Kadcast-NG: A Structured Broadcast Protocol for Blockchain Networks

Elias Rohrer and Florian Tschorsch

***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. In contrast, the Kadcast protocol is a structured peer-to-peer protocol for block and transaction propagation in blockchain networks. Kadcast utilizes the well-known overlay topology of Kademlia to realize an efficient broadcast operation with tunable overhead. We study the security and privacy of the Kadcast protocol based on probabilistic models and analyze its resilience to packet losses and node failures. Moreover, we evaluate Kadcast's block delivery performance, broadcast reliability, efficiency, and security based on advanced network simulations. Lastly, we introduce a QUIC-based prototype implementation of the Kadcast protocol and show its merits through deployment in a large-scale cloud-based testbed.**

***Index Terms*—blockchain, peer-to-peer networks, broadcast**

I. Introduction

BITCOIN [1] 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.

That is, 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 [2, 3, 4]. 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 [5]. As this opens opportunities for fraud [6, 7], 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 [5], it leads to flooding the network with block data, which the current networking paradigm cannot handle. The emergence of third-party relay networks [8, 9] 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.

Kadcast [10] is a new structured broadcast protocol for blockchain networks based on the Kademlia [11] architecture. Kadcast allows for a more efficient broadcast operation with tunable redundancy and overhead. To this end, it exploits the structured overlay topology to delegate broadcast responsibilities for subtrees with decreasing height. In this work, we complement and extend the study of the Kadcast protocol and explore the feasibility of a real-world adoption. We propose a number of protocol extensions in order to further harden Kadcast in the face of security and privacy threats. In particular, we show that Kadcast achieves a high degree of reliability and resilience in the face of random and adversarial node failures, and provide an analysis of transaction privacy.

In addition, we developed *Kadcast-NG*, a QUIC-based [12] prototype implementation of the Kadcast protocol. We use Kadcast-NG to evaluate the approach in global-scale testbed hosted by Google Compute Cloud. Our evaluations show that Kadcast distributes blocks significantly faster than the currently deployed blockchain protocols in both simulated scenarios and real-world deployments. Kadcast furthermore increases the efficiency of block propagation, making room to introduce additional features such as unsolicited block push. Our simulation results also indicate that it achieves similar, often better, security results in terms of the stale block rate.

Our key contributions can be summarized as follows: (i) We present an efficient broadcast protocol for blockchain networks, which exploits Kademlia's structured architecture. (ii) We introduce parallelization to improve the reliability and resilience of our algorithm in a completely adjustable and predictable way. (iii) We discuss attack vectors and provide mitigation strategies against Sybil and Eclipse attacks, and analyze the network's resilience to adversarial nodes obstructing block delivery or trying to deanonymize transaction originators. (iv) We conduct a comprehensive simulation

The authors are with the Distributed Security Infrastructures Group, Technical University of Berlin.
E-mail: elias.rohrer@tu-berlin.de and florian.tschorsch@tu-berlin.de

study and evaluate the performance, reliability, efficiency, and security of Kadcast in comparison to a *vanilla* blockchain protocol, which generalizes the currently prevalent networking paradigm. (v) We provide Kadcast-NG, a prototypical implementation, and confirm its benefits through comparative real-world deployment in a global testbed.

The remainder of this paper is structured as follows. First, we describe primitives of information dissemination currently found in the blockchain landscape in Section II. In Section III, we present the Kadcast protocol and discuss its adjustable redundancy. Subsequently, we analyze Kadcast's security and privacy, and discuss various threats and mitigation strategies in Section IV. We present our findings from the simulative evaluation in Section V, and introduce the results of our testbed deployment in Section VI. In Section VII, we discuss related work, before we conclude the paper in Section VIII.

II. Information Dissemination in Blockchain Networks

The central purpose of blockchain protocols is to keep track of the current state of accounts, which are bound to cryptographic key pairs. To this end, new transactions are disseminated in the peer-to-peer network, whereby they reach *validator* nodes. Elected validators (a.k.a., "round leaders") collect and verify the transactions, decide on an authoritative transaction ordering, and finally batch them into blocks, i.e., updates to the global blockchain state. 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 [1], Proof-of-Stake [13], or PBFT [14]. After their issuance, blocks are broadcast to all nodes in the network, which verify them and apply the updates to the local ledger state. While such blockchain protocols have been proven to be consistent in the partial synchronous network model, they are based on the assumption that blocks are delivered to all nodes in a timely fashion, i.e., faster than a propagation delay limit [15]. As this restricts the interval in which new blocks of a certain size may be safely issued, blockchain scalability and consensus security have been shown to depend on the performance and reliability of the networking layer [2, 5].

Currently, most blockchain networks employ a network stack based on an *unstructured* overlay construction, to which we in the following refer to as the *vanilla* approach. When joining such an unstructured overlay network, nodes retrieve the addresses of a number of other participants by the means of an adequate bootstrapping mechanism. Then, they establish TCP connections to a random subset of *neighbor* nodes $R \subset N$. As nodes are only able to communicate to the rest of the peer-to-peer network via these neighbors, messages are passed hop-by-hop in a store-and-forward manner. In particular, upon arrival of new transactions and blocks, each 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 advertised 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, which potentially introduces a large amount of superfluous messages. Therefore, this kind of broadcast operation exhibits a high message complexity ($\mathcal{O}(N \cdot R)$), which has been shown to have severe consequences on network scalability in the past [16].

Therefore, some blockchain networks try to reduce the net accruing traffic by introducing alternative messaging patterns, such as request-response or compact block [17] propagation schemes. However, while such schemes may reduce the network utilization, they also add at least one round-trip time (RTT) per hop to the message propagation delay. Such delayed block propagation has been shown before to be unfavorable compared to *unsolicited* block propagation [5], if the network is capable of handling the increased load. However, in the *vanilla* case of an unstructured overlay network, unsolicited block relay would amount to blindly flooding the network. As a consequence of the ensuing congestion, blockchain networks would likely face degraded block propagation performance and serious issues regarding the consensus stability.

For more detailed information on the Bitcoin [1] and Ethereum [18] networking layers, we refer the reader to literature [19].

III. The Kadcast Protocol

Kadcast is based on Kademlia [11], 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 operation. 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.

A. Overlay Construction

Kademlia is a 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 overlay. An example of such a tree for a 4-bit address space is shown in Figure 1. Nodes use their local view to traverse the network structure efficiently, yielding a message complexity of $\mathcal{O}(\log N)$. To this end, 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.

