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 global-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.

Problem Statement: Broadcast is accordingly the most commonly used network operation in blockchain networks. 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 into 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.

This 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. In fact, many of the improvement proposals [10, 11, 12, 13] are compatible with our approach presented here.

Kadcast and Kadcast-NG: Kadcast [14] is a new structured broadcast protocol for blockchain networks based on Kademlia [15]. Kadcast utilizes the structured overlay architecture of Kademlia, which can be thought as a binary tree, to delegate broadcast responsibilities for subtrees with decreasing height. This procedure eventually leads to forwarding data along a spanning tree and under the assumption of non-faulty nodes yields an highly efficient broadcast. In order to deal with faulty and malicious nodes, however, we introduce mitigation strategies and reliability mechanisms, most notably, a redundant broadcast strategy. This allows Kadcast to maintain its broadcast efficiency and offers tunable parameters to adjust trade-offs with respect to propagation delays, overhead, reliability, security, and privacy.

In this work, we complement and extend the study of the Kadcast protocol, address some weaknesses, and explore the feasibility of a real-world adoption. We propose a number of protocol extensions 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 when confronted with random and adversarial node failures. Moreover, we provide an analysis of transaction privacy.

In addition, we developed Kadcast-NG, a QUIC-based [16] prototype implementation of the Kadcast protocol. Kadcast-NG resolves the weaknesses our initial UDP-based approach, most notably reliability and flow control, without sacrificing fast and lightweight message exchanges (cf. 0-RTT session resumption). We use our Kadcast-NG prototype 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 deploy-
ments. 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.

**Contributions:** Our key contributions can be summarized as follows: (i) We present an efficient broadcast protocol for blockchain networks, which utilizes Kademlia’s structured architecture. (ii) We introduce parallelization to improve the reliability and resilience of Kadcast 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 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.

While this paper is based on our introduction of Kadcast [14], it complements and extends the contributions significantly. Specifically, we discuss security threats and mitigation strategies, propose ID space randomization, conduct a privacy analysis, and extend the simulation study by implementing and evaluating compact blocks [10] and other messaging schemes. Most notably, though, we developed Kadcast-NG including a pluggable prototype, set up a global testbed, and performed a real-world evaluation.

**Structure:** 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 by some kind of Byzantine fault tolerant consensus mechanism, e.g., Nakamoto-style Proof-of-Work [1], Proof-of-Stake [17], or PBFT [18]. After their creation, blocks are broadcast to all nodes in the network, which verify them and apply the updates to their local ledger state. While blockchain protocols have been proven to be consistent in the partially synchronous network model, they are based on the assumption that blocks are delivered in a timely fashion, i.e., faster than a propagation delay limit [19]. 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 network [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 \( O(N \cdot R) \), which has been shown to have severe consequences on network scalability in the past [20].

Therefore, some blockchain networks reduce traffic by introducing alternative messaging patterns, such as request-response or compact block [10] 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 networking layer of Bitcoin [1] and Ethereum [21], we refer the reader to literature [22].

III. THE KADCAST PROTOCOL

Kadcast is based on Kademlia [15], a distributed hash table (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 $O(N)$. 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 ID$_0$ holds buckets $B_i$, $i = 0 \ldots L - 1$, whereby bucket $B_i$ holds the node information of $k$ nodes with ID$_j$ so that $2^i \leq d$(ID$_0$, 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 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, 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 [23].

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. 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 $\alpha$ closest nodes with respect to the XOR-metric in its own buckets. (2) It queries these $\alpha$ nodes for the ID by sending FIND_NODE messages. (3) The queried nodes respond with a set of $\alpha$ 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 $k$, $\alpha$ 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 [16] enable a more scalable

Figure 1: Example broadcast initiated by node 1111 ($\beta = 1$). Colors indicate node distances in the spanning tree, relative to the initiator.
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
$C \leftarrow C \cup \{c\}$
end for

for $i = 0 \rightarrow h - 1$ do
  $R \leftarrow \text{randomly select} (\beta, B_i)$
  for all $r \in R$ do
    send_chunk($r, c, i$)
  end for
end for

and dynamic approach with short-lived, low-cost transmissions, which complement Kadcast’s design. While this allows for a lightweight protocol design with reduced statekeeping 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 [24]. 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 $\text{CHUNK}$ messages, which carry the data and her routing information. In particular, it assigns a new height $h$, which effectively determines the receiver’s broadcast responsibility. When a node receives a $\text{CHUNK}$ message, 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 $\text{CHUNK}$ with a smaller value for $h$ accordingly.

