FLOKI: Efficient Network-Wide Flooding

Paul Seehofer, Roland Bless, Martina Zitterbart
Institute of Telematics, *Karlsruhe Institute of Technology (KIT)* – firstname.lastname@kit.edu

***Abstract*—We introduce FLOKI, an efficient and scalable approach for network-wide flooding in the network's control plane. FLOKI floods packets in a structured way along replication trees installed in the forwarding plane based on the routing architecture KIRA. Compared to traditional hop-by-hop flooding, this significantly reduces control plane processing overhead. In dense topologies, it also substantially reduces network load and flooding time.**

***Index Terms*—network-wide flooding, autonomic networks**

I. INTRODUCTION

Network-wide flooding is a fundamental primitive in the control plane of network infrastructures, used for disseminating state information or events like topology changes. Traditional hop-by-hop flooding (HHF), as used in link-state routing (LSR) protocols (OSPF [1], ISIS [2]), distributed traffic engineering [3], and in autonomic infrastructures [4], floods an incoming packet to all neighbors. This generates many duplicates especially in dense network infrastructures, resulting in high load in the control plane and long flooding times until the packet is distributed to all nodes [5], [6]. While some recent approaches [6], [7] propose selectively flooding to neighbors, they may introduce significant control plane overhead in dynamic scenarios (see Appendix C).

FLOKI (FLOoding over KIra), is a novel approach to scalable and efficient network-wide flooding for the control plane of networks. Compared to HHF, FLOKI significantly reduces control plane processing overhead, flooding time as well as network load. This is achieved by building FLOKI *replication trees* that comprise all nodes in the network. FLOKI does not flood the packets to all neighbors, but replicates them only to *specific nodes* defined by the replication tree. Using replication trees eliminates the creation of duplicate packets and thus also the need for de-duplication in the control plane. This allows FLOKI to realize its replication trees as pre-calculated *replication rules* in the *forwarding plane*.

FLOKI builds on the routing architecture KIRA [8] that provides scalable and resilient control plane connectivity in autonomic infrastructures in a zero-touch manner. No manual configuration of IP addresses and network areas as in OSPF is needed. FLOKI uses KIRA's existing routing overlay structure and forwarding mechanism to establish its replication tree and replication rules. As a result, FLOKI scales to large networks, as KIRA does.

FLOKI can support any distributed control plane application or protocol that needs to flood packets to all nodes in the network, e.g., for state synchronization.

The authors acknowledge the financial support by the German Federal Ministry of Education and Research (BMBF) in the projects "Open6GHub" (grant number 16KISK010) and "Open6GHub+" (grant number 16KIS2405).

Our main contributions are:

- The novel FLOKI flooding mechanism that operates in the forwarding plane and thus improves flooding performance.
- An efficient mechanism to construct replication trees based on KIRA.
- Our evaluations show that FLOKI reduces control plane processing overhead significantly and substantially reduces network load and flooding time in dense topologies.

II. KIRA

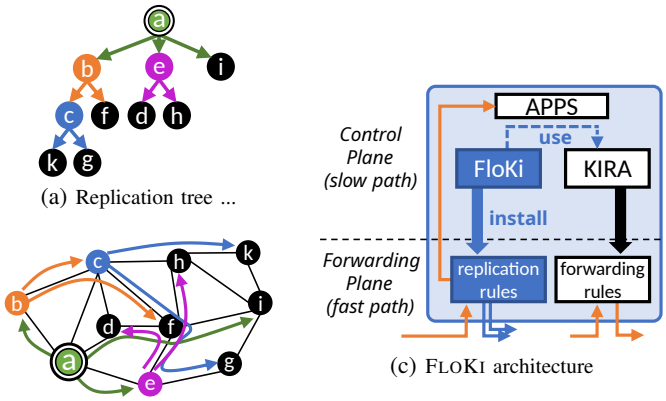
KIRA [8] is a routing architecture that provides scalable, resilient and zero-touch control plane (CP) connectivity. It is designed as the foundation for a resilient CP architecture. We briefly describe core characteristics relevant for FLOKI's design. KIRA builds a tree-like, ID-based overlay structure akin to the peer-to-peer network Kademlia [9]. Nodes randomly generate a NodeID from a large ID-space which puts it somewhere in that structure. Each node maintains a small routing table with $\mathcal{O}(\log n)$ entries. It consists of a series of k-buckets, which hold up to k contacts (other nodes) each. Contacts are selected based on distance in the overlay structure using the XOR-metric between their NodeIDs. Contacts in the first bucket have a large distance, while the last bucket contains contacts from the node's *ID-wise proximity*. The routing table always contains all nodes in a node's ID-wise proximity as contacts.

