, more stable nodes over fresh ones. It thereby also circumvents an eviction bias towards fresh, potentially malicious peers, which hardens the network against security issues, such as the eclipse attacks described in [28].

When a node first joins the network, it has to know the address of at least one bootstrapping node. It therefore sends `PING` messages to known nodes to check whether they are actually online. Additionally, `PING` transmits the sending node's routing information to the recipient, thereby distributing its existence in the network. In fact, similar patterns can be found throughout the protocol, where

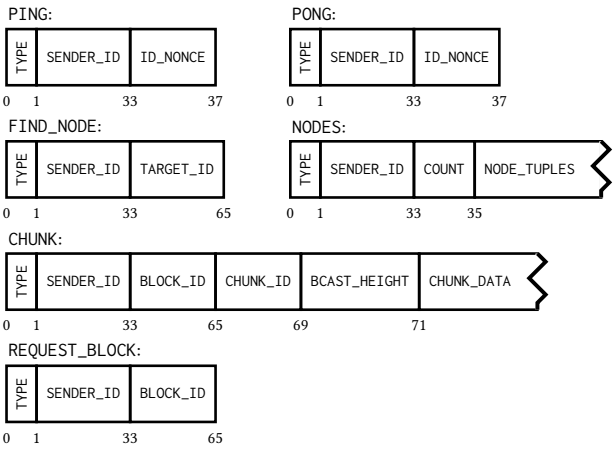

Figure 3: Kadcast message types.

every seen message updates not only the sender's but also the recipient's buckets. This soft-state approach makes for a very lightweight overlay membership management.

After the initial bootstrapping step, each Kadcast node begins discovering the network to update its routing information, which it repeats periodically throughout its lifetime. Initially, the joining node looks up its own `ID`, which returns a set of nodes closely positioned to its own network location. Moreover, each node periodically refreshes every bucket it has not seen some activity from in the last hour: for each such bucket, it picks a random `ID` with appropriate distance and performs a look up to populate its buckets with fresh routing information.

The lookup procedure allows a node to retrieve a set of k nodes closest to a specific `ID` in the address space. The procedure of finding the k closest nodes is carried out by iteratively narrowing down the search space and issuing `FIND_NODE` messages (see Figure 3) to nodes which are closer to the `ID`. To this end, (1) the node looks up the α closest nodes regarding the XOR-metric in its own buckets. (2) It queries these α nodes for the `ID` by sending `FIND_NODE` messages. (3) The queried nodes respond with a set of k nodes they believe to be closest to `ID`. (4) Based on the acquired information, the node builds a new set of closest nodes and iteratively repeats steps (1)-(3), until an iteration does not yield any nodes closer than the already known ones anymore.

Like the bucket size k, α is a globally known parameter determining the redundancy (and hence also the overhead) of the lookup procedure. At the same time, these parameters influence the lookup latency, as the parallel nature of the lookup procedure optimizes the needed delay. Typical parameter values are $k \in [20, 100]$ and $\alpha = 3$. As the Kadcast protocol is not used to store and retrieve values, it does not incorporate other message types found in Kademlia.

3.2 Block Propagation

As described before, most blockchain networks rely on TCP-based transport protocols for block propagation, which ensure the reliable transmission of arbitrarily large data. In contrast, the Kadcast protocol is UDP-based. While this allows for a lightweight protocol with reduced message complexity, it also entails that it has to handle block data serialization and reliable transmission on the application layer. Therefore, when the propagation of a block is initiated, Kadcast first segments its data in packet-sized *chunks* that are then distributed in the network via corresponding messages (see Figure 3) and according to the broadcast procedure (cf. Algorithm 1), which is a modified version of the algorithm in [13].

Algorithm 1 Redundant broadcasting algorithm.

Require: broadcast height h,
chunk data c,
set of known chunks C,
redundancy factor β
if $c \in C$ **then** `abort`
$C \leftarrow C \cup \{c\}$
for $i = 0 \rightarrow h - 1$ **do**
  $R \leftarrow$ `randomly_select` (β, B_i)
  for all $r \in R$ **do**
    `send_chunk`(r, d, i)
  end for
end for

Kademlia's bucket logic partitions the identifier space in subtrees whose sizes depend on their distance to the current node. The Kadcast protocol makes use of this fact to generate a spanning tree that allows for an efficient broadcast operation: the algorithm delegates broadcast responsibilities for subtrees with decreasing height h to other nodes, which recursively repeat the process within their delegated area. Therefore, when a miner initiates the block broadcast, it is responsible for the entire tree with height $h = L$. The miner picks a random peer from each bucket and delegates broadcast responsibilities by sending `CHUNK` messages, which carry the data and her routing information. It assigns a new height h, which effectively determines the receiver's broadcast responsibility. When a node receives a `CHUNK`, it repeats the process in a store-and-forward manner: it buffers the data, picks a random node from its buckets up to (but not including) height h, and forwards the `CHUNK` with a smaller value for h accordingly.

This means, with every step, another set of nodes is designated to be responsible for chunk delivery in their respective subtrees. A simple example for $L = 4$ can be seen in Figure 2: node $\texttt{ID}_0 = 1111$ initiates a broadcast in the network, and sends four `CHUNK` messages with heights $h = 0 \ldots 3$ to one random node picked from each of the respective buckets B_i, $i = 0 \ldots 3$. The receiving nodes repeat this procedure, again issuing messages to nodes from bucket numbers less then their assigned height. Hence, the broadcast operation is performed on decreasing subtree sizes, and therefore guaranteed to terminate in $O(\log n)$ steps. Upon receipt of all chunks required to rebuild a block, the node follows Bitcoin's typical block verification procedure before continuing the broadcast operation.

3.3 Reliability of Block Delivery

If we assume constant transmission times, honest network participants, and no packet loss in the underlying network, the propagation method just discussed would result in an optimal broadcast tree. In this scenario, every node receives the block exactly once and

hence no duplicate messages would be induced by this broadcasting operation. Unfortunately, we cannot make these assumptions and have to consider packet losses, adversarial failures, as well as random failures during transmission.

In the example of Figure 2, if a chunk on its way to node `0000` is corrupted or this node refuses to forward a chunk, the whole bucket B_3, i.e., the right half of the tree, would not receive the block. That is, in the worst case, a single transmission failure could result in a network coverage of fifty percent only. Therefore, our the broadcast algorithm is improved and secured by two different approaches, which both introduce redundancy.

3.3.1 Improving Broadcast Reliability and Performance. First, instead of having a single delegate per bucket, we select β delegates. This severely increases the probability that at least one out of the multiple selected nodes is honest and reachable. It therefore protects the broadcasting operation against random and adversarial node failures on the propagation path. Moreover, this parallelized broadcasting method improves the propagation performance in terms of latency: nodes with the best connection receive the transmitted chunk first and will proceed to propagate the chunks in the bucket. As this repeats on every hop, and Kadcast nodes ignore duplicate chunks, only the fastest routes are used for block delivery.

Secondly, since the internet protocol (IP) only promises best-effort datagram delivery, Kadcast has to consider transmission failures due to corrupted and/or dropped packets on every hop of the propagation. To increase the reliability of its data transport, Kadcast therefore employs a forward error correction scheme based on RaptorQ [38] codes. The adoption of this scheme allows Kadcast nodes to recover transmitted block data after the reception of any s source symbols out of n encoding symbols, which are transmitted via `CHUNK` messages. As this results in more transmitted data overall, an overhead of $n - s$ additionally transmitted symbols per block transmission is introduced. The FEC overhead factor can be adjusted through the parameter $f = \frac{n-s}{s}$. Utilizing FEC gives the receiver the ability to correct errors without the need for retransmissions, which lead to additional delay. We therefore optimize our protocol in terms of latency and accept an additional overhead. However, in order to allow nodes to recover from the rare case that block delivery fails entirely, and to enable the initial bootstrapping of the blockchain, the Kadcast protocol incorporates a simple `REQUEST_BLOCK` message (cf. Figure 3) that allows nodes to query others for specific blocks, and is answered by the corresponding `CHUNK` messages.

In the following, we analyze and discuss our methods for improved broadcast reliability.

3.3.2 Analysis of Parallelized Broadcast. Kadcast implements broadcast redundancy by parallelizing the algorithm. To this end, we introduce the system parameter β, which describes how many distinct delegates per bucket should be selected (and thus how many nodes per bucket should receive a copy of the block). This improved algorithm can be seen in Algorithm 1. Please note that for $\beta = 1$, Algorithm 1 describes the "optimal" broadcast from Section 3.2.

