Scalable and Efficient Link Layer Topology Discovery for Autonomic Networks

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

***Abstract*—Increasingly flexible and dynamic network infrastructures, due to softwarization, virtualization and increasing mobility (5G and 6G networks), challenge network management. To cope with the increasing dynamics and complexity, recent research focuses on autonomic network management solutions that do not require human intervention. However, autonomic network management solutions often require up-to-date topological information during network bootstrapping or to quickly react to dynamic events. Therefore, a fast, efficient, and scalable link-layer topology discovery, providing autonomic network management solutions with topological information in highly dynamic and large-scale networks, is required. This paper introduces *KeLLy* an efficient, scalable link layer topology discovery algorithm for large-scale networks (evaluated with up to 100 000 nodes). *KeLLy* is fast, discovering large topologies within a few seconds, guarantees discovery of all nodes and all links while inducing low, predictable overhead by querying only a subset (generally below 4 percent) of nodes. It achieves these properties regardless of the type of underlying topology.**

***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 increasing as well as the number and types of mobile devices. For instance, drones and satellites are considered to be part of the core network, which will lead to increased dynamics. Furthermore, nomadic network partitions may split from the rest of the network and rejoin later at a different location. Consequently, topologies change more frequently and networks may behave (partly) more in an ad-hoc manner.

Such high flexibility and dynamics, however, make network management increasingly challenging. Therefore, autonomic management solutions that can operate without human intervention (zero-config) during bootstrapping or after unexpected, dynamic network events, are becoming increasingly attractive [1] [2]. These autonomic management solutions often require topological information to perform orchestration or management tasks, e.g., to make service placement decisions. Therefore, a topology discovery algorithm is required that provides this information by discovering the network's current link-layer topology. Such an algorithm needs to be *fast* and *efficient* to allow quick adaptations after dynamic events. It also needs to be *scalable* to support large-scale scenarios with hundreds of thousands of nodes as, e.g., envisioned in future 6G network infrastructures. Furthermore, the *topology discovery* algorithm *itself* should be autonomic, i.e., it should not depend on any human intervention beforehand. This is especially relevant when bootstrapping large-scale softwarized networks. There are a variety of state-of-the-art topology discovery solutions, for SDN, using link-state routing protocols (e.g., OSPF), management protocols (e.g., SNMP) or packet probes (e.g., ICMP, UDP). All fit different use-cases but none fit the particular *combination* of requirements in large-scale autonomic networks: *scalability*, *efficiency* and *autonomy* (see section II for details).

Another key requirement for the manageability of future network infrastructures is robust, zero-config control plane connectivity, as identified in [3]. In this space IETF's ANIMA group has worked on *autonomic control planes* [4] [5], that provide control plane connectivity in a zero-config manner. With KIRA [6] we recently presented an alternative approach, which employs a novel zero-config ID-based routing architecture. We found that its overlay-like structure makes it straightforward to design a lightweight topology discovery algorithm, that reuses its distributed routing state information.

We introduce the algorithm *KeLLy* (***K**IRA-**e**nabled **L**ink-**L**ayer Discover**y***), which is a scalable and efficient link-layer topology discovery algorithm for autonomic networks using KIRA. *KeLLy* is designed with large networks in mind: It is highly scalable, discovering large network infrastructures (evaluated with up to 100 000 nodes) while inducing low communication overhead by querying only a small subset (generally less than 4%) of nodes. *KeLLy* is also fast, discovering large topologies in a matter of seconds. In its most basic form, *KeLLy* guarantees to discover *all nodes* and a high percentage of the links (over 80%) of a topology. Using the zero-config KIRA also makes *KeLLy* itself work in zero-config environments.

This contribution completes earlier preliminary work in [7] by greatly improving the guarantees: It introduces and evaluates an additional link calculation scheme that allows to discover 100% of links in addition to all nodes, therefore completing the discovered topology. Furthermore, this work includes additional extensive evaluations featuring different topologies and additional metrics like the discovery time. Moreover, a greatly extended related work section highlights differences and improvements over related approaches.

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

Approach	Scalability	Efficiency	Autonomy	References
SDN	◐	◐	◐	[8]–[10]
Link-State Routing	◐	●	◐	[11], [12]
Probing	◐	○	◐	[13], [14]
SNMP	◐	◐	◐	[15]
BRSKI	◐	◐	●	[16]
KeLLy	●	●	●	

TABLE I: Comparison of Approaches in Related Work

II. Related Work

This section gives an overview of related work and evaluates along important requirements, for large-scale autonomic networks: *scalability*, *efficiency*, and *autonomy* (also see Table I).

Software-defined Networking (SDN) controllers [8] discover the topology, to maintain their global view, by injecting LLDP (Link-Layer Discovery Protocol) packets into switches which are then sent back to the controller by neighboring SDN switches that receive them. This way the controller can detect each link individually. Some improvements are presented in [9], [10]. However, at least two messages need to be sent for each link in the network and need to be handled by the controller. Furthermore, in large SDN networks a single controller is not enough due to resource limitations and latency between controller and switch. Therefore, in large networks manual division into SDN domains is necessary [17] posing a cyclic dependency: the approach requires the network to be divided into SDN domains, but the division itself requires topological information already. The approach therefore is neither zero-config nor autonomic, in addition to requiring pre-existing control network connectivity, between the controller and its switches, that also needs to be configured autonomically. Furthermore, only the controller can discover the topology.

