KeLLy – Efficient, Scalable Link Layer Topology Discovery

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

***Abstract*—Network infrastructures are becoming increasingly flexible and dynamic not only due to softwarization and virtualization, but also due to increasing mobility in 5G and 6G networks, which consider drones and satellites to be part of the core infrastructure. Since the network topology may change frequently, it becomes challenging to get an up-to-date view of its current state. This paper introduces *KeLLy*, an efficient, scalable link layer topology discovery algorithm focussing on large-scale networks (evaluated up to 100,000 nodes). *KeLLy* discovers various large topologies in seconds, guarantees discovery of all nodes (and a high percentage of links), while inducing low, predictable overhead by querying only a subset (4%) of nodes.**

***Index Terms*—topology discovery, network management, autonomic networks**

I. Introduction

The higher flexibility in emerging network infrastructures leads to more dynamically changing networks. Software-based nodes may be deployed or removed on demand leading, e.g., to frequent changes in the network topology. Moreover, in 5G and future 6G networks the number of services and connected devices is ever increasing as well as the number and types of mobile devices. For instance, mobile nodes, such as drones and satellites, are considered to be part of the core network, which will lead to increasing dynamics. Moreover, nomadic network partitions may arise that split from the rest of the network and rejoin later at a different location. Consequently, topologies may change more frequently than ever and networks may behave more in an ad-hoc manner.

One may even view networks as a pool of resources that dynamically (re-)configures as needed. Resources are switches, routers, end systems, compute and storage nodes as well as devices in the Internet of Things context (e.g., drones, mobile robots) and service entities comprising end-user services and network services. Such high flexibility and dynamics, however, make network management increasingly challenging. Therefore, autonomic management solutions are gaining importance, like the work done in IETF's ANIMA group on autonomic control planes [1], [2]. An important component of such a fully autonomic management solution is a fast, efficient and scalable discovery of the network´s current topology. Moreover, many control plane concepts also require an overview of the current network topology: software-defined networking (SDN) controllers use it to make informed routing decisions and *orchestration mechanisms* perform service placement based on topological information.

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

A basic overview of the topology can be provided by discovering the network's link-layer topology. For highly dynamic infrastructures such a discovery needs to be fast and efficient, because it may need to run repeatedly in case of network partitions, severe network outages, or simply due to high dynamics in general. For example, a newly instantiated SDN controller in a nomadic network may need to rediscover the link-layer topology of the network partition.

This paper introduces *KeLLy* (***K**IRA-**e**nabled **L**ink-**L**ayer Discover**y***) a highly scalable, topology discovery algorithm, providing a topological map of the network in a fast manner with low overhead. It guarantees to discover all nodes and a large portion of a topology's links, while having a predictable constant relative overhead w.r.t. the number of nodes in the topology. *KeLLy* is based on reusing distributed routing state information of the zero-touch routing architecture KIRA [3].

II. KeLLy: Basic Concept

We use $G := (V, E)$ as a graph representation of a link-layer topology, where V denotes the set of nodes and E is the set of links. In general, link-layer topology discovery tries to find a graph $G' := (V', E')$ with the nodes $V' \subseteq V$ and edges $E' \subseteq E$ resembling G as closely as possible.

On a high level, *KeLLy* is a topology discovery algorithm that uses distributed information stored in the routing tables of network devices, which represents a partial view of the real topology. Based on this, a sub-graph $R(v)$ of G can be derived for each node v. By querying the routing tables of a set of nodes $Q \subseteq V$ and merging their partial views a graph approximating the topology of the real network can be obtained: $G' := \bigcup_{v \in Q} R(v)$. In order to do so, *KeLLy* is based on the recently introduced routing architecture KIRA.

KIRA is a zero-touch routing architecture providing control plane connectivity in large-scale networks. For instance, it enables robust reachability between SDN controllers and their controlled switches in a zero-touch manner (i.e., without any configuration). KIRA uses concepts of structured overlay networks (Kademlia [4]), but does not require an underlay providing universal connectivity. Instead, a KIRA node stores underlay *source routes* to a small selection of nodes – its *contacts* – in its routing table. KIRA's link-layer source routes and its routing table structure allows *KeLLy* to discover all nodes and the majority of their links without having to contact every node individually, thereby reducing overhead.