Along the lines of [13], we model the block propagation reliability as the expected node coverage of the broadcast operation, which is based on the average probability of transmission failures. Thus, given the failure probability ϵ and $\beta = 1$, a single broadcast chunk would reach its next hop with probability $p = 1 - \epsilon$. The expected number of nodes receiving this chunk can therefore be expressed by $M = (1 + p)^L$, assuming a balanced distribution tree of height L, which is highly plausible due to the uniform random distribution of node identifiers. It follows that the ratio of covered nodes is

$$m = \frac{M}{2^L} = \left(\frac{1+p}{2}\right)^L.$$

Note however that this expression models the transmission of a *single chunk without redundancy* only. In order to express the coverage of a redundant block broadcast, we need to extent this model.

Therefore, we model the parallel execution of our algorithm as the probability that at least one of the redundantly sent chunks is successfully delivered, i.e., $p_\beta = 1 - \epsilon^\beta$. Moreover, let X be a random variable expressing the number of received chunks. The probability that we receive all s chunks of a block is thus $p_b = P(X = s) = p^s$, which induces a block failure probability of $\epsilon_b = 1 - p_b$. Accordingly, the probability to deliver a block with redundancy β is given by $p_{b,\beta} = 1 - \epsilon_b^\beta$. These observations yield an expected block coverage ratio of

$$m_{b,\beta} = \left(\frac{1+p_{b,\beta}}{2}\right)^L.$$

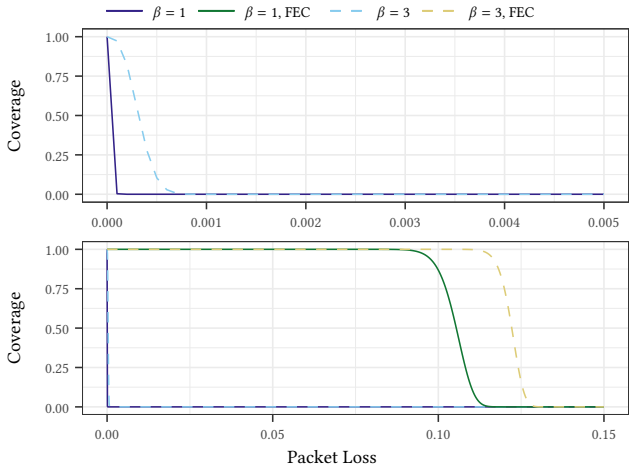

Figure 4: Broadcast reliability (block size of 1 MB, $L = 160$, and FEC overhead factor $f = 0.15$.

In the upper part of Figure 4, we analyze the expected broadcast coverage for different packet loss rates and $\beta \in \{1, 3\}$. We assumed a block size of 1 MB and identifier length $L = 160$. We observe that even the smallest packet loss makes the probability of delivering a block drop immediately, rendering the chance of covering the identifier space virtually impossible (see Figure 4). While a redundancy factor $\beta = 3$ has a positive impact on the block propagation, it is not sufficient to guarantee the reliable transmission of entire blocks over a lossy channel. However, the parallelized broadcast is necessary to compensate adversarial and random node failures, and to improve the propagation performance, as discussed before.

3.3.3 Analysis of FEC-based Block Delivery. Using the RaptorQ forward error correction, a Kadcast node has to successfully receive s or more arbitrary symbols out of the n transmitted in order to recover a full block, an event which can be modeled by a binomial distribution, i.e.,

$$p_{b,f} = P(X \geq s) = 1 - P(X < s) = 1 - \sum_{i=0}^{s-1} p.$$

Figure 4 clearly shows the improved propagation reliability offered by introducing forward error correction with 20% redundancy ($f = 0.15$): the approach keeps the coverage ratio at 100% until around a packet loss rate of 9%. This is quite a number for Internet standards and is even enough to cover the large packet loss rates exhibited by connections towards mainland China.[2] However, this can still be improved by combining the approach with redundancy, i.e., $\beta > 1$. In this case, the success probability is $p_{b,\beta,f} = 1 - (1 - p_{b,f})^{\beta}$, which is shown for $\beta = 3$ in Figure 4 as well. The combination of FEC and parallelization ensures full network coverage, even if on average 12% packets are lost during transmission.

The analysis results highlight that FEC is a favorable way to ensure reliable transmission of data over an unreliable network infrastructure: it allows to significantly increase the reliability of the broadcast while introducing a relatively small linear overhead. In contrast, the overhead introduced with increasing the replication factor β introduces a larger increase in messaging complexity. However, broadcast redundancy is still required in cases where the weak point is not just an unreliable network link, but a malicious node obstructing block delivery.

4 KADCAST SECURITY

As discussed earlier, fast and fair block propagation may be considered security-critical for the consensus layer of blockchain-based systems. However, the peer-to-peer network and the block propagation mechanism can also be attack vectors of system security. In the following, we discuss the security properties of the Kadcast network itself and its broadcast mechanism.

4.1 Threat Model and Mitigation Strategies

The Kadcast design is based on the time-tested and well-studied structured network design of Kademlia [40]. Numerous previous entries study Kademlia's security properties, its behavior when attacked by a range of adversaries, and designs improving on its security [10, 55]. In the following, we discuss the most prevalent adversarial threats to the security of peer-to-peer networks in general and Kadcast in particular.

4.1.1 Sybil attacks. The notion of a Sybil attack [16] describes the possibility of a single adversary to embody a large number of network entities by forging additional identities. By doing so, the adversary aims to out-number the honest nodes participating in a distributed system, effectively increasing the share of malicious nodes f in the system. Moreover, a Sybil attack is especially enticing when the forged identities can be used to trick the system and enable unwanted behavior. In systems based on the Kademlia overlay, Sybil attacks may be used to generate a lot of identities that can fill up a victim's buckets [36, 53]. The ability to run this kind of attack is often a prerequisite to be able to run an Eclipse attack (see next section) on Kademlia-based systems.

In the case of Kadcast, if an adversary can forge arbitrary IDs, she may easily be able to position herself close to a target, thereby increasing the likelihood of receiving lookups and broadcasts from this target. This may enable the adversary to simply refuse block delivery and thereby obstruct the block propagation, which we discuss further in Section 4.2. Hence, we observe that the ability to create valid node identifiers at arbitrary positions in the network is detrimental to the security of the system.

The Kadcast protocol employs a number of countermeasures in order to increase its resilience to Sybil attacks. For one, the node identifiers are generated by hashing the IP addresses of the nodes, i.e., $\texttt{ID} = H(addr)$, which ensures a one-to-one mapping between identifier and network node. This property is validated by other Kadcast nodes by exchanging `PING` and `PONG` messages, i.e., they only accept new identifiers if the node is reachable via the respective address. Note, that this procedure also raises the bar for IP spoofing attacks immensely. Moreover, by using a cryptographic hash function, the proposed generation method ensures that the identifier space is covered randomly but uniformly, making it hard for an adversary to generate identifiers at a specific distance from a target node.

Additionally, the Kadcast protocol can easily be extended to incorporate cryptographic puzzles as Sybil protection, similar to [3, 6, 58]. Along the lines of proof-of-work mining, Kadcast follows a simple scheme: a joining node has to find a nonce, so that the hash of concatenation of its identifier and nonce adheres to a certain difficulty level. This is, the binary value of the hash has to be less than the chosen difficulty target, i.e., $H(\texttt{ID} \,||\, \texttt{ID\_NONCE}) < t_{diff}$, where t_{diff} is a global parameter of the system. Every node that receives a new node identifier validates this property before it inserts the new node to its buckets. It can run the validation quickly, while the node generation can take quite some time, depending on the chosen parameter t_{diff}. Thereby, the inclusion of this hash puzzle scheme seriously impairs the ability of adversary to quickly generate a large number of node identifiers. Moreover, additional effective countermeasures encompass stricter bucket policies which enforce a certain degree of diversity from an AS-level and/or subnet perspective [2, 24].

