BottleNet: Hiding Network Bottlenecks Using SDN-Based Topology Deception

Jinwoo Kim, Jaehyun Nam, Suyeol Lee, Vinod Yegneswaran, Phillip Porras, and Seungwon Shin, *Member, IEEE*

***Abstract*—The robustness of a network's connectivity to other networks is often highly dependent on a few critical nodes and links that tie the network to the larger topology. The failure or degradation to such network bottlenecks can result in outages that may propagate throughout the network. Unfortunately, the presence of the bottlenecks also offers opportunities for targeted *link flooding attacks (LFAs)*. Researchers have proposed a new and promising defense to counter LFAs, referred to as *topology deception*. This strategy centers on hindering the discovery of bottlenecks by presenting false trace responses to adversaries as they perform topological probing of the target network. Even though the goal of topology deception centers on obscuring critical links, node dependencies can be exploited by an adversary. However, current approaches do not consider a wide range of metrics that may reveal important and diverse aspects of network bottlenecks. Furthermore, existing approaches create a simple form of virtual topology, which is subject to relatively easy detection by the adversary, reducing its effectiveness. In this paper, we propose a comprehensive topology deception framework, which we refer to as BottleNet. Our suggested approach can analyze various network topology features both with respect to static and dynamic metrics and then use this information to identify bottlenecks, finally producing complex virtual topologies that are resilient to adversarial detection.**

***Index Terms*—Link flooding attacks (LFAs), topology deception, network robustness, software-defined networking (SDN).**

I. Introduction

WHEREAS the Internet has evolved to be a highly robust communication infrastructure, researchers have also noted its tendency to exhibit topological properties in which node connectivity is by no means evenly distributed. Rather, the topology of Internet segments, regardless of scale, tends to incorporate a few high-degree nodes that facilitate most of the connectivity within the segment's network graph [1]. Unfortunately, from an adversarial perspective, these high-degree nodes represent strategic bottleneck opportunities for disrupting the flow of traffic to all nodes and links (e.g., LFAs) that lay within their connectivity path. Recognition of this concern has not only been raised in academia [2]–[7], but its impact has also been observed in real-world incidents [8], [9].

To address this issue, a variety of *topology deception* systems have been proposed to reduce the likelihood that such network bottlenecks will be discovered and exploited. When an adversary attempts to scan a network, the deception systems aim to hide network bottlenecks, exposing an alternate *virtual topology*. For example, NetHide [10] and Trassare *et al.* [11] manipulate TTLs (time to live) of probing packets to create virtual links, and LinkBait [12] and SPIFFY [13] reroute probing packets to other links to prevent adversaries from collecting target link information. With such a strategy, bottlenecks can remain obfuscated from an adversary's topology construction process. This concept has been actively discussed in both the military [11], [14] and cyber security communities [10], [12], [15] as a promising defense solution.

However, even though prior studies have explored this mitigation strategy, several important limitations still remain: First, they primarily focused on a single metric (e.g., the number of routing paths [11] or flows [10], [12]) to find network bottlenecks, omitting diverse forms (e.g., link cuts [5], bandwidth bottlenecks [16]) not revealed by the limited visibility. This focus may provide another opportunity that exploits other bottlenecks that have yet to be considered by an adversary. Second, the prior work did not produce complex virtual topologies to deceive the adversary. In part, this was due to how virtual topology was modeled in an ad-hoc manner without consideration of complex topological structures [11], [12], [14], [15] or primarily built for improving the usability of network debugging tools rather than security [10]. Because adversaries can exploit diverse topology features [2]–[4], [6], [17]–[19], a single metric or algorithm is insufficient for finding all bottlenecks and producing secure virtual topologies. In addition, it is possible that an adversary can flood arbitrary nodes and links without believing virtual topologies. However, none of the prior studies considered the *blind LFAs* at all.

In this paper, we aim to answer the following research question; *Can we design a comprehensive system that covers diverse bottlenecks by considering all possible types of bottlenecks and generates complex virtual topologies by using the benefits of the existing deception algorithms?* To this end,

Manuscript received November 5, 2020; revised March 7, 2021 and April 11, 2021; accepted April 19, 2021. Date of publication April 26, 2021; date of current version May 21, 2021. This work was supported by the Institute of Civil Military Technology Cooperation Center (ICMTC) funded by the South Korea Government (Defense Acquisition Program Administration (DAPA) and Ministry of Trade, Industry and Energy (MOTIE)) under Grant 18-CM-SW-09. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Mika Ylianttila. *(Corresponding author: Seungwon Shin.)*

Jinwoo Kim, Suyeol Lee, and Seungwon Shin are with the School of Electrical Engineering, KAIST, Daejeon 34141, South Korea (e-mail: jinwoo.kim@kaist.ac.kr; 95leesu@kaist.ac.kr; claude@kaist.ac.kr).

Jaehyun Nam is with AccuKnox, Cupertino, CA 95014 USA (e-mail: jn@accuknox.com).

Vinod Yegneswaran and Phillip Porras are with SRI International, Menlo Park, CA 94025 USA (e-mail: vinod@csl.sri.com; porras@csl.sri.com).

Digital Object Identifier 10.1109/TIFS.2021.3075845

we present BottleNet, a comprehensive topology deception framework that provides a wide range of bottleneck metrics and topology deception algorithms, facilitating the building and deployment of more secure virtual topologies.

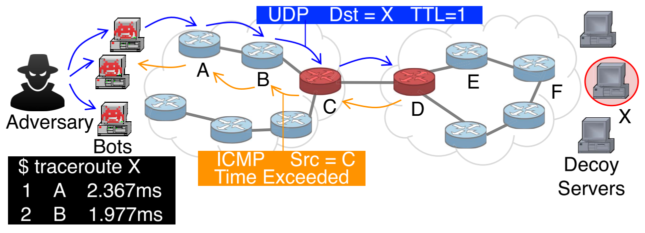
In designing BottleNet, we address four key challenges: First, it is an open question regarding what metrics should be monitored, considering that many factors determine network bottlenecks. Second, given diverse bottleneck types, it is difficult to determine which nodes and links are most vulnerable, such that their failure would severely impact the robustness of the network. Third, designed virtual topologies should be sufficiently complex so that an adversary cannot easily infer real topologies from observed topology snapshots. Finally, the virtual topologies should be realized as a practical solution that can be rapidly deployed in production networks without imposing expensive costs on network operators.

To address the first problem, BottleNet takes the benefits of global network visibility provided by software-defined networking (SDN) [20]. SDN enables centralized control that facilitates network-wide monitoring [21], [22]. With this ability, we analyze two aspects of bottlenecks: (i) static metrics that are graph features of network topology and (ii) dynamic metrics that are fluctuating features according to realtime network flows. Given these metrics, a network operator can pick and choose criteria for observing desired types of bottlenecks. For the second challenge, we design an integer linear programming optimization scheme that chooses κ-ranked bottlenecks with the help of the MCDA (multi-criteria decision analysis) technique. We also design two graph-based heuristics: (i) a deployment node selection algorithm that chooses the best position to deploy virtual networks given a distance threshold, and (ii) a random virtual topology generation algorithm that synthesizes a scale-free network to generate a complex topology structure for making artificial bottlenecks. For the deployment challenge, we leverage the software switches [23] with SDN. Software switches enable rapid instantiations of virtual switches in general-purpose machines, thus helping us deploy virtual networks rapidly. In addition, SDN provides dynamic flow control to detect and react to probing packets in advance. On the basis of these approaches, BottleNet designs diverse actions to orchestrate/manipulate probing packets to hide network bottlenecks and also react to the adversaries who mount blind LFAs.

In summary, our contributions include the followings:

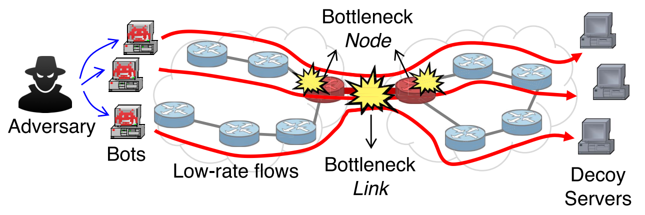
- We introduce a comprehensive set of bottleneck metrics that offer diverse insights of network bottlenecks to a network operator, enabling the discovery of nearly all possible bottlenecks that can be targeted by adversaries.
- We develop diverse heuristics for analyzing a network topology and creating secure virtual topologies using the theory of network graph analysis.
- We implement a prototype of BottleNet on a popular SDN controller, Ryu [24] and evaluate it on real-world network topologies with large-scale simulations and emulations that demonstrate the effectiveness of BottleNet in hiding network bottlenecks.

The rest of the paper is organized as follows: Section II begins with a brief introduction to LFAs and prior topology deception algorithms, and analyzes their limitations with a concrete example scenario. Section III surveys related work pertaining to LFAs and their countermeasures. Section IV presents a system overview. Section V introduces our metrics, algorithms, and topology deployment techniques to find/hide bottlenecks. Section VI presents evaluation results conducted with large-scale simulations. Section VII discusses remaining issues of BottleNet. We conclude the paper in Section VIII.



```
$ traceroute X
1   A   2.367ms
2   B   1.977ms
```

(a) Topology Probing: probe a target network using tracing tools.



(b) Bottleneck Flooding: make low-rate attack flows with decoy servers.

Fig. 1. An example scenario of LFAs.

II. Background and Motivation

This section briefly presents the background of LFAs and how previous topology deception algorithms mitigate it with a detailed scenario. Then, we discuss their limitations and introduce our high-level approach.

A. LFAs

LFAs aim to congest important routers and links in (or between) (a) network(s) for the purpose of cutting off as much network connectivity as possible [3], [4]. Fig. 1 illustrates a typical attack scenario that consists of the two phases:

Topology Probing. To select suitable target nodes or links, an adversary first probes a target network using tracing tools such as `traceroute` [25], which exploits the TTL field of an IP header to discover a path from a source to destination. When the adversary sends low-TTL packets to the destination, TTL values gradually decrement per router, and then TTL = 0 packets elicit probing responses (i.e., ICMP *Time_Exceeded*) of the intermediate routers, enabling the adversary to identify IP-level nodes and links for each path. From the result, the adversary analyzes flow density—the number of flows that pass through a node and link. A high flow density indicates that such nodes and links can be bottlenecks because they are involved in many network flows.

Bottleneck Flooding. Based on the obtained topology information, the adversary generates attack flows that flood the target bottleneck nodes or links by employing a large number of distributed bots. In contrast to traditional DDoS attacks, those flows are essentially difficult to detect because bots use *low-rate* flows, which is indistinguishable from other benign traffic [3]. To produce such flows, the adversary maintains benign sessions with *decoy servers* that are responsible for receiving the attack flows. For this purpose, well-known web/DNS servers are typically used as decoy servers because they are publicly accessible from the Internet [3].

B. Topology Deception

Topology deception is a proactive defense to prevent the topology probing that aims to discover bottleneck nodes and links. It is achieved by deploying a *virtual topology*, the structure of which differs from that of a physical topology by manipulating a view of adversaries [10]–[12], [14], [15]. Below, we describe how they work from the perspective of network operators:

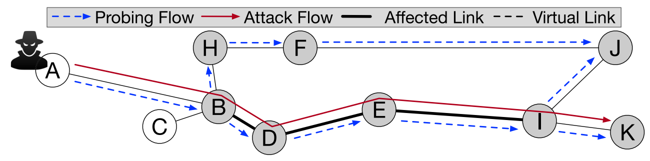
1) Identifying potential bottlenecks. The network operator first identifies which parts (e.g., nodes or links) of a network should be hidden from adversaries. To find potential targets, prior studies mostly leverage simple graph-based metrics such as betweenness centrality (i.e., the number of shortest paths for nodes [11]) or flow density (i.e., the number of network flows for links [10], [12]). Those metrics reveal important nodes and links that have significantly more connectivity than others in a network. If certain nodes and links have high values, the network operator determines them as potential bottlenecks.