Another often applied approach is to reuse information exchanged by *link state routing protocols* (e.g., OSPFv3, IS-IS). Routers exchange link state advertisements (LSAs) that include information about the neighbors they are connected to. Each router discovers the complete network topology and uses it to calculate shortest paths to all other routers. Several topology discovery solutions have been proposed using the information exchanged in LSAs to discover the same topology as the router themselves [11], [12]. These solutions are very efficient as they are passive actors and do not induce any communication overhead. The limitations of this approach are inherited by OSPF: in large-scale networks (more than a few hundred routers) OSPF, like SDN, requires division into routing areas which are generally defined by human operators using their topological knowledge, which is not autonomic and practically infeasible for the large-scale and dynamic nature of future network infrastructures. Even though there are solutions allowing link-state routing protocols to be zero-config they are limited to small simple deployments [18]. There are similar approaches using IS-IS suffering from the same limitations.

More general approaches to topology discovery use tools like *traceroute* or information from control plane or management applications. For example, topology discovery approaches used for Internet measurements mainly build upon the usage of ICMP through tools like *ping* and *traceroute* or UDP probes [14] [13]. In general, these have high overhead in relation to the network size w.r.t. probe count. For instance, *Diamond-Miner* [13] sent more than 6 billion probes to discover 1 million nodes and 3 million links. Furthermore, because they require pre-existing connectivity their autonomy is dependent on the autonomy of the routing protocol in use. Furthermore, these approaches do not make any guarantees in terms of what percentage of nodes or links in a network are discovered. Another challenge to these approaches is that the results also depend on the *vantage point* the probes are sent from. Most such approaches therefore have to deploy multiple nodes in the network, from which probes can be sent.

Furthermore, there are approaches using the network management protocol SNMP. The authors of [15] present an approach using SNMP to query information about each network device and its interfaces from a central node. Same as the *probing*-based approaches this also requires pre-existing connectivity that needs to be established autonomically. In terms of efficiency these approaches need to query the management information in each node once.

To our knowledge [16] is the only related work also looking into topology discovery specifically for autonomic networks. It presents different solutions, based on employing a distributed clustering approach, or using the secure bootstrapping protocol BRSKI [19] to transport the list of neighbors of a newly joined node towards a single central registrar which keeps an up to date map. The authors evaluate their solutions in small topologies (10 nodes). A scalability analysis and experiments in larger topologies are not discussed. All presented solutions send more messages than nodes in the network.

All aforementioned solutions only fulfill parts of the requirements (the requirements relevant to the respective use case) while having limitations in others. *KeLLy*'s main contribution is being strong on all of them, i.e., it is scalable, discovering large-scale networks *and* efficient, contacting only 4% of nodes, *and* autonomic, i.e., it does not depend on any prior human intervention of the network. This set of characteristics makes *KeLLy* the perfect fit for providing topological information in future large-scale, autonomic networks. One potential use case for *KeLLy* in such networks is to break up the cyclic dependency mentioned above: an autonomic management solution can use the discovered link-layer topology to calculate and configure routing areas for link-state routing protocols to enable efficient data plane routing.

III. KeLLy: Basic Concept

KeLLy, and other link-layer topology discovery algorithms, try to discover network devices and their interconnections: For a given link-layer topology, represented by graph $G := (V, E)$ where V denotes the set of nodes and E is the set of links, they try to find a graph $G' := (V', E')$ with nodes $V' \subseteq V$ and edges $E' \subseteq E$ resembling G as closely as possible.

KeLLy is a topology discovery algorithm that uses distributed information stored in routing tables of network devices. The information stored in a node v represents a partial

view of the physical topology, from which a sub-graph $R(v)$ of G can be derived. 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: $G' := \bigcup_{v \in Q} R(v)$. To do so, *KeLLy* is based on the routing architecture KIRA. The following briefly introduce its key characteristics that allow for *KeLLy*'s efficient design.

A. KIRA

KIRA provides a zero-config routing architecture for autonomous control plane connectivity. Its design features, (1) topology-independent, randomly generated IDs (called *NodeIDs*) in order to enable node mobility and multi-homing with stable addresses, (2) a scalable, selective update propagation mechanism achieving fast recovery of connectivity even during severe outages that affect many links simultaneously and (3) it does not require any node-level configuration preventing human-based network configuration errors impacting control plane connectivity.

On startup each KIRA node randomly generates its own NodeID from a large (112-bit) ID space. Nodes are then organized in an ID-based overlay, which is inspired by Kademlia [20], using their NodeID, in which they first discover their ID-based neighbors. For routing, KIRA uses this ID-based overlay to discover paths in the link-layer topology. Most traditional overlay networks, such as Kademlia, are built on top of IP and expect universal connectivity between all nodes to be provided already. In contrast to that, KIRA, like other network layer routing protocols (e.g., OSPF, IS-IS), provides this universal connectivity between all nodes itself and does not depend on a lower layer for that. To bridge the gap between the expectation of universal connectivity in overlay networks and the local point-to-point connectivity provided by link-layer protocols, KIRA stores link-layer source routes for each routing table entry. In order to achieve high scalability, KIRA stores only a small amount of routing table entries (also called *contacts*) in each node. The size of KIRA's routing tables scale only logarithmically in terms of number of entries with network size, while keeping packet stretch low. Small routing tables are especially important for resource constrained devices. These characteristics allow KIRA to be used as an autonomic control plane for large-scale networks (100 000+ nodes), providing robust connectivity to control plane applications. An autonomic control plane, such as KIRA, providing robust zero-config control plane connectivity is a necessary foundation for fully autonomic networks and higher-level autonomic behavior. The connectivity KIRA provides does not replace efficient data plane routing protocols like OSPF, but may be used to configure it in a zero-config manner.

