Evaluating Webpage Performance and DNS latency during a DoS attack on SDN Controller

Kevin Zhao
New York University
kz667@nyu.edu

***Abstract*—Software defined networking (SDN) is a paradigm that allows centralized programmability and visibility of a local area network. SDN is sometimes referred to as a "network operating system," in which the controller provides an abstraction layer to the underlying switches, exchanging messages over a southbound API (ie. OpenFlow)—analogous to a traditional OS providing an abstraction to the underlying hardware. Researchers have surveyed the topic of SDN, proposed models for DDoS mitigation on the southbound interface, weighed the network performance of popular controllers, and enumerated various security challenges and threats against the architecture.**

In this paper, we assess an enterprise controller during a naïve request flooding attack through the following KPIs: the amount of time required for a Ubuntu VM to load each of the top 50 Alexa sites in the U.S. and resolve a DNS query that is mediated by a DNS filtering app at the controller, as well as the controller's ability to maintain operability. A Python script is used to deliver the attack from within the LAN. Amidst the attack, we have a VM on the GNS3 emulator browse each of the top 50 Alexa sites in the U.S. and record their page load times using a sideloaded browser extension. The results indicate negligible disruption to page load time—an important caveat is that DNS queries are no longer mediated by the Network Protector app due to connection loss during the DoS event. Moreover, an increase of <0.02 seconds in DNS resolution time is seen between the worst and best performing cases.

Keywords— SDN, HPE VAN Controller, Denial-of-Service, network performance, GNS3, Raspberry Pi

I. Introduction

Performance and security are tantamount to one another in a vertically integrated or software defined network environment. Organizations are susceptible to increasing configuration and management complexity as a network infrastructure is scaled upward, owing to vendor specific software and the need for specialized and manual configuration, resulting in greater capital expenditure and opportunity costs. Software-defined networking employs commodity switching and a centralized logical backbone to ameliorate these complexities, insofar as performance and security criteria are met. In our literature review, we have found various proposed mechanisms for mitigating DDoS attacks on the southbound interface, performance benchmarking between controllers of choice using an emulation tool like Mininet, which leverages process-based virtualization for each device in the topology [6], and a study measuring page load time where all HTTP/DNS requests are mediated by a Floodlight controller with an artificially added latency of zero to 50ms.

Our key contributions in this paper include:

- architecting a basic software-defined network by leveraging GNS3 and VMWare Workstation, a separate host for our controller (HPE VAN version 2.8.8), and a Raspberry Pi 4 Model B (primarily used for executing the DoS attack) on a residential network
- evaluating DNS latency and impact to page load times when browsing to each of the top 50 Alexa sites browsed in the U.S. using Google Chrome before and during a southbound API (OpenFlow) flooding attack
- testing the resiliency of the Network Protector app (installed atop the HPE VAN controller), particularly its ability to provide DNS-level filtering services as the southbound interface becomes saturated

The organization of the paper is as follows: Section II describes the related work and the limitations of current methods. In Section III, we describe the elements of our experiment and the methodology employed to realize the study. Section IV provides the motivating example behind our work. Section V describes investigates the tests we used in our experimentation with page load time when attacking the controller at different throughput rates. We conclude in Section VI and discuss future work.

II. Related Research

Our work is inspired by existing literature covering DDoS mitigation mechanisms on the OpenFlow channel, the viability of SDN in residential networks, and performance comparisons between controllers of choice.

Approaches to DDoS mitigation on the southbound interface converge on the singular objective of improving accuracy whilst minimizing overhead. Nimbus, for example, is a module installed at the controller to infer statistics from traffic samples sent to a load-balanced, resource-scaled pool of VMs, providing what the author coins, "attack-prevention as a service" [9]. Wang, Jia, and Zhu (2015) propose an "entropy-based lightweight DDoS flooding attack detection model" at an OpenFlow-enabled switch at the network edge to periodically collect flow entries across OF switches for machine learning-based analysis. Other variations like OpenSketch, which distinguishes a measurement data plane for malleable and programmable data collection [8], claim a novel model for intelligent data sampling.