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 $\texttt{ID}_0$

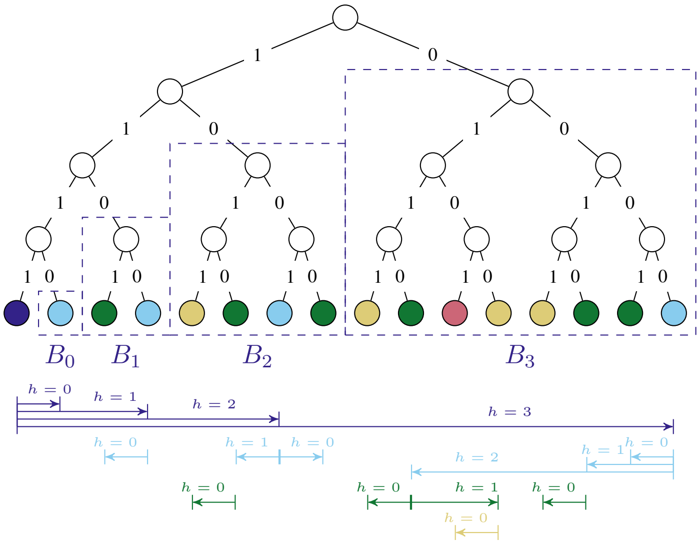

Figure 1: Example broadcast initiated by node `1111` ($\beta = 1$). Colors indicate node distances in the spanning tree, relative to the initiator.

holds buckets $B_i, i = 0 \ldots L - 1$, whereby bucket B_i holds the node information of k nodes with $\mathtt{ID}_j$ so that $2^i \leq d(\mathtt{ID}_0, \mathtt{ID}_j) < 2^{i+1}$. It follows that the node space covered by each bucket is exponential with i. 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 1, the 4 buckets of node $\mathtt{ID}_0 = \mathtt{1111}$ would hold nodes from the ranges `1110`, `110`$\star$, `10`$\star\star$, and `0`$\star\star\star$, respectively. If a node wants to add a new entry to a given bucket that already holds k entries, we employ a least recently used (LRU) drop policy. Before dropping an entry from the list, the peer also checks whether the respective nodes is still reachable, and only drop it otherwise. 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 [20].

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 every seen message updates not only the sender's but also the recipient's buckets. This soft-state protocol design allows for a very lightweight overlay membership management that keeps the footprint of the required information base to a minimum.

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.

Algorithm 1 Redundant broadcasting algorithm.

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

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 to nodes which are closer to the `ID`. To this end, (1) the node looks up the α closest nodes with respect to 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.

B. Message Propagation

As described before, most blockchain networks rely on TCP-based transport protocols for block and transaction propagation, which ensure the reliable transmission of arbitrarily large data by retransmitting missing segments in case of packet loss. This method implicitly assumes long-lived connections and requires additional state-keeping in terms of connection management and is therefore less scalable. Retransmissions and head-of-line blocking might introduce additional delays and unpredictable message overhead. In contrast, transport protocols such as UDP or QUIC [12] enable a more scalable and dynamic approach with short-lived, low-cost transmissions, which complement Kadcast's design. While this allows for a lightweight protocol design with reduced state-keeping and tunable per-link message complexity, it also entails handling data serialization and reliable transmission on the application layer.

Therefore, when the propagation of a block or a transaction is initiated, Kadcast first segments the data in *chunks* that are then distributed in the network according to the broadcast procedure described in Algorithm 1, which is a modified version of the algorithm in [21]. 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 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.

Consequentially, 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 1: node $\mathtt{ID}_0 = \mathtt{1111}$ initiates a broadcast in the network, and sends four `CHUNK` messages with heights $h = 0 \dots 3$ to one random node picked from each of the respective buckets $B_i, \ i = 0 \dots 3$. 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 $\mathcal{O}(\log n)$ steps. Upon receipt of all chunks required to rebuild a block or transaction message, the node follows Bitcoin's typical verification procedure before continuing the broadcast operation.

C. Reliability of UDP-based Message 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 required data exactly once and hence no duplicate messages would be introduced by this broadcasting operation. Unfortunately, we cannot make these assumptions and have to consider packet losses, as well as adversarial and random failures during transmission. In the example of Figure 1, 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 corresponding message. That is, in the worst case, a single transmission failure caused by the unreliable UDP transport protocol could result in a network coverage of fifty percent only. Therefore, the broadcast algorithm is improved and secured by two different approaches, which both introduce redundancy.

In the following, we present means of improving the broadcast reliability and provide a first analytical assessment of the respective approaches.

1) Improving Broadcast Reliability and Performance: Firstly, 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 message delivery.

Secondly, Kadcast has to consider transmission failures due to corrupted and/or dropped packets on every hop of the propagation. When Kadcast is implemented on top of an unreliable transport protocol, such as UDP, it therefore needs to increase the reliability of the transmission and hence employs a forward error correction scheme based on RaptorQ [22] 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 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 message delivery fails entirely, and to enable the initial bootstrapping of the blockchain, the Kadcast protocol incorporates a simple `REQUEST` message that allows nodes to query others for specific blocks or transactions, and is answered by the corresponding `CHUNK` messages.

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 message). This improved algorithm can be seen in Algorithm 1. Please note that for $\beta = 1$, Algorithm 1 describes the "optimal" broadcast from Section III-B.

Along the lines of [21], we model the 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

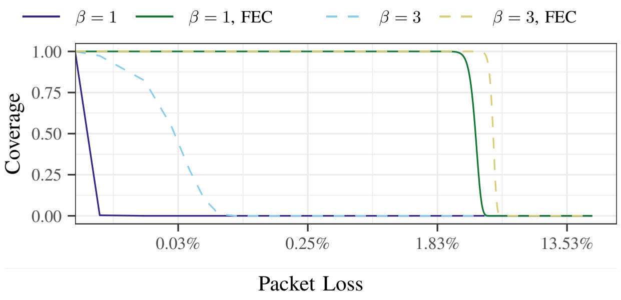

Figure 2: Reliability of UDP-based block propagation over unreliable channels (block size of 1 MB, $L = 160$, and FEC overhead factor $f = 0.15$).

the coverage of a redundant broadcast, we need to extend 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 or transaction is thus $p_b = P(X = s) = p^s$, which induces a failure probability of $\epsilon_b = 1 - p_b$. Accordingly, the probability to deliver a message with redundancy β is given by $p_{b,\beta} = 1 - \epsilon_b^\beta$. These observations yield an expected coverage ratio of

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

As the transmission of blocks is only successful, if all chunks are received, the propagation of entire blocks is not as reliable. To this end, we analyze the expected UDP-based block broadcast coverage in the face of different packet loss rates, $\beta \in \{1, 3\}$, and an assumed block size of 1 MB. The results are shown in Figure 2: we observe that without redundancy even the smallest packet loss makes the probability of delivering a block drop immediately, hence rendering the chance of covering the entire network virtually impossible. While a redundancy factor $\beta = 3$ has a positive impact on the block propagation, it is not sufficient on its own to guarantee the reliable transmission of entire blocks over a lossy channel. However, the parallelized broadcast is still necessary either way to compensate adversarial and random node failures, and to improve the propagation performance, as discussed before.

3) Analysis of FEC-based Block Delivery: In order to further increase the block transmission reliability, a Kadcast node employing the RaptorQ forward error correction 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 2 clearly shows the improved transmission reliability offered by introducing forward error correction with 15% redundancy ($f = 0.15$): this approach ensures that broadcasted blocks reach full network coverage for packet loss rates up to around 9%.

However, this can still be improved by combining the FEC 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 2 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 UDP-based 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 or transaction delivery.

IV. Kadcast Security and Privacy

Fast and fair block propagation is considered security-critical for the consensus layer of blockchain-based systems. However, the peer-to-peer network and the block propagation mechanism may also become themselves subject to attacks on security and privacy. In the following, we therefore discuss the security and privacy properties of the Kadcast network protocol. In particular, we make the security and privacy threats transparent and provide a series of mitigation strategies.

A. Adversary and Threat Model

We assume an active adversary who joins the peer-to-peer network. More specifically, the adversary has the means and resources to run and maintain a number of nodes. The adversary's intention is to attack the network as a whole or to isolate specific nodes by occupying strategic positions in the network. Please note that this includes, both, attacks on security as well as privacy of the network and its users.

Numerous previous entries study Kademlia's security properties, its behavior when attacked by a range of adversaries, and designs improving on its security [23, 24, 25, 26, 27, 28, 29, 30]. In the following, we discuss the most prevalent adversarial threats to the security of blockchain peer-to-peer networks in general and Kadcast in particular.