2) Designing a virtual topology. The network operator designs a virtual topology that can effectively hide the potential bottlenecks. Prior work commonly used a *graph* to model the virtual topology. Its goal is to make a dissimilar virtual topology from the original physical topology, preventing adversaries from inferring the original graph structure. Hence, most topology deception algorithms focus on the manipulation of the graph structure (see §II-C for more details).

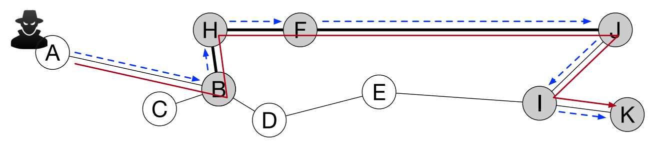
3) Deploying the virtual topology. Finally, the network operator deploys deception logics to network devices to implement the virtual topology. For this, network devices (i.e., routers and switches) should support packet header modifications [10] or arbitrary packet generation [11] to display virtual nodes and links in the tracing tools. In addition, network devices often need to alter their deception logics when the physical topology changes or a certain virtual topology is excessively exposed. However, these approaches are limited to the legacy devices because its control-plane intelligence highly depends on the operator's manual configurations. For this reason, previous work leveraged programmable network devices (e.g., OpenFlow [12], [15], P4 [10]) that can flexibly reroute or modify probing packets according to deception logics. In addition, NFV (network function virtualization) or software switches (e.g., OpenvSwitch) have been used [14], [15] as well to instantiate virtual networks in the data plane.

C. A Motivating Example and Challenges

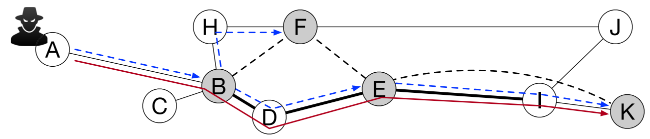
Fig. 2a presents an example scenario, where the core links D–E and E–I are vulnerable to LFAs because they are involved in many network flows—high flow density. To discover those bottlenecks, the adversary attempts to scan the target network



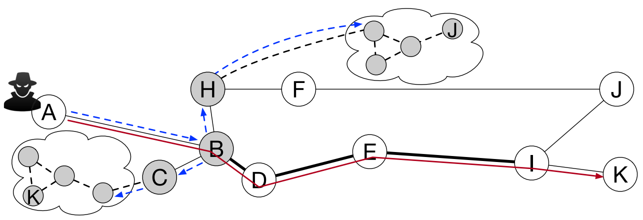
(a) A physical topology that consists of 10 routers and traces of the adversary's probing/attack flows when no defense is deployed.



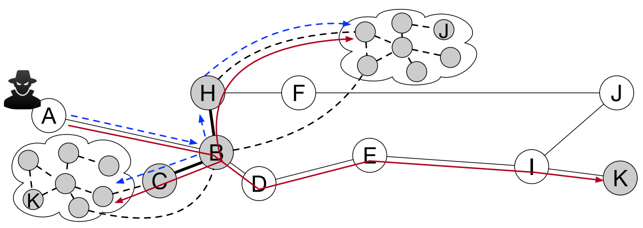
(b) The FR-based approach [12], [13] redirects probing packets to other physical nodes and links (i.e., H, F).



(c) The ON-based approach [10], [11] adds virtual links denoted (i.e., dashed lines) with by manipulating packet headers.



(d) The VN-based approach (i.e., [14], [15]) redirects probing flows to virtual networks (i.e., clouds).



(e) BOTTLENET leverages all the techniques to create more complex virtual topologies and is also able to reroute attack flows to mitigate LFAs.

Fig. 2. A motivating example to compare generated virtual topologies by different approaches. The blue arrows indicate the physical paths of probing flows. The red arrows denote the path of attack flows. The thick lines denote affected links that can be flooded if the adversary mounts LFAs. The grey nodes represent visible nodes to the adversary.

from the node A to K and J. Below, we show how three different topology deception algorithms work:

- *Flow rerouting (FR)-based approach* [12], [13] focuses on redirection of probing flows to other physical nodes and links. This strategy forces an adversary to believe that other nodes and links are bottlenecks. As shown in Fig. 2b, the adversary's flows are rerouted to the link H–F and F–J; hence, she can be deceived that those links are bottlenecks, not the links D–E and E–I.
- *Overlay network (ON)-based approach* [10], [11] aims at creating virtual (overlay) links layered upon a physical network by manipulating probing packets. For example, in Fig. 2c, packet headers such as TTLs and IP addresses are modified so that packets are not expired on some

nodes (e.g., H, D, I). This creates virtual links B–F, F–E, and E–K (e.g., dashed lines).

- *Virtual network (VN)-based approach* [14], [15] aims to add virtual nodes and links in the physical topology to make the topology search space larger. In Fig. 2d, the virtual networks (i.e., clouds) are deployed to the nodes C and H. In addition, the virtual networks respond with the IP addresses of physical nodes, such as K and J. This approach can force the adversary to construct a significantly different topology view.

With the in-depth analysis of prior studies, we observe that they have the following limitations:

L1. Limited Bottleneck Metrics. Previous work [10]–[12] only uses a single graph property of a network as a metric to determine whether or not a link is a bottleneck. However, we argue that it is not sufficient to find all possible bottlenecks that may exist in practice. In Fig. 2a, whereas the links D–E and E–I are determined to be bottlenecks due to the high number of flows, the links H–F and F–J can be vulnerable if they have low available bandwidth at the current time. However, none of the previous works considered such *dynamic* metrics. If the links H–F and F–J are included in the virtual topology as shown in Fig. 2b, they expose attack points to the adversary who can estimate link bandwidth using measurement tools [26].

L2. Limited Complexity of Virtual Topologies. To make virtual topologies secure, it should be guaranteed that adversaries cannot easily infer physical topologies. However, we observed that existing algorithms are limited in creating secure topologies. For example, in Fig. 2b, possible graph operations of the ON-based approach are to add virtual links and remove existing physical nodes. However, they reduce the possible number of nodes and links that adversaries will discover, consequently leading to the creation of a limited range of virtual topologies. From our theoretical analysis, the maximum number of possible virtual topology instances is $O(2^{\frac{n(n-1)}{2}})$ in the ON-based approach, which is bound to the number of physical nodes n. Furthermore, adversaries can find commonly observed nodes and links from many topology snapshots. On the other hand, the VN-based approach [14], [15] did not consider the complex structure of virtual networks.

L3. Lack of Defense Against Blind LFAs. Existing FR- and ON-based approaches did not control any production traffic to protect bottleneck links. For example, in Fig. 2c, if an adversary sends attack flows that are highly similar to benign traffic from A to K, they pass through the bottleneck links D–E, E–I. Unfortunately, if the adversary knows this routing policy, they can mount blind LFAs even though they do not know exact topology information. For instance, they can send a large volume of attack flows aimed to target virtual links B–F and F–E, and then the underlying core links B–D, D–E, and E–I are flooded. Whereas the VN-based approach focused on redirecting probing flows to isolated network space, none of the previous work utilized VNs to mitigate attack flows.

D. Our Approach

BOTTLENET addresses the limitations with three high-level approaches. To address L1, BOTTLENET provides richer metrics from both aspects of bottlenecks: topological and performance metrics. For example, if they want to consider both the number of network flows and available link bandwidth as criteria, BOTTLENET then makes an optimized decision to cover all potential bottleneck nodes and links that display a high possibility of being targets. To address L2, BOTTLENET combines existing deception algorithms to make more complex virtual topologies with proactive and reactive ways: we proactively analyze a target network and generate complex virtual networks, the structure of which is scale-free [27], and we reactively enforce rules that dynamically manipulate routes of probing packets and its headers. For example, in Fig. 2e, BOTTLENET redirects probing packets to VNs so as to hide the intermediate links D–E and E–I. In addition, BOTTLENET modifies packet headers to make overlay links from node B to the VNs. As a result, the complexity of those networks becomes larger than the prior VN-based approaches [14], [15]. To address L3, inspired from the traditional virtual honeypot concept [28], BOTTLENET quarantines flows to virtual networks when their volumes are likely to congest bottleneck links. It consequently minimizes the impact of attack flows when adversaries mount blind LFAs aimed to flood virtual nodes or links even under the obfuscation.

III. RELATED WORK

LFAs. LFAs have recently garnered significant attention as a new threat to network robustness. In contrast to traditional DDoS attacks, LFAs target intermediate core links to cause significant damages to a large number of victims (see §II-A for more details). The Coremelt attack [4] first demonstrated that it is possible to cause congestion on core links using pair-wise bot-to-bot traffic. The Crossfire attack [3] showed that distributed bots can use publicly accessible servers to make a number of indistinguishable attack flows on bottleneck links. The key insight is that such network bottlenecks actually exist at both intra- and inter-domain network topology regardless of scales, which has been demonstrated by Kang and Gligor [7].

There are multiple defending approaches to mitigate LFAs, and we compared our solutions with them as follows:

Reactive Traffic Engineering. Detecting LFAs is a challenging problem considering that an adversary utilizes legitimate traffic when attacking network bottlenecks. To catch the hidden behavior of LFA attackers, researchers have proposed a variety of reactive approaches. They assume that LFA bots send constant-rate traffic to maximize attack effects in a long time period and may not comply with instructions of upstream ASs. Paying attention to the fact, SPIFFY [13] proposed an SDN-based traffic engineering technique that temporarily expands end-to-end bandwidth. Codef [29] proposed an inter-AS collaboration approach that reroutes traffic on the request of a peer router's request. In the circumstance of those rerouting instructions, bots may not correctly adjust their flow rates/paths, which is evidence of detection. However, reactive countermeasures are limited in that they do not prevent LFA attackers from discovering target bottlenecks.

Topology Deception. Table I shows a comparison that systematically analyzes differences between the existing topology deception and BOTTLENET for supported metrics and

TABLE I
COMPARISON OF BOTTLENET AND PREVIOUS WORK FOR TOPOLOGY DECEPTION

	Purpose	Static Metrics	Dynamic Metrics	Deception Algorithms
[11]	Important node hiding	BC	✘	ON
NetHide [10]	Usable virtual topology construction	FD	✘	ON
RDS [15]	Virtual networks with dynamic features	✘	LL, ALB	VN
[14]	Moving target defense framework	✘	✘	VN
LinkBait [12]	Exposure of bait links	FD	✘	FR
SPIFFY [13]	Emulation of fake bandwidth	✘	✘	FR
BOTTLENET	Diverse metrics & Complex topologies	BC, CC, DC, MC+	ATL, CBR, ALB, LL+	FR, ON, VN

deception algorithms (For a comprehensive list of metrics, readers should refer to §V-A.). Here, we briefly review prior studies according to the aforementioned categories (§II-C):

FR-based approach. LinkBait [12] reroutes flows into other physical links (namely, bait links) that are fake bottleneck links to conceal high flow density links. SPIFFY [13] can also be seen as a topology deception because it emulates fake expanded bandwidth to remove bandwidth bottlenecks from an adversary's view. Whereas the FR-based approach is straightforward in achieving the goal with the help of traffic engineering, they consequently make other physical nodes and links appear as target bottlenecks.

ON-based approach. Trassare *et al.* [11] showed that it is possible to build a fake overlay by manipulating traceroute responses to hide high betweenness centrality nodes. NetHide [10] further advanced the ON-based approach by proposing an optimization algorithm that computes a secure and useful virtual topology to hide high flow density[1] links while maximizing the utility of tracing tools. However, the ON-based approaches are limited in that they can only build a simple virtual topology due to the restricted possible graph operations upon a physical topology.

VN-based approach. RDS (Reconnaissance Deception System) [15] designed a virtual network-based deception system that imitates the performance characteristics of a real network by emulating a virtual link latency and bandwidth. Aydeger *et al.* [14] proposed an SDN/NFV moving target defense system that constructs virtual networks using NFV and orchestrates suspicious flows using SDN. Although their approach is similar to ours, they did not consider the complex topology structure at all.

