Bridging Missing Gaps in Evaluating DDoS Research

Lumin Shi, Samuel Mergendahl, Devkishen Sisodia, Jun Li
{luminshi,smergend,dsisodia,lijun}@cs.uoregon.edu
University of Oregon

Abstract

While distributed denial-of-service (DDoS) attacks become stealthier and more disruptive, real-world network operators often ignore academic DDoS defense research and instead rely on basic defense techniques that cannot adequately defend them. In fact, prior to the deployment of a DDoS defense solution, a network operator must understand its impact specifically on their network. However, without a sound empirical analysis of the solution, which is often the case even for the most cited academic work, the network operator may fear its poor defense efficacy or its adverse effects on legitimate traffic. In this work, we elaborate on the critical missing gaps in DDoS defense evaluation and propose a new evaluation platform to help produce the missing defense analytics. To identify the impact of a defense solution in realistic network settings, our platform offers to emulate a mini Internet topology with realistic IP address space allocation and generate representational, closed-loop background traffic specific to particular networks. As such, our platform fulfills the prominent missing gaps in current DDoS research. In the end, we conduct some experiments to demonstrate the correctness and efficiency of our platform.

1 Introduction

Advanced distributed denial-of-service (DDoS) attacks, such as the CrossFire attack [14] and CICADAS [15], seriously challenge the efficacy of the rudimentary DDoS defense strategies typically deployed by network operators. In particular, the most commonly deployed DDoS defense systems consist of two main components: a simple threshold-based DDoS detection/classification system, such as FastNetMon [9], and a coarsely grained mitigation solution such as Remotely Triggered Black Hole (RTBH) [18]. Unfortunately, these simple defense strategies struggle against the aforementioned, advanced DDoS attacks. A threshold-based detection solution rarely finds an appropriate balance between false positive and false negative rates, and a coarsely grained mitigation solution, by definition, filters legitimate traffic.

Despite over two decades of research to improve DDoS defense, network operators continue to choose these basic DDoS defense strategies regardless of their limitations. To draw insights into this discrepancy, Kokulu *et al.* recently surveyed security operation centers and found that network operators must obtain a thorough quantitative analysis of any security solution prior to deployment [17]. Without a comprehensive performance test of a potential security system in an environment similar to the network in which the system will deploy, the network operator cannot justify the risk associated with the adoption of new research defense systems. Clearly, to increase real-world DDoS defense deployment, the research community should conduct rigorous defense evaluation of DDoS defense solutions. However, by surveying well-received DDoS defense papers, we found that even highly-cited solutions frequently fail to evaluate their approach under realistic deployment scenarios. We highlight a few prominent missing gaps in evaluating DDoS defense solutions below.

Lack of insights into advanced attacks in action. While researchers continue to discover advanced DDoS attacks [14, 15, 31, 32], researchers rarely have the opportunity to study these attacks in action. To better defend against real-world attacks that leverage advanced attack mechanisms [6, 10], researchers should be able to run and study these attacks in a high-fidelity network environment. Preparing such a network environment is not a trivial task. For example, to correctly study pulsing attacks, such as CICADAS [15], that exploit TCP congestion control, the network must provide a traffic generator [30] to provide realistic background traffic that honors network condition changes (e.g., congestion control).

Lack of closed-loop networks for DDoS detection. As the most valuable outcome of a DDoS detection solution is to facilitate DDoS mitigation, we must evaluate the collaboration of different detection and mitigation solutions. In other words, rather than evaluate DDoS detection and mitigation separately, we need to keep all DDoS defense components in the loop to evaluate each component individually and the entire defense as a whole, thus a closed-loop evaluation. For example, a detection solution must work with abrupt network changes caused by the mitigation effort. Indeed, Vishwanath *et al.* [36] found many network systems, including DDoS detection solutions, can present biased conclusions if evaluated under non-closed-loop networks.

Lack of collateral damage analysis in filter-based mitigation. Despite the network community is slowly adopting fine-grained mitigation solutions for disseminating filtering rules (e.g., BGP Flowspec [24]), the limited hardware filter space [7, 12] is insufficient in mitigating DDoS attacks that do not contain common packet signatures [13, 32]. Network operators often use limited filters to mitigate attacks but at the cost of disabling access to non-attacking networks. Soldo *et al.* [34] proposed a set of filter generation methods to study the trade-off between limited filters and collateral damage. However, the performance of such these methods is highly dependent on the IP address locality. Therefore, we must provide an emulation environment with realistic IP addresses, only then can we evaluate such DDoS mitigation solutions with confidence.