1) Sybil attacks: The notion of a Sybil attack [31] 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 outnumber the honest nodes participating in a distributed system, effectively increasing the share of malicious nodes 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 [27, 29]. The ability to run this kind of attack is often a prerequisite to be able to run Eclipse attacks (see next section) on Kademlia-based systems. In the case of Kadcast, if an adversary can forge arbitrary IDs and position herself close to a target, she may be able to increase the likelihood of receiving lookups and broadcasts from this node. This may enable the adversary to simply refuse block delivery and thereby obstruct the block propagation. 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.

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 and then further exploit the protocol. They have been shown to enable double-spending and selfish-mining attacks [20, 32]. In the past, the feasibility of Eclipse attacks on the Kademlia protocol have been studied in literature [27, 29, 30]. 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.

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. Additionally, previous work [33] highlighted that node operators have become targets of DDoS in the past. In the worst case, this results to a node failure, which might impair the broadcast operation.

4) Transaction Privacy: Previous works highlighted that the propagation of messages on the network layer may compromise user privacy. In particular, it has been shown that malicious nodes may passively monitor the network and link transaction data to identifying information such as IP addresses, thereby possibly deanonymizing their origin [34, 35, 36]. Furthermore, the notion of AS-level adversaries monitoring the data flows of blockchain networks has been discussed in literature [37, 38]. Such passive attacks on privacy are a threat to the Kadcast network protocol in particular, since broadcast messages include a height field, which is additional information that may allow an adversary monitoring transaction propagation to infer her distance from the transaction's origin.

B. Best Practice Mitigation Strategies

The Kadcast protocol employs a number of countermeasures in order to increase its resilience to the various threats. Later in Sections IV-C–IV-E, we present Kadcast-specific countermeasures. Here, we concentrate on best practice mitigation strategies and how they can be integrated into Kadcast.

First of all, Kadcast follows a bucket eviction policy that favors older, more stable nodes over newly acquired node addresses. By following this policy, we increase an adversary's efforts, requiring her to run nodes for longer periods. Even stricter bucket policies, which enforce a certain degree of diversity from an AS-level and/or subnet perspective [37, 38], are conceivable. The general approach effectively prevents the adversary from easily supplying all nodes known to the victim.

In addition, the Kadcast protocol can easily be extended to incorporate cryptographic puzzles as Sybil and Eclipse protection, similar to [28, 39, 40]. Along the lines of proof-of-work mining, 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. That 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 system parameter. 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}. The inclusion of this hash puzzle scheme impairs the ability of an adversary to quickly generate a large number of node identifiers. Evidently, the use of cryptographic puzzles also has its difficulties with respect to difficulty adjustments and a skewed distribution of computational power.

In general, we can observe however that by safeguarding the node identifiers and enforcing rigorous bucket policies, Kadcast follows best practices for Sybil and Eclipse protection [32].

In order to avoid DoS attacks, blockchains like Bitcoin employ a store-and-forward propagation policy: each time a node receives a new block, it 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 sticks to this DoS protection scheme.

Lastly, communication between nodes should be encrypted to mitigate network-level threats. Unfortunately, many blockchain networks, e.g., Bitcoin, do not employ this mitigation strategy. For Kadcast, we propose to encrypt messages at the transport layer [41], which hinders an AS-level adversary from passively monitoring the network traffic. More specifically, we suggest to implement a trust-on-first-sight public key pinning policy and to derive the node identifiers from the corresponding public keys. We consider this is an easy way to bind the protocol to cryptographic identities without necessitating a dedicated (possibly centralized) public-key infrastructure (PKI).

C. Identifier Space Randomization

As just discussed, it is essential for network security and privacy to impair an adversary's capability to determine and exploit a node's position in any distribution graph for a given block or transaction broadcast. In particular, an adversary should not be able to pre-populate nodes' buckets with pre-calculated identifiers in a targeted fashion. To this end, Kadcast employs *identifier space randomization* in order to facilitate distribution graphs that cannot be easily anticipated by an adversary. However, a complete randomization of the identifier space would abandon the network structure and hence also render Kadcast's efficient broadcast algorithm based on the XOR-metric non-deterministic. We therefore implement pseudo-randomized views of the identifier space for each broadcast operation. In this regard, additional temporary routing tables are created for each block or transaction broadcast, in which buckets are populated based on a salted distance metric $d_s(x, y) = (x \oplus y \oplus s)_{10}$, where s describes a salt value derived from the message data to be propagated, e.g., the hash of the corresponding block or transaction message. By applying such a temporary transformation to the node identifier space, we make sure that, on the one hand, node locations cannot be predicted by any adversary, but that on the other hand the network view of all nodes remains the same on a per-broadcast basis.

D. Obstruction of Block Delivery

The reliability of Kadcast depends 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 foresee or cheat the placement mechanism, i.e., has to hash node identifiers like everyone else, resulting in a uniform coverage of the randomized identifier space. In fact, this is a set of very conservative assumptions, 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 III-C.

Table I: Markov Simulation Results

Parameters				Results	
N	M	ϵ	β	p_ϵ	p_d
11,000	1,000	0.1	3	0.00075	0.993
12,000	2,000	0.2	3	0.0046	0.957
13,000	3,000	0.3	3	0.0122	0.888
11,000	1,000	0.3	5	0.00065	0.994
15,000	5,000	0.5	5	0.0041	0.963

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 as well as identifier space randomization. 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. In this model, 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` [42], 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 I; 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 with more than 96% probability in a highly adversarial environment where 50 % of nodes are controlled by the adversary.

E. Privacy of Transaction Propagation

In the following, we discuss what privacy Kadcast can provide during transaction propagation and what improvements to the baseline protocol should be considered in other to further mitigate the possibility for attacks on user privacy.

1) Prepending a Random Walk: As the Kadcast protocol is designed with privacy in mind, its protocol messages generally do not include unnecessary information that may be utilized by an adversary in order to gain additional knowledge on their origin. However, one necessary exception to this rule is the *height* field that is part of the broadcast messages carrying block and transaction data. As this field is initialized with value L and is decreased with every subsequent forward, an adversary receiving a broadcast message is able to infer her distance from the broadcast originator in the propagation graph. This is particular worrisome on the first hop, as an adversary receiving a message with height $L-1$ would be able to conjecture that the sender of this message is also its origin. While this may be no issue in the case of block propagation—which can be assumed to be not privacy-critical—it may be very detrimental to user's privacy in the case of transaction propagation.

Borrowing from the idea of the Dandelion protocol [43, 44], Kadcast therefore improves the privacy of transaction propagation by prepending the spreading process with a privacy phase, i.e., introducing an initial random walk. To this end, the originator of a transaction broadcast initially sets the height field to an otherwise unused magic number (e.g., $L+1$) in order to signal that the broadcast is still in the privacy phase and sends corresponding messages to β initial peers. After receiving these messages, each of these peers reads the height field and throws a weighted coin in order to decide whether to continue the random walk, i.e., forward the message to a single other peer, or immediately initiate the spreading of the transaction message as discussed above. It follows that the expected length of the random walk can easily be adjusted through the weight parameter of the coin toss.

2) Resilience to Passive Attacks on Privacy: In order to study the privacy that Kadcast can provide, we in the following assume scenarios in which an adversary controls an embedding of M out of N nodes that passively monitor the network for propagated transactions. We furthermore assume that the adversary has no knowledge of additional information that would allow her to infer the propagation path, but is only capable of passively recording from which node the controlled observation points received a transaction. That is, the adversary classifies observations according to a so-called *first-spy* estimator model and assumes the first encounter to be the transaction's origin [43]. As additional confounding factors—such as considering varying verification times, link latencies, or bandwidths—would only increase entropy and hence reduce an adversary's capabilities, we omit modelling any message delivery timings, but assume that network messages arrive in the order in which they were initially sent.