KIRA's link-layer source routes and its overlay-like structure allow *KeLLy* to discover all nodes and the majority of their links without having to discover every link individually, thereby reducing overhead.

B. Applying KeLLy to KIRA

The structure and contents of KIRA's routing tables enable a straightforward derivation of the sub-graph $R(v)$ of a node v:

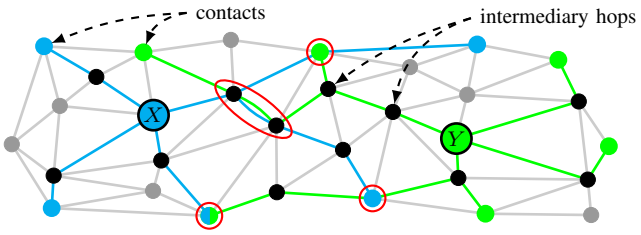


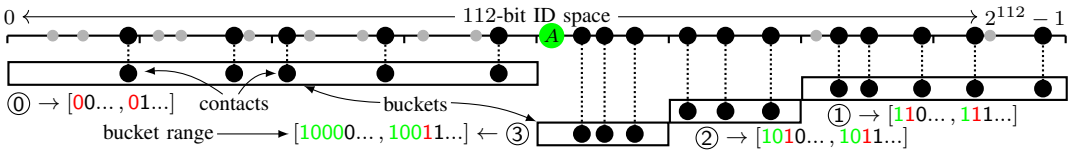
Fig. 1: Topology discovered by two nodes X and Y. Common components $R(X) \cap R(Y)$ are marked by red circles.

$R(v)$ consists of the 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 enables queries targeting a *section* of the ID space instead of existing individual nodes that 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.

C. 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 [20]. 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 in a list of k-buckets each storing up to k contacts at most. Each bucket covers a specific range in 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.



i	lcp	$\lvert range\rvert$
⓪	0	2^{111}
①	1	2^{110}
②	2	2^{109}
③	$[3, 112)$	2^{109}

Fig. 2: A node A (with $ID_A = 10000...$) with its location 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.

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 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 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 to the new last bucket. This ensures that contacts in the direct ID-based neighborhood are not replaced, upholding the invariant that each node knows how to reach its ID-wise neighbors.

D. Bucket Completeness

If a bucket contains all nodes with IDs in its bucket range that are actually present in the network it is *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 [21]. 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.

IV. 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 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, 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. Query Count Lower-bound

As the efficiency of the discovery algorithm is important the number of queried nodes $|Q|$ for discovering all nodes should be kept to a minimum. A first estimation of the lower-bound can be made as follows: All nodes are discovered if each node appears at least once in the discovery range of a queried node. The maximal number of nodes in a discovery range depends on the bucket size k. Each complete bucket can

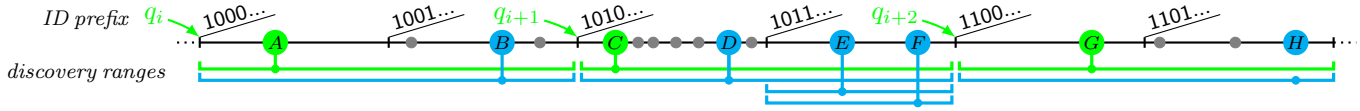


Fig. 3: A section of the ID space with nodes placed according to their NodeID. For selected nodes (A-H) their respective discovery ranges are shown. Queries q_i in the ID space and the respective responding nodes with their discovery ranges are colored green.

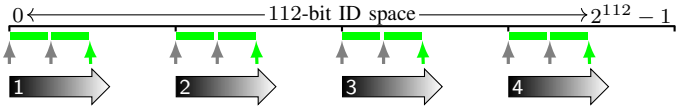


Fig. 4: *Discovery ranges* covering parts of the ID space of an in progress *KeLLy* algorithm with four *ID Space Walks* (1–4) in parallel.

contain up to $k-1$ nodes. With the two last buckets defining the discovery range one can discover a maximum of $2k-2$ nodes per query. If all discovery ranges contain this number of nodes the lower limit for the number queries $|Q|$ in relation to the overall number of nodes $n = |V|$ in the network is:

$$\text{query\_ratio} = \frac{|Q|}{n} = \frac{n}{2k-2} \cdot \frac{1}{n} = \frac{1}{2k-2} \approx \Omega(\frac{1}{2k}) \quad (1)$$

The query_ratio depends on k only and not on n. An example: to find all nodes in network with $n = 10\,000$ nodes and a bucket size of $k = 20$, at least 250 or 2.5% of nodes need to be queried. Under the assumption that the last two buckets are about half full directly after a bucket split (see section III-C) a query ratio between 2.5% and 5% is a realistic estimation.