For efficient forwarding, KIRA establishes *label switched paths (LSPs)* to all contacts in the nodes' forwarding planes.

III. FLOKI IN A NUTSHELL

FLOKI builds replication trees along which it replicates packets in order to flood them to all nodes in the network. Multiple replication trees are used, one for each node as root, which is used when that node originates a packet. Each node selects *one* set of child nodes that it uses for replication when receiving a packet. Depending on where in a replication tree a node is located, it will replicate the packet to *all* (when the node is the originator), a *subset* (replication node), or *none* (leaf node) of its child nodes.

For the selection of the child nodes (see section IV), the existing overlay structure of KIRA is used. A node *delegates* the replication in a sub-tree of the overlay structure to each selected child node. In this way, the replication trees of child nodes are reused partially, and it can be guaranteed that the resulting replication topology is a tree that contains all nodes *exactly once*, either as a replication node or as a leaf node. Fig. 1a shows an example of a replication tree with node a as root, i.e., flooding of the packet originates at a. Fig. 1b



(a) Replication tree ...

(b) ... in link-layer topology

(c) FLOKI architecture

Fig. 1. FLOKI replication tree (a, b) and internal flooding procedure (c)

shows the embedding of this replication tree in the link layer topology.

FLOKI uses the LSPs established by KIRA to forward packets between neighbors in the replication tree, e.g., between nodes b and f. Intermediate nodes on the LSP simply forward the packet using label switching but do not replicate it, e.g., node c on the LSP from b to f. Different LSPs in a replication tree may overlap, i.e., use the same links. In this case, multiple replicas of a packet pass through that same link, e.g., link (c, f) is used by two LSPs. Moreover, not all links are used to flood to all nodes, e.g., link (h, i) or (a, c).

Each of the replication or leaf nodes is only targeted by one LSP of the replication tree, and thus receives only exactly one replica. Consequently, each node needs to process a packet only once in the control plane and not multiple times, as needed in traditional HHF (cf. section III-B) greatly reducing processing overhead. Node c, for example, receives the packet to be replicated from b as it is one of b's child nodes in the replication tree. The other replica from b to f passes through c's forwarding plane according to forwarding rules from KIRA without being processed or replicated by c's control plane.

A. Replication Rules

When constructing the replication trees, FLOKI installs corresponding replication rules in the forwarding plane (cf. Fig. 1c). In this way, packet replication happens in the fast path and does not involve any per-packet processing overhead in the control plane. The replication rules are pre-calculated after KIRA established its routing table and FLOKI selected child nodes. Thus, replication rules are readily available if a packet needs to be flooded through the network.

Replication rules are applied when a packet reaches the destination of KIRA's LSP it follows. For example, node b is the destination of the LSP from node a to b. Consequently, the packet is (1) replicated to the child nodes of b (c and f) and (2) delivered to the corresponding CP application entities (e.g., link state routing protocol) locally in node b.

If the KIRA routing table of a node changes (e.g., in response to a topology change) the affected replication rules must be re-calculated, potentially involving the selection of alternative child nodes from the updated routing table. Due to

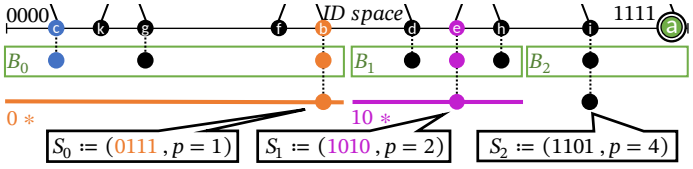


Fig. 2. Child nodes selected from node a's routing table (4-bit NodeIDs)

KIRA's small routing tables a topology change will only affect the routing tables of a small portion of all nodes.