Consequently, 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 $\text{ID}_0 = 1111$ initiates a broadcast in the network, and sends four $\text{CHUNK}$ 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 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 $B_0$, 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 $\beta$ 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 (FEC) scheme based on RaptorQ [25] 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 $\text{CHUNK}$ 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 a 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. In order to allow nodes to recover from the rare case that message delivery fails, and to enable the initial bootstrapping of the blockchain, the Kadcast protocol also incorporates a simple $\text{REQUEST}$ message that allows nodes to query others for specific blocks or transactions, and is answered by the corresponding $\text{CHUNK}$ messages.

2) Analysis of Parallelized Broadcast:

Kadcast implements broadcast redundancy by parallelizing the algorithm. To this end, we introduce the system parameter $\beta$, 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 [24], 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 $\epsilon$ 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.

In order to express the coverage of a redundant broadcast, we need to extend this model. We therefore model the parallel execution of our algorithm as the probability that at least one of the redundantly sent chunks is successfully delivered, i.e., $p_2 = 1 - \epsilon^\beta$. 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 $\beta$ 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.,

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

Figure 2 clearly shows the improved transmission reliability offered by introducing FEC 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 $\beta$ 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 Kademia’s security properties, its behavior when attacked by a range of adversaries, and designs improving on its security [26, 27, 28, 29, 30, 31, 32, 33]. 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 [34] 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 [30, 32]. 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 [23, 35]. In the past, the feasibility of Eclipse attacks on the Kademlia protocol have been studied in literature [30, 32, 33]. 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 [36] 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 in the network 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 [37, 38, 39]. Furthermore, the notion of AS-level adversaries, i.e., adversaries monitoring the data flows of blockchain networks within an entire autonomous system (AS), has been discussed in literature [40, 41]. 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 [40, 41], 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 [31, 42, 43]. 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., $H(\text{ID}\|\text{ID}_\text{NONCE}) < t_{\text{diff}}$, where $t_{\text{diff}}$ 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 $t_{\text{diff}}$. 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 that by safeguarding node identifiers and enforcing rigorous bucket policies, Kadcast follows best practices for Sybil and Eclipse protection [35].

In order to avoid Denial-of-Service (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 [44], 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 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 \textit{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

\[b_{s,i} = \frac{2^i}{2L \cdot N},\]

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 \(\beta\) 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 \(\beta\), as we discussed and analyzed in Section III-C.

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 \(s_1\), 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 \textit{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 \(p_{a,i}\). The chosen node may be either honest or malicious. If it is honest (again, probability \(p_h = 1 - p_e = 1 - (M/N)^\beta\)), 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 \textit{fail} state \(s_f\) with probability \(p_m = 1 - p_h\). Once in the \textit{success} or \textit{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 \texttt{markovchain} [45], and simulated the success probability of block propagation for different shares of malicious nodes \(\epsilon\) and redundancy factors \(\beta\). 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 adversarial.

### 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 order 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 L and is decremented with every subsequent forwarding step, an adversary receiving a broadcast message is able to infer her distance from the originator of the broadcast in the propagation graph. This is particular worrisome for the first hop, as an adversary receiving a message with height L − 1 would be able to infer that the sender of this message is also its origin. While this may be no issue for block propagation (blocks can be assumed to be not privacy-critical) it may be detrimental to user privacy for transaction propagation.

Borrowing from the idea of the Dandelion protocol [46, 47], 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 [46]. 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.

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 d1 = β(L − 1) initial bucket delegates. This probability of detection is the probability that at least on malicious node is initially chosen, i.e.,

\[
p_m = m_1 = 1 - \left( \frac{N - M}{N} \right)^{d_1}.
\]

By utilizing the privacy phase, the number of delegates directly contacted by the originator is reduced from d1 = β(L − 1) to d1 = β, 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 share of malicious nodes and the parallelization factor β (L = 160).