C. ID Space Walk

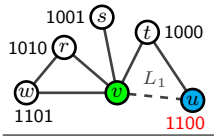
Now a sequential algorithm can be built using a structured approach of *walking* across the ID space: starting from an ID a, which is queried first, the approach waits for the query response to return the routing table information of the node closest to the queried ID and calculates its discovery range. This is achieved by analyzing which buckets were not full at the queried node and combining their respective bucket ranges into a single discovery range $[x, y]$. Next, it queries the next ID $(y+1)$ following the end of the discovery range. This is repeated until the ID space walk reaches an ID $b \leq y$. At this point all nodes in the range $[a, b]$ have been discovered. This algorithm minimizes the number of required queries as no discovery range is queried twice. 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 F, as it has a longer common prefix with the *query ID* and therefore a smaller distance based on the XOR metric.

This algorithm can be run in parallel over disjoint sections of the ID space that can be determined by slicing it (see Figure 4). For maximum parallelization each section should require only as few as possible sequential queries. On the other hand if the sections are too small, i.e. smaller than the discovery ranges, multiple queries will return redundant information. To prevent that, the discovering node can estimate the required amount of sections by calculating the discovery range of its own routing table and rounding down the number of sections to the next power of two. This way each section only requires an ID space walk with a few sequential queries. Using this estimation the discovering node can run an independent ID space walk for each section. When all of them reach the end of their respective sections the whole ID space has been covered by discovery ranges and therefore all nodes are discovered.

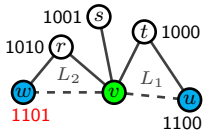
D. Completing the Link Set

With the previously presented ID Space Walk algorithm *KeLLy* can guarantee to discover *all nodes* (i.e., $V' = V$), but it cannot guarantee that *all links* are discovered. If the discovered link set E' is not yet complete, a complementary link calculation algorithm can calculate undiscovered links by using minor enhancements to KIRA. This link calculation algorithm requires knowledge of two parameters for each node v: the *node degree* $D_G(v)$ (number of directly connected nodes, the *physical neighbors* – PNs) and a *physical neighbor ID sum* $\Sigma_{\oplus}PN_G(v)$. The PN ID sum of a node v is calculated by using the set of its physical neighbors $PN(v)$ and calculating the XOR sum over all of them: $\Sigma_{\oplus}PN_G(v) := \bigoplus_{i=0}^{D_G(v)} PN_i(v)$. Each node can calculate and provide both parameters on its own. The second step is to make these parameters available to *KeLLy*. This can be achieved with a minor enhancement to a KIRA node's join process. In KIRA a node repeatedly checks for new nodes in its ID-wise neighborhood by sending a FindNodeRequest that is routed to the ID-wise closest neighbor. The parameters can be disseminated to all nodes in an ID-wise neighborhood by enhancing the response to contain a list of the closest contacts, with both their node's degree and the PN ID sum for each of them. This makes it easy for *KeLLy* to employ them: Nodes receiving a *KeLLy* query include both parameters for all contacts in their last two buckets in addition to the source routing path. With these simple modifications, after the initial *KeLLy* run the parameters are available to the discovering node for each node in the network.

These parameters in combination with the previously discovered topology allow the discovering node to easily *calculate* some of the not yet discovered links. By checking whether the node degree $D_G(v)$ discovered by *KeLLy* matches the node degree $D_{G'}(v)$ in the incomplete discovered topology graph, nodes with undiscovered links can easily be identified. The simplest case (see fig. 5) for link calculation is a node that has exactly one undiscovered link. For such a node an *expected* PN ID sum ($\Sigma_{\oplus}PN_{G'}(v)$) can be calculated using the physical neighbors present in the discovered graph G'. The difference between the expected PN ID sum and the *real* PN ID sum ($\Sigma_{\oplus}PN_G(v)$), calculated by the node itself,



real degree	$D_G(v)$	5
degree in G'	$D_{G'}(v)$	4
real PN ID sum	$\Sigma_{\oplus}PN_G(v)$	1010
expected PN ID sum	$\Sigma_{\oplus}PN_{G'}(v)$	0110
PN ID sum difference	$\Delta\Sigma_{\oplus}PN(v)$	1100



real degree	$D_G(v)$	5
degree in G'	$D_{G'}(v)$	3
real PN ID sum	$\Sigma_{\oplus}PN_G(v)$	1010
expected PN ID sum	$\Sigma_{\oplus}PN_{G'}(v)$	1011
PN ID sum difference	$\Delta\Sigma_{\oplus}PN(v)$	0001
potential neighbor u	$\Delta\Sigma_{\oplus}PN(v) \oplus ID(u)$	1101

Fig. 5: Two link calculation examples with 4-bit NodeIDs showing an excerpt of a topology with a node v and one undiscovered Link L (top) and two undiscovered links L_1 and L_2 (bottom) respectively. Table with parameters of node v on the right of each example.