By contrast, DELTA [1] takes a proactive approach to satisfying performance and security criteria through its "blackbox fuzzing" framework, which aims to unravel vulnerabilities in the flow operations between and within the SDN stack. Control flow vulnerabilities are taxonomized as symmetric, asymmetric, intra-controller and "non-flow" in nature. Their study on popular controllers like Floodlight yields

seven zero-day scenarios—two of which compromise the availability of the SDN controller by altering the payloads of ECHO and STAT telemetry messages to induce a connection teardown.

Although we have briefly described some studies on DDoS mitigation in the SDN context, the study assumes a successful attack and therefore, our topology is free of interloping firewall/IDS appliances. We elect to use a Python script through a Raspberry Pi to deliver the flooding attack despite DELTA's capability of producing a similar asymmetric attack due to relative ease of deployment and limited scope of experimentation.

Taylor, Guo, Shue, and Najd (2017) discuss the feasibility of converting residential-caliber routers into commodity switches capable of connecting to cloud-based SDN controllers over the OpenFlow protocol. From a sample of 270 U.S.-based participants on a residential internet connection, 90% were within 50ms RTT of two VMs hosted by Amazon, Google, Microsoft, or Digital Ocean—latency values that fell in top 10 percent precluded adequate performance. Their findings are incorporated into a test of page load time, rationalized by the frequency of web browsing on a residential connection and that for each of the top 100 Alexa sites, the number of short requests "from DNS, website servers, CDN servers or advertisement networks" that would require controller elevation serve as the "worst-case scenario". To emulate wide-area network latency, the on-premise controller induced an added delay of 0ms, 25ms, or 50ms (e.g. through a modification at the host OS kernel using a command like *sudo tc qdisc*). At the 50th percentile, the median PLT increased from four seconds (without OpenFlow) to six seconds (when connected to the controller with an artificially induced 50ms latency). Similarly, our study measures page load time, but within the context of a DoS attack and DNS filtering by an SDN application.

Papers focused on conducting a network performance evaluation of SDN controllers lend insight into the various KPIs, measurement techniques, and tools employed. Badotra and Panda (2019) simulate a hierarchical network topology of 27 hosts and 13 OpenvSwitch switches using Mininet in one Ubuntu VM while two other VMs respectively host the OpenDayLight and ONOS controllers. These instances are tethered to a layer-2 bridge created on the hypervisor. They predicate from their gathered metrics like burst rate, throughput, bandwidth, and response times between control plane messages such as those querying for tables, ports, and flow statistics using Wireshark and iPerf, that the OpenDaylight controller outperforms the ONOS controller. Zhu et al. (2019) provides a list of benchmarking metrics & tools used to evaluate 9 controllers in different network scenarios. Some metrics of interest include: (a) throughput—defined as the number of PACKET_IN and PACKET_OUT messages per unit time, (b) latency—defined as the round trip time between controller and OpenFlow switch, and (c) flow installation rate—the rate at which flows can be installed on each OpenFlow switch along a complete flow path. Although they state that metric outcomes are highly dependent upon the setup environment, their paper does not illustrate the topologies used during their assessment with different number of switches.

III. Methodology/Empirical Evidence/Hypothesis

This section describes the testbed architecture, metric-gathering toolshed, and experimental methodology. By measuring page load times and DNS response times, we seek to validate the scope of impact that a naïve DoS attack could have on an SDN network.