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>M</td>
</tr>
<tr>
<td>11,000</td>
<td>1,000</td>
</tr>
<tr>
<td>12,000</td>
<td>2,000</td>
</tr>
<tr>
<td>13,000</td>
<td>3,000</td>
</tr>
<tr>
<td>13,000</td>
<td>3,000</td>
</tr>
<tr>
<td>15,000</td>
<td>5,000</td>
</tr>
</tbody>
</table>

**Table I: Markov Simulation Results**
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 [14, 48] blockchain simulation framework, which is based on the time-discrete network simulator ns-3 [49]. The parameters that fuel our simulation are therefore generally chosen in reference to real-world measurements of the Bitcoin peer-to-peer network [48]. 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 [48]. 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 [50]). 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 \( \beta \): 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 [51].

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.$^1$

$^1$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 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.

Figure 5: Overhead ratios in dependence of applied messaging pattern and parametrizations for $f$ and $\beta$. The results show that Kadcast’s overhead immensely depends on the applied messaging pattern and that it increases linearly with the redundancy factors $\beta$ 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 empirically evaluated the performance of Kadcast in the face of an adversary obstructing block delivery. For this, we set up simulation scenarios in which a fraction $\epsilon$ 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 $\epsilon$ and $\beta$; 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 $\beta$ is also clearly visible, ensuring 99% coverage for $\beta = 3$ as long as $\epsilon \leq 0.3$. For $\beta = 5$, Kadcast can even handle higher shares of malicious nodes ($\epsilon \geq 0.4$).

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 under Degraded Network Conditions

In order to evaluate the Kadcast protocol in more heterogeneous networking environments, we reproduced the previous scenarios in a resource restricted setting. We utilize the geographic topology model (Geo) from [48], where node deployments resemble a geographic distribution with heterogeneous network conditions (delay, bandwidth) on the underlay as measured in the Bitcoin network. Due to lower bandwidths, larger latencies, packet losses, and more complex structure of this model, more network effects come into play. We accordingly expect a degraded performance with a more pronounced long tail and more outliers.

The results shown in Figure 7 confirm our expectations. But despite the heterogeneous network conditions, the results follow the same tendencies as discussed before: in general, Kadcast provides faster block propagation than vanilla. In particular when compact block relaying is enabled, the performance improvements are clearly visible. Nevertheless, the results also show that the UDP-based Kadcast implementation may exhibit degraded performance when facing slower or more congested networking environments. In particular for the unsolicited block propagation, we can see the first signs of network congestion in Kadcast: even though the average propagation delays are comparable with vanilla, 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.

In a first conclusion, we can confirm that Kadcast improves the efficiency of blockchain networks, particularly in combination with compact blocks. We, however, can also see a weakness of the UDP-based approach in Kadcast for restricted network environments as it can lead to network congestion and therefore to degraded performance. This is not surprising as our data transport does not include any congestion control mechanisms. We accordingly consider implementation of congestion control in Kadcast as a logical, necessary next step, which we will address in the next section with Kadcast-NG. On a more general note, our simulation results show that degraded performance can be mitigated by compact block relaying schemes. We therefore consider adopting such a messaging pattern to be beneficial for blockchain networks in general. This could help to further alleviate network bottlenecks in heterogeneous environments and hence foster decentralization.
VI. KADCAST-NG AND TESTBED DEPLOYMENT