$\Delta\Sigma_{\oplus}PN(v) = \Sigma_{\oplus}PN_G(v) \oplus \Sigma_{\oplus}PN_{G'}(v)$ is the NodeID of the missing physical neighbor. The undiscovered link is therefore $L = (v, \Delta\Sigma_{\oplus}PN(v))$. The top of fig. 5 shows an example of this case with a node v and an adjacent undiscovered Link $L = (u, v)$. For simplicity the example uses 4 bit NodeIDs indicated next to each node. On the right side the figure details the *real* and *expected* parameters of node v. The *real* PN sum is calculated by v itself over the IDs of all physical neighbors ($r - u$ and w) while the expected PN sum is calculated only over the physical neighbors present in G' ($r - t$ and w). An XOR of both eliminates the IDs of nodes $r - t$ and w as they appear twice, leaving only the ID of u.

For a node v with two undiscovered links another approach is necessary, because the PN sum difference consists not of a single NodeID but of two missing neighbor NodeIDs XOR'ed with each other $\Delta\Sigma_{\oplus}PN(v) = ID(u) \oplus ID(w)$. Both u and w are from the set of incomplete nodes $U = \{x \in V | \Delta D(x) > 0\}$. A naïve solution to this problem is to simply iterate over all NodeIDs of incomplete nodes $ID(u) | u \in U$ and for each of them check whether a matching node w with a NodeID $ID(w) = \Delta\Sigma_{\oplus}PN(v) \oplus ID(u)$ is also in U. If that is the case u and w are the two missing neighbors. Collisions are highly unlikely, due to KIRA's large NodeIDs. This brute force approach has to check $|U|$ potential neighbors for each node with two undiscovered links, i.e., its complexity is $\mathcal{O}(|U|)$ per node or $\mathcal{O}(|U|^2)$ for all nodes. This does not scale well when the set of nodes with undiscovered links U is too large. Instead of iterating over all incomplete nodes, this approach can be used on a reduced set of nodes within 2-hop distance (in G') to v. Thus, if one of the two missing neighbors is within 2-hop distance it can still be calculated. This works well because of two reasons: First, real network topologies often contain triangles (e.g., u, v and t in fig. 5) in clusters of nodes and second, if the undiscovered link would provide an efficient shortcut (for a large distance in G') it should have appeared in KIRA routing tables more often – KIRA prefers shorter paths to its contacts – and would therefore have been discovered by *KeLLy* already. The set of potential neighbors gets significantly smaller by using this approach. The second example in fig. 5 shows the same scenario with two undiscovered links L_1 and L_2. The updated metrics table on the right shows that when using u as a potential neighbor the matching NodeID $\Delta\Sigma_{\oplus}PN(v) \oplus ID(u)$ is equal to the NodeID of the second missing neighbor u.

Algorithm 1 Link Set Completion

INPUT: $G' := (V', E')$, D_G, $\Sigma_{\oplus}PN_G$

```
PriorityQueue pq ← {}
for v ∈ {V' | ΔD(v) > 0} do
    pq.add(priority=ΔD(v), item=v)
while (ΔD(v), v) in pq do
    ΔΣ⊕PN(v) ← Σ⊕PN_G(v) ⊕ Σ⊕PN_G'(v)
    switch ΔD(v) do
        case 1                                  ▷ 1 undiscovered links
            u ← ΔΣ⊕PN(v)
            E' ← E' ∪ {(v, u)}
            pq.update(priority=ΔD(u), item=u)
        case 2                                  ▷ 2 undiscovered links
            // Get potential missing neighbors in 2 hops
            n ← incompleteNodesWithinTwoHops(G', v)
            for u in n do
                w ← ID(u) ⊕ ΔΣ⊕PN(v)
                if w in V and ΔD(w) > 0 then
                    E' ← E' ∪ {(v, u), (v, w)}
                    pq.update(priority=ΔD(u), item=u)
                    pq.update(priority=ΔD(w), item=w)
                    break
        case > 2                                ▷ No calculation possible
            break
return G' := (V', E')
```

These two approaches for nodes with one or two undiscovered links can be combined into an iterative algorithm (see algorithm 1). The input for this algorithm is the graph G' and the node degree and PN ID sum parameters discovered by *KeLLy* (D_G and $\Sigma_{\oplus}PN_G$). Using these the algorithm first traverses the list of all nodes identifying the ones with undiscovered links. Each such node is inserted in a priority queue (pq) using the number of undiscovered links $\Delta D(v) = D_G(v) - D_{G'}(v)$ as priority. Following that the algorithm keeps taking items from this priority queue and applies the two approaches above until either the queue is empty or it only contains items with more than two undiscovered links $\Delta D(v) > 2$. With each successfully calculated link an update to the priority queue is required. If a link (u, v) is calculated when working on node u the number of missing links of node v also decreases by one. This may result in node v now also having one or two missing links only. The priority queue allows to first work on the *easier* nodes with only one undiscovered link, before working on the *harder* to calculate nodes with two undiscovered links.

With this simple algorithm over 99% of the links are discovered in large networks, which should be enough for most use cases. For use cases like traffic engineering where every link may be interesting a second pass over nodes with two undiscovered links using the brute force approach discovers the remaining links in nearly all scenarios. Alternatively, a simple *re-query* strategy can be employed: Sending additional *KeLLy* queries, e.g., in tranches of $0.25\% \cdot n$, targeting nodes

with a high number of undiscovered links, running link calculation in-between reaches 100% with lower compute overhead.