To facilitate sound empirical evaluation of DDoS defense solutions, we propose, design, and develop an emulation platform for evaluating DDoS defense solutions with high fidelity. Referred to as the **DDoS SandBox** (or simply, the SandBox), it has the following capabilities:

1. Generation of inferred Internet topologies at the level of autonomous systems (ASes);
2. Assignment of realistic IP address spaces to ASes;
3. Routing and packet forwarding functions of each AS;
4. Packet-level mimicry of real network traffic that honors network condition changes (e.g., congestion); and
5. A simple usage model with high experiment portability.

2 Related Work: Emulation Systems

Dummynet [5] and Modelnet [35] are two early network emulation systems that support unmodified applications. They have fixed components that render the discussed missing gaps above difficult to close. For example, Modelnet employs Click [16], a software router, to configure and route traffic in the emulation network. This fixed design choice imposes burdens in deploying different routing implementations, which makes BGP-related DDoS defense techniques [1, 26, 28] difficult to evaluate. Later emulation platforms, such as described in [2,3,19,27,37], unanimously adopted Linux namespaces [22] to provide each process its own abstracted and isolated system resources with a single system kernel. For example, one type of Linux namespace, the Linux network namespace [21], can assign a unique network stack to each process, which allows the process to have its own routing table. Thus, network emulation tools can easily spawn "*hosts*" by assigning processes different network namespaces. Notably, mininet [19], a popular network emulation tool, also relies on Linux namespaces. Mininet then utilizes *cgroup* [20] to partition system resources to each process (e.g., set a maximum CPU utilization for a host – a process in its own namespaces). A more recent work, Containernet [27], extends mininet to support Docker. It encourages users to create self-containing software images to mitigate deployment issues.

A namespace-based emulation system offers low overhead when compared to virtual machines (VM) and many VM-based testbeds; the former emulates everything with a single operating system (OS) kernel while the latter emulates OS kernels but could incur additional interface translation and storage overheads. Our SandBox leverages Containernet to realize its capabilities. Specifically, the Docker support in Containernet allows us to create self-contained software images quickly, and its underlying mininet programming interfaces enable us to program the links between emulation nodes. These features create a solid foundation for us to close the missing gaps in a foreseeable amount of time.

3 DDoS SandBox

3.1 Design Considerations

Using DDoS SandBox. The DDoS SandBox facilitates network operators to evaluate a defense solution in an emulation environment that mimics a real network. Meanwhile, DDoS researchers can also utilize the SandBox to gain insights into different DDoS attacks and defense solutions. Emulating a distributed system with a wide range of applications is a challenging task. Experimenters often run into issues such as fixing software dependencies or configuring network routers manually in an experiment. Thus, in the SandBox, we reduce the *deployment friction* by automating as many components as possible. In the meantime, the system remains fully customizable. For example, users can drop into a shell terminal of an arbitrary router node, and add or remove its network interfaces as they wish.

A mini Internet. Both advanced attacks and defense research projects require the emulation system to provide support for basic network functions (e.g., correct router ICMP behavior for *traceroute*). Ideally, the system should provide a *mini* Internet that is functionally equivalent to the real Internet. Only then can we study the latest DDoS attacks and evaluate defense solutions in a closed-loop environment. This includes but is not limited to 1) realistic AS-level IP space assignments based on the real-world BGP announcements, 2) BGP routing, and 3) closed-loop background traffic that reacts to the network conditions in real-time.

Elastic emulation fidelity. Today, anyone can have easy access to bare-metal servers with 100-core processors and 100s GBytes of memory from major cloud providers [33]. The modern hardware can easily emulate a small to medium network at high fidelity. However, emulation fidelity is not only about scalability. An experiment may require specific hardware for emulation (e.g., using a programmable switch