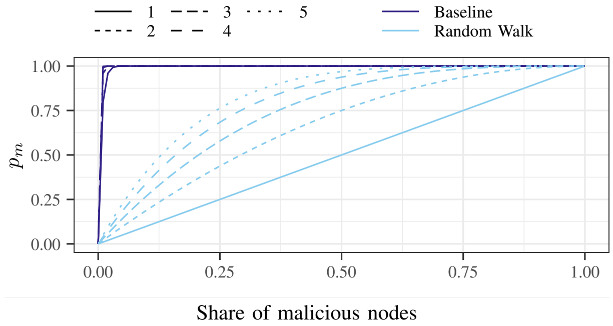

Figure 3: Probability of detection with and without a prepended privacy phase in dependence of the share of malicious nodes and the parallelization factor β ($L = 160$).

As a consequence, the adversary estimates the correct transaction origin, if *any* of the M malicious nodes receive a propagated message first, i.e., it is selected by the origin as one of the $d_1 = \beta(L-1)$ initial bucket delegates. This *probability of detection* is the probability that at least on malicious node is initially chosen, i.e.,

$$p_m = \overline{p_h} = 1 - \left(\frac{N-M}{N}\right)^{d_1}.$$

By utilizing the privacy phase, the number of delegates directly contacted by the originator is reduced from $d_1 = \beta(L-1)$ to $d_1 = \beta$, hence resulting in a lower probability of detection.

Figure 3 shows the probability of detection with and without a prepended privacy phase in dependence of the parallelization factor β. We observe that in the baseline Kadcast protocol, an adversary would indeed have a really high probability of guessing the originator of a transaction correctly. However, we also can see that the privacy phase immensely improves this and depending on the degree of parallelization may even come close to the optimal probability proportional to the share of malicious nodes M/N. Finally, we observe that there is a trade-off between privacy and reliability: on the one hand, the more nodes β the transaction originator initially contacts, the higher is the probability that a passive adversary gains information on the transaction's origin. On the other hand, as we previously explicated, a too low β value may leave the broadcast operation open for censorship attacks.

V. Evaluation: Network Simulations

In this section, we evaluate the block distribution performance, the 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.

A. Simulation Model

Our network simulation study utilizes the network-centric `bns` [10, 45] blockchain simulation framework, which is based on the time-discrete ns-3 network simulator [46]. The parameters that fuel our blockchain simulation are therefore

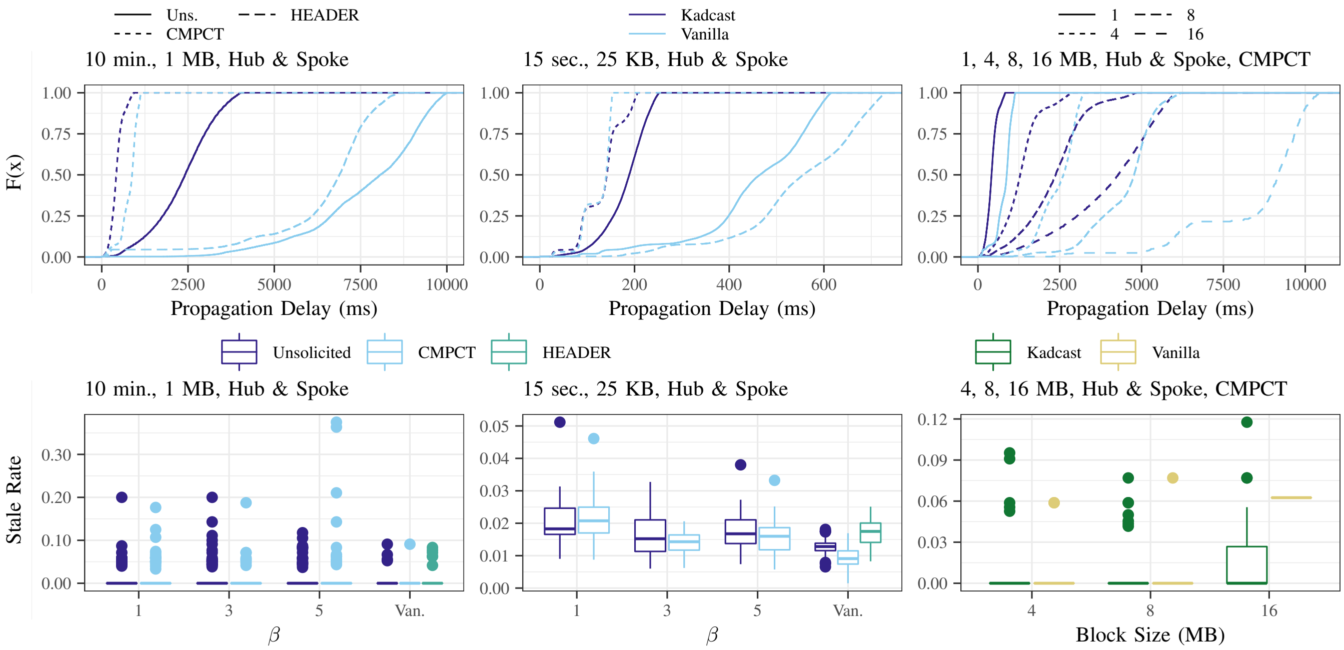

Figure 4: Block propagation delay and resulting stale rates for Bitcoin-like and Ethereum-like parametrizations, as well as for simulation scenarios with higher block size limits.

generally chosen in reference to real-world measurements of the Bitcoin peer-to-peer network (cf. [45]). Furthermore, bns allows us to evaluate a UDP-based Kadcast implementation in comparison to a TCP-based *vanilla* blockchain network protocol stack.

While our evaluation is based on a number of different setups, the results described in the following 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 miners generated blocks and initiated broadcast operations employing one of the networking stacks, i.e., the unstructured *vanilla* baseline or the *Kadcast* protocol. We furthermore implemented and evaluated the impact of different messaging patterns, i.e., *header*, *cmpctblock*, and *unsolicited* block propagation for the vanilla stack, as well as *cmpctblock* and *unsolicited* propagation for Kadcast. In the Kadcast case, if not stated otherwise, the results are based on the default parameters $L = 64$, $k = 100$, $\alpha = 3$, $\beta = 3$, and an FEC overhead factor $f = 0.05$.

In order to analyze the protocol behavior under different network conditions, we furthermore evaluated all protocol variants in the Hub & Spoke, as well as in the geographic topology models provided by the bns framework [45]. The Hub & Spoke model is a typical setup for the assessment of peer-to-peer overlays, and while it captures some network effects, it does not rely on additional assumptions about the underlying topology. As it furthermore assumes the Internet connectivity to not be a bottleneck, it creates an idealized simulation scenario that gives us the capability to assess the networking stacks based on a neutral, common ground. The geographic topology model however captures a heterogeneous and partly resource-restricted environment that enables simulations in more complex network scenarios, which incorporate a high degree of network effects.

B. Protocol Evaluation

In order to show the benefits of the Kadcast protocol in different environments, we created simulation scenarios with parametrizations resembling 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 [47]). 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 4, 8, and 16 MB.