4.1.2 Eclipse attacks. All peer-to-peer networks rely on some kind of routing scheme that allow nodes to decide where to forward data or which nodes to query for a specific data item. However, these routing decisions are made on the basis of an underlying data structure, the routing table. Eclipse attacks describe a family of attacks on peer-to-peer networks in which the adversary manipulates the routing tables of its targets to contain only nodes controlled by the adversary. Once she isolated her target from the rest of the network, the adversary is in full control of the data streams coming from and to the target node. This may be used by the adversary to completely block data delivery, selectively obstruct data transmission, or even foist spurious data.

In blockchain networks, Eclipse attacks are a serious threat, since they could be used to monopolize the connections of a target node

[2] Kaiser et al. describe that, induced by the Chinese "Great Firewall", connections exhibit 6.9% packet loss, which leads to artificially delayed block propagation of Chinese Bitcoin miners [31].

and then further exploit the protocol. They have been shown to enable double-spending and selfing-mining attacks [28, 39]. In the past, Eclipse attacks on the Kademlia protocol have been studied in literature [36, 37, 53]. These studies show that, *if* an adversary would come to control a large number of node identifiers, she may try to flood all buckets of a target node with addresses of nodes in her control. This technique could be used to isolate Kadcast nodes from the rest of the network. The Kadcast protocol however includes strong Sybil protection to mitigate this possibility. Moreover, Kadcast follows a bucket eviction policy, which favors older, more stable nodes over newly acquired node addresses. This policy impedes the adversaries capability of supplying all nodes known to the victim. In conclusion, by safeguarding the node identifiers through the means of IP binding and cryptographic puzzles, as well as applying rigorous bucket policies, Kadcast follows best practices for Sybil and Eclipse protection [39].

4.1.3 Denial-of-Service attacks. Broadcast protocols aim to distribute information to all nodes in the network. This inherent asymmetry immediately raises the question on whether they allow an adversary to flood the network with arbitrary data, i.e., how susceptible they are to denial-of-service (DoS) attacks. In order to avoid these kind of attacks, blockchains like Bitcoin employ a store-and-forward propagation policy: each block a node receives is first stored and validated (i.e., check the proof of work), before it is announced to neighbors. This way, an adversary trying to flood the network with fake block data would have to solve a proof-of-work hash puzzle for each of the forged blocks, making it a very unattractive attack vector. The Kadcast protocol adapts this DoS protection: every node first validates received blocks before they are forwarded in the broadcast tree.

Additionally, previous work [56] highlighted that node operators have become targets of DDoS in the past. In the worst case, this results to a node failure, which possibly impairs the broadcast operation, as we discuss further in the following sections.

4.2 Obstruction of Block Delivery

Kadcast relies on the responsiveness and compliance of delegate nodes. An adversary however may have an interest to obstruct the block delivery. To this end, she could position herself on the distribution path during the broadcast operation, and refuse to comply when chosen as delegate. In the following, we will elaborate and analyze this general attack vector.

First, an adversary may try to prevent a specific node from publishing a new block. In order to intercept *outgoing* blocks generated by a target node, an adversary needs to fill every bucket of the target with malicious nodes. We assume that an adversary is able to spawn M out of N nodes, but cannot cheat the placement mechanism, i.e., has to hash node identifiers like everyone else, resulting in a uniform coverage of the identifier space. In fact, this is a very conservative assumption, since we neglect the previously discussed bucket filling and eviction policies that would heavily skew this towards stable and honest nodes. Moreover, for the sake of censoring outgoing blocks, all buckets are equally attractive targets, since the covered space does not only determine the amount of affected nodes, but in equal manner the probability to be selected by the target's broadcast operation. For example, as the bucket size in a network of N nodes can be estimated to be

$$b_{s,i} = \left\lfloor \frac{2^i}{2^L} \cdot N \right\rfloor,$$

a successful attack on the transmission to bucket B_{L-1} may lead to only a coverage of $N/2$ nodes. However, the required number of nodes in this bucket space is also proportionally harder to acquire for the adversary. Due to Kadcast's parallel route selection, it becomes highly unlikely that all β nodes per bucket are picked from the adversary's pool. In particular, when the adversary can acquire control of M nodes, we can assume that the same share, $\epsilon = M/N$, describes the situation in every bucket and hence determines the failure probability of a single broadcast operation. Accordingly, this would result in a parallelized broadcast failure probability of $p_\epsilon = (M/N)^\beta$, which exponentially decreases with the redundancy factor β, as we discussed and analyzed in Section 3.3.1.

The more interesting case is an adversary trying to interfere with the block delivery *to* a specific node. As discussed before, a true Eclipse attack is unfeasible in the Kadcast network, since it strictly applies best practices and enforces a uniform coverage of the identifier space. However, in the following we analyze security of block delivery when faced with an adversary that is able to spawn a certain amount of network nodes, i.e., attempting a Sybil attack. In order to calculate the probability of successful block delivery in face of such an attacker, we model the broadcast operation as a simple Markov chain, which is depicted in Figure 5. The block propagation starts in an arbitrary distance i from the target. For instance, if the origin would fall in bucket B_2, the model only needs to consider the operations in height two or smaller. The broadcast operation succeeds, when the block is delivered to the target node, and only honest nodes were visited during path traversal.

The initial state in the Markov model is s_i, and without loss of generality we can assume it to start at s_{L-1}, the state representing broadcast in the largest bucket. From state i, the Kadcast algorithm can delegate the targeted node directly and transition to the *success* state s_d with probability

$$p_{d,i} = 1 - \left(\frac{b_{s,i} - 1}{b_{s,i}} \right)^\beta.$$

Alternatively, the algorithm chooses some other node in the bucket with probability $\overline{p_{d,i}}$. The chosen node may be either honest or malicious. If it is honest (again, probability $p_h = 1 - p_\epsilon = 1 - (M/N)^\beta$), the broadcast operation continues and the model transitions to state s_{i-1}. If it is malicious, it would obstruct the block delivery, and hence the model transitions to the *fail* state s_f with probability $\overline{p_h} = 1 - p_h$. Once in the *success* or *fail* state, the fate of the broadcast operation is decided, hence the Markov model reaches a steady state after a maximum of $L - 1$ state transitions.

We implemented the Markov chain model utilizing the R package `markovchain` [51], and simulated the success probability of block propagation for different shares of malicious nodes ϵ and redundancy factors β. As these simulations assume the source to be in the bucket of highest distance $(L-1)$, they yield worst-case estimations for the steady-state success probability. The results are shown in Table 1; even for adversaries that control 10 % of network nodes, Kadcast delivers block with more than 99 % probability. Moreover, by adjusting the redundancy factor, Kadcast is able to deliver blocks

Table 1: Markov Simulation Results

Parameters				Results	
N	M	ϵ	β	p_ϵ	p_d
11000	1000	0.1	3	0.00075	0.993
12000	2000	0.2	3	0.0046	0.957
13000	3000	0.3	3	0.0122	0.888
11000	1000	0.3	5	0.00065	0.994
15000	5000	0.5	5	0.0041	0.963

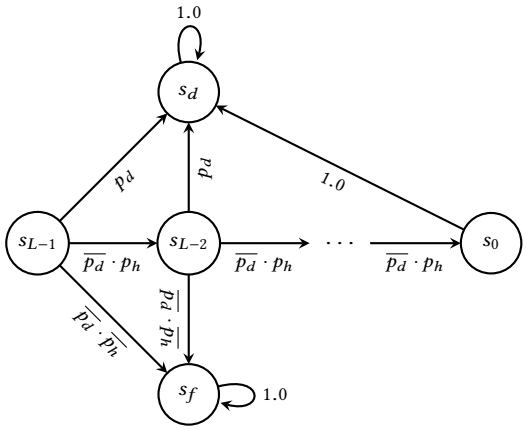

Figure 5: Markov chain model.

with more than 96% probability in a highly adversarial environment where 50 % of nodes are controlled by the adversary.

4.3 Security Implications of Network-Layer Properties

One of the central ideas of blockchain-based systems is that valid blocks extend the blockchain and eventually lead to a global consensus, as long as the majority of peers are honest and follow the protocol [42]. However, this is based on the assumption that peers receive new blocks soon after their creation. In this regard, recent analytical works [26, 34, 45] focussing on the consensus layer proved that the consistency guarantees of blockchain protocols only hold when blocks are delivered in a timely fashion.

In an early study, Decker and Wattenhofer highlighted the importance of the block propagation delay for the security of the Bitcoin system [15]. They showed that block propagation in Bitcoin's unstructured overlay network follows a long-tailed distribution in which 5% of the peers wait more than 40 seconds for new blocks. The authors could show that the delay increases the probability of blockchain forks.