B. Comparison with Hop-by-Hop Flooding

Traditional hop-by-hop flooding (1) utilizes every link in the network and (2) delivers duplicate packets to nodes. Because of this, each packet needs to be processed in the nodes' control plane to perform de-duplication. Only after this CP processing the packet can be flooded to all neighbors.

FLOKI does not require such a detour to the control plane because no de-duplication is required and the replication rules are thus placed directly in the forwarding plane.

IV. ESTABLISHING THE REPLICATION TREES

To understand how FLOKI's replication trees are established, we first give a short description of the replication procedure. It is based on an existing *overlay broadcast algorithm* [10], [11] that is used for information distribution in Kademlia-based peer-to-peer networks. FLOKI's procedure differs in its use, network-wide flooding in link-layer topologies, and improves it by implementing the replication in the nodes' forwarding planes.

At its core, the procedure works by each node being responsible for replicating the packet in a sub-tree of the ID-based overlay. Each node fulfills this *replication responsibility* by (1) replicating the packet to all nodes in its ID-wise proximity and (2) dividing the remaining sub-tree into smaller sub-trees, *delegating* the replication responsibility to chosen *child nodes* within the respective sub-tree. The nodes in the ID-wise proximity are the *leaf nodes* and will not replicate further. The other child nodes will apply the same procedure recursively in their delegated sub-tree.

A. Child Node Selection

Each node selects child nodes from its local KIRA routing table (cf. Appendix A). It selects one contact from each k-bucket as each of them corresponds to a sub-tree of the overlay (cf. Algorithm 1 in the appendix). For the last bucket, which contains contacts in the ID-wise proximity, it selects all contacts. These are the leaf nodes. For each child node the node calculates the length of the ID-prefix p of the corresponding sub-tree. p is encoded into the packet header of each replica. The result is a set of sub-tree delegations $\{S_i := (c_i, p_i)\}$, where c_i is the selected child node's NodeID and p_i is the prefix length of the corresponding sub-tree. For the leaf nodes, we set p to the length of the NodeID as they do not replicate further. Fig. 2 shows the child nodes selected by node a from its routing table buckets (B_0, B_1, B_2). The figure also shows the prefix and the range (in the ID space) of the

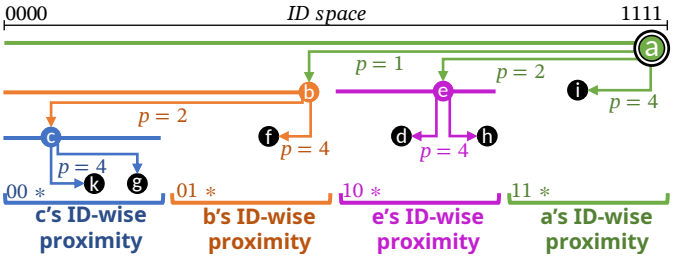


Fig. 3. Replication tree with responsibilities originating from node a.

sub-tree delegated to each child node c_i colored accordingly. For example, when node a replicates to its first child node (c_0) it delegates the replication responsibility in the sub-tree $0*$ to that node. The sub-trees of all selected child nodes combined comprise the whole overlay structure that covers all nodes.

B. Using Child Nodes during the Flooding Procedure

When receiving a packet the value p determines the sub-tree the receiving node should replicate in. It will replicate only to all child nodes within that sub-tree.

Fig. 3 shows an example of a packet originated at node a. Since it is the originator (indicated by a $p = 0$ in the packet header) it replicates to all child nodes. Accordingly, it sends three replicas and encodes the respective p_i in the packet header. The $p = 1$ value in the replica to b tells it to only replicate to child nodes in the sub-tree with prefix $0*$. Selecting child nodes along the routing table structure ensures that each node can always recursively sub-divide the sub-tree they are responsible for into smaller parts during the flooding procedure. In the end four replicating nodes (a, b, c, e) will have replicated in their ID-wise proximity which together comprise all possible nodes in the 4-bit ID-space.

C. Efficient Replication in the Fast Path

The replication rules require the following primitives in the forwarding plane: (1) matching on p encoded in the IPv6 destination address, (2) replicating the packet, and for each replica (3) modifying the IPv6 destination address (to update p) and (4) pushing the label for the LSP to the child node. These primitives are available in devices with programmable data planes (P4, eBPF), in some ISP-grade router architectures (used for ingress replication in multicast over VPN [12]), and in recent versions of SRv6 standards [13].