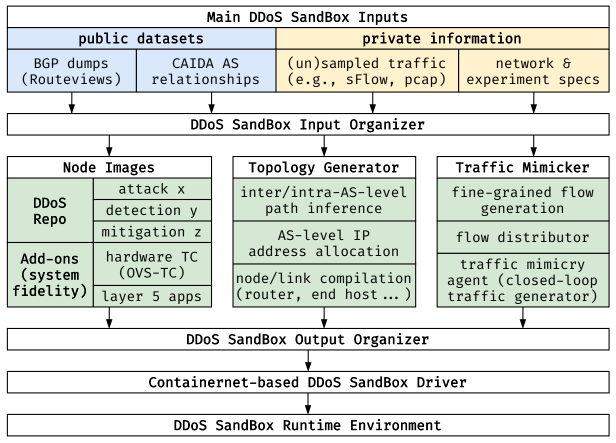
Main DDoS SandBox Inputs
public datasets
private information
BGP dumps (Routeviews)
CAIDA AS relationships
(un)sampled traffic (e.g., sFlow, pcap)
network & experiment specs
DDoS SandBox Input Organizer
Node Images
DDoS Repo
attack x
detection y
mitigation z
Add-ons (system fidelity)
hardware TC (OVS-TC)
layer 5 apps
Topology Generator
inter/intra-AS-level path inference
AS-level IP address allocation
node/link compilation (router, end host ...)
Traffic Mimicker
fine-grained flow generation
flow distributor
traffic mimicry agent (closed-loop traffic generator)
DDoS SandBox Output Organizer
Containernet-based DDoS SandBox Driver
DDoS SandBox Runtime Environment

Figure 1: DDoS SandBox Architecture

for DDoS detection in the data plane). The emulation system must also consider supporting additional physical hardware, and as such, we can both offload the emulation load and increase the overall emulation fidelity.

3.2 System Components

We introduce the system inputs and main SandBox components at a high level, as shown in Fig. 1. The blue colored boxes are open source software or dataset that anyone can acquire, the yellow colored boxes are information private to our users, and the green colored boxes are the main SandBox components.

Input. The SandBox creates a functional mini Internet with the following inputs: BGP table dumps from RouteViews [25], CAIDA AS relationships [4], and traffic traces (e.g., sFlow or a pcap trace, of a network). Typically, a user will input a benign traffic trace to generate the AS-level topology and background traffic in which the user can later inject attack traffic using the DDoS repository of attacks (which we describe further in the **Node Images** paragraph). However, the user can also leverage attack traces to study a specific instance of an attack. We reduce the input effort from users by automating data processing tasks for the public datasets. Additionally, users can feed network and experiment specifications to increase the emulation fidelity. For example, network operators can specify their intra-AS topologies and their upstream AS's partial intra-AS topologies to evaluate a defense solution of choice.

Topology Generator. The primary objective of Topology Generator is to create the blueprint of a mini Internet. Specifically, it leverages the user input data to: 1) create an inferred AS-level network that mimics a section of the Internet with regard to the input trace file, and assign realistic IP prefixes to each AS, 2) automate the network configuration for each network device (e.g., populate router and BGP configuration files that include information such as BGP prefix ownership and neighbors), and 3) attach nodes (e.g., traffic generators, DDoS bots, DDoS defense modules) to their belonging ASes. Of course, if the user provides detailed intra-AS topology information, the SandBox will reflect such information in the blueprint. Otherwise, we only create one router to represent an AS. While users are welcome to implement their own routers in the SandBox, we believe many users may not have strong preferences of the router implementation, as long as it is a realistic component. Thus, we provide a reference router that is based on Quagga routing suite [11].

Node Images. In the SandBox, we define any computational device, virtual or physical, as a node, and we package the software environment of the nodes as node images to improve the experiment portability. For example, if we instantiate a node using our reference router image, the node can 1) read/announce BGP prefix ownership populated by Topology Generator and 2) configure its forwarding table based on BGP announcements from its neighbors. Users can create a variety of node images containing any applications. For example, we can create an end host image with the ability to generate background network traffic by including a traffic generator. Similarly, to study different DDoS defense solutions, we plan to create a DDoS repository containing images of well-received attack strategies and defense solutions, thereby reducing the need for users to obtain attack traffic traces. We can then evaluate defense solutions under different attacks in realistic network environment.