Eyal and Sirer discovered the feasibility of *selfish mining* attacks on the Bitcoin protocol [21]. They show that a malicious miner could gain an advantage by withholding mined blocks. The main idea is to let other miners "waste" their computational power on an old block, while the selfish miner (secretly) mines a new block. When other miners find and propagate a block solution, the selfish miner quickly broadcasts its secret block and therefore still has a chance to "win the race" for the longest chain. The success of this attack, however, heavily depends on the attacker's mining power and the network share it can reach with its secret block before any concurrent block. Thus, reducing the propagation delay would also reduce the attack surface for selfish mining attacks, as the time window selfish miners have to react becomes smaller.

Recently, the impact of block propagation on Bitcoin's resilience towards selfish mining and double spending [32] attacks was further investigated by Gervais et al. [27]. The authors show that the occurrence of stale blocks, i.e., blocks that do not get included in the final longest chain and therefore do not contribute to its security, can have severe negative consequences for the performance and the security of proof-of-work-based blockchain systems. Their results suggest that the stale block rate is influenced by the propagation delay, and that it heavily depends on the employed propagation algorithm.

We therefore conclude that a central goal of any block distribution algorithm should be to efficiently make use of the available resources and to minimize the propagation delay as it has a positive effect on the security and fairness of blockchain-based systems.

5 EVALUATION

In this section, we evaluate the block distribution performance, broadcast reliability and efficiency, as well as the security impact of the Kadcast protocol on an empirical basis. For this, we gathered data from a comprehensive network simulation study, which are discussed in the following.

5.1 Simulation Model

Our network simulation study is based on a new simulator for blockchain networks, which we now introduce.

5.1.1 `bns` - *Blockchain Network Simulator.* In order to capture the networking aspects of blockchain networks in general and to evaluate the characteristics of the Kadcast design in particular, we implemented `bns`, a new blockchain network simulation whose architecture is able to incorporate interchangeable networking modules and network topology models. The simulation is based on ns-3 [48], an advanced event-discrete network simulator, which allows to model the networking protocols to a high degree of detail and enables time-independent simulations. In the simulation framework, we developed a prototype application implementing a rudimentary blockchain node logic, which runs on network nodes that are connected according to one of the network topology models. Moreover, the application is able utilize one of the implemented networking stacks, i.e., the UDP-based Kadcast networking protocol, as well as an instantiation of the TCP-based VanillaCast networking stack for comparison. The `bns` simulator allows to easily setup and analyze large-scale network simulations for a large set of adaptable parameters for the consensus layer, such as block sizes, block intervals, and simulation time, as well as networking layer parameters, such as network size, network topology, number of miners, unsolicited or header-based block relay, etc. We make the simulator source code publicly available to the research community.[3]

[3]The source code can be retrieved from https://gitlab.tubit.tu-berlin.de/rohrer/bns-public.

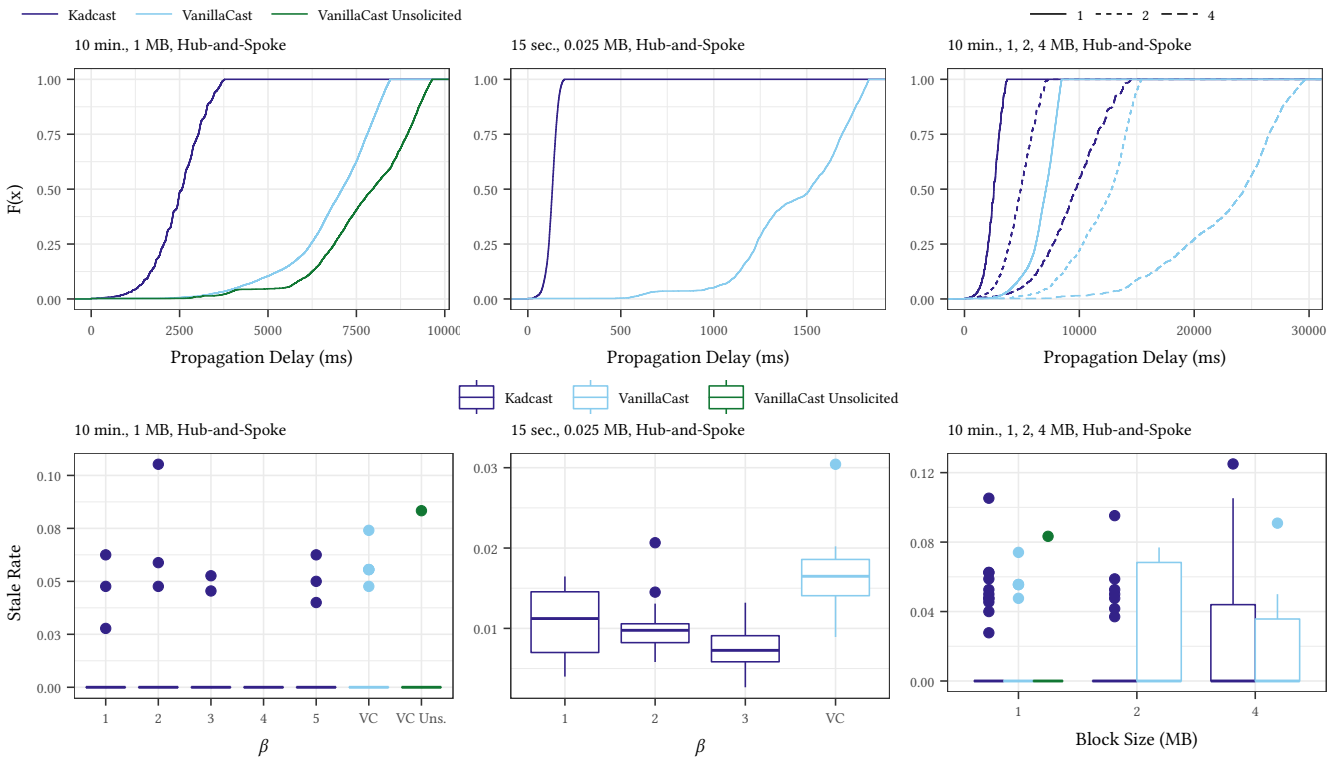

Figure 6: Block propagation delay and resulting stale rates for Bitcoin-like and Ethereum-like parametrizations, as well as for simulation scenarios with higher block size limits.

5.1.2 Simulator Parametrization. The parameters that fuel our blockchain simulation are mainly chosen in reference to the Bitcoin network. For example, the simulator draws its block size distribution from uniform random sampling over Bitcoin's daily average block sizes of the last year. Moreover, the mining difficulty is considered a network-wide fixed parameter, set to Bitcoin's current value. Likewise, we simulate 16 mining nodes configured with a certain hash power, according to the currently deployed mining pools found in the Bitcoin network [5]. Please note that these 16 mining pools in sum contribute approximately the total hash power. Based on the individual hash power, each miner samples the next mining events from an exponential distribution function, which simulates a Poisson process in accordance to the target block interval. As a result, we get a realistic model of block generation in blockchain networks and at the same time are able to simulate competing mining nodes and their block propagation in the network.

While our evaluation is based on a number of different setups, the results described are based on scenarios simulating the mining process in networks with $N = 500$ nodes. Every scenario was repeated 30 times for different seed values to ensure statistical significance of the conducted measurements. During the three hour simulation time, the 16 miners generated blocks and initiated broadcast operations employing one of the two networking stacks, i.e., VanillaCast or Kadcast. In the Kadcast case, if not stated otherwise, the results are based on the parameters $k = 100$ and $\alpha = 3$. In order to evaluate the protocol behavior under different network conditions, we furthermore implemented two network topology models, which we now discuss in detail.

5.1.3 Hub-and-Spoke Topology Model. As a baseline for the protocol evaluation, we use a simple hub-and-spoke topology where all nodes are connected to one central node representing "the Internet". Our assumption is that the Internet is not a bottleneck and therefore set the hub's link capacities to 100 Gbps and client bandwidth to 50 Mbps. Moreover, we sample link latencies from the publicly available data set of measured end-to-end median latencies in the Bitcoin network [43]. The hub-and-spoke model is a typical setup for the assessment of peer-to-peer overlays, and while it captures network effects, it does not rely on additional assumptions about the underlying topology. This creates an idealized simulation scenario that gives us the capability to assess the networking stacks based on a neutral, common ground.