V. Evaluation Methodology

For the evaluation of FLOKI we use an OMNeT++-based KIRA implementation. We compare FLOKI to the following generic HHF approach we implemented in OMNeT++.

A. Generic HHF Approach

A node that receives a flooding packet checks whether it has seen this packet already, by using a unique identifier consisting of an ID and a sequence number. If it is a duplicate, the packet is dropped. If not, the node stores the identifier and floods the packet to all its neighbors besides the one from which the packet was received. The nodes will also send hop-by-hop acknowledgments (ACKs) back to the sender.

B. Control Plane and Forwarding Plane

We model the nodes' control planes using a FIFO processing queue. CP processing times are uniformly drawn from $[100, 800]\,\mu s$ [14]. The CP has to process control messages of KIRA and flooded packets by HHF.

Packets that are flooded by FLOKI do not enter the CP. Instead, they are delayed by a static *forwarding plane processing delay* of 3.8 µs [15], in addition to queueing and transmission delay depending on the current utilization of the link and the packet size, respectively. We use 10 Gbit/s links with a link delay of 1 µs ($\approx 428\,m$ in fiber).

C. Warm-up and Flooding Packets

We start each simulation run with a 15 s *warm-up period* to let KIRA converge. In the remaining 45 s, we periodically trigger randomly selected nodes to originate a flooding packet. We choose five nodes every five seconds to create some contention of flooding packets resulting in a total of 45 flooding packets being originated from randomly selected locations in the topologies. Although the number of flooding packets is rather small, we already observe control plane queuing delay caused by duplicates in case of HHF.

D. Evaluated Metrics

- *Received Replicas*: The number of times a node receives a packet in the CP. The optimal value is one replica per node, i.e., no duplicates. With HHF, the node must process it before the flooding procedure continues.
- *Link Stress*: The number of replicas of a flooding packet forwarded over each link in the network. HHF's ACKs are not counted in this metric.
- *Flooding Time*: The time between originating the flooding packet and when it was received by all nodes for the first time.

VI. Scalability

In order to evaluate FLOKI's scalability, we run experiments with different network size. We use *power-law clustered* (PLC) [16] and data-center *fat tree* [17] topologies with 100 to 10k nodes.

We expect FLOKI to outperform HHF with respect to CP processing as it forwards replicas solely in the forwarding plane. Fig. 4 (left) shows the number of replicas each node processes in its control plane during the flooding procedure. With FLOKI this value always stays at the optimal value of exactly one for all topologies and sizes, i.e., each node processes exactly one replica of the flooding packet. Meanwhile, HHF induces much higher processing overhead of between 5 and 70 replicas per flooding packet. As expected, it induces the highest processing overhead in large and dense data center topologies. Although the processing overhead is also higher in power-law topologies, it does not increase with the size of the topology, which can be explained by the average node degree staying the same in contrast to the data center topologies. Still, the maximum number of received replicas increases anyway

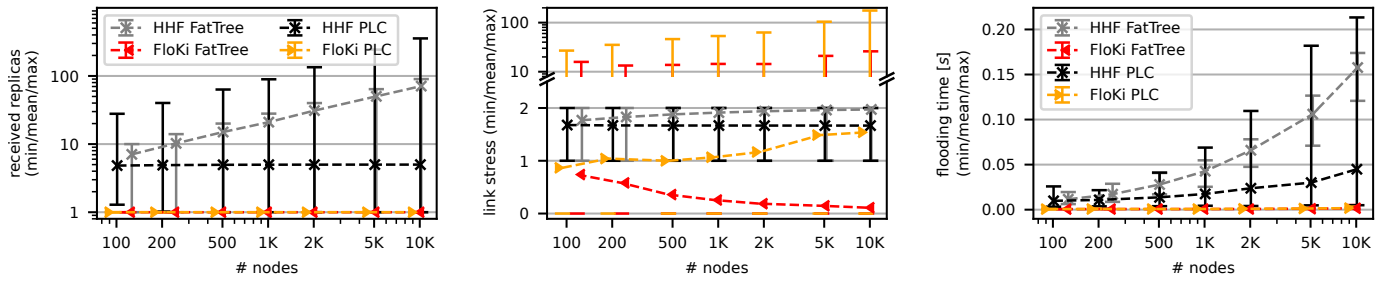


Fig. 4. Received replicas per node (left), link stress (middle) and flooding time (right) in power-law (PLC) and fat tree topologies with 100 to 10 000 nodes.

because of the power-law characteristic: *hub nodes* with high node degree also receive many duplicates.