1) Block Propagation Delay: As a first study, we investigated the performance of Kadcast compared to different instantiations of vanilla broadcasting. Figure 4 shows the block propagation delay to reach 90% of all nodes as cumulative distribution functions $F(x)$: as expected, the block distribution time heavily depends on the block size and block intervals, as well as the employed messaging pattern. The Kadcast protocol, however, delivers blocks significantly faster compared to vanilla in all cases. For example, Kadcast exhibits a mean propagation time of 436 ms to deliver compact blocks with a 1 MB block size limit (Bitcoin-like scenario, upper plot), which is close to twice as fast as vanilla with enabled compact blocks. The mean propagation delay for unsolicited Kadcast propagation is 2,349 ms, 63% faster than header-based, and even 70% faster than unsolicited vanilla propagation in Bitcoin-like scenarios.

The improved propagation speed is also reflected by an overall faster network coverage: while it takes vanilla in the Bitcoin case with enabled compact blocks 1,133 ms to reach 90% of the network, Kadcast is able to reach the same number of nodes around 22% faster. Unsolicited propagation via Kadcast reaches 90% coverage even respectively 53% and 60% faster than header-based and unsolicited propagation in the unstructured vanilla networking layer.

Furthermore, as shown in the bottom-left of Figure 4, Kadcast's performance is competitive with vanilla's in the case of smaller intervals and smaller block sizes: in the Ethereum-like scenario, unsolicited Kadcast is able to deliver blocks on average more than 60 % faster than the unsolicited vanilla baseline, and even 66% faster than the header-based propagation. However, the vanilla stack performs slightly better in the compact block case, in which it is around 7% faster than Kadcast. For the larger block sizes of 4 MB, 8 MB, and 16 MB (cf. the bottom-right plot), Kadcast is on average also able to consistently deliver blocks more than 50% faster.

These results highlight on the one hand the messaging pattern has a significant impact on the performance of block propagation. On the other hand, we conclude that Kadcast is able to immensely speed up the block distribution and may be beneficial to a wide variety of blockchain networks.

2) Impact on Consensus Stability: The effect of quicker block propagation is also reflected in the median stale rate, i.e., rate of blocks that are mined, but do not become part of the final blockchain. As increased blockchain forks and wasted mining power weaken consensus security, the stale rate is an indicator for how the networking layer impacts security [5].

The boxplots in the lower part of Figure 4 show the stale rate in dependence of the applied messaging pattern and 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 vanilla 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 of around 1-2%. In this Ethereum-like scenario, the influence of the messaging pattern is again clearly visible, as compact block based propagation beats unsolicited and header-based propagation every time. In accordance with the results of the previous section, the compact block based vanilla variant achieves the lowest median stale rate of 1.3%.

The simulations with compact block relaying of larger blocks indicate that the additional stress on the network layer negatively impacts consensus security: while vanilla and Kadcast can mostly retain a median stale rate of zero, the number of outliers increase in both cases. Though, in the case of the 16 MB block size limit, a clear difference becomes apparent, as Kadcast can retain an average stale rate of 1.6% versus vanilla's 6%.

In summary, the improved block propagation of Kadcast leads to a median stale rate that is comparable and often better than the results of the vanilla stack. This indicates that the consensus security of blockchain systems could benefit from employing the Kadcast protocol. 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 [48].

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 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.[1]

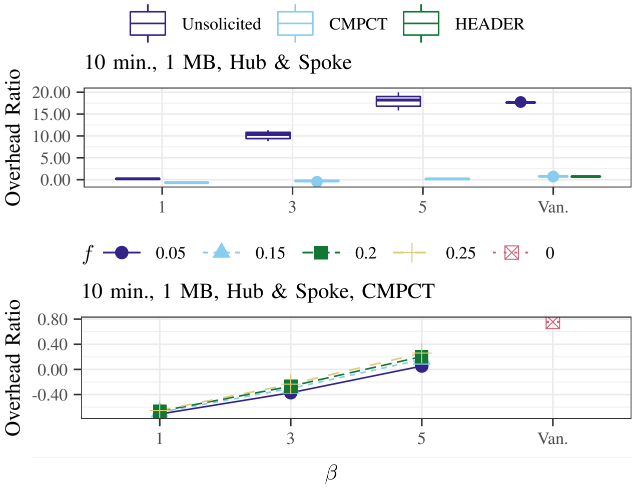

Figure 5: Overhead ratios in dependence of applied messaging pattern and parametrizations for f and β.

Figure 5 shows the resulting overhead ratios in dependence of the applied messaging patterns and different parametrizations of the redundancy parameters β and f. Firstly, we observe that Kadcast's overhead immensely depends on the applied messaging pattern and that it increases linearly with the redundancy factors β and f. As expected, unsolicited block propagation overall results in a significantly higher overhead than the propagation through headers or compact blocks. We moreover note that throughout all parametrizations, Kadcast's overhead remains below its vanilla counterpart: for $\beta = 1$, Kadcast even with unsolicited block relay results in an overhead ratio below the header-based and compact block variants of vanilla, and for $\beta = 5$, it is comparable to unsolicited vanilla propagation. In the bottom half of Figure 5, we can finally observe that the overhead of compact block propagation through Kadcast is consistently below the vanilla counterpart. This shows the adjustability of the Kadcast approach, which allows for a fast block relay with low and controllable overhead.

C. 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 6 shows the network coverage in dependence of ϵ and β: while Kadcast

[1] Note, that since we only consider block propagation for the traffic estimation and due to the existence of stale blocks, the overhead ratio may assume values below 1 or even 0, which however does not impair the validity of this metric.

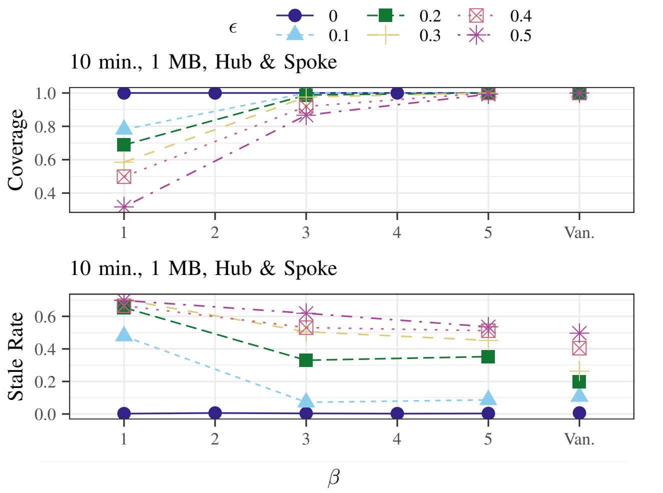

Figure 6: Network coverage and stale rate, when a share ϵ of adversarial nodes is introduced to the network.

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 vanilla'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 6). 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 vanilla networking layer.

D. Protocol Behavior in Heterogeneous and Resource-Restricted Environments

Additionally, in order to evaluate the Kadcast protocol in more heterogeneous 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 7 follow the same tendencies as discussed before: in general, Kadcast provides faster block propagation than vanilla. And again, it does so especially when compact block relaying is enabled.

Nevertheless, the results also show that the UDP-based Kadcast implementation may exhibit degraded performance when facing slower or more congested networking environments. In the case of unsolicited block propagation through Kadcast, we can already see the first signs of congestion: even though the average propagation delays are comparable with the vanilla stack, the propagation delay distribution exhibits a longer tail. The degraded network performance is also reflected by higher stale rates for $\beta > 1$. As the effect is only exhibited with unsolicited block propagation, enabling compact block relaying ascertains low stale rates even in the face of these restricted and heterogeneous environments.