5.1.4 Geographic Topology Model. As a counterpart to the idealized the hub-and-spoke model, we created a more topology model based on geographic clustering of nodes. Again, we aligned the basic model with the Bitcoin network, as we retrieved the publically available node list from the Bitnodes website [4], which yields a node distribution for the seven interconnected geographic clusters: North America, South America, Europe, Asia, Africa, China, and Oceania. Simulated nodes are distributed accordingly and are connected to their respective regional hubs. For each of the regions, we parametrized link latency distributions with measurements from

the iPlane [29] dataset and upload and download bandwidth distributions with broadband data from Speedtest.net [52]. In order to consider the findings of [31], we furthermore modeled China as a separate region and appointed a packet loss rate of 6.9 % to links in the region. Note that since we do not consider hosted nodes with server-grade connections, the geographic topology model captures a heavily resource-restricted environment that enables simulations in more complex network scenarios which foster a high degree of network effects.

5.2 Protocol Evaluation

In order to show the benefits of the Kadcast protocol in different environments, we created simulation scenarios with parametrizations mimicking Bitcoin (10 min. block interval and 1 MB block size limit) and Ethereum (15 sec. block interval, proportionally smaller block size limit of 25 KB [19]). Moreover, in light of the debates on block size limits in the Bitcoin community, we additionally analyzed the block propagation for increased block size limits of 2 MB and 4 MB.

5.2.1 Block Propagation Delay. As a first study, we investigated the performance of Kadcast compared to different instantiations of VanillaCast. The upper half of Figure 6 shows the block propagation delay to reach 90% of all nodes as cumulative distribution function $F(x)$: as expected, the block distribution time depends on the block size limit, that is, larger blocks take longer to propagate. The Kadcast protocol, however, delivers blocks significantly faster compared to VanillaCast in all cases. For example, Kadcast exhibits a mean propagation time of 2,500 ms to deliver blocks with a 1 MB block size limit (Bitcoin-like scenario, left plot), 4,312 ms faster than VanillaCast, and even 5,242 ms faster than unsolicited VanillaCast block propagation, which interestingly experiences additional queueing delay and other network effects stemming from its increased overhead. Compared to both cases, request-response and unsolicited VanillaCast, this amounts to 30 % faster block propagation in the Bitcoin-like scenario. Furthermore, Kadcast's performance really excels in the case of smaller intervals and smaller block sizes: in the Ethereum-like scenario (middle plot), Kadcast is able to deliver blocks on average more than 90 % faster than the VanillaCast baseline.

The improved propagation speed is also reflected by an overall faster network coverage: while it takes VanillaCast in the Bitcoin case 8,461 ms to reach 90% of the network (and unsolicited even 9645 ms), Kadcast is able to reach the same number of nodes more than 50% faster, at 3,784 ms. In the Ethereum case, Kadcast again fares even better and covers 90 % of the network 90 % faster than the unstructured VanillaCast networking layer. For the larger block sizes of 2 MB and 4 MB (right plot), Kadcast was also able to deliver blocks more than 50% faster on average and in total.

The results highlight that Kadcast is able to immensely speedup the block distribution in current blockchain systems, and that while it scales for larger block sizes, it meets the requirements of networks with quicker block intervals in particular.

5.2.2 Impact on Consensus Stability. The effect of the quicker block propagation is reflected in the median stale rate, i.e., rate of blocks that are mined, but do not become part of the final blockchain. As

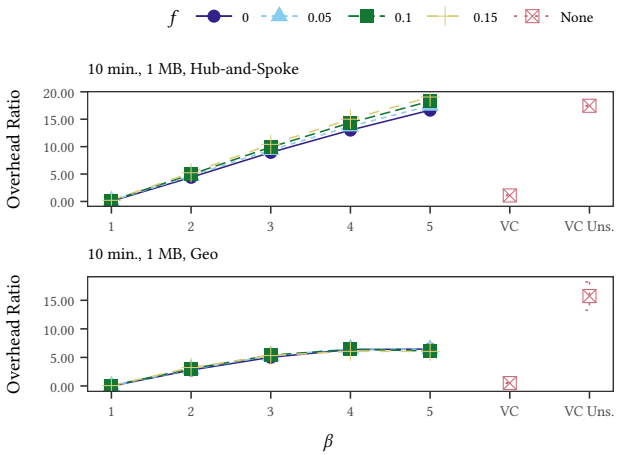

Figure 7: Overhead ratios for Hub-and-Spoke and geographic topologies, and parametrizations for f and β.

increased blockchain forks and wasted mining power weaken consensus security, the stale rate is an indicator for how the networking layer impacts security [27].

The boxplots in the lower half of Figure 6 show the stale rate in dependence of different choices for the redundancy parameter β: in the Bitcoin-like scenario Kadcast achieves a median stale rate of zero, barring the occasional outliers. This is comparable to the VanillaCast and unsolicited VanillaCast cases, which exhibit similar behavior. However, in the case of a decreased block interval, the ratio of propagation delay to block interval gets much larger, resulting in an overall increased stale rate. In this Ethereum-like scenario, VanillaCast exhibits a median stale rate of 0.017, which Kadcast is able to divide in half, achieving an overall median stale rate of 0.0085. Additionally, the influence of the performance increase for higher β values is visible, leading even to a median stale rate of 0.0073 for $\beta = 3$. The simulations with larger block size limits indicate that the additional stress on the network layer negatively impacts consensus security: while VanillaCast and Kadcast can retain a median stale rate of zero, the number of outliers increase in both cases. While Kadcast still fares better in the 2 MB case, VanillaCast exhibits better stale rates in the 4 MB scenarios, which suffer from more network effects, as we discuss further in Section 5.4.

In summary, the improved block propagation of Kadcast leads to a median stale rate that is comparable and often better than VanillaCast. This indicates that the consensus security of blockchain systems could benefit from employing the Kadcast protocol, especially networks with lower block intervals, such as Ethereum. Moreover, since blocks reach a larger share of the network much faster, the adoption of Kadcast could help to mitigate time-dependent adversarial mining strategies, such as selfish mining [21].

5.2.3 Broadcast Efficiency. In order to confirm the adjustability and efficiency of the Kadcast protocol, we recorded the total amount of traffic t_{total} produced during our simulation time. Furthermore, we accumulated the block sizes of all blocks generated during this time, t_{blocks}. As all blocks need to be transmitted to each node at least

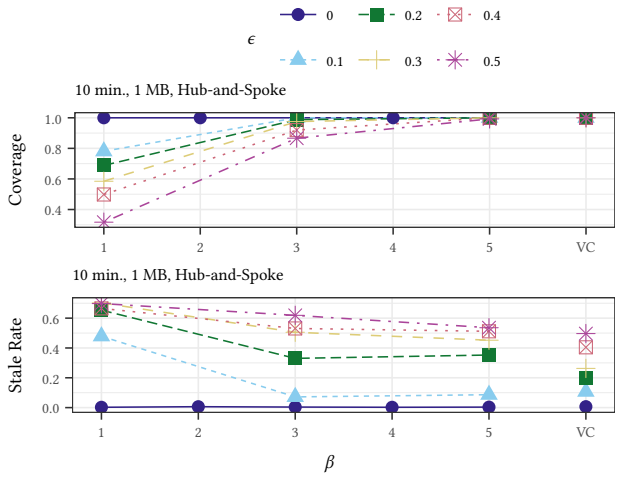

Figure 8: Network coverage and stale rate, when a share ϵ of adversarial nodes is introduced to the network.

once, the minimum amount of traffic for the broadcast operation can be calculated as $N \cdot t_{blocks}$. Accordingly, we define the overhead ratio as $r_o = (t_{total} - N \cdot t_{blocks})/(N \cdot t_{blocks})$, which describes how much additional traffic was generated during a simulation run, including all signaling messages.

The upper part of Figure 7 shows the results in the Hub-and-Spoke scenarios for different parametrizations of the redundancy parameters β and f: we observe that Kadcast's overhead increases linearly with β and f. We also note that for $\beta = 1$, Kadcast's overhead ratio is below the relay-response based VanillaCast, and that for $\beta = 5$ it is comparable to unsolicited VanillaCast. This shows the adjustability of the Kadcast approach, which allows for a fast, unsolicited block relay with controllable overhead.