E. Incremental Updates

To integrate topological updates caused by dynamic events in an incremental fashion another simple mechanism can be employed. The main problem is to transfer the information about a failed link (or node) from a physical neighbor detecting the failure to the node discovering and subsequently monitoring the topology. Since not every network node is queried during the initial *KeLLy* discovery, most will not know which node to send the update to. However, due to the structured way of walking the ID space each node knows that the *first* node q in its own discovery range was queried during the initial discovery. This node then knows which node(s) are currently monitoring for topological changes. Therefore, a topology update can be sent to monitoring nodes using the node q as an intermediary target. A single link failure (or new link) will thus trigger two update messages (as both adjacent nodes detect it) to each monitoring node. Preliminary evaluations show that this concept works and achieves low overhead and updates the topology graph in monitoring nodes quickly.

V. Evaluation

The evaluation of *KeLLy* is based on a KIRA implementation using the OMNeT++ simulation framework. In each configuration the simulation is repeated 10 times with different seeds to show the deviation of the results. Figures in the following sections show the mean result and the 2σ deviation across the runs as error bars. Even though *KeLLy*'s contribution is mostly qualitative, i.e., supporting autonomic networks, we also compare it to other similar active approaches (SDN/SNMP) in terms of resource usage. The following key metrics, among others, are investigated:

- *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*: The percentage of links discovered $|E'|/|E|$ is important, as it identifies the accuracy of the discovered graph G' compared to the physical topology. *KeLLy* achieves different link coverages in different stages and scenarios.
- *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.
- *Discovery Time*: The duration of the discovery process is another important metric. For KeLLy this is can be measured in number of query round-trip times.

A. Scalability

Overall, the presented evaluations show that *KeLLy* works well even for very large topologies while inducing acceptable overhead. Networks such as the Internet follow a power law node-degree distribution [22]. Therefore, to evaluate the influence of the network size on the efficiency of *KeLLy*,

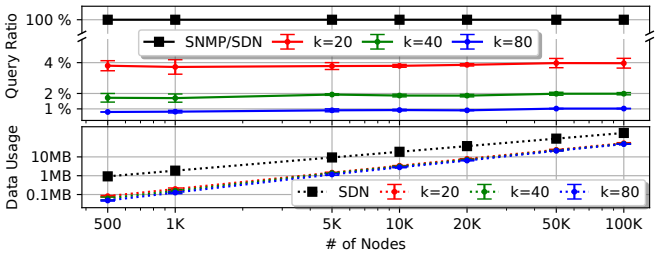


Fig. 6: Query ratio and data usage in power law topologies

simulations in differently sized, randomly generated power law topologies are compared. The topologies were generated using the algorithm introduced by Holme and Kim [23] 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 6 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. In comparison an SDN or SNMP-based approach needs to contact 100% of nodes. With 100 000 nodes an SDN-based approach would actually send 700 000 messages, one from the controller to each switch and two for each link back, while *KeLLy* induces only 8000 messages (4 000 queries and responses). Additionally, the figure shows the transmitted data for each configuration. As the routing tables grow larger 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 14 Byte. Interestingly, the overhead is the same irrespective of k: the larger routing tables with higher bucket size k balance out with the lower number of queries. Practical implementations could limit the number of parallel queries and use pacing to evenly distribute data arriving in the respective responses over (a longer) time period depending on connectivity of the discovering node. Because the messages sent by SDN-based approaches are very small (267 Byte LLDP packets [8]) the total data usage only comes down to around three times that of *KeLLy*, but each message has to be sent and/or processed individually in the SDN controller.

The top of fig. 7 shows that the link coverage without any link calculation 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*

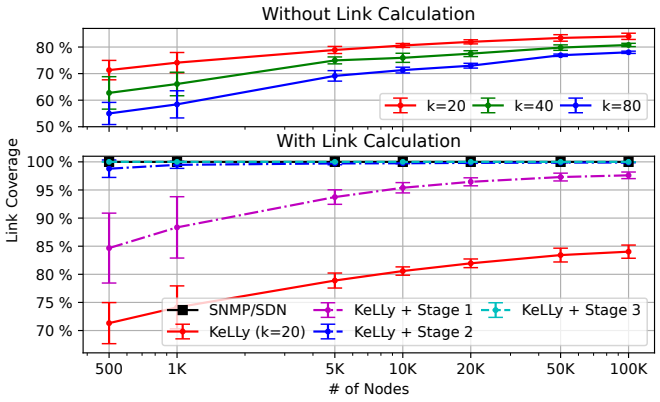


Fig. 7: Link coverage with and without link calculation in power law topologies

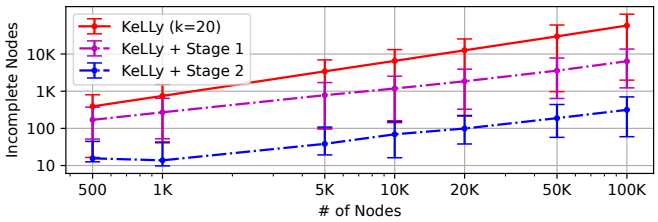


Fig. 8: Incomplete nodes after different stages of link calculation

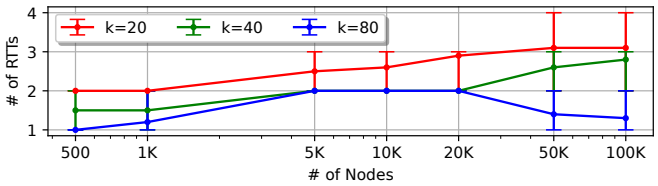


Fig. 9: Discovery time in RTTs (*mean* whiskers *min/max*)

point, because all discovered source routing paths originate from the queried node. Therefore, a lower query ratio (when using 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.