A. Testbed

The networking core spans three physical hosts, each tethered to a physical switch on their gigabit network interfaces.

a) Device 1: Equipped with GNS3 and VMware Workstation Pro, this host simulates an SDN-based LAN which uses NAT overload to communicate with the physical LAN

b) Device 2 (Sony VAIO VPCEB23FX, 2.26GHz Core i3-350M, 4GB RAM): hosts the HPE VAN SDN controller virtual machine in a type-2 hypervisor (VirtualBox)

c) Device 3 (Raspberry Pi 4 Model B 4GB 4GB RAM): delivers Python-based DoS attack and records network statistics into a text file every 15 seconds from active interface eth0

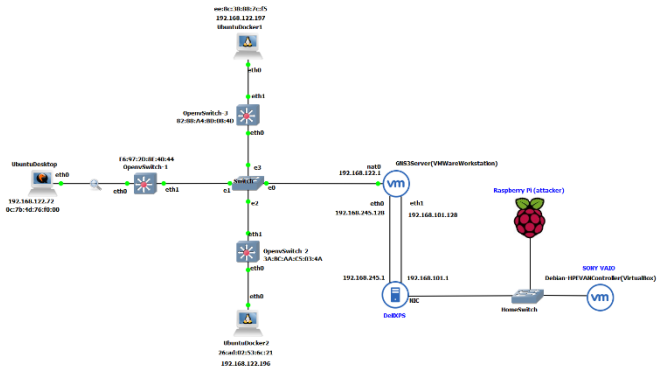

Fig. 1. A diagram of our testbed. The 192.168.122.0/24 network is virtualized on our Ubuntu GNS3 server. The server instance runs on the type-2 VMWare Workstation Pro hypervisor. The hypervisor is run as a process on a Dell XPS 9560. The XPS, Raspberry Pi 4, and SONY VAIO devices are physically linked to a gigabit switch.

B. Background

Before proceeding with experimentation, we must chiefly establish how DNS traffic moves across the testbed while the Network Protector app is active. When website example.edu is requested at the Ubuntu desktop, one or more DNS queries are produced (we focus on the IPv4 A record) and forwarded to OpenvSwitch-1 an OpenFlow flow has already been installed at OpenvSwitch-1 directing all UDP port 53 traffic to be elevated to the controller. The query is encapsulated in an OpenFlow PACKET_IN message and sent to the controller. The controller replies with a PACKET_OUT message containing the unmodified query if the site has not been blacklist in the Network Protector app, or a response containing an NXDOMAIN code if explicitly blocked. If the query is not blacklisted, it will be retransmitted by the commodity switch for standard DNS resolution.

C. *Test Scripts and Tools*

To automate the testing and metric gathering process, we authored and built upon Bash/Python scripts and used Wireshark, iPerf, sysstat, and the Page Load Timer extension (provided under MIT license).

Prior to each page load time measurement, the bash script executes commands to clear the OS DNS cache and browsing cache to ensure that we prompt the maximum number of OpenFlow elevations to the controller.

iPerf is used to measure the bandwidth between two hosts, where one endpoint acts as a client and the other as a server. Through a series of trials, we identified a bandwidth disparity of 390 Mbps to 930 Mbps between the Debian guest running in VirtualBox and serving the controller and host OS (Windows) on a gigabit NIC and 78 Mbps to 86 Mbps when the NIC is limited to 100Mbps. By running sysstat on the Raspberry Pi to record the throughput rate every 15 seconds over the course of each trial, we were able to determine that the Python script is able to maintain an average rate within 99% of the available bandwidth on the Windows host.

The Page Load Timer extension sideloaded into Google Chrome is used to measure the total page load time based on the sum of attributes gathered through the Navigation Timing API. We modify the source code to automatically download a CSV file containing the website URL and load time for each of the 50 sites. A Python script is used to consolidate the CSV files.

The attack script leverages the Python scapy, struct, and socket modules to craft OpenFlow 1.0 messages, encapsulate them in TCP packets, and send them into a live socket. To emulate flooding behavior, it arranges a handshake with the controller before sending more than 3000 OpenFlow messages into an open socket per loop iteration. Given a maximum transmission unit (MTU) size of 1514 bytes, each packet yields 20 bogus PACKET_IN requests.