To gain insight into the network load created, the center of Fig. 4 shows the induced link stress. As expected, HHF causes link stress between one and two for both topologies and all sizes because it sends at least one and at most two replicas over the same link (one for each direction). FLOKI, however, forwards replicas along paths of its replication tree, which may overlap, resulting in the same link being used multiple times, while other links may not be used at all. For FLOKI the link stress in the power-law topology increases logarithmically because the number of KIRA routing table buckets also increases logarithmically with network size. This results in a higher number of recursions and, therefore, replication trees with higher depth. The structure of the topology induces a maximum of up to 150 replicas on individual central links: The shortest path between neighbors in the replication tree will often lead over the central *hub nodes* using the same links. For the fat tree topologies the opposite trend can be observed, i.e., with increasing network size the average link stress decreases significantly. This is because FLOKI, unlike HHF, does not flood over all links, but only along the LSPs in its replication tree. The number of redundant links increases significantly with increasing network size, but FLOKI will only use as many as it needs.

The results for the flooding time (cf. right of Fig. 4) show a significant reduction for FLOKI compared to HHF. Moreover, FLOKI's flooding time does not increase significantly with growing network sizes. In contrast, the results for HHF show significant increases. This has two reasons: First, with increasing network size the diameter of the power-law topology increases. This means that a flooding packet has to traverse more hops (and be processed by them) to reach all nodes in the topology. Second, the increasing node degree (in PLC only for the hub nodes) results in a significantly higher number of duplicates being processed by these nodes. These fill the processing queue and need to be processed in the node's control plane before the next non-duplicate flooding packet can be processed and flooded. This heavily depends on the number of packets simultaneously flooded (in our case 5).

Takeaways: FLOKI significantly reduces control plane processing overhead independent of the network size, which also translates to significantly lower flooding times. While the cost is a less evenly distributed network load compared to HHF, this is unlikely to cause problems in practice: the most heavily stressed link varies depending on the originator, and links in central network positions are typically provisioned for higher load.

VII. RELIABILITY

FLOKI's replication trees guarantee that each node receives the flooding packet exactly once. However, link failures can break the connectivity of FLOKI's replication tree if the failed link is part of it. While KIRA will re-converge quickly after the link failure is detected and the replication trees are updated accordingly, flooding packets sent in the meantime may be lost because they try to use the broken link. HHF is less affected due to redundantly flooded packets.

To evaluate the impact on the delivery ratio of flooding packets we evaluate FLOKI in scenarios with link failures.

After the warm-up time of 15 s we introduce a *failure event* every 10 s in which 1, 5 or 50 links fail simultaneously. Nodes adjacent to a failed link send a flooding packet 1 ms after the failure. This simulates a CP application informing others of the topology change.

Results: Even with 50 simultaneous link failures FLOKI still achieves very high delivery ratios ($> 99.7\%$). We also implemented a reliability extension based on acknowledgments and retransmissions along each LSP used in FLOKI's replication tree. With this we reach all nodes for slightly higher network load (see appendix B for details and more results). Although HHF achieves a 100% delivery ratio by itself, it results in significantly higher flooding times with increasing failure rate due to the increased number of flooded packets. For example, with 50 failed links in a 10k nodes power law topology FLOKI achieves a mean flooding time of $\approx 200\,ms$ while HHF requires multiple seconds.

VIII. CONCLUSION

We introduce FLOKI, an efficient flooding mechanism using existing connectivity provided by the routing architecture KIRA. FLOKI floods in a structured way along replication trees that are established by installing pre-calculated replication rules in the nodes' forwarding planes. With this, FLOKI creates no duplicates during flooding, addressing key scalability challenges of traditional hop-by-hop flooding approaches in dense topologies. Moreover, FLOKI's pre-calculated replication rules result in significantly lower flooding times in dense topologies. We evaluated FLOKI in different topologies and show that it performs especially well in dense data-center topologies, where it also significantly reduces network load.