In summary, none of the existing work considered a complex graph structure and multiple types of metrics, and this limitation leads to the design of insecure virtual topologies.

IV. BOTTLENET OVERVIEW

This section presents our threat model and terminologies, and introduces an overview of the BOTTLENET architecture.

A. Problem Scope

Threat Model. We assume that an adversary seeks to discover network bottlenecks through topology probing for the purpose of executing LFAs on targeted nodes and links [3], [4]. The bottlenecks include important routers and links that are involved in critical network connectivity that can be defined by a variety of factors such as network flows, routing paths, and available bandwidth. As noted earlier, a common technique used to find routing bottlenecks is to gather topology information about network flows and routing paths using topology probing tools such as `traceroute` [25], which has been popularly used for network debugging [30] or large-scale topology measurements [31]. Here, we do not consider inaccuracy issues of traceroute, due to IP aliasing or load balancing [31], [32], because the objective of topology probing is to obtain an approximate topology view for identifying targets [3]. We also assume that the adversary can perform topology probing from both inside and outside a network. Finally, we assume that the adversary can measure end-to-end bandwidth using bandwidth probing tools (e.g., `iPerf [26]`, `PathNeck [16]`), meaning that they can approximately discover bandwidth-constrained links.

Deployment Model. Our target network is a WAN (wide area network) because an adversary performs LFAs typically over a large geographical area [3], [4]. We envision that all switches in the target network are OpenFlow-enabled devices that allow a network operator to directly control and monitor all visible flows (e.g., Google B4 [33]). The operator, who is in charge of a single ISP (internet service provider), may run a (logically) centralized SDN controller to control OpenFlow switches with a number of SDN applications. BOTTLENET can be deployed as a security application upon the controller especially built for preventing bottlenecks from being exposed from adversarial topology scanning because many network operators employ similar solutions to protect their networks (e.g., honeypots [28], network tarpits [34]). Note that BOTTLENET needs periodic network monitoring to obtain global visibility for current network topologies and statistics. Although this approach may raise concerns regarding scalability due to the massive number of collected data, state-of-the-art SDN controllers are designed as distributed platforms that provide high performance, enabling built-in support for monitoring large-scale networks [35], [36]. Finally, we envision that the operator provides deployable host machines (i.e., available nodes) within their networks to deploy virtual networks as a form of VMs or containers (e.g., Planetlab [37]).

B. Notations

In this study, we depict a physical topology by a graph $G = (V, E)$, where V represents the set of physical routers (or switches), and E represents the set of physical links between the routers. We assume that a network operator wants to protect a set of bottleneck nodes denoted by V_b (or bottlenecks links denoted by E_b). They wish to hide V_b from an adversary's topology probing to make the prober

[1]Note that the flow density is a subset of our betweenness centrality.

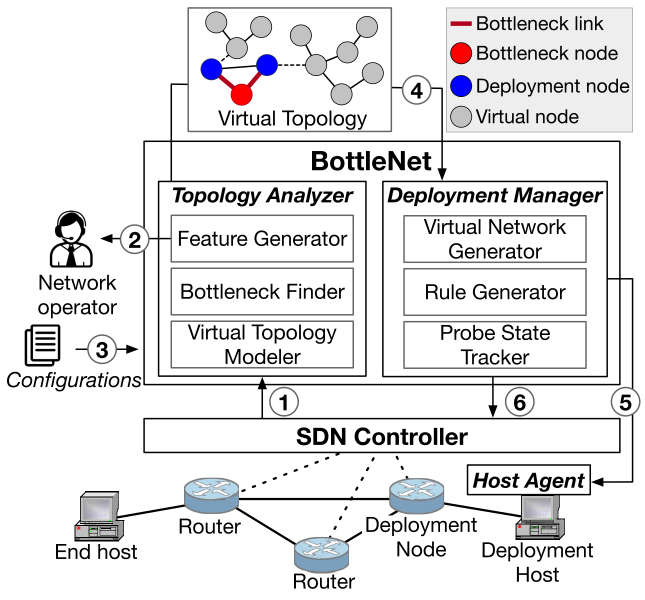


Fig. 3. BOTTLENET system overview.

believe that V_b are not bottlenecks in the physical topology G. Deployment nodes are denoted by V_d, which are selected from available nodes V_a supported by the operator. A virtual network is denoted by G_v that consists of virtual nodes V_v and links E_v. Formally, we state that the goal of generating and deploying the virtual network G_v is to reduce/hide bottleneck metrics of bottleneck nodes V_b (or bottleneck links E_b) and to increase the metrics of virtual nodes V_v (or virtual links E_v).

C. System Workflow

Fig. 3 illustrates BOTTLENET's main components and the overall workflow that operates with the following sequence:

Topology Analyzer. is responsible for analyzing a target network topology and finding potential bottlenecks. ① The *metric generator* periodically collects a current topology snapshot and network statistics for routers and links from an SDN controller. ② Based on the data, it calculates static and dynamic metrics, each of which denotes the topological and performance properties of network bottlenecks, respectively. ③ The network operator then gives BOTTLENET the configuration parameters: the number of bottlenecks κ that they want to protect, the preferred bottleneck metric types M, the available nodes V_a for deploying virtual networks, and the distance threshold τ that denotes allowed hop distances from deployment nodes to bottlenecks. ④ The *bottleneck finder* locates bottlenecks by solving an ILP optimization, and the *virtual topology modeler* designs the *virtual topology* that specifies the location of deployment nodes and the necessary number of virtual nodes and links to hide the target bottlenecks.

Deployment Manager. aims to deploy the virtual topology into a physical network by enforcing necessary rules and network configurations. ⑤ The *virtual network generator* instantiates virtual networks (modeled on the virtual topology) to virtual switches and links with concrete network configurations and deploys them to the deployment hosts. ⑥ The *rule generator* generates detection and redirection rules that monitor probing packets and reroutes/modifies them when deception is required. When potential probing packets are detected, it first reroutes the packets to the deployed virtual networks and reports them to the *probe state tracker* module, which then manages states for the subsequent behavior of probers. If probers subsequently send suspicious traffic (e.g., long-lived TCP flows with a constant rate, such as the Crossfire attack [3]) to virtual networks, the rule generator blocks or quarantines the flows. The probe state tracker also manages known IP addresses of benign probers to allow benign network debugging and topology measurements.

Host Agent. is a lightweight module running on the deployment host attached to the designated deployment node so as to deploy virtual networks. Instructed by the deployment manager with a private control channel, the host agent then realizes the virtual topology with virtual switches that produce probing responses and simulate arbitrary dynamic network properties to hinder the inference of real topologies.

TABLE II
SUPPORTED BOTTLENECK METRICS IN BOTTLENET

Category	Metric (abbr.)	Type	
		Node	Link
Static	Betweenness Centrality (BC)	✓	✓
	Closeness Centrality (CC)	✓	
	Mincut Centrality (MC)		✓
	Degree Centrality (DC)	✓	
Dynamic	Aggregate Traffic Load (ATL)	✓	
	Consumed Bandwidth Ratio (CBR)		✓
	Available Link Bandwidth (ALB)		✓
	Link Latency (LL)		✓

V. SYSTEM COMPONENTS

This section elaborates on metric definition, deception algorithms, and deployment workflows of virtual networks.

A. Bottleneck Metric Definition

Below, we summarize bottleneck metrics currently supported by BOTTLENET (see Table II). In what follows, we denote a single node by u, and denote a direct link from node u to v by (u, v).

1) Static Metrics: One aspect we consider when designing metrics is that bottlenecks result from a fundamental structure of network topologies. This consideration motivates the use of the following 4 graph metrics:

Betweenness Centrality (BC). It denotes the number of times a node or link acts as a bridge along the *shortest paths* between two nodes [38]. If a node or link is included in many routing paths, we consider that they are likely to be bottlenecks. Because network operators typically adopt *policy-based routing*, we use the modified version proposed by Schuchard *et al.* [19] who replaced the shortest paths to policy-based routing paths. We define the modified BC for a node u as $BC(u) = \sum_{s \neq u \neq t \in V} \frac{path_{st}(u)}{path_{st}}$, where $path_{st}$ is the number of routing paths between s and t, and $path_{st}(u)$ is the number of those paths that contain the node u.

Closeness Centrality (CC). It indicates the average *hop distances* of a target node from others. Kang *et al.* showed that routing bottlenecks are commonly discovered around the midpoint hops of routing paths from the large-scale traceroute measurement [7]. Based on this fact, we conjecture that the high closeness centrality nodes are likely to be adjacent to bottleneck nodes and links. The closeness centrality of a node u is defined with $CC(u) = \frac{1}{\sum_{v \in V} d(u,v)}$, where $d(u,v)$ is the geodesic distance between node u and v.

Mincut Centrality (MC). The LFA attackers may seek to minimize the number of target links while cutting as many connections as possible [3]. In this regard, attacking *link cuts* is one of the efficient ways to choose target links minimally [5]. To consider this approach, we design a new centrality metric that captures how often a link is involved in the *s-t mincut*, which is a minimum set of cuts from source s to destination t, for all pairs of nodes. We define the mincut centrality of a link (u,v) with $MC(u,v) = \sum_{s \in V} \sum_{t \neq s \in V} mincut_{st}(u,v)$, where $mincut_{st}(u,v)$ is the number of times a link (u,v) is included in *s-t* mincut between all pairs of nodes.

Degree Centrality (DC). The degree of a node determines the criticality of a node failure in the network, and it has been the most popular indicator used in prior studies for network robustness problems [2], [6], [18]. If a node has a high degree, it indicates that the node has many neighbor nodes; hence the failure of the node can severely affect a significant number of network connections. The degree centrality is defined with $DC(u) = \frac{deg(u)}{\sum_{v \in V} deg(v)}$, where $deg(u)$ is the degree of the node u. Here, we compute a normalized degree of a node u by dividing it by the sum of all node degrees.

2) Dynamic Metrics: BOTTLENET leverages the capability of OpenFlow protocols [39] to extract the following 4 dynamic metrics that locate bandwidth bottlenecks in SDN networks:

Aggregate Traffic Load (ATL). It indicates the sum of the overall incoming traffic load rate that a node u receives. To extract this metric, we leverage the *Aggregate_Flow_Statistics* OpenFlow message. It retrieves the aggregate statistics for active flow entries from an OpenFlow switch. Specifically, we refer to the *byte_count* field that represents the accumulative number of byte counts for alive network flows. BOTTLENET periodically queries it to all switches every t seconds and computes a rate of the change of aggregate byte counts.

Consumed Bandwidth Ratio (CBR). It represents the utilized bandwidth ratio of a link by considering the current bitrate of active flows. It can be computed using the OpenFlow *Port_Statistics* message that retrieves statistical information of all ports from a switch. In particular, we leverage the *tx_bytes* field that denotes the number of transmitted bytes and *duration_sec* that indicates the elapsed time the port has been activated. With these two fields, we can compute $TX_{u,v}(t)$, which is the number of transmitted bytes from node u to v per time t. Finally, we define the consumed bandwidth ratio of a link (u,v) as $CBR(u,v) = \frac{(TX_{u,v}(t_0) - TX_{u,v}(t_1)) \cdot 8}{t_1 - t_0}$, where t_0 and t_1 are the previous and current timestamp, respectively.

Available Link Bandwidth (ALB). It displays how much a link can receive further flows based on the link capacity and used bit rate. It is the most important dynamic metric for determining whether or not a link is a bandwidth bottleneck.

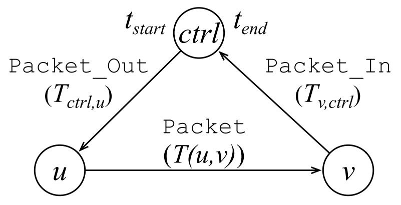


Fig. 4. Measuring a link latency $LL(u,v)$ of a link (u,v).