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

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

As we have seen in the previous section, Kadcast may exhibit degraded performance in restricted network settings when it is implemented on top of UDP. We therefore consider it necessary to implement the Kadcast protocol together with an appropriate transport protocol including congestion control. While we could implement congestion control on top of UDP ourselves, we opt for the QUIC [16] transport protocol as a base for Kadcast-NG. 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 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 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 [52]. The kadcast-ng codebase is open source and publicly available. Following the separation of concerns principle and for the sake of comparability, kadcast-ng is designed to run on top of the Bitcoin Core daemon bitcoind [53], which is the reference implementation of the vanilla Bitcoin protocol. To this end, kadcast-ng interacts with bitcoind through its remote procedure call (RPC) [54] and ZeroMQ [55] interfaces only, while leaving the enforcement of the consensus protocol to bitcoind. This allows kadcast-ng to merely act as a relay node that receives and submits transaction and block data which it (de-)serializes using the rust-bitcoin [56] library implementation. At the time of writing, rust-bitcoin does not (yet) support Bitcoin’s compact blocks [10]. The kadcast-ng prototype implementation is therefore also limited to unsolicited block and transaction propagation. As transport protocol and to establish overlay connections, the kadcast-ng prototype utilizes the asynchronous Quinn [57] 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 set up a testbed that enables the evaluation in a global and real-world network setting. To this end, we deployed 1,000 instances of type e2-small in different regions of the Google Compute Cloud. Google Cloud’s e2-small instances feature a shared CPU (vCPUs), 2 GB of memory, and up to 1 Gbps egress bandwidth. We build upon the results from [48], who measured the Bitcoin network and derived a network model. According to the measured node distribution, we deploy node instances close to the regional center of the geographic topology model. Specifically, we deployed nodes in the Google Cloud zones: us-west1-a (NA), southamerica-east1-a (SA), europewest3-a (EU), asia-south1-a (AS), australia-southeast1-a (OC), and asia-east2-a (CN). 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 [48]. 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 [58]. 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 [10] 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 feas-
ibility of a global-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 improve-
ments for blockchain networks.

Orthogonal to the core of the Kadcast broadcast protocol, a 
number of contributions deal with private transaction relaying. 
For example, Venkatakrishnan, Fanti, and Viswanath [46] and 
Fanti et al. [47] propose protocol redesigns that improve 
anonymity of transaction propagation in the Bitcoin network. 
Similarly, Naumenko et al. [11] recently proposed a more 
bandwidth-efficient protocol for transaction dissemination in 
the Bitcoin network. While 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. Yet, for improved 
privacy, we in fact build upon the main idea proposed by 
Venkatakrishnan, Fanti, and Viswanath and extended Kadcast 
in this paper. We therefore also provide an example for how 
these protocols can be integrated in Kadcast.

The Graphene protocol [12] proposes a more efficient block 
transmission protocol based on Bloom filters, specifically In-
vertible 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 
Velocity protocol [13] 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.

Decker and Wattenhofer [6] deploy a highly connected high 
capacity node, which effectively reduces the network diameter. 
While the approach has a positive effect on the propagation 
delay, it also leads to a more centralized network design. 
Kadcast is able to improve the propagation delay without 
sacrificing decentralization.

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Method</th>
<th>Structured Overlay</th>
<th>Unstructured Overlay</th>
<th>Block Propagation</th>
<th>Tx Propagation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kadcast</td>
<td>Broadcast delega-</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>BIP 152 [10]</td>
<td>Compact blocks</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Graphene [12]</td>
<td>Set reconciliation</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>Velocity [13]</td>
<td>Fountain codes</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
</tbody>
</table>

O “can be used for” ☐ “designed for”

Third-party relay networks, such as bloXroute [9] or 
FIBRE [8] aim to improve the block distribution. While the 
emergence of these proposals clearly show the urgency of 
the problem, we consider them orthogonal to the goal of 
improving the peer-to-peer networks of blockchain systems 
themselves. Additionally, results 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 relay networks involve some central peer management 
and coordination, they do not meet the blockchain design goals 
of decentralization.

In general, the Kadcast protocol dictates the overlay struc-
ture and the distributed algorithm for information dissemina-
tion only. Messaging formats and handshakes between peers 
still can make use of improvements such as compact blocks as 
in Bitcoin’s improvement proposal BIP 152 [10] or alike. 
Principally, Kadcast is compatible with related work such as 
the previously mentioned Erlay [11], Graphene [12], and 
Velocity [13]. Similarly, Kadcast can be combined with an ad-
ditional relay network—if desired. We summarize, categorize, 
and compare related protocols in Table II.

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 broadcast opera-
tion based on the overlay structure Chord [60]. Furthermore, 
a number of entries are concerned with the broadcasting 
operation in Kademlia [15]-based overlay networks [24, 61, 
62]. Of these contributions, we highlight the work by Czirkos 
and Hossz [24], 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 global-scale cloud-based testbed. 
Finally, through our work, we hope to further the discussion 
about alternative transport protocols in the blockchain space.