Traffic Mimicker. Traffic Mimicker utilizes input traffic traces of a real-world network to create closed-loop background traffic in the SandBox. It generates fine-grained flow snapshots of ongoing flows based on the input traffic trace. For example, every second Traffic Mimicker may generate a snapshot of each 5-tuple network flow in the trace. Then, the flow distributor distributes a series of flow snapshots of each flow to the responsible end hosts with matched IPs. The example below shows a Comcast residential IP *23.24.100.123* communicating with *Amazon.com* at IP *205.251.242.103* (the snapshots were taken every second).

```
_________ Example Output of Traffic Mimiker _________
 ['23.24.100.123', 2333, '205.251.242.103', 443,
  'TCP', 1, [50, 10, ...], [500, 1000, ...]]
```

We also see that the residential IP sends 50 KB of data to Amazon in the first second, then 10 KB of data in the next second. In return, Amazon.com sends 500 KB and then 1 MB of data back in two seconds. Finally, each traffic mimicry agent creates corresponding network sockets and communicates with the destination specified in each flow snapshot.

3.3 Proof of Concept (PoC) Implementation

We choose Containernet as the PoC driver to implement the blueprint produced by Topology Generator. Specifically, we use Containernet to instantiate node images in Docker (e.g., the reference router image) and link instantiated nodes. The implementation also allows us to pass hardware network interfaces to a router node to achieve high-performance *traffic control* (*TC*) [23,29] support.

4 Experiments

Goals. To conduct a preliminary evaluation of our PoC, we first validate the correctness of Topology Generator via traceroute tests, and we then evaluate the single-host system scalability by analyzing network instantiation time with respect to the number of routers in the network. We plan on testing other aspects of the DDoS SandBox, such as the Traffic Mimicker, in our future work.

Setup. To create the AS topology, we feed a sampled sFlow trace from an IXP in the United States as our *only* private information input and two public information datasets (shown in Fig. 1) into the SandBox. Furthermore, we also attached several *traceroute*-enabled end hosts to two ASes within the AS topology. We conduct our evaluations on two distinct environments: 1) a Hyper-V VM utilizing three cores on an Intel i7-4970 processor with 24 GB of memory, and 2) a bare metal Amazon EC2 C5d instance utilizing 96 virtual cores with 192 GB of memory. Each Quagga router represents a single AS, and Docker version 19.03 is used to create and instantiate the router Docker image.

Traceroute Test. To test network connectivity and show that each AS is allocated realistic IP addresses, we perform several traceroute tests between ASes in the SandBox environment. An example traceroute is shown in Fig. 2. This traceroute (without *TC* policies) shows the route packets take between a randomly selected educational network and an arbitrary IP at a major cloud provider. The packets generated at the educational network flow through its upstream AS and Internet2 to reach the cloud IP. This reflects the real-world Internet AS topology, and we can find a corresponding AS-level path on *bgpview.io*. Furthermore, note that the SandBox provides a realistic IP assignment to each AS based on its real-world prefix ownership.

```
root@a12145h0:/# traceroute 3.13.0.2
traceroute to 3.13.0.2 (3.13.0.2), 30 hops max, 60 byte packets
 1  129.82.0.1 (129.82.0.1)  0.457 ms  0.426 ms  0.417 ms
 2  129.82.255.255 (129.82.255.255)  0.416 ms  0.410 ms  0.406 ms
 3  129.19.255.251 (129.19.255.251)  0.489 ms  0.357 ms  0.371 ms
 4  64.57.31.245 (64.57.31.245)  0.377 ms  0.379 ms  0.381 ms
 5  3.13.0.2 (3.13.0.2)  0.570 ms  0.568 ms  0.565 ms
```

Figure 2: An example traceroute test in DDoS SandBox

Network Instantiation Time. We benchmark the scalability of the DDoS SandBox by measuring the time it takes to: 1) create the reference router nodes, 2) populate configuration files for each node, 3) set up *veth* devices for each node, and 4) allocate IP addresses to each node. The main factor that affects network instantiation time is the number of nodes (or routers in this case) that need to be created in the SandBox. Note, there are many factors other than the number of nodes that can affect the system instantiation time, albeit to a smaller degree – for example, if an AS has a high degree of neighboring connections, the system may spend more time in creating *veth* devices and generating Quagga routing configuration files. Fig. 3 shows the network instantiation