To obtain this information, we subtract *CBR* from a predefined link capacity, and then we can obtain the available link bandwidth for a link (u,v) as: $ALB(u,v) = C(u,v) - CBR(u,v)$, where $C(u,v)$ denotes the capacity of the link (u,v).

Link Latency (LL). It measures the latency of a link between two switches to know whether the link experiences congestion. Because the OpenFlow protocol does not support such a metric, we leverage the Skowyra *et al.*' latency estimation method [40] as illustrated in Fig. 4. First, the controller denoted by $ctrl$ generates an arbitrary measurement packet and instructs the switch u to send it to switch v with a *Packet_Out* message. The switch v then receives the packet and sends it to the controller with *Packet_In*. The controller then records the time t_{start} that indicates the time of sending the *Packet_Out* and t_{end} that represents the time of receiving the *Packet_In*. If we know $T_{ctrl,u}$ and $T_{v,ctrl}$ that denotes propagation delays of a control channel among $ctrl$, u, and v, we can estimate the link latency from u to v as follows: $LL(u,v) = t_{end} - t_{start} - (T_{ctrl,u} + T_{v,ctrl})$.

B. Topology Analyzer

The goal of the topology analyzer is to find potential bottlenecks and generate a secure virtual topology that cannot be easily inferred by an adversary. We now present an optimization method that chooses the most vulnerable bottleneck nodes and links based on an operator's inputs and propose graph-based algorithms that design complex virtual topologies.

1) Optimized Bottleneck Node Selection: BOTTLENET aims to find κ number of bottlenecks based on the input metric types M preferred by a network operator. The key question is how we choose the most vulnerable bottleneck nodes and links across heterogeneous types of metrics. To address this issue, we adopt MCDA (multi-criteria decision analysis), which is commonly used to choose an optimized goal across multiple criteria. To employ MCDA, we first need to vectorize the computed metric set and preprocess heterogenous types because it is possible that each metric vector has different dimensions.

For example, suppose that we are given $M = \{m_1, m_2, \ldots, m_r\}$, a set of computed metric vectors for r number of required metric types chosen by an operator. Consider a specific case where m_1 is the *BC* for all nodes, and m_2 is the *ALB* for all links. The two metrics have different vector dimensions; m_1's length is the number of nodes, whereas m_2's length is the number of links. To equalize those heterogeneous dimensions, we relax the dimension of link metric vectors into the node dimension; for a metric value of a link (u,v), we aggregate it to the *source* node u. For example, in Fig. 2a, if *ALB* of the link B–D is 0.7 and B–H is 0.9. Then, we assign 1.6 to the source node B by aggregating those two values.

Now, we build a decision matrix $D = \begin{pmatrix} m_{11} & \cdots & m_{1r} \\ \cdots & m_{ij} & \cdots \\ m_{n1} & \cdots & m_{nr} \end{pmatrix}$, where m_{ij} denotes a metric value of i-th node for j-th metric type, and n denotes the number of nodes. To cast this problem into ILP (integer linear programming) formulation, we calculate a node weight w_i from the matrix:

$$w_i = \frac{\sum_{j=1}^{n} \alpha_j \cdot m_{ij}}{\sum_{k=1}^{m} \sum_{j=1}^{n} \alpha_j \cdot m_{kj}},$$

that denotes the weighted sum of all selected metrics for the i-th node. Note that we leave α_j that is a biased weight of a metric type j, and it can be customized by the operator's preferences. We now formulate the bottleneck node selection problem as ILP:

$$\max. \sum_{u \in V} w(u) x_u$$

$$\text{s.t.} \sum_{u \in V} x_u \leq \kappa \quad \text{(C1)}$$

$$x_u \in [0, 1], u \in V \quad \text{(C2)}$$

The objective function is to find κ nodes that maximize the sum of metric values. The constraint (C1) ensures that we select κ nodes among all physical nodes V. In the constraint (C2), the decision variable x_u indicates whether node u is a bottleneck node or not. As a result, we obtain a set V_b that consists of κ bottleneck nodes v_b.

2) Deployment Node Selection: Based on the bottleneck node set V_b, the next step is to select deployment nodes V_d where virtual networks are deployed. Recall that BOTTLENET receives available nodes denoted by V_a and distance threshold τ as inputs from a network operator. To choose the best-fit deployment nodes among them, we consider the following constraints: In terms of cost-effectiveness, the operator may want to minimally use deployment nodes to save resources. On the other hand, the operator wants to deploy virtual networks close to bottleneck nodes as much as possible to maximize the deception effects. This problem can be defined as a vertex cover problem that chooses a minimum node cover set that is adjacent to all links in a graph. However, this has been referred to as NP-hard, and it is possible that we cannot select an adjacent node to bottlenecks in cases where there are no best-fit available nodes from the input V_a.

We address the problem by presenting the approximation algorithm. The key idea behind this approach is to choose a minimum node set that is placed in a position close to bottleneck nodes in terms of hop distances within the distance threshold τ. Consider the deployment node selection example shown in Fig. 5a that analyzes the topology dataset of an Abilene network [41]. We assume that a network operator wants to hide $\kappa = 2$ bottleneck nodes based on the static metric *BC*. Then, the selected bottleneck nodes are nodes D and E because they display high metric values. There are three available nodes for deployment: nodes B, I, and K.

BOTTLENET first calculates distance scores according to hop distances from each available node to each bottleneck node and greedily selects the closest node in turn. For example, node B is placed in the closest locations to both nodes E and D, leading to assigning high distance scores. If the selected node is located within the distance threshold τ, we mark that the node can cover the bottleneck nodes. Suppose that the threshold distance $\tau = 2$; then, we need not choose an additional deployment node because node B can cover both E and D.

Algorithm 1 Minimum Deployment Node Set Selection

Input: Physical topology $G = (V, E)$,
Set of bottleneck nodes V_b,
Set of available nodes V_a,
Distance threshold τ

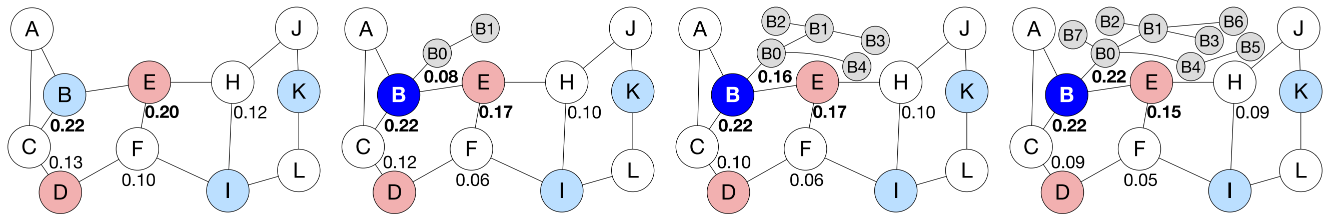
Output: Set of deployment nodes V_d

```
1: procedure GETDEPLOYMENTNODES($G$, $V_b$)
2:     $V_d \leftarrow \emptyset$
3:     for $v_a \in V_a$ where $v_a \notin V_b$ do
4:         $d(v_a) \leftarrow \frac{\sum_{v_b \in V_b} d(v_h, v_b)}{|V_b|}$
5:     while $V_b \neq \emptyset$ do
6:         $v_a^{min} \leftarrow$ node with minimum $d(v_a)$ from $V_a$
7:         $V_{neighbor} \leftarrow$ DistanceToNeighbor $v_a^{min}$, $\tau$
8:         $V_{covered} \leftarrow V_{neighbor} \cap V_b$
9:         if $V_{covered} \neq \emptyset$ then
10:            append $v_a^{min} \rightarrow V_d$
11:            $V_b \leftarrow V_b / V_{covered}$
12:    return $V_d$
```

Algorithm 1 illustrates the pseudo code of the algorithm. It first computes average distances from each available node v_a to all bottleneck nodes (lines 2 to 4). Then, it chooses an available node v_a^{min} that has the minimum average distance, and investigates whether τ-neighbor nodes $V_{neighbor}$, which denotes the set of nodes located within the distance threshold τ in turn (lines 5 to 8). If any bottleneck node is adjacent to v_a^{min}, the algorithm then chooses it as a deployment node that covers the adjacent bottleneck nodes denoted by $V_{covered}$, and removes the covered ones from the set V_b (lines 9 to 11). The iteration is repeated until all bottleneck nodes are covered.

3) Virtual Topology Generation: Once locations are determined, the virtual topology modeler builds virtual topologies that can effectively hide the bottleneck nodes by adding virtual networks to deployment nodes. The problem is what topology types should we use and how many virtual nodes and links are required.

We observe that the existence of network bottlenecks primarily originates from the fundamental topological nature of the communication networks—known as *scale-free networks*. It is one of a representative theory to illustrate network structure in the real world [27]. In scale-free networks, node degree distributions are known to follow the power law, meaning that a few nodes have most of the links; the more a node has links, the more it can be a bottleneck because the node becomes a hub node between others, which leads to a significant increase of degree on a few nodes. Although the existence of the scale-free networks has been debated by researchers since its emergence [42], we focus on the fact that this structure allows us to make *biased distributions* of link densities, which can



(a) Initial state. (b) After first step. (c) After 4th steps. (d) After 7th steps.

Fig. 5. A working example of the algorithms in Abilene. The red nodes denote target bottleneck nodes that have the highest metrics when $\kappa = 2$, the blue node denotes a chosen deployment node among the available nodes (cyan colors) when $\tau = 2$. The numbers indicate normalized values of BC. The grey nodes represent virtual nodes.

synthesize arbitrary bottlenecks. Those particular ones become attractive targets to the LFA attackers.

From the aspect of a graph feature, the scale-free network tends to create bottleneck nodes and links that have high centrality values inside its structure. For this reason, attaching it to the deployment nodes (remotely placed from targets) would reduce the static metrics of physical nodes and links relatively (see Fig. 5 for illustrations). Based on this approach, we design a virtual topology generation algorithm by leveraging a Barabasi-Albert (BA) model [43] that generates random scale-free networks. The BA model incrementally adds a new node to a graph and connects a link based on the random node selection with a certain probability. Here, the probability is computed as proportional to the number of links that existing nodes already have. Inspired by this strategy, our algorithm pursues to add a new link to nodes that have higher bottleneck metrics when a new node is added.

Algorithm 2 takes as inputs the physical topology G, a target bottleneck node v_b, a target deployment node v_d, and a node weight $w(v, G)$, which denotes the aggregated bottleneck metric values (computed in §V-B) of the node v in the graph G. The purpose of the algorithm is to generate virtual networks (subgraph) G_v that have at least one new bottleneck node inside its structure. It first adds an initial virtual node v_0 to the empty set G_v and a link from v_d to v_0 (lines 2 to 4). For each iteration, it adds a new virtual node v_v^{new} to G_v and a virtual link to an existing virtual node of G_v with a biased node selection probability $p(v)$, which is proportional to the node weight w that the virtual node v_v already has (lines 5 to 9). Thus, the virtual node that has high weight is more likely to have new links. The algorithm repeats the addition and linking of a new node, until V_v has at least one virtual node, the metric of which is greater than the target bottleneck node v_b (lines 10 to 13).

We illustrate the algorithm with a working example depicted in Fig. 5. As noted earlier, the target bottleneck nodes are E and D; therefore, node B is selected as a deployment node. BOTTLENET then begins an iteration by attaching initial two virtual nodes B0 and B1 to node B (Fig. 5b). When new virtual nodes are added, the algorithm recomputes a target metric and selection probability of existing virtual nodes for connecting a new link. During four iterations, the algorithm incrementally appends new virtual nodes B2, B3, and B4. Here, B2 and B3 are connected to B1, and B4 is connected to B0 as a result of the biased random node selection (Fig. 5c). After three more iterations, nodes B5, B6, and B7 are added, and B6 and B7 are linked to B1 and B0, respectively (Fig. 5d). The algorithm is terminated when the target metric of the virtual node B0 is greater than that of the nodes E and D.