References

[1] J. Moy, "OSPF Version 2." RFC 2328, Apr. 1998.
[2] ISO, "Intermediate System to Intermediate System intra-domain routeing information exchange protocol," Standard 10589:2002, International Organization for Standardization, Geneva, CH, 2002.
[3] A. Krentsel *et al.*, "A Decentralized SDN Architecture for the WAN," in *ACM SIGCOMM*, 2024.
[4] C. Bormann, B. E. Carpenter, and B. Liu, "GeneRic Autonomic Signaling Protocol (GRASP)." RFC 8990, May 2021.
[5] A. Retana and R. White, "Optimizing Link-State Protocols for Data Center Networks," in *The Internet Protocol Journal*, vol. 16, 2013.
[6] T. Li, P. Psenak, H. Chen, L. Jalil, and S. Dontula, "Dynamic Flooding on Dense Graphs." RFC 9667, Oct. 2024.
[7] R. White, S. Hegde, T. Przygienda, L. Jalil, and D. Voyer, "IS-IS Distributed Flooding Reduction," Internet-Draft draft-ietf-lsr-distoptflood-08, Internet Engineering Task Force, Apr. 2025. Work in Progress.
[8] R. Bless *et al.*, "KIRA: Distributed Scalable ID-based Routing with Fast Forwarding," in *IFIP Networking*, 2022.
[9] P. Maymounkov *et al.*, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems*, Springer, 2002.
[10] Z. Czirkos, G. Bognár, and G. Hosszú, "Packet loss and overlay size aware broadcast in the kademlia p2p system," *International Journal on Communication*, vol. 4, no. 1, 2013.
[11] E. Rohrer and F. Tschorsch, "Kadcast: A Structured Approach to Broadcast in Blockchain Networks," in *1st ACM Conference on Advances in Financial Technologies*, 2019.
[12] E. C. Rosen *et al.*, "Ingress Replication Tunnels in Multicast VPN." RFC 7988, Oct. 2016.
[13] D. Voyer *et al.*, "Segment Routing Replication for Multipoint Service Delivery." RFC 9524, Feb. 2024.
[14] A. Shaikh and A. Greenberg, "Experience in black-box ospf measurement," in *ACM SIGCOMM Workshop on Internet Measurement*, Association for Computing Machinery, 2001.
[15] Arista, *7280R3 Series Data Center Switch Router*, 2025.
[16] P. Holme and B. J. Kim, "Growing scale-free networks with tunable clustering," *Phys. Rev. E*, vol. 65, Jan 2002.
[17] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *ACM SIGCOMM*, 2008.
[18] E. Rohrer and F. Tschorsch, "Kadcast-NG: A Structured Broadcast Protocol for Blockchain Networks," *IEEE/ACM Transactions on Networking*, vol. 31, no. 6, 2023.
[19] R. White and M. Aelmans, "Recent Developments in Link State on Data-Center Fabrics," in *The Internet Protocol Journal*, vol. 23, 2020.
[20] T. Eckert, M. H. Behringer, and S. Bjarnason, "An Autonomic Control Plane (ACP)." RFC 8994, May 2021.

Appendix

A. KIRA Routing Tables

KIRA uses a list of k-buckets, holding up to k contacts each. Each bucket holds contacts from different sub-trees of the ID-based overlay, together ensuring that each node knows some nodes from all parts of the overlay, while knowing all ID-wise neighbors. Fig. 5 shows a visualization of the ID-based overlay as a binary tree. We use 4-bit NodeIDs and a bucket size of $k = 3$ for simplicity. It highlights the routing table structure, i.e., the list of k-buckets (B_0, B_1, B_2) of the node with NodeID 1111 (a) and the contacts stored in them. Which contact is stored in which bucket is determined by the length of the longest common prefix between the node's own and the contacts' NodeIDs. The first bucket B_0 contains all contacts with a longest common prefix of length zero with the node's own NodeID, i.e., nodes having a different first bit in their NodeIDs (the left half of the overlay $0*$ in Fig. 5). The second bucket B_1, in turn, contains all contacts with the longest common prefix of length one, i.e., the quarter of the ID space containing nodes having the same first, but