5.3 Protocol Behavior under Attack

We moreover empirically evaluated how the Kadcast protocol fares in the face of an adversary obstructing block delivery. For this, we set up simulation scenarios in which a fraction ϵ of nodes were marked as adversarial and henceforth would cease to forward blocks. The upper part of Figure 8 shows the network coverage in dependence of ϵ and β: while Kadcast of course reaches 100 % network coverage for $\epsilon = 0$, its block propagation is severely hindered when malicious nodes are introduced and no redundancy exists ($\beta = 1$). However, the effect of the redundancy factor β is also clearly visible, ensuring 99 % coverage for $\beta = 3$ when $\epsilon \leq 0.3$ and for $\beta = 5$, when the share of malicious nodes would be even higher.

Interestingly, while VanillaCast's network coverage is not impaired by the introduction of adversarial nodes, it does exhibit degraded propagation performance due to the almost fragmented network. In fact, the resulting stale rates of both protocols are very similar, when confronted with such a powerful adversary (cf. lower part of Figure 8). The results show that, with a reasonably chosen set of parameters, Kadcast is resilient to a large amount of adversarial nodes and compares to the currently deployed VanillaCast networking layer.

5.4 Protocol Behavior in Complex and Resource-Restricted Environments

Additionally, in order to evaluate the Kadcast protocol in more complex networking environments, we reproduced the previously introduced scenarios in the resource restricted geographic topology model. Due to the lower bandwidths, larger latencies, packet losses, and more complex structure of this model, much more network effects come into play here. However, the results shown in Figure 9 follow the same tendencies as discussed before: in general, Kadcast provides much faster block propagation than VanillaCast. And again, it does especially well in the Ethereum-like networking scenarios with smaller, higher frequency blocks.

Nevertheless, the results also show that Kadcast exhibits degraded performance when facing a congested networking environment. In the lower part of Figure 7, we can already see the first signs of congestion: for $\beta > 3$, the overhead ratio stagnates, indicating that packet losses occur. In Figure 9, the results for the 4 MB block size limit clearly show a significantly increased propagation delay. Similar congestion effects can be seen for the unsolicited VanillaCast, where high overhead even in the 1 MB case forces the network traffic to a standstill at times.

The degraded network performance is also reflected by higher stale rates for networking protocols. However, as the effect is more taxing on the protocols with unsolicited block propagation, it highlights the robustness of the request-response and TCP-based VanillaCast in heavily congested networks. In particular, we therefore hold the introduction of a suitable congestion control mechanism to the Kadcast protocol as an important avenue of future research, especially if blockchain networks such as Bitcoin would opt to incorporate larger, more bandwidth hungry blocks. However, as our simulation results show degraded performance for larger block sizes and in congested networks for all networking stacks, the community should on the contrary think about reducing block sizes and intervals. We deem this to be generally preferable from a networking perspective, since it would lead to more uniform traffic patterns and effectively would help reduce network effects during peak traffic flows. Thereby, it also would alleviate network bottlenecks and foster decentralization.

6 RELATED WORK

In recent years, a large body of work proposed improvements for blockchain networks. Orthogonal to our approach, a number of contributions deal with transaction privacy. For example, Venkatakrishnan et al. and Fanti et al. propose protocol redesigns that improves anonymity of transaction propagation in the Bitcoin network [22, 57]. We, in contrast, are mainly concerned with block propagation.

The Graphene protocol [44] proposes a more efficient block transmission that augments the concept of compact blocks. Similarly, the recently proposed Velocity protocol [9] uses FEC on top of the existing network architecture. While improving on some aspects, such as the messaging overhead of the current block delivery method in Bitcoin, these protocols do not fundamentally change the prevalent block propagation model.

Third-party relay networks, such as the FIBRE network [25] or bloXroute [35] are supposed to improve the block distribution.

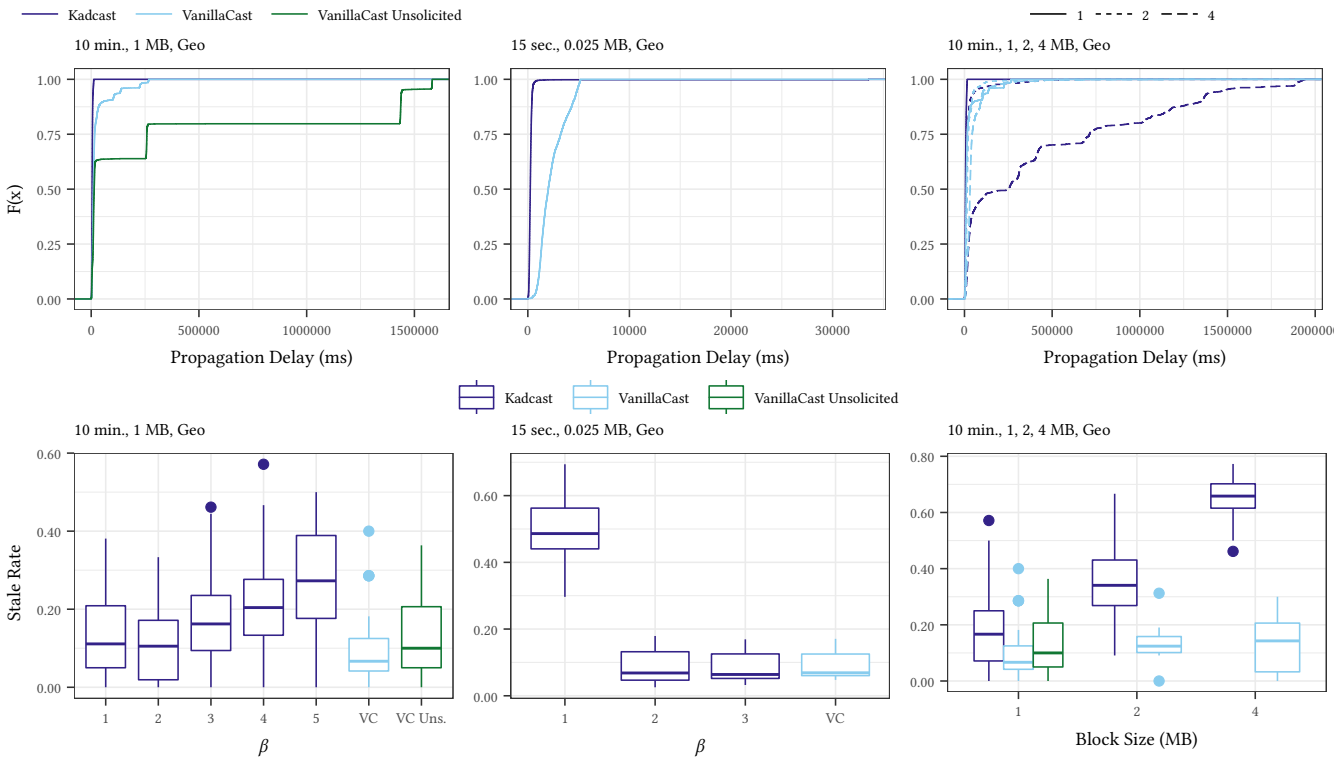

Figure 9: Block propagation delay and resulting stale rates for Bitcoin-like and Ethereum-like parametrizations, as well as for simulation scenarios with higher block size limits in the geographic network topology model.

While the emergence of these proposals clearly show the urgency of the problem, we deem them orthogonal to the goal of improving the peer-to-peer networks of blockchain systems themselves. First results however suggest [27] that a separate relay network has a negligible effect over switching to a faster, i.e., unsolicited block propagation scheme. Moreover, since such networks require central and manual coordination, they do not meet the blockchain design goals of decentralization.

A number of projects provide simulation frameworks for the Bitcoin network. The Shadow simulator, for example, was extended to incorporate the Bitcoin Core logic [30, 41]. Shadow focuses on simulating accurate application behavior by executing the application's actual source code. The simulation developed by Gervais et al., on the other hand, abstracts from the actual Bitcoin implementation and models its network behavior based on simplifying assumptions [27]. For example, the authors assume a network model that is congruent to the overlay's TCP connections, which does not capture network effects. Moreover, both simulators were not developed with interchangeable networking stacks in mind.