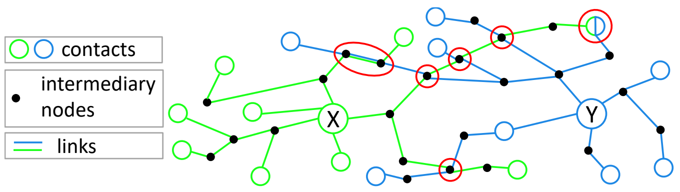


Fig. 1: Topology discovered by two nodes X and Y. Common components $R(X) \cap R(Y)$ are marked by red circles.

A. Applying KeLLy to KIRA

The source routes stored in KIRA's routing tables enable a straightforward derivation of the sub-graph $R(v)$ of a node v: $R(v)$ consists of node v, contacts of v, as well as all links and nodes along the link-layer source routing paths. Figure 1 shows two exemplary sub-graphs $R(X)$ and $R(Y)$ from nodes X and Y. When querying multiple nodes, their resulting sub-graphs often overlap, either because the nodes have shared contacts or the source routing paths have intersections, i.e., common nodes and links. Querying routing tables of other nodes is a key concept of KIRA as it is used to discover new contacts. KIRA's *FindNodeRequest* message allows searching for nodes close to some destination ID by recursively routing closer in the ID space with each overlay hop, until it reaches the node closest to the queried ID. The same mechanism can be used for the routing table queries required for *KeLLy*. This allows to send a query targeting a particular *section* of the ID space rather than to an existing individual node that would need to be known beforehand.

Using this routing table querying concept of KIRA the main remaining challenge is that a node using *KeLLy* must determine when it has discovered all nodes in the network. The trivial solution of querying every node in the network would negate the advantage of the additional information gathered from the source routes in the routing tables. Therefore, an exit condition identifying when all nodes have been discovered while maintaining low overhead is essential.

B. KIRA Routing Tables

The structure of KIRA's routing tables is essential for *KeLLy*'s design. KIRA inherits the basic structure of its routing table from Kademlia [4]. This structure implies the important invariant that each node knows how to reach its *ID-wise closest neighbors*. These neighbors are determined using the XOR metric $d(X, Y) = X \oplus Y$ as distance between two NodeIDs X and Y. Like in Kademlia, a KIRA node stores routing table entries for known nodes (also called *contacts*) in a list of k-buckets each storing up to k contacts at most. Each bucket covers a specific range from the ID space. This *bucket range* is defined as follows: the bucket with index i contains contacts with NodeIDs having a longest common prefix of length i with the node's own ID. The bucket range of the first bucket therefore includes all NodeIDs with a different leading bit than the node's own ID. The respective range in the ID space includes half of all possible IDs. With each consecutive bucket (i.e., increasing i) the size of the corresponding range in the ID space halves, as they are defined by a longer longest common prefix, fixing more leading bits to be equal to the node's own ID. Consequently, each routing table comprises $\mathcal{O}(\log n)$ contacts at most.

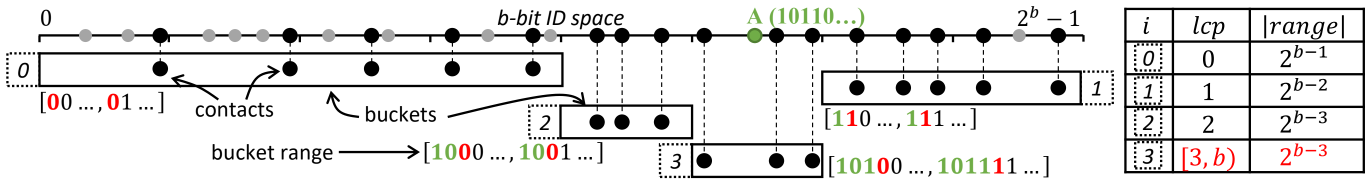
Figure 2 shows an example detailing the structure of the routing table of a node A with four buckets. At the top it shows the ID space as a line with nodes (including A) placed at the location of their NodeID on that line. The bucket range, shown next to each bucket, indicates the range of IDs of all potential contacts in that range. The range of the first bucket (index 0) spans half of the ID space. Because of the limited bucket space (here $k = 5$) only a selection of the nodes in this range can be stored as contacts. In the example, this is the case for the first two buckets. The last bucket in the routing table is special: its covered ID space range is defined by the length of the longest common prefix being equal *or larger* (instead of a specific length) than the bucket's index. On the right-hand side of the figure a table shows the properties of each bucket. The list of buckets grows dynamically depending on the number of nodes in the network: if the last bucket (storing the ID-wise closest contacts) is full and a new contact falls into the respective range, the list is extended by a new k-bucket. This means that the prior last bucket range is effectively split into two ranges (halving its original range) and some of its contacts will move over to the new last k-bucket. This ensures that the contacts in the direct ID-based neighborhood never need to be replaced, upholding the invariant that each node knows how to reach its ID-wise neighbors.