The link calculation algorithm that can be used by any node using *KeLLy*, allows to invest compute power to increase the link coverage of the discovered network topology. To evaluate this, we run the algorithm on the graphs discovered by *KeLLy*. In order to check different compute-performance trade-offs we ran different versions of the algorithm. The first, least compute intensive stage, only uses nodes with a single undiscovered link. In the second stage we additionally use the approach for nodes with two undiscovered links using the reduced node set for potential neighbors. In the third stage the "brute force" approach uses all node pairs of the remaining incomplete nodes to discover the remaining links adjacent to nodes with two undiscovered links. The bottom of fig. 7 shows the link coverage for *KeLLy* without link calculation and $k = 20$ and each of the three stages. One can see that the first stage brings significant improvements already reaching 97% link coverage for 100 000 nodes. With the second stage there is another significant improvement leading to more than 99% of links being discovered. In smaller topologies similar improvements are achieved, but because *KeLLy* itself reaches lower link coverages the results are overall lower compared to larger topologies. Using the third stage with the brute force approach all topology sizes reach 100% link coverage. The additionally transmitted parameters for link completion do not significantly impact the total data usage of *KeLLy*. This is because the parameters are only transmitted once per node in the network, leading to an additional 1.6 MBs of data for 100 000 nodes (14 Byte neighbor sum + 2 Byte node degree).

To give an estimation of the compute overhead of the final most compute intensive third stage using the brute force approach fig. 8 shows the number of incomplete nodes, adjacent to undiscovered links, with increasing topology size after stage one and two of the link calculation. This number grows linearly with topology size. The third stage of complexity $\mathcal{O}(|U|^2)$ needs to check 350 potential neighbors for each of the 350 incomplete nodes in the 100 000 node scenario (if all of them have two missing links). Given that a node running *KeLLy* in such a large network has to have the resources to store a graph with that many nodes, it should also be able to invest that amount of computing power. Otherwise, it always has the option to send a few additional *KeLLy* queries to fill in the remaining gaps. This would on average take 350 additional queries for a total of ≈ 4300 or a 4.3% query ratio.

B. Discovery Time

Another important metric for a topology discovery algorithm is how long the discovery process takes until it produces a usable result. For *KeLLy*, this largely depends on the number of iterations needed. When using multiple parallel ID space walks the number of iterations is determined by the maximum number of iterations over all ID space walks. As each iteration corresponds to a query and response in the physical topology and under the assumption that there is no bandwidth bottleneck (which is likely due to *KeLLy*'s low data usage) the time needed for single iteration is bounded by the maximum query round-trip time (RTT). Figure 9 shows *KeLLy*'s discovery time in RTTs. The results show the estimation of the optimal parallelization factor, i.e., the number of parallel ID space walks, is very accurate leading to very few purely sequential queries even for very large networks. Even for a large RTT of 500 ms (worst-case Frankfurt–Auckland ping), *KeLLy* discovers a topology within a few seconds ($0.5\,\mathrm{s} \cdot 4 = 2\,\mathrm{s}$) requiring only four or less iterations irrespective of the network size.

C. Topological Versatility

One of KIRA's advantages is that it works well across a wide range of topologies, due to its topology-independent NodeIDs and overlay-based routing concepts. It is to be expected, that *KeLLy* therefore works equally well independent of the network's topology. In order to confirm that, *KeLLy* was run on various synthetic topologies of the same size with different characteristics. The topologies include a Holme-Kim power law graph [23], small world graphs (Watts–Strogatz graphs [24] with different values for p), a random and a random geometric graph (each with average node degree 8), a 100×100 grid and a fat tree topology [25] used in data center

networks. The evaluations across these topologies, cf. fig. 10, confirm the aforementioned expectations: the query ratio is the same (4% of nodes) for all evaluated topologies. The discovery overhead on the other hand varies, being higher in topologies with large diameters (Watts–Strogatz graphs with small p, random geometric graphs, and the grid) as this causes the average length of the source routing paths to be larger. *KeLLy*'s overhead is lower than when using SDN-based approaches, except for the sparse grid topology with a much fewer of links.

Furthermore, fig. 11 shows the achieved link coverage over the different topologies with and without link calculation. Interestingly, the link coverage is different for all topologies. The link coverage ranges from 74% in power law topologies all the way up to almost 100% for the grid topology. This can be explained by the number of shortest paths between any two nodes in these topologies: in power law topologies there is often only a low number of shortest paths between any two nodes leading over *hub nodes* with many connections, while in the grid there are numerous paths with the same length because of the Manhattan distance. This lets *KeLLy* discover the same paths repeatedly in the power law topologies while many different paths are discovered in the routing tables in the grid topology. With link calculation all topologies see significant improvements in link coverage. In most topologies the link coverage reaches 100% or very close to it without the compute-intensive third stage. An outlier is the random geometric topology, which reaches only slightly over 90% with the second stage. In this case, not even the brute force approach improves on this. The remaining undiscovered links are in clusters of nodes having three or more undiscovered links each. When employing the *re-query* strategy described in section IV-D the link coverage reaches 100% in the random geometric topology as shown in the figure. During this re-query period an additional 3.5% – for a total of 7.3% – of nodes were queried. In all other topologies the link calculation was enough on its own and no re-querying was necessary.