Beyond cryptocurrencies, contributions focusing on the broadcast in structured peer-to-peer networks are also relevant for our work. El-Ansary et al. [17] realize a perfect broadcasting operation based on the overlay structure of the Chord [54] peer-to-peer distributed hash table. Furthermore, a number of entries are concerned with the broadcasting operation in Kademlia [40]-based overlay networks [13, 46, 47]. Of these contributions, we highlight the work by Czirkos and Hosszú [13], as parts of Kadcast are based on the proposed scheme. However, to the best of our knowledge, we are first to adopt and evaluate a broadcasting algorithm based on a structured peer-to-peer network in the setting of a real-world application with respective additional requirements, e.g., in terms of security.

7 CONCLUSION

In this work, we presented Kadcast, a new protocol for fast, efficient, and secure block propagation for the Bitcoin network. While this initial entry focused on improving the block distribution, other operations could immensely benefit from Kadcast's superior performance and low overhead. Finally, we hope to initiate a discussion about alternative transport protocols in the blockchain space.

REFERENCES

[1] 2002. *IPTPS '02: Proceedings of the 1st International Workshop on Peer-to-Peer Systems* (2002-03).

[2] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. 2017. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. 375–392.

[3] Ingmar Baumgart and Sebastian Mies. 2007. S/Kademlia: A practicable approach towards secure key-based routing. In *ICPADS '07: Proceedings of the 13th International Conference on Parallel and Distributed Systems* (2007-12). 1–8.

[4] bitnodes. 2019. Homepage. Retrieved May 21, 2019 from https://bitnodes.earn.com

[5] blockchain.info. 2018. Hashrate Distribution. Retrieved May 6, 2018 from https://blockchain.info/pools?timespan=4days

[6] Nikita Borisov. 2006. Computational Puzzles as Sybil Defenses. In *P2P '06: Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing* (2006).

171–176.

[7] Miguel Castro and Barbara Liskov. 1999. Practical Byzantine Fault Tolerance. In *OSDI '99: Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation* (1999-02). 173–186.

[8] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. 2003. Making gnutella-like p2p systems scalable. In *SIGCOMM '03: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (2003-08). 407–418.

[9] Nakul Chawla, Hans Walter Behrens, Darren Tapp, Dragan Boscovic, and K Selçuk Candan. 2019. Velocity: Scalability Improvements in Block Propagation Through Rateless Erasure Coding. In *ICBC '19: Proceedings of the 1st International Conference on Blockchain and Cryptocurrency* (2019-05).

[10] Thibault Cholez, Isabelle Chrisment, and Olivier Festor. 2009. Evaluation of Sybil Attacks Protection Schemes in KAD. In *AIMS '09: Proceedings of the 3rd International Conference on Autonomous Infrastructure, Management and Security* (2009). 70–82.

[11] Matt Corallo. 2016. BIP 152: Compact Block Relay. https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki

[12] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. 2016. On Scaling Decentralized Blockchains - (A Position Paper). In *BITCOIN '16: Proceedings of the 3rd Workshop on Bitcoin Research* (2016-02). 106–125.

[13] Zoltán Czirkos and Gábor Hosszú. 2013. Solution for the broadcasting in the Kademlia peer-to-peer overlay. 57, 8 (2013), 1853–1862.

[14] Suhas Daftuar. 2015. BIP 130: sendheaders message. https://github.com/bitcoin/bips/blob/master/bip-0130.mediawiki

[15] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the bitcoin network. In *P2P '13: Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing* (2013-09). 1–10.

[16] John R. Douceur. 2002. The Sybil Attack, See [1], 251–260.

[17] Sameh El-Ansary, Luc Onana Alima, Per Brand, and Seif Haridi. 2003. Efficient Broadcast in Structured P2P Networks. In *IPTPS '03: Proceedings of the 2nd International Workshop on Peer-to-Peer Systems* (2003). 304–314.

[18] Ethereum Project. 2014. A next-generation smart contract and decentralized application platform. https://github.com/ethereum/wiki/wiki/White-Paper

[19] Etherscan.io. 2019. Ethereum Block Size History. Retrieved May 24, 2019 from https://etherscan.io/chart/blocksize

[20] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. 2016. Bitcoin-NG: A Scalable Blockchain Protocol. In *NSDI '16: Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation* (2016-03). 45–59.

[21] Ittay Eyal and Emin Gün Sirer. 2014. Majority is not enough: Bitcoin mining is vulnerable. In *FC '14: Proceedings of the 18th International Conference on Financial Cryptography and Data Security* (2014-03). 436–454.

[22] Giulia C. Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. 2018. Dandelion++: Lightweight Cryptocurrency Networking with Formal Anonymity Guarantees. 2, 2 (2018), 29:1–29:35.

[23] Giulia C. Fanti and Pramod Viswanath. 2017. Deanonymization in the Bitcoin P2P Network. In *NIPS '17: Proceedings of 30th Annual Conference on Neural Information Processing Systems* (2017-12).

[24] Sebastian Feld, Mirco Schönfeld, and Martin Werner. 2014. Analyzing the Deployment of Bitcoin's P2P Network under an AS-level Perspective. In *ANT '14: Proceedings of the 5th International Conference on Ambient Systems, Networks and Technologies* (2014-06). 1121–1126.

[25] Fast Internet Bitcoin Relay Engine (FIBRE). 2017. Homepage. Retrieved August 1, 2017 from http://bitcoinfibre.org

[26] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *EUROCRYPT '15: Proceedings of the 34th International Conference on the Theory and Applications of Cryptographic Techniques* (2015-04). 281–310.

[27] Arthur Gervais, Ghassan Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the Security and Performance of Proof of Work Blockchains. In *CCS '16: Proceedings of the 23nd ACM SIGSAC Conference on Computer and Communications Security* (2016-10).

[28] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. 2015. Eclipse Attacks on Bitcoin's Peer-to-Peer Network. In *USENIX Security '15: Proceedings of the 24th USENIX Security Symposium* (2015-08). 129–144.

[29] iPlane. 2019. An Information Plane for Distributed Services. web.eecs.umich.edu/~harshavm/iplane/.

[30] Rob Jansen and Nicholas Hopper. 2012. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. In *NDSS '12: Proceedings of the Network and Distributed System Security Symposium* (2012).

[31] Ben Kaiser, Mireya Jurado, and Alex Ledger. 2018. The Looming Threat of China: An Analysis of Chinese Influence on Bitcoin. abs/1810.02466 (2018).

[32] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. 2012. Double-spending Fast Payments in Bitcoin. In *CCS '12: Proceedings of the 19th ACM Conference on Computer and Communications Security* (2012-10). 906–917.

[33] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In *CRYPTO '17: Proceedings of the 37th Conference on Advances in Cryptology* (2017-08). 357–388.

[34] Lucianna Kiffer, Rajmohan Rajaraman, and Abhi Shelat. 2018. A Better Method to Analyze Blockchain Consistency. In *CCS '18: Proceedings of the 25nd ACM SIGSAC Conference on Computer and Communications Security* (2018-10). 729–744.

[35] Uri Klarman, Soumya Basu, Aleksandar Kuzmanovic, and Emin Gün Sirer. 2018. bloXroute: A Scalable Trustless Blockchain Distribution Network WHITEPAPER.

[36] Michael Kohnen, Mike Leske, and Erwin P. Rathgeb. 2009. Conducting and Optimizing Eclipse Attacks in the Kad Peer-to-Peer Network. In *NETWORKING '09: Proceedings of the 8th International IFIP-TC 6 Networking Conference* (2009). 104–116.

[37] Thomas Locher, David Mysicka, Stefan Schmid, and Roger Wattenhofer. 2010. Poisoning the Kad Network. In *ICDCN '10: Proceedings of the 11th International Conference on Distributed Computing and Networking* (2010). 195–206.

[38] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder. 2011. RaptorQ Forward Error Correction Scheme for Object Delivery. RFC 6330 (Proposed Standard). , 69 pages. https://www.rfc-editor.org/rfc/rfc6330.txt

[39] Yuval Marcus, Ethan Heilman, and Sharon Goldberg. 2018. Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network. 2018 (2018), 236.

[40] Petar Maymounkov and David Mazières. 2002. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric, See [1], 53–65.

[41] Andrew Miller and Rob Jansen. 2015. Shadow-Bitcoin: Scalable Simulation via Direct Execution of Multi-Threaded Applications. In *CSET '15: Proceedings of the 8th Workshop on Cyber Security Experimentation and Test* (2015-08).

[42] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system.