Algorithm 2 Virtual Topology Generation

```
Input: Physical topology $G = (V, E)$,
    Target bottleneck node $v_b$,
    Target deployment node $v_d$,
    Node weight $w(v, G)$ for all nodes $v$ in topology $G$,
Output: Virtual network $G_v$
 1: procedure GenerateVirtualTopology($G, v_b, v_d, w$)
 2:     $G_v \leftarrow \emptyset$
 3:     add an initial virtual node $v_0 \rightarrow G_v$
 4:     add a virtual link $(v_d, v_0) \rightarrow G_v$
 5:     do
 6:         add a new virtual node $v_v^{new} \rightarrow G_v$
 7:         calculate $w(v_v, G_v), \forall v_v \in G_v$
 8:         compute $p(v_v) \leftarrow \frac{w(v,G_v)}{\sum_{u \in G_v} w(u,G_v)}, \forall v_v \in G_v$
 9:         $v_v \leftarrow$ select a node $v_v$ with the prob. $p(v_v), \forall v_v \in G_v$
10:         add a virtual link $(v_{new}, v_v) \rightarrow G_v$
11:         $H \leftarrow G \cup G_v$
12:         recalculate $w(v, H), \forall v \in H$
13:     while max $w(v, H) < w(v_b, H), \forall v \in H$
14:     return $G_v$
```

C. Deployment Manager

The goal of the deployment manager is to instantiate virtual networks on a physical network based on the generated topology model. This task involves diverse actions such as configuring rules for detecting and modifying TTL values of probing packets, and installing virtual switches on a target deployment node. Here, we present how to realize these techniques using SDN and virtual switches.

1) Virtual Network Deployment: We consider several requirements to deceive an adversary into thinking that virtual topologies are real: First, we should be able to dynamically activate or deactivate virtual networks by instructions of network operators. The reason is that BOTTLENET needs to periodically re-generate virtual topologies to be secure from the adversary's topology inference or when bottleneck locations are changed. Second, virtual networks should emulate realistic network parameters such as IP addresses and performance metrics. For example, if we use private IP addresses, an adversary will not believe the topology snapshot; thus, we need to assign public IP addresses to virtual networks

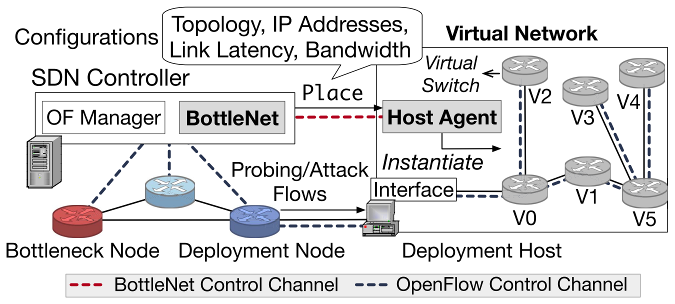
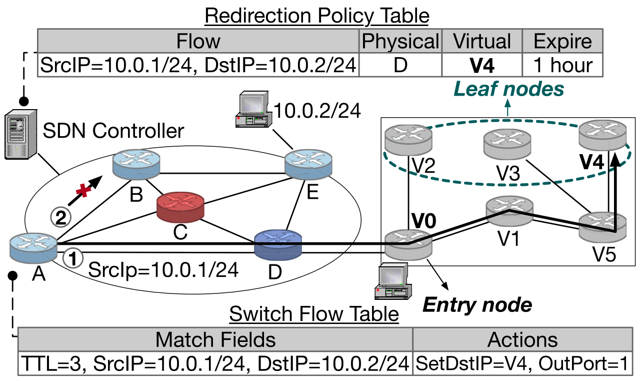


Fig. 6. A workflow of deploying virtual networks.



Redirection Policy Table

Flow	Physical	Virtual	Expire
SrcIP=10.0.1/24, DstIP=10.0.2/24	D	**V4**	1 hour

Switch Flow Table

Match Fields	Actions
TTL=3, SrcIP=10.0.1/24, DstIP=10.0.2/24	SetDstIP=V4, OutPort=1

Fig. 7. An example of rule enforcement. A is an edge node that receives the external probing packets in the network boundary, and C is the bottleneck node that needs to be hidden. The probing packets which TTL=3 are likely to expire on C, so they are redirected to D and arrive at V4 in the end.

(refer to §VII to see how we address this issue). Furthermore, because virtual switches are emulated on a local network stack in a deployment host, the performance metrics between virtual nodes can be observed as infeasible values (e.g., too many low RTTs) when suitable constraints are not applied. Finally, received probing packets by virtual networks should be monitored by BOTTLENET so that we can track probers' subsequent behavior.

BOTTLENET implements virtual networks using network virtualization techniques [23], [44] that enable the instantiation of virtual switches on general-purpose machines. Fig. 6 shows the workflow of deploying virtual networks. When receiving instructions from BOTTLENET, the host agent composes virtual networks, the link relationship of which follows the modeled virtual topologies. To redirect probing packets into virtual networks, one of the virtual switches is connected to the deployment node through an interface of a host machine (we call it an *entry node*.). To avoid the usage of dedicated control channels for virtual networks, all virtual switches use in-band OpenFlow channels where all control packets are transmitted into the assigned interface.

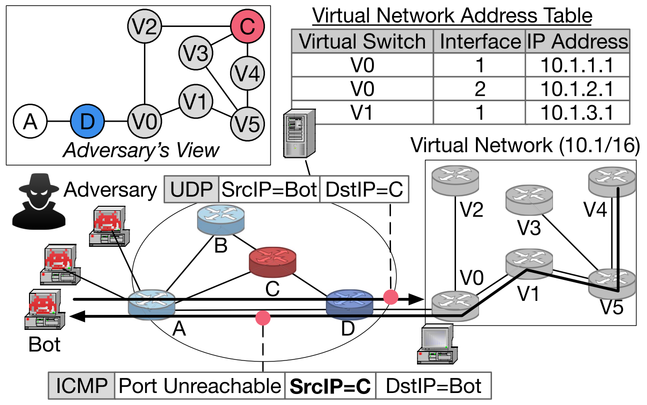
To enable dynamic control of virtual networks, we design a customized control channel between BOTTLENET and the host agent. The control channel carries an encrypted message that contains configuration information that specifies a virtual topology, allocated IP addresses, and performance constraints (e.g., link latencies and bandwidth). Here, a network operator can specify their preferences for the performance configurations, but BOTTLENET essentially utilizes the dynamic metrics as reference values to emulate real environments. One limitation is that it is difficult to implement the ultra-high bandwidth used in core links and nodes (e.g., 40Gbps, 100Gbps) in commodity machines. However, adversaries may not measure such large bandwidth precisely due to various QoS constraints in the wild (e.g., TCP fair-share rates [13]).

2) Detection and Redirection Rule Enforcement: BOTTLENET generates rules that detect potential probing packets and reroute them to virtual networks. We prioritize traceroute-type packets (i.e., low TTLs) as the most important criterion regarding whether or not a flow sender is a suspicious prober. To hide bottlenecks from an adversary, we need to avoid the case when traceroute-type packets expire on bottleneck nodes because this triggers switches to reply with ICMP *Time_Exceeded* packets. A strawman solution is to detect all TTL = 1 packets. However, if an adversary sends a TTL$>$1 packet that is crafted to become TTL = 1 on the bottleneck nodes, it will bypass the TTL = 1 detection rule.

A key benefit of OpenFlow is to enable monitoring such types of packets with fine-grained match fields; if incoming packets detected on edge nodes have a certain TTL value that is matched with the *hop distance* to bottleneck nodes, it will be likely to expire on bottleneck nodes. To implement this logic, we precompute routing paths from each edge node to bottleneck nodes based on current forwarding behavior. We then generate the following OpenFlow match fields: *TTL* as the length of the path, and *SrcIP* from the source traffic class that sends probing packets, and *DstIP* from the destination traffic class that passes by bottleneck nodes. If a packet header is matched with those fields, BOTTLENET considers it as a potential probing packet. Note that TTL is not currently specified as a standard OpenFlow match field, but we found that many data-plane extensions (e.g., Open vSwitch [45]) support the TTL field to complement the limited match capability.

When a potential probing packet is detected, BOTTLENET decides its redirected destination as shown in Fig. 7. We first need to choose *physical* destinations of a probing packet that is among the deployment nodes. Here, we preferentially select the nearest deployment node from detected locations by computing the shortest path because we aim to minimize exposure of physical nodes and links. We then decide *virtual* destinations of a probing packet among the virtual nodes. To maximize deception effects, we want to make adversaries discover as many virtual nodes as possible. For this purpose, we greedily choose the farthest virtual node from the entry node: Using the virtual network graph, we compute a minimum spanning tree rooted at the entry node and find a set of *leaf nodes* located in the bound of the tree. We then periodically choose a different virtual destination among the set in a round-robin manner so that all leaf nodes are fairly discovered. Finally, we install flow rules for which the match fields are the same as the detected probing packet headers so that subsequent probing flow can be redirected to the chosen virtual nodes.

3) Handling Direct Topology Probing: BOTTLENET also needs to handle *direct* probing packets that may reveal the identity of virtual switches. For example, traceroute terminates its stream of probing packets when receiving ICMP


Virtual Network Address Table
Adversary's View
Adversary
UDP SrcIP=Bot DstIP=C
Virtual Network (10.1/16)
Bot
ICMP Port Unreachable SrcIP=C DstIP=Bot
A
B
C
D
V0
V1
V2
V3
V4
V5

Virtual Switch	Interface	IP Address
V0	1	10.1.1.1
V0	2	10.1.2.1
V1	1	10.1.3.1

Fig. 8. An example of handling direct topology probing.

Port_Unreachable messages, indicating that a port in the target host is not opened (in case of UDP). This response reveals the IP address of a virtual switch (e.g., node V4 in Fig. 8) as a final destination, which is unexpected by adversaries. To address this issue, BOTTLENET instructs the *leaf nodes* of the virtual network to produce a fake direct response for which the source IP address is the one a sender intended (e.g., node C in Fig. 8). As a result, the adversary is deceived that the target node is located in an adjacent location of the leaf nodes, leading to the construction of completely different topologies from the adversary's perspective.

4) Mitigating Blind LFAs: An adversary can conduct blind LFAs aimed to flood arbitrary links despite not being aware of the exact path information. This is possible because we do not control the routing path of non-probing flows initially; hence, attack flows can pass through bottleneck links (e.g., link A–C in Fig. 9). For this, deployed virtual networks can also be utilized for mitigating the LFAs, such as a traffic scrubbing center that temporarily reroutes attack flows for the purpose of reducing target load. The traffic scrubbing center has been employed as a primary DDoS defense measure by many popular websites (e.g., CloudFlare [46]); thus, using virtual networks as traffic scrubbing services may encourage network operators to deploy BOTTLENET to protect their networks from not only topology probing but also actual flooding traffic.

The challenge is that it is impossible to differentiate attack flows from benign ones due to the low-rate (indistinguishable) property of LFAs [3]. We address this problem by selectively rerouting all flows in a probabilistic manner: Arbitrary portions of all flows are routed to virtual networks temporarily when a link experiences congestion. For this, BOTTLENET leverages the `select` action in the OpenFlow `group` table feature [39]. The `select` action enables an OpenFlow switch to choose one of multiple actions in one bucket. Each action has an assigned `weight` that denotes the probability of selecting the action, determined by hashing an incoming packet header.