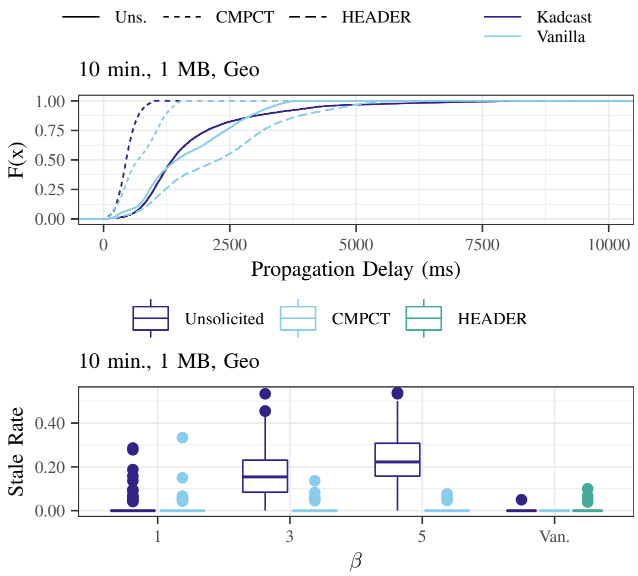

Figure 7: Block propagation delays and stale rates for different parametrizations in the geographic topology model.

However, as the results highlight the robustness of TCP-based vanilla networking stack in comparison to the UDP approach of this Kadcast implementation in heavily congested networks, we consider the implementation of Kadcast on the basis of a suitable congestion control mechanism to be a logical next step. As our simulation results furthermore show that degraded performance may be mitigated by compact block relaying schemes, more blockchain networks should consider adopting such a beneficial messaging pattern. This could help to further alleviate network bottlenecks in heterogeneous environments and hence foster decentralization.

VI. Evaluation: Testbed Deployment

In the following, we introduce our prototypical implementation of the Kadcast protocol and evaluate the properties of Kadcast in a global-scale testbed deployment.

A. Kadcast-NG: A QUIC-based Prototype Implementation

In order to enable testing and evaluation in real-world networks, we developed `kadcast-ng`, a prototypical node implementation of the Kadcast protocol written in Rust [49]. The `kadcast-ng` codebase is open source and publicly available.[2] Following the separation of concerns principle and for the sake of comparability, `kadcast-ng` is designed to run on top of the Bitcoin Core [50] reference implementation of the *vanilla* Bitcoin protocol. To this end, the prototype leaves the enforcement of the consensus protocol to `bitcoind`, with which it interacts through the RPC [51] and ZeroMQ [52] interfaces. This allows `kadcast-ng` to merely act as a relay node that receives and submits transaction and block data

[2] https://git.tu-berlin.de/rohrer/kadcast-ng-public

which it (de-)serializes using the rust-bitcoin [53] library implementation.[3]

As we've seen in the previous section, Kadcast may exhibit degraded performance in congested network settings when it is implemented on top of the unreliable UDP transport protocol. We therefore base the kadcast-ng prototype implementation on the reliable QUIC [12] transport protocol. QUIC fits the design philosophy of Kadcast very well: It features data transmission via multiple concurrent streams, which eliminates head-of-line blocking and therefore reinforces the benefits of Kadcast's parallelized broadcast operation. In contrast to the comparatively slow TCP handshake, QUIC furthermore facilitates connection establishment in one or even zero RTTs. This is an important feature, since it enables kadcast-ng to continue to follow the lightweight soft-state approach in which short-lived connections are only established when they are needed. Moreover, QUIC is an encrypted and authenticated transport protocol which provides many benefits to the security and privacy of the Kadcast protocol, as previously discussed. In this regard, the kadcast-ng prototype utilizes the asynchronous Quinn [54] library implementation of the QUIC protocol and establishes an encrypted overlay network based on a trust-on-first-sight principle.

B. Testbed Setup

In accordance with real-world blockchain networks, we create a testbed scenario that enable the evaluation based on a large and distributed network setting. To this end, we deployed 1,000 instances of type e2-small in different regions of the Google Compute Cloud. In particular, we follow the measured node distribution and deploy the instances close to the regional center of the geographic topology model introduced in [45].[4]

All node instances are provisioned with Ubuntu Linux 20.10, Bitcoin Core in regtest mode, and kadcast-ng. Of all deployed nodes, 15 nodes are chosen as "miners" based on the geographic mining distribution. They are then configured with a certain *mining rate*, which denominates at which exponentially distributed rate they should produce blocks to induce a network-wide average block interval of 10 minutes. According to this parameter, new blocks are generated through the invocation of bitcoind's generatetoaddress remote procedure call, which in the regtest mode allows to instantaneously create blocks, i.e., simulate the mining process without actually searching for a proof-of-work solution. In order to fill the blocks, new transactions are created based on a *transaction rate* parameter, which we assume to be uniformly distributed over all nodes. To this end, on average a total of 2,000 transactions should be created for every block interval through the distributed invocation of the sendtoaddress RPC. This parameter corresponds to the current average number of transactions per block in the Bitcoin network [55].

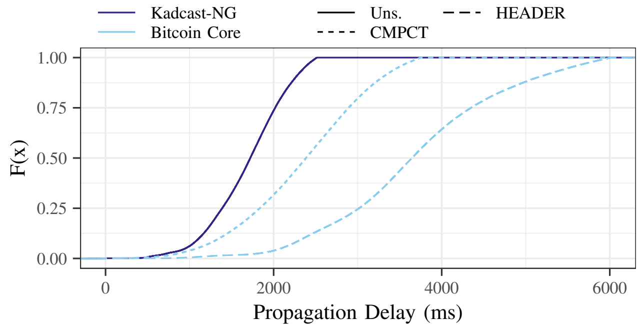

Figure 8: Block propagation delays for the QUIC-based kadcast-ng prototype and Bitcoin Core in a testbed of 1,000 nodes.

To enable this distributed issuance of transactions, we deploy a pre-generated blockchain and funded wallets to each individual node during provisioning.

As before, we configure kadcast-ng with the default parameters $k = 100$ and $\beta = 3$, and employ an initial waiting period to ensure that all nodes have finished their initial bootstrapping before the experiment's measurement period of 24 hours starts.

C. Block Propagation Performance

In order to study the block propagation performance of the kadcast-ng prototype, we evaluated its unsolicited broadcast in comparison to the Bitcoin Core baseline with and without compact block [17] relaying enabled. Over the measurement period of 24 hours an average of 143 blocks were produced and propagated in the described testbed scenarios of 1,000 nodes, each of which recorded the time the blocks arrived from the ZeroMQ interface.

The results are shown in Figure 8: firstly, we observe that the characteristics of the block propagation behavior concur with the simulated scenarios, thereby validating our prior results. Secondly, we observe that also in the testbed deployment Kadcast is able to show its benefits, resulting a block delivery time that is on average 43% faster than the header-based propagation of Bitcoin Core, and even still 27% faster than the case with enabled compact block relay.

The evaluation of the QUIC-based kadcast-ng prototype implementation therefore does not only show the general feasibility of a large-scale deployment of the Kadcast protocol, but also once more underlines its significant impact on the block propagation performance in many scenarios.

VII. 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 private transaction relaying. For example, Venkatakrishnan, Fanti, and Viswanath [43] and Fanti *et al.* [44] propose protocol redesigns that improve anonymity of transaction propagation in the Bitcoin network. Moreover, Naumenko *et al.* [56] recently proposed a more bandwidth-efficient protocol for transaction dissemination in the Bitcoin network. While

[3] At the time of writing, rust-bitcoin does not (yet) support Bitcoin's compact blocks [17]. The kadcast-ng prototype implementation is therefore currently also limited to unsolicited block and transaction propagation.