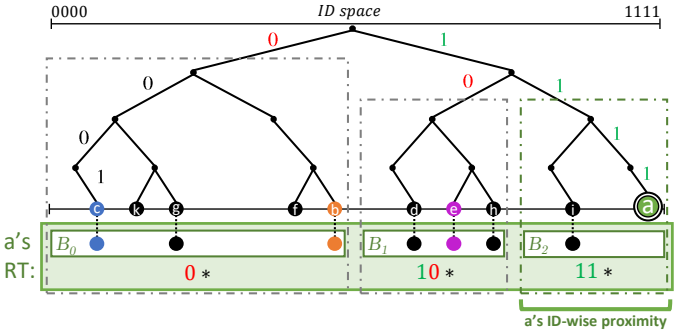


Fig. 5. KIRA's ID-based overlay structure and node a's routing table

different second bit ($10*$). Each bucket corresponds to a sub-tree of the ID-based overlay as highlighted by gray borders in the figure. The last bucket B_n (B_2 in Fig. 5) is special in that it always covers the remaining sub-tree of the ID-based overlay containing the node itself, i.e., all nodes with a longest common prefix of length n or more ($11*$ in Fig. 5). This is the node's *ID-wise proximity*. In KIRA the list of k-buckets is not static, but grows dynamically. Whenever the last bucket is full and a new contact should be added to it, the node will split the last bucket into two new buckets, each covering one half of the sub-tree of the previously last bucket. This ensures KIRA's key invariant, that a node always knows how to reach all nodes in its ID-wise proximity.

Because the first buckets are responsible for a large sub-tree of the ID-based overlay but can only hold k contacts, the node only knows how to reach a very small fraction of the nodes in that sub-tree directly. As the sub-trees become smaller for the following buckets, with constant k, this fraction increases, reaching 1 in the last bucket.

B. Reliability Extension

We designed an optional reliability extension for FLOKI that can be used to ensure quick reliable delivery of flooding packets to all nodes in the presence of packet loss (e.g., due to link failures).

Previous work on reliable flooding in overlay networks [18] used redundant paths to increase the delivery ratio, but this cannot guarantee reliable delivery and introduces significant additional overhead. Instead we introduce acknowledgments and retransmissions along the replication tree: Whenever a node replicates a flooding packet it is also delivered locally to FLOKI's reliability extension (like any other CP application). It stores the packet in a local cache, together with

Algorithm 1 Selecting child nodes with sub-trees

Require: buckets $B_0, \ldots, B_n$
▷ *All buckets except last bucket* ◁
for $B_i \in B_0, \ldots, B_{n-1}$ **do**
  ▷ *choose contact with lowest hop distance* ◁
  $c \leftarrow \{x \in B_i \mid \forall y \in B_i,\ hops(x) \leq hops(y)\}$
  $S := S \cup \{(c, p = i + 1)\}$
▷ *add all leaf nodes from the ID-wise proximity* ◁
$S := S \cup \{(x, IDBitLength) \mid x \in B_n\}$
return $S_0, \ldots, S_m$

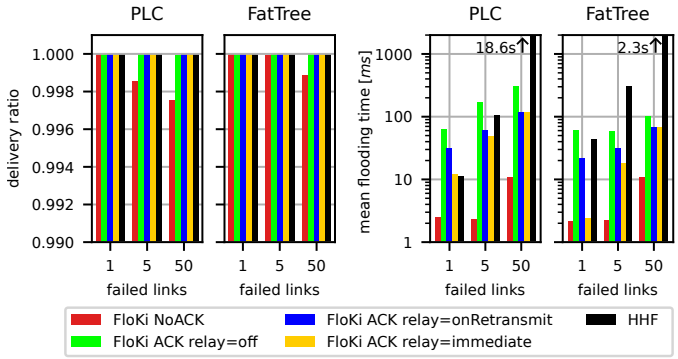


Fig. 6. Delivery ratio and flooding time under link failures.

the child nodes to which the packet was replicated, and starts a retransmission timer. Furthermore, the node sends an acknowledgment back to its parent in the replication tree.