Fig. 9 describes how BOTTLENET mitigates LFAs with the above solution. Consider that the link A–C is likely to experience congestion when the adversary sends attack flows to a decoy server. If detected, BOTTLENET installs a group entry to a suitable node (e.g., A) so that 20% of flows are redirected to the virtual network by assigning the weight 20, whereas the remaining 80% flows are forwarded to the

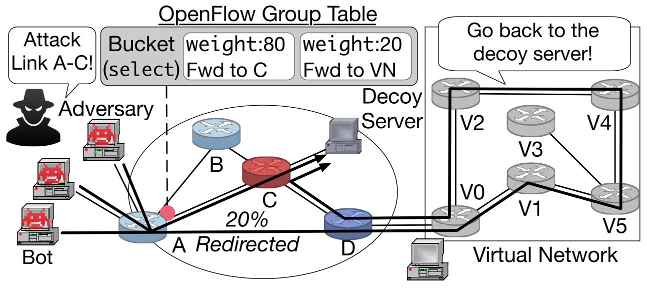


Fig. 9. An example of mitigating LFAs using a virtual network.

actual destination. The redirected flows are forwarded from the virtual network to the target decoy server after the flows visit the virtual nodes. This strategy alleviates the congestion of the link A–C, while preventing flows from being flooded using the imposed link constraints to the virtual network (see §V-C.1). We refer readers to §VI-B to see how this solution is effective in reducing the dynamic metrics of bottlenecks.

VI. EVALUATION

In this section, we evaluate the effectiveness of BOTTLENET's algorithms and conduct a performance benchmark on large-scale network topologies.

Implementation and Environments. We implemented a BOTTLENET prototype as SDN applications with 1000+ lines of code in Python upon Ryu [24], one of the popular SDN controllers. To collect dynamic metrics, we modified Ryu's core module `ryu.topology.switches`, which is responsible for collecting topology information from the data plane. We use PuLp [47] to model and solve the ILP optimization problem. We leverage Open vSwitch [23], which enables modeled virtual nodes to be rapidly instantiated into virtual switches on general-purpose machines. To implement virtual host nodes in virtual networks, we utilize Docker containers [44] that enables the deployment of micro services to process redirected traffic. All experiments were performed with a 40-core 2.2 GHz Intel Xeon E5-2630 and 64 GB RAM.

We evaluate BOTTLENET to answer the following questions: (i) Can modeled virtual topologies generate significantly different topology views? (ii) Can modeled virtual topologies effectively reduce metrics of bottleneck nodes? (iii) Is BOTTLENET scalable for building large-scale virtual networks?

A. Verifying Deception Algorithms

We first verify the effectiveness of BOTTLENET's topology deception algorithms on large-scale topologies for an operator's diverse input parameters such as the number of bottleneck nodes κ and the number of virtual nodes N per virtual network. We use Mininet [48] to model large-scale router-level topologies of Topology-Zoo [49] database for which the size is summarized in Table III. To simulate probing scenarios, we assign arbitrary C class IP addresses to each router interface and generate responses from routers when a traceroute packet is detected. In the simulation, we examine two experimental cases: (i) κ is fixed as 30% of total nodes for each topology, and N is changed, which indicates the case when the number of virtual nodes per virtual network varies for protecting fixed bottleneck locations, (ii) N is fixed as 30%

TABLE III
ROUTER-LEVEL TOPOLOGY DATA SIMULATED IN EXPERIMENTS

	Abilene	AT&T	SWITCH	OTEGlobe
# routers	11	25	42	83
# links	14	57	63	99

of total nodes and κ is varied from 1 to a maximum number of nodes, which represents the case when the protected locations become wider with the same number of virtual nodes. Note that the variables N and κ both affect the total number of virtual nodes and links; thus, we used one of them to a fixed value (i.e., 30% of total nodes) to observe how the other influences the experimental results.

Effectiveness of Topology Similarity Reduction. The key question here is how *different* a topology an adversary obtains after deploying virtual networks. To quantify the similarities between a physical network topology and an adversary's virtual topology, we measure the graph edit distance (GED), which denotes the total cost of transforming one graph to another one in terms of graph operations (e.g., node/link insertions and deletions). We consider the worst-case scenario that an adversary can fully discover all nodes and links with repeated probing trials. Thus, we measure GED by comparing the entire original physical topology with a virtual topology where virtual networks are attached. The less the similarity, the more an adversary cannot easily infer the original topology; therefore, bottlenecks are unlikely to be targeted.

Fig. 10a shows that the similarities linearly decrement as bottleneck node coverage κ increases. We observe that a similarity reduction is highly affected by a graph structure as virtual nodes are more deployed for many target bottleneck nodes. For example, when we deploy virtual nodes for 10% bottleneck nodes, 60% similarity is reduced in OTEGlobe, whereas 21% is reduced in AT&T. The reason for this difference is that the former has highly skewed degree distributions, meaning that the top 10% bottleneck nodes have the most connectivities in a graph. Therefore, it is quite effective in reducing similarities by deploying virtual nodes on them. In the case of varying the number of deployed virtual nodes N on the 30% fixed bottleneck nodes, similar reduction trends among topologies are observed because the target bottleneck nodes are unchanged (see Fig. 10b). Note that deploying 10 virtual nodes for each 30% of bottleneck nodes guarantees at least a 50% similarity reduction in all topologies.

Effectiveness of Static Metric Reduction. We now examine how the topology-generation algorithm is effective in reducing the static metrics of target bottleneck nodes. To quantify this, we compute the node weight w that denotes the total bottleneck metrics for the static metrics BC, CC, DC, and MC. Then, we calculate $w_{before}(v_b)$, weight values for all bottleneck nodes V_b before applying the topology-generation algorithm. After deploying virtual topologies, we recompute $w_{after}(v_b)$ for the extended topology. Based on these values, we measure the metric reduction ratio (MR) with the following definition:

$$MR(v_b) = 1 - \frac{avg(w_{after}(v_b))}{avg(w_{before}(v_b))},$$

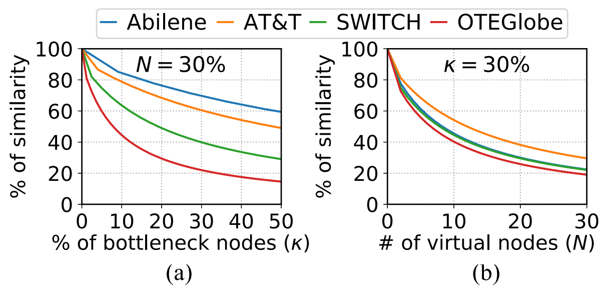


Fig. 10. Measured topology similarity for variants of κ and N.

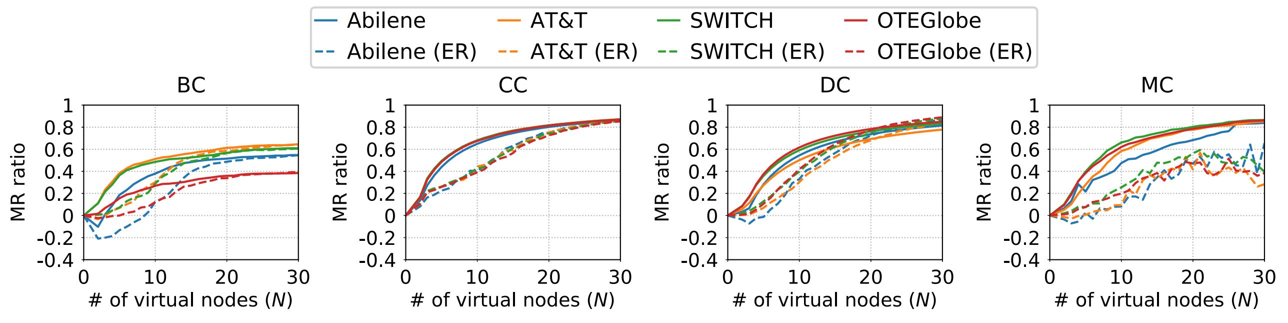
where avg denotes the average weights of bottleneck nodes.

To assess the performance of our algorithm, we use the Erdos-Renyi (ER) graph model [50] as a baseline. Whereas our algorithm generates a scale-free graph for which the degree distributions are biased, the ER model produces a uniformly connected graph, where all nodes have nearly similar degrees.

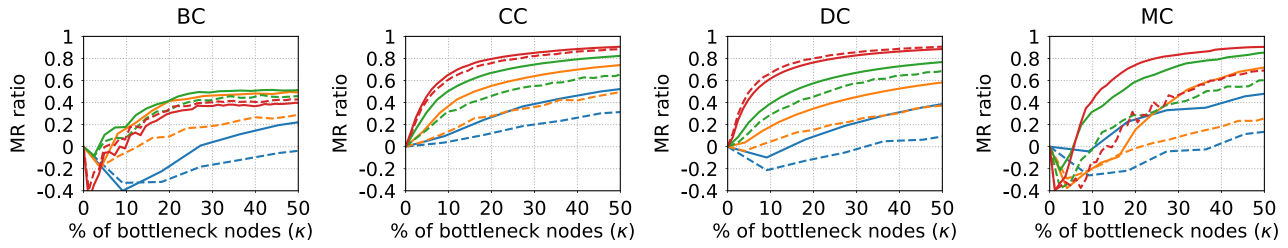
Fig. 11a illustrates measured MR when N is changed. Overall, we conclude that deploying 10 virtual nodes for each bottleneck node is the most efficient solution to protect a total of 30% of bottleneck nodes. After that point, it turns out that there are no significant differences of the reduction rate from the ER model. In particular, ER also performs well for all metrics after $N = 20$. However, it costs a lot, given the number of virtual nodes required. For example, deploying 20 virtual nodes to 30% of bottleneck nodes in SWITCH requires the deployment of 260 virtual nodes in total to achieve a 60% reduction rate, whereas our algorithm requires only 100 virtual nodes. Thus, our algorithm can make deployment nodes save system resources.

Fig. 11b shows how BOTTLENET can protect a wide area of bottleneck nodes as κ increments. Although our algorithm rather increases metrics when nodes are deployed on few deployment nodes, the performance becomes better after $\kappa = 10\%$ of a network size. Hence, the wide deployment of virtual networks is more effective for reducing bottleneck metrics. The reason is that virtual nodes evenly distributed in a network topology contribute to an increase of their static metrics, making bottleneck nodes appear as normal nodes, whereas virtual nodes seem attractive targets.

Our algorithm particularly shows better performance when BC and MC metrics are used. The reason is that they are strongly affected by a graph structure rather than network topology size. For example, MC metrics become larger on a scale-free graph because there are many link-cuts due to the existence of bottleneck nodes. On the other hand, the ER model generates particularly few link-cuts because links are uniformly connected. This property leads to significant fluctuations of the MR ratio when the ER model is applied to reduce MC metrics. The structure-dependent trends are also observed in the BC metric. As the virtual network grows, a few nodes are involved in many routing paths, which makes the BC values of the nodes significantly larger than others. In the case of CC, DC, it turns out that they are primarily affected by the total number of virtual nodes. This is why the ER model's



(a) κ is fixed as 30% of a network size and N is variable.



(b) N is fixed as 30% of a network size and κ is variable.

Fig. 11. Measured Metric Reduction (MR) ratio for different parameters: # of virtual nodes N and # bottleneck nodes κ.

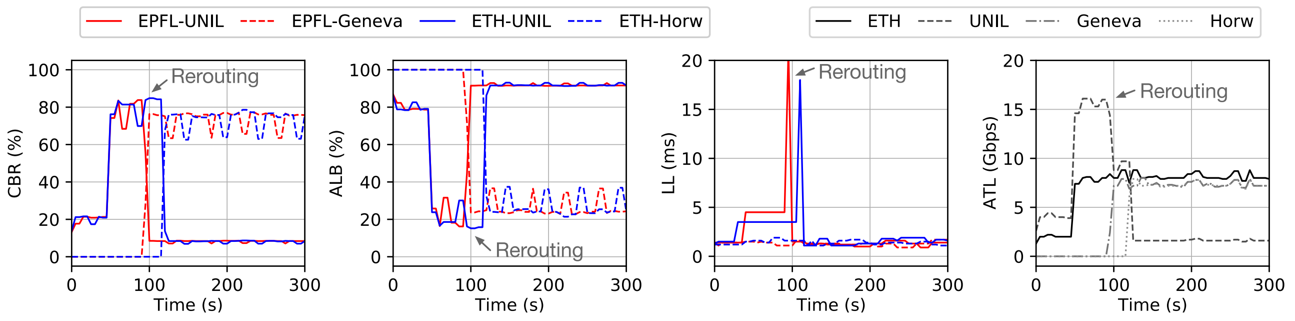


Fig. 12. Results for effectiveness of rerouting algorithms to reduce the dynamic metrics measured in the scenario shown in Fig. 13.