[4] We deployed nodes in the zones: us-west1-a (NA), southamerica-east1-a (SA), europe-west3-a (EU), asia-south1-a (AS), australia-southeast1-a (OC), and asia-east2-a (CN)

these prior works are concerned with transaction propagation, Kadcast is mainly concerned with block propagation, which is currently not considered to have additional privacy requirements.

The Graphene protocol [57] proposes a more efficient block transmission protocol based on bloom filters, more specifically Invertible Bloom Lookup Tables (IBLTs). This scheme augments the concept of compact blocks to enable higher bandwidth utilization by only retrieving transactions missing from the transaction mempool, which however accepts the possibility of an increased worst-case block propagation delay. Similarly, the recently proposed Velocity protocol [58] 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 network and block propagation model. In particular, they do not address issues that are inherent to unstructured overlay networks. Kadcast however also tackles the overlay structure and the dissemination protocol.

Third-party relay networks, such as the FIBRE network [8] or bloXroute [9] are supposed to improve the block distribution. 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 [5] 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.

Beyond cryptocurrencies, contributions focusing on the broadcast in structured peer-to-peer networks are also relevant for our work. El-Ansary *et al.* [59] realize a perfect broadcasting operation based on the overlay structure of the Chord [60] peer-to-peer distributed hash table. Furthermore, a number of entries are concerned with the broadcasting operation in Kademlia [11]-based overlay networks [21, 61, 62]. Of these contributions, we highlight the work by Czirkos and Hossz [21], 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.

VIII. Conclusion

In this work, we studied Kadcast, a new protocol for fast, efficient, and secure block and transaction propagation in blockchain networks. We analyzed Kadcast's reliability, security, and privacy, and showed its merits through extensive simulative evaluations. Furthermore, we provided a prototype implementation and utilized it to confirm our prior results through deployment in a large-scale cloud-based testbed. Finally, through our work, we hope to further the discussion about alternative transport protocols in the blockchain space.

References

[1] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2008.

[2] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. E. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song, and R. Wattenhofer, "On scaling decentralized blockchains - a position paper," in *BITCOIN '16: Proceedings of the 3rd Workshop on Bitcoin Research*, Christ Church, Barbados, Feb. 2016, pp. 106–125.

[3] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse, "Bitcoin-ng: A scalable blockchain protocol," in *NSDI '16: Proceedings of the 13th USENIX Symposium on Networked Systems Design and Implementation*, Santa Clara, CA, USA, Mar. 2016, pp. 45–59.

[4] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "SPECTRE: a fast and scalable cryptocurrency protocol," *IACR Cryptology ePrint Archive*, vol. 2016, p. 1159, 2016.

[5] A. Gervais, G. Karame, K. Wst, V. Glykantzis, H. Ritzdorf, and S. Capkun, "On the security and performance of proof of work blockchains," in *CCS '16: Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security*, Vienna, Austria, Oct. 2016.

[6] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *P2P '13: Proceedings of the 13th IEEE International Conference on Peer-to-Peer Computing*, Trento, Italy, Sep. 2013, pp. 1–10.

[7] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *FC '15: Proceedings of the 19th International Conference on Financial Cryptography and Data Security*, Isla Verde, Puerto Rico, Jan. 2015, pp. 507–527.

[8] F. I. B. R. E. (FIBRE). "Homepage." (2017), [Online]. Available: http://bitcoinfibre.org.

[9] U. Klarman, S. Basu, A. Kuzmanovic, and E. G. Sirer, *Bloxroute: A scalable trustless blockchain distribution network whitepaper*.

[10] E. Rohrer and F. Tschorsch, "Kadcast: A structured approach to broadcast in blockchain networks," in *AFT '19: Proceedings of the first ACM conference on Advances in Financial Technologies*, Zurich, Switzerland, Oct. 2019.

[11] P. Maymounkov and D. Mazires, "Kademlia: A peer-to-peer information system based on the XOR metric," in *IPTPS '02: Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, USA, 2002, pp. 53–65.

[12] A. Langley *et al.*, "The QUIC transport protocol: Design and internet-scale deployment," in *SIGCOMM '17: Proceedings of the 2017 Conference of the ACM Special Interest Group on Data Communication*, Los Angeles, CA, USA, 2017, pp. 183–196.

[13] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *CRYPTO '17: Proceedings of the 37th Conference on Advances in Cryptology*, Santa Barbara, CA, USA, Aug. 2017, pp. 357–388.

[14] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *OSDI '99: Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation*, New Orleans, LA, USA, Feb. 1999, pp. 173–186.

[15] J. A. Garay, A. Kiayias, and N. Leonardos, "Full analysis of nakamoto consensus in bounded-delay networks," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 277, 2020. [Online]. Available: https://eprint.iacr.org/2020/277.

[16] Y. Chawathe, S. Ratnasamy, L. Breslau, N. Lanham, and S. Shenker, "Making gnutella-like p2p systems scalable," in *SIGCOMM '03: Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Karlsruhe, Germany, Aug. 2003, pp. 407–418.

[17] M. Corallo. "Bip 152: Compact block relay." (Apr. 2016), [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0152.mediawiki.

[18] V. Buterin. "A next-generation smart contract and decentralized application platform." (2014), [Online]. Available: https://ethereum.org/en/whitepaper/.

[19] T. Neudecker and H. Hartenstein, "Network layer aspects of permissionless blockchains," *IEEE Communications Surveys and Tutorials*, vol. 21, no. 1, pp. 838–857, 2019.

[20] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *USENIX Security '15: Proceedings of the 24th USENIX Security Symposium*, Washington, DC, USA, Aug. 2015, pp. 129–144.

[21] Z. Czirkos and G. Hossz, "Solution for the broadcasting in the kademlia peer-to-peer overlay," *Computer Networks*, vol. 57, no. 8, pp. 1853–1862, 2013.

[22] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder, *RaptorQ Forward Error Correction Scheme for Object Delivery*, RFC 6330 (Proposed Standard), RFC, Fremont, CA, USA: RFC Editor, Aug.

2011. DOI: 10.17487/RFC6330. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6330.txt.

[23] G. Urdaneta, G. Pierre, and M. van Steen, "A survey of DHT security techniques," *ACM Comput. Surv.*, vol. 43, no. 2, 8:1–8:49, 2011.

[24] T. Cholez, I. Chrisment, and O. Festor, "Evaluation of sybil attacks protection schemes in KAD," in *AIMS '09: Proceedings of the 3rd International Conference on Autonomous Infrastructure, Management and Security*, Enschede, The Netherlands, 2009, pp. 70–82.

[25] P. Wang, J. Tyra, E. Chan-Tin, T. Malchow, D. F. Kune, N. Hopper, and Y. Kim, "Attacking the kad network," in *4th International ICST Conference on Security and Privacy in Communication Networks, SECURECOMM 2008, Istanbul, Turkey, September 22-25, 2008*, Istanbul, Turkey, 2008, p. 23.

[26] ——, "Attacking the kad network - real world evaluation and high fidelity simulation using DVN," *Security and Communication Networks*, vol. 6, no. 12, pp. 1556–1575, 2013.

[27] M. Steiner, T. En-Najjary, and E. W. Biersack, "Exploiting KAD: possible uses and misuses," *Computer Communication Review*, vol. 37, no. 5, pp. 65–70, 2007.