[43] Karlsruhe Institute of Technology DSN. 2018. Bitcoin Monitoring. Retrieved April 26, 2018 from https://dsn.tm.kit.edu/bitcoin/

[44] A. Pinar Ozisik, Gavin Andresen, George Bissias, Amir Houmansadr, and Brian Neil Levine. 2017. Graphene: A New Protocol for Block Propagation Using Set Reconciliation. In *CBT '17: Proceedings of the 1st International Workshop on Cryptocurrencies and Blockchain Technology* (2017-09). 420–428.

[45] Rafael Pass, Lior Seeman, and Abhi Shelat. 2017. Analysis of the Blockchain Protocol in Asynchronous Networks. In *EUROCRYPT 17: Proceedings of the 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2017). 643–673.

[46] Antonio Delgado Peris, José M. Hernández, and Eduardo Huedo. 2012. Evaluation of the Broadcast Operation in Kademlia. In *HPCC '12: 14th IEEE International Conference on High Performance Computing and Communication* (2012). 756–763.

[47] Antonio Delgado Peris, José M. Hernández, and Eduardo Huedo. 2016. Evaluation of alternatives for the broadcast operation in Kademlia under churn. 9, 2 (2016), 313–327.

[48] ns-3 Network Simulator. 2018. Homepage. Retrieved May 5, 2018 from https://www.nsnam.org

[49] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. 2016. SPECTRE: A Fast and Scalable Cryptocurrency Protocol. 2016 (2016), 1159.

[50] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure High-Rate Transaction Processing in Bitcoin. In *FC '15: Proceedings of the 19th International Conference on Financial Cryptography and Data Security* (2015-01). 507–527.

[51] Giorgio Alfredo Spedicato. 2017. Discrete Time Markov Chains with R. *The R Journal* 9, 2 (2017), 84–104.

[52] Speedtest.net. 2019. Global Index. https://www.speedtest.net/global-index.

[53] Moritz Steiner, Taoufik En-Najjary, and Ernst W. Biersack. 2007. Exploiting KAD: possible uses and misuses. 37, 5 (2007), 65–70.

[54] Ion Stoica, Robert Tappan Morris, David R. Karger, M. Frans Kaashoek, and Hari Balakrishnan. 2001. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (2001). 149–160.

[55] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. 2011. A survey of DHT security techniques. 43, 2 (2011), 8:1–8:49.

[56] Marie Vasek, Micah Thornton, and Tyler Moore. 2014. Empirical analysis of denial-of-service attacks in the Bitcoin ecosystem. In *BITCOIN '14: Proceedings of the 1st Workshop on Bitcoin Research* (2014-03). 57–71.

[57] Shaileshh Bojja Venkatakrishnan, Giulia C. Fanti, and Pramod Viswanath. 2017. Dandelion: Redesigning the Bitcoin Network for Anonymity. (2017).

[58] Vivek Vishnumurthy, Sangeeth Chandrakumar, and Emin Gun Sirer. 2003. Karma: A secure economic framework for peer-to-peer resource sharing. In *P2PEcon '03: Proceedings of the 1st Workshop on Economics of Peer-to-Peer Systems* (2003-06).

[59] Gavin Wood. 2014. Ethereum: A Secure Decentralised Generalised Transaction Ledger. http://gavwood.com/Paper.pdf

A BITCOIN'S PEER-TO-PEER NETWORK

The backbone of Bitcoin is an unstructured peer-to-peer network: every peer establishes 8 outgoing TCP connections to a random set of neighbor peers. Some peers additionally accept incoming connections (by default up to 117).[4] Once a peer established a TCP connection to another peer, it initiates a handshake and starts exchanging address (ADDR) messages in order to advertise itself to the network and to update its peer cache. Next, the joining peer proceeds to fetch blocks missing from its current local state of the blockchain.

When a peer issues new transactions, it announces them through inventory (INV) messages that are periodically broadcast in the network: every peer sends INV messages to each of its neighboring peers in intervals corresponding to a Poisson process. At the time of writing, the default average interval between messages is set to be 5 seconds. This artificial delay was introduced to improve the privacy properties of the system, as observing the origin of a transaction allows to link it to the IP address of the sender and thereby enables deanonymization attacks.[5] Each peer receiving an INV message checks if it knows the announced transactions and retrieves missing data items by sending a corresponding GETDATA request, which is answered by TX messages. After the peer verified the received transactions, it continues the broadcast by adding them to the queued inventory messages. Up until Bitcoin Core v0.12.0, inventory messages were also the default way to announce new blocks in the network. Peers receiving such messages would then request block headers via GETHEADERS and block data via GETDATA messages. These would be answered by HEADERS and BLOCK messages, respectively. Note that the delay just described does not affect the inventory-based block propagation, since INV messages containing blocks are sent right away.

With the update to protocol version 70012, a new default method for block propagation was introduced [14]: after the initial handshake, each peer signals its support for the new propagation method by sending a SENDHEADERS message. From this point on, new blocks are announced directly via the HEADERS message, which reduces the messaging overhead and propagation delays. Note that when more than one block has to be announced, Bitcoin Core falls back to the old INV method. Additionally, the option for *compact block relay* [11] allows a peer to request block announcements to be sent in a more bandwidth efficient manner. In particular, it allows the peer to only retrieve transaction data it is missing from an announced block, which can severely reduce the bandwidth overhead of block propagation, but is prone to induce an additional latency overhead.

B ETHEREUM'S PEER-TO-PEER NETWORK

The notion of *smart contracts* is no alien concept to Bitcoin: transactions can do more than transferring funds from one address to another. In fact, Bitcoin transactions hold bytecode, which is executed by all peers validating the transaction. The code determines whether to commit a state transition, making the network one large distributed state machine.

While this perception is sort of an afterthought in Bitcoin, the Ethereum [18, 59] project is a blockchain-based platform specifically dedicated to the distributed execution of Turing-complete smart contracts. Even though the scope of Ethereum is different, the blockchain design and the corresponding network protocols share more than a few similarities with Bitcoin. Interestingly, Ethereum's peer management is based on Kademlia, which is used to populate a peer lookup table. However, for block and transaction propagation Ethereum uses an unstructured TCP-based overlay network, very similar to Bitcoin. Each peer maintains a limited number of connections, currently $\texttt{MaxPeers} = 25$, of which $\lceil \texttt{MaxPeer}/2 \rceil = 13$ are outbound connections.[6] Just like in Bitcoin, the contract code is first propagated in the form of a transaction broadcast: each peer receiving a transaction directly forwards it to all neighbors who do not already know about it. When smart contract transactions reach a miner, they are validated (i.e., executed) and consolidated to blocks. These are then again broadcast, whereby blocks are only directly forwarded to subset of $\sqrt{n}$ neighbors, and are otherwise advertised via inventory messages. A peer receiving an advertisement waits 500 ms before further advertising the block. All peers in the network accept the longest known chain of blocks[7] as the currently valid blockchain.

We observe that Ethereum is indeed built on a hybrid networking stack that utilizes Kademlia primitives for peer discovery, but still relies on an unstructured overlay for information propagation. Even though most of the network operations resemble those of Bitcoin, Ethereum makes use of unsolicited transaction and block propagation, which is probably needed to allow for much tighter block synchronization intervals demanded by Ethereum's short block time of approximately 15 s. This suggests on the one hand that Ethereum may have worse privacy guarantees than Bitcoin, which introduces an artificial transaction propagation delay to obfuscate the distance of an observer to the source of the transaction. On the other hand, unsolicited relaying increases the bandwidth overhead of the broadcast procedure. While we did not recreate every detail of the Ethereum network, the experiments with smaller block intervals clearly suggest that the Ethereum peer-to-peer network may especially benefit from Kadcast's improved performance and efficiency.

[4] Note that we only consider *full nodes*, i.e., peers running the Bitcoin Core software and holding a full copy of the blockchain.

[5] Note that this method of obscuring the origin of a transaction replaced the *trickling* method and was introduced with Bitcoin Core version 0.12.0. However, as Fanti and Viswanath highlight in [23], the new method does not improve much on the old: both exhibit rather poor anonymity properties.

[6] There are a number of different implementations. Here, we consider `geth`, the most prevalent Ethereum software.

[7] In contrast to claims in [59], Ethereum does not implement GHOST [50] as chain selection policy, but still relies the longest-chain policy while rewarding miners of stale blocks if they include appropriate uncle blocks [27].