C. Bucket Completeness

If a bucket contains all nodes with IDs in its bucket range that are actually present in the network we call it *complete*. In a converged network state, this is the case when there exist fewer than k nodes with their NodeIDs in the respective ranges. Due to each node generating a uniformly distributed random NodeID at startup, all NodeIDs are spread evenly across the whole ID space. Therefore, buckets with a large bucket range (i.e., the first buckets) also have a large share of the nodes to select k contacts from and are therefore rarely complete. Because of the dynamically growing bucket list, the last bucket is always complete: if a new contact should be entered into an already full last bucket, the bucket list is extended by another bucket having a smaller bucket range and therefore a lower number of nodes as potential contacts. Besides the last bucket, in a converged network state, other *non-full* buckets (fewer than k contacts), are also complete [5]. In such a bucket no contact was replaced, as there was enough space for all of them. On average the second to last bucket will be non-full, because its bucket range has the same size as the last bucket, on average containing the same number of contacts.

III. KeLLy Design

One of the main concepts in *KeLLy*'s design are *discovery ranges* in the ID space, that are defined by the *complete* buckets in each node's routing table. Because the range of a *complete* bucket contains only the nodes stored as contacts in that bucket this ID-range is regarded as *discovered* by that



i	lcp	$\lvert range \rvert$
0	0	2^{b-1}
1	1	2^{b-2}
2	2	2^{b-3}
3	$[3, b)$	2^{b-3}

Fig. 2: A node A having $ID_A = 10110...$ with its position in the ID space and its routing table. The table on the right shows the bucket index i, the longest common prefix (lcp) and the size of the range of each bucket. Common prefix with A in green; first different bit in red.

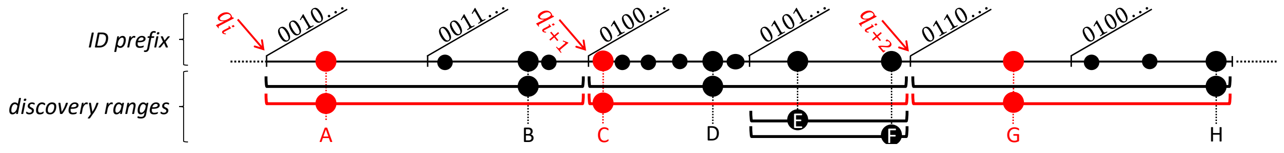


Fig. 3: A section of the ID space with nodes placed according to their NodeID. For selected nodes (A-H) their respective discovery ranges are shown. Queries q_i in the ID space and the respective responding nodes with their discovery ranges are colored red.

bucket. The discovery range $DR(v)$ of a node v is defined by the bucket ranges of all complete buckets in its routing table combined. Because the discovery range is generally defined by the last two buckets of a node's routing table, with bucket ranges containing the ID-based neighborhood of the node, it encloses a range around the node's own ID. Furthermore, the size of the discovery range $|DR(v)|$ depends on the depth of the routing table: the larger the index of the last bucket the smaller the bucket range. The number of buckets in the nodes' routing tables, and therefore also the size of their discovery range, is about the same for all nodes in a network, due to the uniform distribution of NodeIDs. Figure 3 shows a section of the ID space with nodes and their respective discovery ranges. The discovery ranges of nodes ID-wise close to each other are *identical* (e.g., A & B), because they all share the same common prefix and therefore the same complete buckets in their routing tables. Discovery ranges can theoretically completely overlap other smaller discovery ranges ($C - F$), but according to our experiments that occurs very rarely, because it requires strongly unevenly distributed NodeIDs in that range. Nodes ID-wise a bit further away (e.g., A & C) have *consecutive* discovery ranges. This has two advantages, a first, it does not matter which node in each discovery range is used and second no ID-range needs to be discovered redundantly.