When the retransmission timer expires, the packet is retransmitted. Since the original child node did not respond, the node instead selects a *new* child node from the same sub-tree for the retransmission. However, in ID-wise proximity, we can only retransmit the packet to the same leaf nodes, i.e., the retransmission will succeed only after KIRA has established a new LSP. Since this may take some time, i.e., $\mathcal{O}(100\,ms)$, we investigated a faster *relay mechanism*.

Relays: When retransmitting to a leaf node, the node selects another node from its ID-wise proximity as a relay, to which it will retransmit the packet. The relay will then replicate the packet in its ID-wise proximity (which is the same as that of the original node). For example, in Fig. 1b node c could use node k as a relay if, e.g., the link c to f fails, making the LSP to the other leaf node g unusable. To avoid loops, the packet also needs an additional flag indicating that it is a relayed packet.

Instead of using this mechanism only after the retransmission timer expires, we can also use it immediately if a packet arrives when KIRA has already marked an LSP to a leaf node as broken. This sums up to the following variants of FLOKI:

- `NoACK`: Without the reliability extension
- `ACK relay=off`: With just acknowledgments and retransmissions
- `ACK relay=onRetransmit`: With additionally relaying on retransmission for a leaf node
- `ACK relay=immediate`: With additionally relaying immediately when an invalid LSP to a leaf node has been detected

We evaluate the reliability extension with a 100 ms retransmission timeout.

Results: The results for the *delivery ratio* (cf. left of Fig. 6) confirm that FLOKI's reliability extension ensures delivery to all connected nodes independent of the relay configuration.

The results for the flooding time are shown on the right of Fig. 6. In general, there is a trend to higher flooding times with higher link failure rates. This trend is much more pronounced for HHF. While HHF achieves lower flooding times for a low failure rate it increases drastically with additional failing links especially in the power-law topology. In these HHF will not finish the flooding of packets from one failure event before the next event happens. This is because of the high contention between flooded packets at the central hub nodes.

For FLOKI the trend is less significant leading to much lower flooding times with a high failure rate even without the relay mechanism. Here only the higher traffic volume, due to the increasing number of flooded packets, increases the queuing delay in the nodes' fast paths. With the relay mechanism active the flooding times reduce further even below the retransmission timeout of 100 ms. This is the result of some of the replication trees not being affected by the link failure or the immediate relay successfully bypassing a failed link. It is important to note, that almost all nodes ($> 99.7\%$) will receive the packet much earlier, as can be seen in the results without the reliability extension (`NoACK`).

If we now include the ACKs for both HHF and FLOKI's reliability extension, the link stress roughly doubles for both. FLOKI also produced a duplicate in about 5% of nodes (lost ACKs leading to unnecessary retransmissions).

Takeaways: FLOKI achieves reliable delivery by using acknowledgments, retransmissions and relays. The large amount of duplicates leads to significantly higher flooding times for HHF in drastic failure scenarios.

C. Related Work

Flooding Topologies in Link State Routing: The scalability problems of HHF observed in our evaluations have been known for some time to cause issues for link-state routing in dense data-center networks [5], [19]. This initiated ongoing work [6] in the IETF trying to reduce the number of duplicates being created by calculating flooding topologies, which are sparser than the physical topology. Link state advertisements (LSAs) are then only flooded along these topologies. These are either calculated by a central leader which then distributes the flooding topology to all nodes, or decentralized on *all* nodes. The calculation itself has high computational overhead compared to FLOKI's child node selection process, because it takes the whole physical topology as input. Furthermore, any network event affecting the calculated topology will require a recalculation in *all* nodes (decentralized), or re-distribution to all nodes (centralized).

Overlay Broadcast: The *overlay broadcast* mechanisms that inspired FLOKI's [11], [18] are used in peer-to-peer networks over the Internet and are therefore unaware of the specifics of the underlay. They also replicate packets in the control plane.

Autonomic Control Planes (ACPs): ACPs [20] have been proposed as part of an autonomic networking infrastructure. The ACP functions as a resilient foundation for distributed autonomic functions, which implement core network services (e.g., efficient routing, traffic engineering, ...). One of the ACP's components is to provide a flooding mechanism, e.g., for synchronizing state or service discovery. The solution suggested in [4] uses an HHF approach which will likely result in the same scalability issues observed.