MR becomes better than our algorithms for *DC* when many virtual nodes are deployed.

B. Verifying Rerouting Algorithms

Next, we verify the effectiveness of dynamic metric reduction when an adversary mounts blind LFAs (§V-C.4); thus, congestion occurs on (non-)bottleneck links. For this, we emulated a rerouting scenario for mitigating LFAs in the SWITCH topology as depicted in Fig. 13. In the scenario, we divided the network into several groups: sites A, B, and C, where A and B send attack flows (green and red arrows) to the target area C (decoy servers). The purpose of this attack is to make congestion on bottleneck links ETH–UNIL and EPFL–UNIL because those links are located in important positions in the network. For bandwidth configurations, we uniformly set the link capacity to be 10 Gbps for all links. We generate background traffic among each site to consume approximately 20% of the link capacity. Regarding attack flows, we deployed 200 bots to 5 switches in each site A, B, and each bot sends 6 Mbps attack flows to site C to consider the low-rate property of LFAs. As such, the aggregated traffic becomes about 8 Gbps on the bottleneck links, which is likely to saturate the link capacity when additional traffic is requested. We assume that a

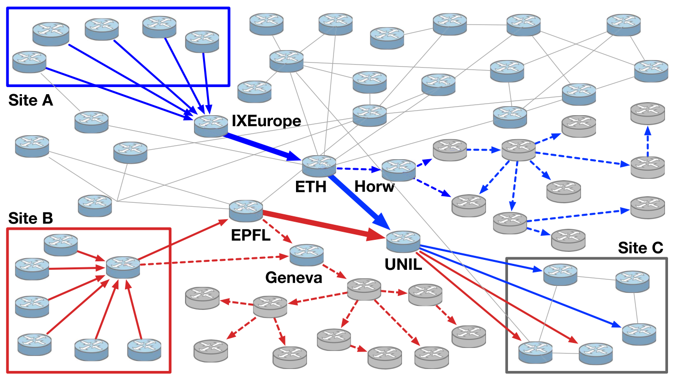


Fig. 13. The SWITCH topology used for testing dynamic metric reduction. The blue and red arrows denote attack flows sent from Site A and B to Site C, respectively. The dashed lines indicated rerouted flows and the grey nodes represent deployed virtual nodes.

network operator instructs BOTTLENET to execute the reroute action to avoid those links becoming congested.

Effectiveness of Dynamic Metric Reduction. Fig. 12 illustrates measured dynamic metrics during the execution of the rerouting scenario. As shown in the left two plots, the *CBR* of the bottleneck links EPFL–UNIL and ETH–UNIL became 80% when LFAs are mounted, whereas *ALB* was

TABLE IV
PROCESSING TIME (MS) TO GENERATE DYNAMIC METRICS

	ATL	CBR	ALB	LL
Statistics Collection	73.01 ms (99.76%)	98.44 ms (99.10%)	98.44 ms (99.10%)	0.79 ms (91.29%)
Post Processing	0.18 ms (0.24%)	0.89 ms (0.90%)	0.89 ms (0.90%)	1.12 ms (8.71%)

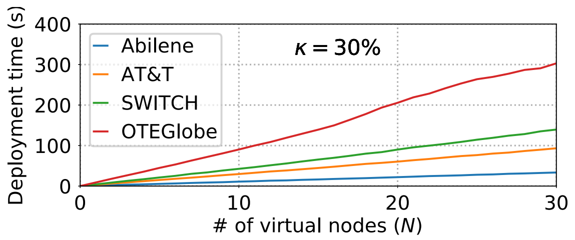


Fig. 14. Deployment time (s) per virtual nodes N.

20%. At the time that `reroute` actions were executed, their link capacity was restored to 90% as attack flows are rerouted to virtual networks deployed at the deployment nodes Geneva and UNIL. This situation is illustrated as *CBR* of the entry links to deployment nodes, such as EPFL–Geneva and ETH–Horw, was increased to 75%, whereas their *ALB* was decreased to 25%. We also observed that the bandwidth of rerouted attack flows fluctuated due to the imposed constraints on virtual networks (see §VI-B). Before the rerouting action, the SDN controller had to proactively install many forwarding rules along the rerouted path to avoid service disruption. This method led to a temporary saturation of the OpenFlow control channel, displayed as temporary high *LL* (i.e., 18–20 ms) at the rerouting time due to delayed link measurement (see Fig. 4).

C. Performance Benchmark

In this evaluation, we conducted a performance benchmark to measure the system overhead of BOTTLENET to generate dynamic metrics and virtual switches.

Latency of Generating Dynamic Metrics. BOTTLENET requires two sub-stages for generating dynamic metrics: (i) statistics collection, which gathers raw network statistics from switches, and (ii) post-processing, which synthesizes dynamic metrics using defined equations in §V-A. We measure the processing time of each step as shown in Table IV, where parentheses denote each proportion of total time. The result indicates that all dynamic metrics are generated in less than 1 second, and among them, obtaining *CBR* from all links requires the longest processing times as it pulls out all port statistics from switches. We note that the statistics collection occupies a large proportion of the metric-generation process. This limitation can be significantly improved if BOTTLENET is supported by scalable SDN controllers. For example, the recent performance testing result of ONOS [35] has achieved a 36.9 ms elapsed time when installing 200 flows rules [51].

Time to Deploy Virtual Switches. Fig. 14 illustrates the measured deployment time when virtual topologies are converted into virtual switches according to N. For $k = 30\%$ of a network size, BOTTLENET deploys 10 virtual nodes in less than a 1 minute for a country-scale network topology such as AT&T and SWITCH and in less than 2 minutes for a continental-level network topology OTEGlobe. From this result, we argue that BOTTLENET can be used on a practical network environment if a reliable control channel and host agent are correctly installed on available hosts.

VII. DISCUSSION

Network Probing in SDN. As noted, adversaries need to find bottlenecks by probing the network before conducting LFAs. Hence, handling probing packets is the most critical operation in our system. Technically, SDN-enabled switches do not support traceroute [52]; thus, we may consider that if SDN-enabled switches are fully deployed in a network, LFAs are not so feasible; adversaries have a problem in targeting bottlenecks. However, network probing is an essential option in understanding network status and debugging network failures; hence, several researchers have proposed traceroute tools tailored for SDN [53].

As such, we also enable traceroute in our system by leveraging a few OpenFlow protocol features. When installing flow rules with *Flow_Mod*, BOTTLENET inserts the *decrement_IP_TTL* action into the beginning of the rule action chain. This approach makes all incoming packets decrease their TTL values when matched in switch flow tables. Although the *decrement_IP_TTL* is an optional field in the current OpenFlow protocol specification [39], we observe that network vendors include it as a supported feature in the state-of-the-art product specifications [54]. To detect expired packets for which the TTL = 1, a controller needs to instruct OpenFlow switches to configure the *invalid TTL* flag to receive *Packet_In* messages that include an expired packet header. When detected, the controller generates a *Packet_Out* message, which instructs a switch to send ICMP *Time_Excceded* packets to a sender with a router IP address. With this methodology, BOTTLENET can control network probing sent by attackers and help operators understand network status as well.

Topology Inference Attacks. Adversaries may attempt to infer the structure of the physical topology by observing common nodes and links that are frequently shown in probing responses. To prevent this issue, BOTTLENET periodically re-runs the topology-generation algorithm to mutate the structure of virtual topologies for a certain epoch. Here, dynamic metrics that vary by network status can be unpredictable seed values for the node selection probability (§V-B). Hence, it results in the deployment of virtual networks into different node locations from the previous epoch.

Network Address Allocation. Our virtual networks require public IP address blocks to answer with valid responses that can deceive adversaries. We envision that network operators provide BOTTLENET with a public IP address range from unused address blocks as an investment for enhancing the security of their network. Indeed, there have been numerous prior efforts that have successfully leveraged *unused* IP addresses for monitoring suspicious activities [55] and using honeynets to hinder scanning activities [28], [34]).

Partial Deployment. One of the limitations in the current design is that BOTTLENET requires all switches to support SDN (i.e., OpenFlow) for collecting fine-grained bottleneck metrics. We note that BOTTLENET can also support a partial-deployment model. In this model, BOTTLENET can obtain the required metrics from traditional network management protocols such as SNMP and NetConf.

Control-Plane Resilience. An adversary can exploit SDN capabilities if they know that a target network is managed by a centralized SDN controller. For example, they can mount control-plane DDoS attacks [56], [57] that saturate control channels between SDN switches and the controller by triggering several *Packet_In* messages. This issue can be addressed if network operators adopt a more resilient control-plane, such as physically distributed controllers (e.g., ONOS [58], OpenDaylight [36]) or more intelligence data-planes [59], [60] that alleviate control message flooding.

VIII. CONCLUSION

Protecting critical network nodes and links from external threats is an extremely onerous task due to the openness of the Internet. An adversary can easily identify network bottlenecks on the Internet through active topology probing, making it difficult for an operator to fortify attack points. In this paper, we present BOTTLENET, which is a comprehensive network topology deception framework that provides diverse bottleneck metrics and complex topology generation algorithms. Our framework allows for the definition of diverse bottleneck metrics in terms of graph and performance properties for nodes and links. Thus, an operator can devise the desired inputs based on network policy or conditions, and BOTTLENET selects the best deployment nodes with the assistance of its node selection algorithms. By deploying virtual networks, our evaluations demonstrate how it is possible to manipulate the topology view of adversaries with proactive topology deception algorithms and reactive manipulation rules.

ACKNOWLEDGMENT

The authors thank the anonymous TIFS reviewers for their valuable comments. They would also like to thank Eduard Marin and Mauro Conti for joining important discussions on this article and Jeongyoon Moon for initial experiments.

REFERENCES

[1] A. Dhamdhere and C. Dovrolis, "The Internet is flat: Modeling the transition from a transit hierarchy to a peering mesh," in *Proc. 6th Int. Conf. (Co-NEXT)*, 2010, pp. 1–12.

[2] R. Albert, H. Jeong, and A.-L. Barabási, "Error and attack tolerance of complex networks," *Nature*, vol. 406, no. 6794, pp. 378–382, Jul. 2000.

[3] M. S. Kang, S. B. Lee, and V. D. Gligor, "The crossfire attack," in *Proc. IEEE Symp. Secur. Privacy*, May 2013, pp. 127–141.

[4] A. Studer and A. Perrig, "The coremelt attack," in *Proc. Eur. Conf. Res. Comput. Secur.*, 2009, pp. 37–52.

[5] S. M. Bellovin and E. R. Gansner, "Using link cuts to attack Internet routing," in *A Technical Report From Columbia University*, 2013. [Online]. Available: https://academiccommons.columbia.edu/doi/10.7916/D84J0MT0

[6] R. Cohen, K. Erez, D. ben-Avraham, and S. Havlin, "Breakdown of the Internet under intentional attack," *Phys. Rev. Lett.*, vol. 86, no. 16, pp. 3682–3685, Apr. 2001.

[7] M. S. Kang and V. D. Gligor, "Routing bottlenecks in the Internet: Causes, exploits, and countermeasures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2014, pp. 321–333.

[8] (2016). *Large DDoS Attacks Cause Outages at Twitter, Spotify, and Other Sites*. Accessed: Apr. 7, 2021. [Online]. Available: https://techcrunch.com/2016/10/21/many-sites-including-twitter-and-spot%ify-suffering-outage/

[9] (2013). *Can a DDoS Break the Internet? Sure! Just Not all of it*. Accessed: Apr. 7, 2021. [Online]. Available: https://arstechnica.com/information-technology/2013/04/can-a-ddos-break%-the-internet-sure-just-not-all-of-it/

[10] R. Meier, P. Tsankov, V. Lenders, L. Vanbever, and M. Vechev, "NetHide: Secure and practical network topology obfuscation," in *Proc. Secur. Symp. USENIX*, 2018, pp. 693–709.