A. Exit Condition

KeLLy uses discovery ranges to define an exit condition when finding a *spanning sub-graph* of the network topology. If all possible IDs (within the respective ID space) are covered by the discovery range of one of the queried nodes then all nodes are *guaranteed* to be discovered in one of them. To achieve that, each distinct discovery range needs to be queried at least once. So in order to determine whether all nodes have been found, one simply keeps track of the discovery ranges in the overlay ID space and keeps querying undiscovered ranges of the ID space until the *whole* ID space has been covered.

B. ID Space Walk

Now a sequential algorithm can be built using a structured approach of *walking* across the ID space: starting from an ID a, which is queried first, the approach waits for the query response to return the routing table information of the node closest to the queried ID and calculates its discovery range. Next, it queries the next ID following the end of the discovery range. Starting by querying ID 0 and repeating until the end of the ID space (ID $2^b - 1$) the whole ID space is discovered. This algorithm minimizes the number of required queries. The downside of proceeding this way is that each iteration needs to wait for the query to return before starting the next query. This mechanism is also depicted in fig. 3 for a section of the ID space. The first query q_i is routed to node A, the next node according to the XOR metric, which returns its complete routing table (with $\mathcal{O}(\log n)$ entries). The next query (q_{i+1}) is placed consecutively after the corresponding discovery range and is routed to node C. q_{i+2} queries the next consecutive ID, which is routed to node G. Node G is closer, by the XOR metric than node F, as it has a longer common prefix with the *query ID* and therefore a smaller distance based on the XOR metric. This is also easy to parallelize by slicing the ID space into sections and starting an ID space walk for each of them.

IV. Evaluation

KeLLy's evaluation is based on a KIRA implementation using the OMNeT++ simulation framework. In the evaluation multiple parallelized ID space walks were used. Due to space limitations we focus on *KeLLy*'s scalability characteristics. The following metrics are used to evaluate *KeLLy*:

- *Query Ratio*: The main objective of *KeLLy* is to achieve low overhead w.r.t. number of messages sent during the discovery. Therefore, the fraction of nodes $|Q|/n$ queried during topology discovery is a key metric.
- *Link Coverage*: *KeLLy* in its current form does not guarantee to find all links. The percentage of links discovered $|E'|/|E|$ is an important metric, as it identifies the accuracy of the discovered graph G' compared to the real topology.
- *Data Usage*: This metric identifies the amount of data transmitted during discovery. For each query the number of transmitted NodeIDs (contacts and nodes on source-routing paths) is multiplied by the 14 Byte NodeIDs used in KIRA. Summing over all queries gives *KeLLy*'s total data usage.

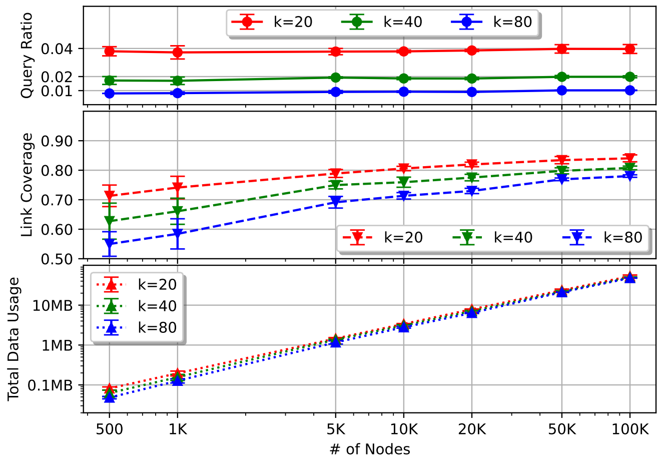


Fig. 4: Query ratio, link coverage and data usage in randomly generated power law topologies. Mean and 2σ deviation (10 runs)