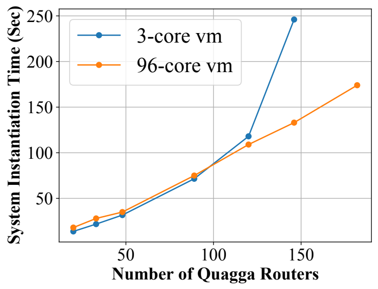

Figure 3: DDoS SandBox Instantiation Time

time with respect to the number of routers instantiated in the network. The two machines spend a similar amount of time to instantiate networks that have a limited amount of routers (i.e., less than 120). In fact, due to its high processor frequency, the 3-core Hyper-V VM performs slightly faster than the 96-core Amazon EC2 VM. However, for networks with more than 120 routers, the instantiation time for the Hyper-V machine increases exponentially due to its limited memory space, whereas the Amazon EC2 VM continues to follow a linear trend in instantiation time. Fig. 3 clearly shows that the SandBox can instantiate relatively large-scale networks within a relatively short amount of time.

5 Future Work and Conclusion

The DDoS SandBox is an ongoing project that is under active development. We are integrating more reference node images into the SandBox, including Traffic Mimicker, and reducing the required efforts to include physical devices. We plan to experiment with the scalability of the SandBox across multiple servers and investigate different approaches to extrapolate experiment results to the Internet scale. We are investigating alternatives to Containernet such as Container Network Interface (CNI) plugins [8] that have better support and system compatibility in the long run. Our ultimate goal for the DDoS SandBox is to run sound empirical DDoS attack and defense experiments. We hope results derived from the SandBox can draw more attention from network operators, and pave the way for the deployment of various defense systems. We will first implement the advanced DDoS attacks and well-received defense solutions, and provide them as reference node images in the SandBox for users to run general DDoS experiments. To the best of our knowledge, this is the first attempt to bridge the missing gaps for sound empirical DDoS experiments, and we have shown initial success in bridging those gaps in our PoC. Our PoC system is open source, and for more design details, the code is available on: `https://github.com/DDoS-SandBox`.

References

[1] Joe Abley and Kurt Erik Lindqvist. Operation of Anycast Services. https://tools.ietf.org/html/rfc4786, 2006.

[2] Jeff Ahrenholz. Comparison of core network emulation platforms. In *Military Communications Conference (MILCOM)*, 2010.

[3] G. Bonofiglio, V. Iovinella, G. Lospoto, and G. Di Battista. Kathará: A container-based framework for implementing network function virtualization and software defined networks. In *2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, 2018.

[4] CAIDA. AS Relationships. https://www.caida.org/data/as-relationships/, 2020.

[5] Marta Carbone and Luigi Rizzo. Dummynet revisited. *ACM SIGCOMM Computer Communication Review*, 40(2):12–20, 2010.

[6] Catalin Cimpanu. 'Carpet-bombing' DDoS attack takes down South African ISP for an entire day. https://www.zdnet.com/article/carpet-bombing-ddos-attack-takes-down-south-african-isp-for-an-entire-day/, 2019.

[7] Cisco. IP Addresses and Services Configuration Guide for Cisco NCS 540 Series Routers, IOS XR Release 6.3.x. https://www.cisco.com/c/en/us/td/docs/iosxr/ncs5xx/ipaddress/63x/b-ip-addresses-cg-63x-ncs5xx/b-ip-addresses-cg-63x-ncs5xx_chapter_01.html.

[8] Cloud Native Computing Foundation. CNI - the Container Network Interface. https://github.com/containernetworking/cni, 2020.

[9] FastNetMon. Testimonials. https://fastnetmon.com/client-testimonials, 2020.

[10] Töma Gavrichenkov. Reflection DDoS last week (was: syn flood attacks from NL-based netblocks). https://seclists.org/nanog/2019/Aug/415, 2019.

[11] Paul Jakma and David Lamparter. Introduction to the quagga routing suite. *IEEE Network*, 28(2):42–48, 2014.

[12] Juniper. IP Addresses and Services Configuration Guide for Cisco NCS 540 Series Routers, IOS XR Release 6.3.x. https://www.juniper.net/documentation/en_US/junos/topics/concept/firewall-filter-ex-series-overview.html.

[13] Min Suk Kang, Virgil D Gligor, Vyas Sekar, et al. Spiffy: Inducing cost-detectability tradeoffs for persistent link-flooding attacks. In *NDSS*, 2016.

[14] Min Suk Kang, Soo Bum Lee, and Virgil D. Gligor. The Crossfire Attack. In *2013 IEEE Symposium on Security and Privacy*, 2013.