[11] S. T. Trassare, R. Beverly, and D. Alderson, "A technique for network topology deception," in *Proc. MILCOM IEEE Mil. Commun. Conf.*, Nov. 2013, pp. 1795–1800.

[12] X. Ding, F. Xiao, and M. Zhou, "Active link obfuscation to thwart link-flooding attacks for Internet of Things," 2017, *arXiv:1703.09521*. [Online]. Available: http://arxiv.org/abs/1703.09521

[13] M. S. Kang, V. D. Gligor, and V. Sekar, "SPIFFY: Inducing cost-detectability tradeoffs for persistent link-flooding attacks," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, pp. 53–55.

[14] A. Aydeger, N. Saputro, and K. Akkaya, "Utilizing NFV for effective moving target defense against link flooding reconnaissance attacks," in *Proc. MILCOM IEEE Mil. Commun. Conf. (MILCOM)*, Oct. 2018, pp. 946–951.

[15] S. Achleitner, T. La Porta, P. McDaniel, S. Sugrim, S. V. Krishnamurthy, and R. Chadha, "Cyber deception: Virtual networks to defend insider reconnaissance," in *Proc. 8th ACM CCS Int. Workshop Manag. Insider Secur. Threats*, Oct. 2016, pp. 57–68.

[16] N. Hu, L. Li, Z. M. Mao, P. Steenkiste, and J. Wang, "Locating Internet bottlenecks: Algorithms, measurements, and implications," in *Proc. Conf. ACM Special Interest Group Data Commun.*, 2004, pp. 41–54.

[17] P. Holme, B. J. Kim, C. N. Yoon, and S. K. Han, "Attack vulnerability of complex networks," *Phys. Rev. E, Stat. Phys. Plasmas Fluids Relat. Interdiscip. Top.*, vol. 65, no. 5, May 2002, Art. no. 056109.

[18] D. Magoni, "Tearing down the Internet," *IEEE J. Sel. Areas Commun.*, vol. 21, no. 6, pp. 949–960, Aug. 2003.

[19] M. Schuchard, A. Mohaisen, D. F. Kune, N. Hopper, Y. Kim, and E. Y. Vasserman, "Losing control of the Internet: Using the data plane to attack the control plane," in *Proc. 17th ACM Conf. Comput. Commun. Secur. (CCS)*, 2010, pp. 726–728.

[20] S. Shin, L. Xu, S. Hong, and G. Gu, "Enhancing network security through software defined networking (SDN)," in *Proc. 25th Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2016, pp. 1–9.

[21] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and elastic DDoS defense," in *Proc. Secur. Symp. USENIX*, 2015, pp. 817–832

[22] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Y. Yau, and J. Wu, "Realtime DDoS defense using COTS SDN switches via adaptive correlation analysis," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 7, pp. 1838–1853, Jul. 2018.

[23] (2020). *Open vSwitch*. Accessed: Apr. 7, 2021. [Online]. Available: http://openvswitch.org

[24] (2020). *Ryu*. Accessed: Apr. 7, 2021. [Online]. Available: https://osrg.github.io/ryu/

[25] (2006). *Traceroute for Linux*. Accessed: Apr. 7, 2021. [Online]. Available: http://man7.org/linux/man-pages/man8/traceroute.8.html

[26] (2020). *iPerf3*. Accessed: Apr. 7, 2021. [Online]. Available: https://software.es.net/iperf/

[27] A.-L. Barabási and E. Bonabeau, "Scale-free networks," *Sci. Amer.*, vol. 288, no. 5, pp. 60–69, 2003.

[28] N. Provos, "A virtual honeypot framework," in *Proc. Secur. Symp. USENIX*, 2004, pp. 1–14.

[29] S. B. Lee, M. S. Kang, and V. D. Gligor, "CoDef: Collaborative defense against large-scale link-flooding attacks," in *Proc. 9th ACM Conf. Emerg. Netw. Exp. Technol.*, Dec. 2013, pp. 417–428.

[30] (2020). *A Practical Guide to (Correctly) Troubleshooting With Traceroute*. Accessed: Apr. 7, 2021. [Online]. Available: https://archive.nanog.org/meetings/nanog45/presentations/Sunday/RAS_tra%ceroute_N45.pdf

[31] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 133–145, Oct. 2002.

[32] B. Augustin *et al.*, "Avoiding traceroute anomalies with Paris traceroute," in *Proc. 6th ACM SIGCOMM Internet Meas. (IMC)*, 2006, pp. 153–158.

[33] S. Jain *et al.*, "B4: Experience with a globally-deployed software defined wan," in *Proc. ACM SIGCOMM Conf. SIGCOMM*, Aug. 2013, pp. 1–12.

[34] L. Alt, R. Beverly, and A. Dainotti, "Uncovering network tarpits with degreaser," in *Proc. 30th Annu. Comput. Secur. Appl. Conf. (ACSAC)*, 2014, pp. 3–14.

[35] P. Berde *et al.*, "ONOS: Towards an open, distributed SDN OS," in *Proc. Workshop Hot Topics Softw. Defined Netw.*, 2014, pp. 1–6.

[36] (2020). *OpenDaylight*. Accessed: Apr. 7, 2021. [Online]. Available: https://www.opendaylight.org

[37] (2020). *PlanetLab*. Accessed: Apr. 7, 2021. [Online]. Available: https://www.planet-lab.org

[38] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 40, no. 1, p. 35, Mar. 1977.

[39] (2014). *OpenFlow Switch Specification Version 1.3.5*. Accessed: Apr. 7, 2021. [Online]. Available: https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-swit%ch-v1.3.5.pdf

[40] R. Skowyra *et al.*, "Effective topology tampering attacks and defenses in software-defined networks," in *Proc. 48th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2018, pp. 374–385.

[41] (2005). *Abilene Network*. Accessed: Apr. 7, 2021. [Online]. Available: https://web.archive.org/web/20120324103518/http://www.internet2.edu/pub%s/200502-IS-AN.pdf

[42] A. D. Broido and A. Clauset, "Scale-free networks are rare," *Nature Commun.*, vol. 10, no. 1, pp. 1–10, Dec. 2019.

[43] R. Albert and A.-L. Barabási, "Statistical mechanics of complex networks," *Rev. Modern Phys.*, vol. 74, no. 1, p. 47, 2002.

[44] (2020). *Docker*. Accessed: Apr. 7, 2021. [Online]. Available: https://www.docker.com

[45] (2020). *Open vSwitch Extensions*. Accessed: Apr. 7, 2021. [Online]. Available: http://docs.openvswitch.org/en/latest/topics/ovs-extensions/

[46] (2020). *CloudFlare*. Accessed: Apr. 7, 2021. [Online]. Available: https://www.cloudflare.com/

[47] (2020). *PuLP*. Accessed: Apr. 7, 2021. [Online]. Available: https://coin-or.github.io/pulp/

[48] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proc. 9th ACM SIGCOMM Workshop Hot Topics Netw. (Hotnets)*, 2010, pp. 1–6.

[49] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[50] P. Erdős and A. Rényi, "On the evolution of random graphs," *Publ. Math. Inst. Hung. Acad. Sci.*, vol. 5, no. 1, pp. 17–60, 1960.

[51] (2020). *ONOS Master: Experiment I—Single Bench Flow Latency Test*. Accessed: Apr. 7, 2021. [Online]. Available: https://wiki.onosproject.org/display/ONOS/Master%3A+Experiment+I+-+Sing%le+Bench+Flow+Latency+Test

[52] J. Cao *et al.*, "The CrossPath attack: Disrupting the SDN control channel via shared links," in *Proc. Secur. Symp. USENIX*, 2019, pp. 19–36.

[53] P. Tammana, R. Agarwal, and M. Lee, "CherryPick: Tracing packet trajectory in software-defined datacenter networks," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, Jun. 2015, pp. 1–7.

[54] (2020). *OpenFlow Configuration Guide, Cisco IOS XE Gibraltar*. Accessed: Apr. 7, 2021. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/prog/configuration/16%11/b_1611_programmability_cg/OpenFlow.html

[55] V. Yegneswaran, P. Barford, and D. Plonka, "On the design and use of Internet sinks for network abuse monitoring," in *Recent Advances in Intrusion Detection*. Berlin, Germany: Springer, 2004, pp. 146–165.

[56] S. Shin and G. Gu, "Attacking software-defined networks: A first feasibility study," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, 2013, pp. 165–166.

[57] M. Zhang, G. Li, L. Xu, J. Bi, G. Gu, and J. Bai, "Control plane reflection attacks in SDNs: New attacks and countermeasures," in *Research in Attacks, Intrusions, and Defenses*. Cham, Switzerland: Springer, 2018, pp. 161–183.

[58] (2020). *Open Network Operating System (ONOS)*. Accessed: Apr. 7, 2021. [Online]. Available: https://wiki.onosproject.org/

[59] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 413–424.

[60] H. Wang, L. Xu, and G. Gu, "FloodGuard: A DoS attack prevention extension in software-defined networks," in *Proc. 45th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw.*, Jun. 2015, pp. 239–250.

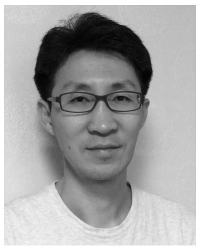
Jinwoo Kim received the B.S. degree in computer science and engineering from Chungnam National University and the M.S. degree from the Graduate School of Information Security, KAIST, where he is currently pursuing the Ph.D. degree with the School of Electrical Engineering. His research interests include software defined networking (SDN) security, designing a system for enchaining network security, and network theory.

Jaehyun Nam received the B.S. degree in computer science and engineering from Sogang University, South Korea, and the M.S. and Ph.D. degrees from the School of Computing, KAIST. He is currently a Technical Advisor with AccuKnox. His research interests include networked and distributed computing systems. He is especially interested in performance and security issues in cloud computing environments.

Suyeol Lee received the B.S. degree from the School of Electrical Engineering, KAIST, where he is currently pursuing the M.S. degree with the School of Electrical Engineering. His research interest includes the analysis of illicit bitcoin activities using graph-based deep learning.

Vinod Yegneswaran received the A.B. degree from the University of California at Berkeley, Berkeley, CA, USA, in 2000, and the Ph.D. degree from the University of Wisconsin, Madison, WI, USA, in 2006, all in computer science. He is currently the Senior Computer Scientist of SRI International, Menlo Park, CA, pursuing advanced research in network and systems security. His current research interests include SDN security, malware analysis, and anti-censorship technologies. He has served on several NSF panels and program committees of security and networking conferences, including the IEEE Security and Privacy Symposium.

Phillip Porras received the M.S. degree in computer science from the University of California at Santa Barbara, Santa Barbara, CA, USA, in 1992. He is currently an SRI Fellow and the Program Director of the Computer Science Laboratory, Internet Security Group, SRI, Menlo Park, CA. He has participated on numerous program committees and editorial boards, and participates on multiple commercial company technical advisory boards. He continues to publish and conduct technology development on numerous topics, including intrusion detection and alarm correlation, privacy, malware analytics, active and software defined networks, and wireless security.



Seungwon Shin (Member, IEEE) received the B.S. and M.S. degrees in electrical and computer engineering from KAIST and the Ph.D. degree in computer engineering from the Electrical and Computer Engineering Department, Texas A&M University. He is currently an Associate Professor with the School of Electrical Engineering, KAIST. He is also the Corporate Vice President at Samsung Electronics, leading the Security Team in the IT and Mobile Communications Division. His research interests include software-defined networking security, the IoT security, Botnet analysis/detection, DarkWeb analysis, and cyber threat intelligence.