Overall, the presented evaluations show that *KeLLy* works well even for very large topologies while inducing acceptable overhead. Real networks such as the Internet follow a power law node-degree distribution [6]. Therefore, to evaluate the influence of the network size on the efficiency of *KeLLy*, simulations in differently sized, randomly generated power law topologies are compared. The topologies were generated using the algorithm introduced by Holme and Kim [7] with the parameters $m = 3$ and $p = 0.5$. The expectation is, that in larger topologies the same query ratio is required to find all nodes, because it only depends on the bucket size k. Firstly, in all simulations *KeLLy* discovered all nodes of the topology, confirming that the algorithm works as expected. Figure 4 shows evaluations with up to 100,000 nodes. As expected, the query ratio stays the same at roughly 4% of all nodes for a bucket size of $k = 20$. Furthermore, the query ratio decreases proportionally to k^{-1} to 2% and 1% for $k = 40$ and $k = 80$, respectively. The figure also shows that the link coverage increases with the size of the topology. This has two reasons: first, the size of the routing tables increases with growing topology size leading to more contact information being gathered per query and secondly, the average path length of the source routing path to each contact is longer in larger topologies and thus contains more links that are discovered per contact. Another characteristic impacting link coverage is the number of queries sent during a discovery: Each query discovers information from a different *vantage point*, because all discovered source routing paths originate from the queried node. Therefore, a lower query ratio, with larger k leads to a lower link coverage. Furthermore, with larger k each query discovers more nodes, but the source routing paths overlap more often close to the queried node. Additionally, the figure shows the transmitted data for each configuration. As the routing tables grow bigger and the absolute number of queries increases, it is expected that the amount of data transmitted also increases. It is important to note that, because information about each node needs to be transmitted at least once, sub-linear overhead is impossible. Still, for 100,000 nodes the overhead is approximately 52 MByte of data when using NodeIDs of size 112 bit. Interestingly, the overhead is the same irrespective of k: the larger routing tables with higher bucket size k balance out with the lower amount of queries required. When comparing *KeLLy* to an approach used by SDN controllers based on LLDP (link-layer discovery protocol), like [8], there are a couple of key differences: With 100,000 nodes that algorithm would induce 700,000 messages, one from the controller to each switch and two for each link back, while *KeLLy* induces only 8,000 messages (4,000 queries and responses). Even though the total data usage only comes down to around three times that of *KeLLy* (267 Byte LLDP packets [8]), each of these messages has to be sent and processed in the SDN controller. In return, the LLDP-based approach achieves 100% link coverage.

KIRA works well across a wide range of topologies due to its topology-independent NodeIDs and overlay-based routing concepts. In order to confirm that *KeLLy* inherits this characteristic, it was run on various synthetic topologies of the same size with different characteristics. Our evaluations have shown, that the query ratio is the same (4% of nodes) for all topologies while data usage depends on the topology's diameter.

V. Related Work

Other link-layer topology discovery approaches come with different problems w.r.t. overhead, scalability and their suitability for autonomic networks. LLDP-based solutions used in SDN controllers [8]–[10] and solutions based on data-plane routing protocols [11], [12] run into scalability issues in large-scale networks, unless the network is split into multiple domains by manual configuration [13]. Other more general ICMP-based solutions [14]–[16] are also affected by this indirectly, because they require a functional data plane routing. *KeLLy* building on the zero-touch KIRA can circumvent this issue. An autonomic management solution could derive network domains for data plane routing or SDN using the topological map discovered by *KeLLy*.

VI. Conclusion & Future Work

Emerging autonomic network management solutions dealing with the increasing dynamics and complexity of network infrastructures require topological information. Providing this information should also be done in an autonomic manner. We introduced *KeLLy*, an efficient, scalable link-layer topology discovery algorithm focussing on large-scale networks. It achieves low message overhead (querying only 4% of nodes) by re-using distributed routing information. Furthermore, *KeLLy* guarantees to discover all nodes, and achieves high link coverage ($> 80\%$). In future work we will present a concept that allows to invest some compute power to calculate remaining missing links by transmitting small amounts of additional information. First evaluations have shown promising results. Furthermore, a concept for integrating incremental changes to the discovered topology during network dynamics will be investigated.