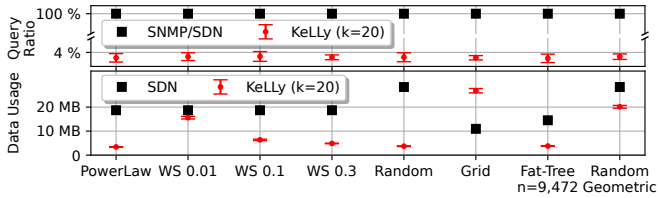


Fig. 10: Query ratio and data usage in different topologies with 10 000 nodes. WS=Watts–Strogatz graph

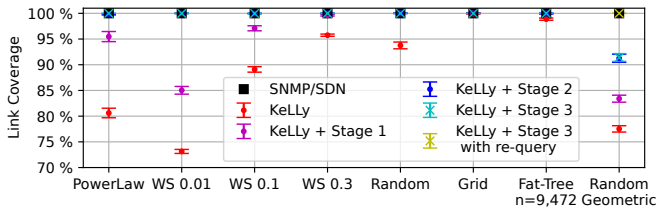


Fig. 11: Link coverage with link calculation in different topologies

VI. Conclusion & Future Work

Emerging autonomic network management solutions dealing with the increasing dynamics and complexity of network infrastructures may highly profit from actual topological information. Providing this information should also be done in an autonomic manner. With *KeLLy* we introduced the first topology discovery algorithm combining efficiency, scalability and autonomy. It achieves low message overhead (querying only 4% of nodes) by re-using distributed routing information. Furthermore, *KeLLy* guarantees to discover all nodes and when investing minimal amounts computing power also all links. In additon to this work's evaluations based on network simulation, future work may include evaluations of a real implementation using network emulation or testbed deployment.

References

[1] E. Coronado *et al.*, "Zero touch management: A survey of network automation solutions for 5G and 6G networks," *IEEE Communications Surveys & Tutorials*, vol. 24, no. 4, 2022.

[2] K. Dzeparoska *et al.*, "Towards a self-driving management system for the automated realization of intents," *IEEE Access*, vol. 9, 2021.

[3] R. L. Aguiar, D. Bourse *et al.*, "NetworldEurope Strategic Research and Innovation Agenda 2022," Dec. 2022.

[4] M. H. Behringer *et al.*, "A Reference Model for Autonomic Networking," RFC 8993, May 2021.

[5] T. Eckert (Ed.), M. Behringer (Ed.), and S. Bjarnason, "An Autonomic Control Plane (ACP)," RFC 8994, May 2021.

[6] R. Bless *et al.*, "KIRA: Distributed Scalable ID-based Routing with Fast Forwarding," in *IFIP Networking*, 2022, pp. 1–9.

[7] P. Seehofer, R. Bless, and M. Zitterbart, "KeLLy: Scalable, Efficient Link-Layer Topology Discovery," in *IEEE/IFIP NOMS*, 2023.

[8] F. Pakzad *et al.*, "Efficient topology discovery in OpenFlow-based Software Defined Networks," *Computer Communications*, vol. 77, 2016.

[9] S. Khan *et al.*, "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 *et al.*, "Tedp: An enhanced topology discovery service for software-defined networking," *IEEE Comm. Letters*, vol. 22, no. 8, 2018.

[11] M. F. Rabbi Ur Rashid *et al.*, "Combining SPF and source routing for an efficient probing solution in IPv6 topology discovery," in *GIIS*, 2014.

[12] M. Li *et al.*, "IPv6 network topology discovery method based on novel graph mapping algorithms," in *IEEE ISCC*, 2013.

[13] K. Vermeulen *et al.*, "Diamond-Miner: Comprehensive Discovery of the Internet's Topology Diamonds," in *17th USENIX NSDI*, 2020.

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

[15] Y. Breitbart *et al.*, "Topology discovery in heterogeneous ip networks," in *Proceedings IEEE INFOCOM 2000*, vol. 1, 2000.

[16] P. Ghaderi, J. W. Atwood, and L. Narayanan, "Topology Discovery in Autonomic Networks," in *IEEE/IFIP NOMS*, 2023.

[17] A. D. Ferguson, S. Gribble *et al.*, "Orion: Google's software-defined networking control plane," in *18th USENIX NSDI 21*, 2021.

[18] A. Lindem *et al.*, "OSPFv3 Autoconfiguration," RFC 7503, 2015.

[19] M. Pritikin *et al.*, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)," RFC 8995, 2021.

[20] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer Information System based on the XOR Metric," in *IPTPS*, 2002.

[21] 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

[22] D. Krioukov *et al.*, "On compact routing for the internet," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 3, p. 41–52, jul 2007.

[23] P. Holme and B. J. Kim, "Growing scale-free networks with tunable clustering," *Phys. Rev. E*, vol. 65, p. 026107, Jan 2002.

[24] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, vol. 393, no. 6684, pp. 440–442, Jun 1998.

[25] M. Al-Fares *et al.*, "A scalable, commodity data center network architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, aug 2008.