[15] Yu-Ming Ke, Chih-Wei Chen, Hsu-Chun Hsiao, Adrian Perrig, and Vyas Sekar. CICADAS: Congesting the Internet with Coordinated and Decentralized Pulsating Attacks. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, 2016.

[16] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click Modular Router. *ACM Trans. Comput. Syst.*, 18(3):263–297, August 2000.

[17] Faris Bugra Kokulu, Ananta Soneji, Tiffany Bao, Yan Shoshitaishvili, Ziming Zhao, Adam Doupé, and Gail-Joon Ahn. Matched and Mismatched SOCs: A Qualitative Study on Security Operations Center Issues. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, 2019.

[18] Warren Kumari and Danny McPherson. Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF). https://tools.ietf.org/html/rfc5635, 2009.

[19] Bob Lantz, Brandon Heller, and Nick McKeown. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, 2010.

[20] man7.org. cgroups - Linux control groups. http://man7.org/linux/man-pages/man7/cgroups.7.html, 2020.

[21] man7.org. ip-netns - process network namespace management. http://man7.org/linux/man-pages/man8/ip-netns.8.html, 2020.

[22] man7.org. namespaces - overview of Linux namespaces. http://man7.org/linux/man-pages/man7/namespaces.7.html, 2020.

[23] man7.org. tc - show / manipulate traffic control settings. http://man7.org/linux/man-pages/man8/tc.8.html, 2020.

[24] Pedro Marques, Nischal Sheth, Robert Raszuk, Barry Greene, Jared Mauch, and Danny McPherson. Dissemination of Flow Specification Rules. https://tools.ietf.org/html/rfc5575, 2009.

[25] David Meyer. University of Oregon Route Views Project. *University of Oregon*, 2001.

[26] Giovane C.M. Moura, Ricardo de O. Schmidt, John Heidemann, Wouter B. de Vries, Moritz Muller, Lan Wei, and Cristian Hesselman. Anycast vs. DDoS: Evaluating the November 2015 Root DNS Event. In *Proceedings of the 2016 Internet Measurement Conference*, IMC '16, 2016.

[27] M. Peuster, H. Karl, and S. van Rossem. MeDICINE: Rapid prototyping of production-ready network services in multi-PoP environments. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, 2016.

[28] Matthew Prince. A Brief Primer on Anycast. https://blog.cloudflare.com/a-brief-anycast-primer/, 2011.

[29] Ido Schimmel. Mellanox OVS TC. https://github.com/Mellanox/mlxsw/wiki/OVS, 2019.

[30] Bianca Schroeder, Adam Wierman, and Mor Harchol-Balter. Open versus closed: A cautionary tale. In *3rd Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX, 2006.

[31] Huasong Shan, Qingyang Wang, and Calton Pu. Tail Attacks on Web Applications. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS'17, 2017.

[32] Lumin Shi, Devkishen Sisodia, Mingwei Zhang, Jun Li, Alberto Dainotti, and Peter Reiher. The Catch-22 Attack (Poster). In *Annual Computer Security Applications Conference (ACSAC)*, 2019.

[33] Julien Simon. Now Available: New C5d Instance Sizes and Bare Metal Instances. https://aws.amazon.com/blogs/aws/now-available-new-c5d-instance-sizes-and-bare-metal-instances/, 2019.

[34] Fabio Soldo, Athina Markopoulou, and Katerina Argyraki. Optimal Filtering of Source Address Prefixes: Models and Algorithms. In *IEEE INFOCOM 2009*, April 2009.

[35] Amin Vahdat, Ken Yocum, Kevin Walsh, Priya Mahadevan, Dejan Kostić, Jeff Chase, and David Becker. Scalability and Accuracy in a Large-Scale Network Emulator. *SIGOPS Oper. Syst. Rev.*, 36:271–284, December 2003.

[36] Kashi Venkatesh Vishwanath and Amin Vahdat. Evaluating distributed systems: Does background traffic matter? In *USENIX Annual Technical Conference*, pages 227–240, 2008.

[37] Philip Wette, Martin Dräxler, Arne Schwabe, Felix Wallaschek, Mohammad Hassan Zahraee, and Holger Karl. Maxinet: Distributed emulation of software-defined networks. In *2014 IFIP Networking Conference*, pages 1–9. IEEE, 2014.