References

[1] M. H. Behringer, B. E. Carpenter, T. Eckert, L. Ciavaglia, and J. C. Nobre, "A Reference Model for Autonomic Networking," RFC 8993, May 2021. [Online]. Available: https://www.rfc-editor.org/info/rfc8993

[2] T. Eckert (Ed.), M. Behringer (Ed.), and S. Bjarnason, "An Autonomic Control Plane (ACP)," RFC 8994, May 2021. [Online]. Available: https://www.rfc-editor.org/rfc/rfc8994

[3] R. Bless, M. Zitterbart, Z. Despotovic, and A. Hecker, "KIRA: Distributed Scalable ID-based Routing with Fast Forwarding," in *2022 IFIP Networking Conference (IFIP Networking)*, 2022, pp. 1–9.

[4] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer Information System based on the XOR Metric," in *International Workshop on Peer-to-Peer Systems*. Springer, 2002, pp. 53–65.

[5] S. Roos, H. Salah, and T. Strufe, "Comprehending Kademlia Routing – A Theoretical Framework for the Hop Count Distribution," 2013. [Online]. Available: https://arxiv.org/abs/1307.7000

[6] D. Krioukov, k. c. claffy, K. Fall, and A. Brady, "On compact routing for the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 3, p. 41–52, jul 2007. [Online]. Available: https://doi.org/10.1145/1273445.1273450

[7] P. Holme and B. J. Kim, "Growing scale-free networks with tunable clustering," *Phys. Rev. E*, vol. 65, p. 026107, Jan 2002. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.65.026107

[8] F. Pakzad, M. Portmann, W. L. Tan, and J. Indulska, "Efficient topology discovery in OpenFlow-based Software Defined Networks," *Computer Communications*, vol. 77, pp. 52–61, 2016.

[9] S. Khan, A. Gani, A. W. Abdul Wahab, M. Guizani, and M. K. Khan, "Topology discovery in software defined networks: Threats, taxonomy, and state-of-the-art," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 303–324, 2017.

[10] E. Rojas, J. Alvarez-Horcajo, I. Martinez-Yelmo, J. A. Carral, and J. M. Arco, "Tedp: An enhanced topology discovery service for software-defined networking," *IEEE Communications Letters*, vol. 22, no. 8, pp. 1540–1543, 2018.

[11] M. F. Rabbi Ur Rashid, C.-H. Lung, M. St-Hilaire, B. Nandy, and N. Seddigh, "Combining SPF and source routing for an efficient probing solution in IPv6 topology discovery," in *2014 Global Information Infrastructure and Networking Symposium (GIIS)*, 2014, pp. 1–6.

[12] M. Li, J. Yang, C. An, C. Li, and F. Li, "IPv6 network topology discovery method based on novel graph mapping algorithms," in *2013 IEEE Symposium on Computers and Communications (ISCC)*, 2013, pp. 000 554–000 560.

[13] A. D. Ferguson, S. Gribble, C.-Y. Hong, C. Killian, W. Mohsin, H. Muehe, J. Ong, L. Poutievski, A. Singh, L. Vicisano *et al.*, "Orion: Google's software-defined networking control plane," in *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*, 2021, pp. 83–98.

[14] K. Vermeulen, J. P. Rohrer, R. Beverly, O. Fourmaux, and T. Friedman, "Diamond-Miner: Comprehensive Discovery of the Internet's Topology Diamonds," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 479–493.

[15] R. Beverly, "Yarrp'ing the Internet: Randomized high-speed active topology discovery," in *Proceedings of the 2016 Internet Measurement Conference*, 2016, pp. 413–420.

[16] J.-F. Grailet and B. Donnet, "Travelling Without Moving: Discovering Neighborhood Adjacencies," in *Network Traffic Measurement and Analysis Conference*, 2021.