[28] I. Baumgart and S. Mies, "S/kademlia: A practicable approach towards secure key-based routing," in *ICPADS '07: Proceedings of the 13th International Conference on Parallel and Distributed Systems*, Hsinchu, Taiwan, Dec. 2007, pp. 1–8.

[29] M. Kohnen, M. Leske, and E. P. Rathgeb, "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*, Aachen, Germany, 2009, pp. 104–116.

[30] T. Locher, D. Mysicka, S. Schmid, and R. Wattenhofer, "Poisoning the kad network," in *ICDCN '10: Proceedings of the 11th International Conference on Distributed Computing and Networking*, Kolkata, India, 2010, pp. 195–206.

[31] J. R. Douceur, "The sybil attack," in *IPTPS '02: Proceedings of the 1st International Workshop on Peer-to-Peer Systems*, Cambridge, MA, USA, 2002, pp. 251–260.

[32] Y. Marcus, E. Heilman, and S. Goldberg, "Low-resource eclipse attacks on ethereum's peer-to-peer network.," *IACR Cryptology ePrint Archive*, vol. 2018, p. 236, 2018.

[33] M. Vasek, M. Thornton, and T. Moore, "Empirical analysis of denial-of-service attacks in the bitcoin ecosystem," in *BITCOIN '14: Proceedings of the 1st Workshop on Bitcoin Research*, Barbados, Mar. 2014, pp. 57–71.

[34] P. Koshy, D. Koshy, and P. McDaniel, "An analysis of anonymity in bitcoin using p2p network traffic," in *FC '14: Proceedings of the 18th International Conference on Financial Cryptography and Data Security*, Barbados, Mar. 2014, pp. 469–485.

[35] A. Biryukov, D. Khovratovich, and I. Pustogarov, "Deanonymisation of clients in bitcoin p2p network," in *CCS '14: Proceedings of the 21st ACM Conference on Computer and Communications Security*, Scottsdale, AZ, USA, Nov. 2014, pp. 15–29.

[36] G. C. Fanti and P. Viswanath, "Deanonymization in the bitcoin P2P network," in *NIPS '17: Proceedings of 30th Annual Conference on Neural Information Processing Systems*, Long Beach, CA, USA, Dec. 2017.

[37] S. Feld, M. Schnfeld, and M. Werner, "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*, Hasselt, Belgium, Jun. 2014, pp. 1121–1126.

[38] M. Apostolaki, A. Zohar, and L. Vanbever, "Hijacking bitcoin: Routing attacks on cryptocurrencies," in *SP '17: Proceedings of the 38th IEEE Symposium on Security and Privacy*, San Jose, CA, USA, 2017, pp. 375–392.

[39] N. Borisov, "Computational puzzles as sybil defenses," in *P2P '06: Proceedings of the 6th IEEE International Conference on Peer-to-Peer Computing*, Cambridge, United Kingdom, 2006, pp. 171–176.

[40] V. Vishnumurthy, S. Chandrakumar, and E. G. Sirer, "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*, Berkeley, CA, USA, Jun. 2003.

[41] E. Rescorla and N. Modadugu, *Datagram Transport Layer Security Version 1.2*, RFC 6347 (Proposed Standard), RFC, Updated by RFCs 7507, 7905, Fremont, CA, USA: RFC Editor, Jan. 2012. DOI: 10.17487/RFC6347. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6347.txt.

[42] G. A. Spedicato, "Discrete Time Markov Chains with R," *The R Journal*, vol. 9, no. 2, pp. 84–104, 2017.

[43] S. B. Venkatakrishnan, G. C. Fanti, and P. Viswanath, "Dandelion: Redesigning the bitcoin network for anonymity," *Proceedings of the ACM on Measurement and Analysis of Computing Systems (POMACS)*, 2017.

[44] G. C. Fanti, S. B. Venkatakrishnan, S. Bakshi, B. Denby, S. Bhargava, A. Miller, and P. Viswanath, "Dandelion++: Lightweight cryptocurrency networking with formal anonymity guarantees," *POMACS*, vol. 2, no. 2, 29:1–29:35, 2018.

[45] E. Rohrer and F. Tschorsch, "Blockchain layer zero: Characterizing the bitcoin network through measurements, models, and simulations," in *LCN '21: Proceedings of the 46th IEEE International Conference on Local Computer Networks*, Online, 2021.

[46] nsnam. "Ns-3: A discrete-event network simulator for internet systems." (2021), [Online]. Available: https://www.nsnam.org.

[47] Etherscan.io. "Ethereum block size history." (2019), [Online]. Available: https://etherscan.io/chart/blocksize.

[48] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *FC '14: Proceedings of the 18th International Conference on Financial Cryptography and Data Security*, Barbados, Mar. 2014, pp. 436–454.

[49] R. Team. "Rust. a language empowering everyone to build reliable and efficient software." (2021), [Online]. Available: https://www.rust-lang.org/ (visited on 07/01/2021).

[50] Bitcoin Core. "Homepage." (2021), [Online]. Available: https://bitcoincore.org (visited on 07/07/2021).

[51] Bitcoin Project. "Bitcoin rpc api." (2021), [Online]. Available: https://developer.bitcoin.org/reference/rpc/index.html (visited on 07/01/2021).

[52] The ZeroMQ Authors. "Zeromq. an open-source universal messaging library." (2021), [Online]. Available: https://zeromq.org/ (visited on 07/01/2021).

[53] Rust Bitcoin Developers. "Rust bitcoin library." (2021), [Online]. Available: https://github.com/rust-bitcoin/rust-bitcoin (visited on 07/01/2021).

[54] J.-C. B. Dirkjan Ochtman Benjamin Saunders. "Quinn. pure-rust quic protocol implementation." (2021), [Online]. Available: https://github.com/quinn-rs/quinn (visited on 07/01/2021).

[55] Blockchain.com. "Bitcoin. average transactions per block." (2021), [Online]. Available: https://www.blockchain.com/charts/n-transactions-per-block (visited on 07/01/2021).

[56] G. Naumenko, G. Maxwell, P. Wuille, A. Fedorova, and I. Beschastnikh, "Erlay: Efficient transaction relay for bitcoin," in *CCS '19: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, London, UK, Nov. 2019, pp. 817–831.

[57] A. P. Ozisik, G. Andresen, B. N. Levine, D. Tapp, G. Bissias, and S. Katkuri, "Graphene: Efficient interactive set reconciliation applied to blockchain propagation," in *SIGCOMM '19: Proceedings of the 2019 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Beijing, China, 2019, pp. 303–317.

[58] N. Chawla, H. W. Behrens, D. Tapp, D. Boscovic, and K. S. Candan, "Velocity: Scalability improvements in block propagation through rateless erasure coding," in *ICBC '19: Proceedings of the 1st International Conference on Blockchain and Cryptocurrency*, Seoul, South Korea, May 2019.

[59] S. El-Ansary, L. O. Alima, P. Brand, and S. Haridi, "Efficient broadcast in structured P2P networks," in *IPTPS '03: Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*, Berkeley, CA, USA, 2003, pp. 304–314.

[60] I. Stoica, R. T. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan, "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*, San Diego, CA, USA, 2001, pp. 149–160.

[61] A. D. Peris, J. M. Hernndez, and E. Huedo, "Evaluation of the broadcast operation in kademlia," in *HPCC '12: 14th IEEE International Conference on High Performance Computing and Communication*, Liverpool, United Kingdom, 2012, pp. 756–763.

[62] ——, "Evaluation of alternatives for the broadcast operation in kademlia under churn," *Peer-to-Peer Networking and Applications*, vol. 9, no. 2, pp. 313–327, 2016.