Because the ping utility merely provides the round trip time of a ICMP request/reply message pair, which is independent of DNS resolution latency, we start Wireshark in-line between the Ubuntu desktop and OpenvSwitch-1 with a capture filter on DNS query/response packets for IPv4 A records as each of the top 50 Alexa sites in the U.S. is pinged—ergo resolved in the process.

IV. Testbed Evaluation

To understand what impact a successful flooding attack on OpenFlow listening TCP port 6633 has on DNS resolution time and page load time, we evaluate our testbed under several scenarios:

a) Standard DNS mediation by Network Protector app where the virtual NIC is bridged to a gigabit NIC and DNS cache is flushed after each ping

b) Standard DNS mediation by Network Protector app where the virtual NIC is bridged to a gigabit NIC and DNS cache is *not* flushed after each ping

c) Standard DNS mediation by Network Protector app as listening port 6633 is flooded and DNS cache is *not* flushed after each ping

d) Standard DNS mediation by Network Protector app as listening port 6633 is flooded, DNS cache is *not* flushed after each ping, and NIC to limited to Fast Ethernet 100Mbps

Based on the results given by Figure 2, the scenario where we have bounded the network device to operate at 100Mbps full duplex before delivering the attack presents the lowest page load time for the longest probability intervals on the y-axis. The quantitative results appear counterintuitive, but this is a mark of the OpenFlow 1.0 specification—the switch's attempt to contact the controller is stymied by the onslaught of bogus PACKET_IN

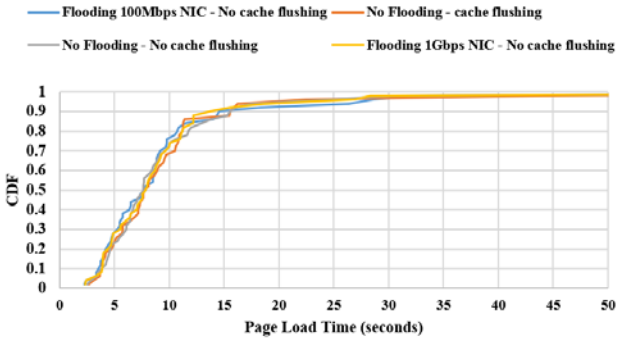

Fig. 2. A graph depicting the cumulative distribution function in relation to page load time across four test scenarios

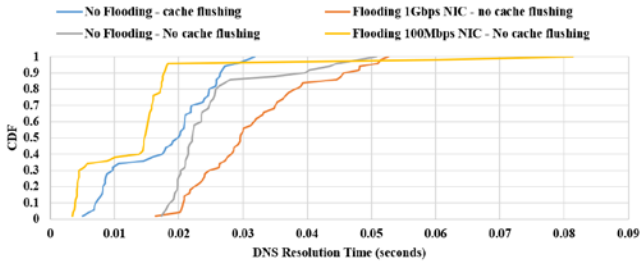

Fig. 3. A graph depicting the cumulative distribution function in relation to DNS resolution time across four test scenarios

messages that the controller must process. Consequently, the switch enters "emergency mode," resets the current TCP connection, and deletes all "normal entries" [4]. In the absence of flow entries, the switch converts into a traditional MAC learning switch. This means that DNS queries are no longer being mediated by the controller but rather immediately forwarded out to a public DNS service.

To summarize the results of Figure 2, the page load times at each percentile up to the 85th across the test scenarios vary by no more than two seconds. However, each scenario was only tried once and as a result, the PLT for our current (and only) trial cannot be interpreted as part of a larger set to distinguish what is anomalous. Because of the wide range of factors (e.g. DNS resolution, web server latency/load, and local computing resource availability) that may significantly influence page load

times, more trials for each scenario must be performed to reach definitive results.

Similarly, the same scenario (d) in Figure 3 yields the quickest resolution time for nearly all sites and achieves this by bypassing the Network Protector app in the absence of flow entries on OpenvSwitch-1. In other words, operability of the network remains intact, but the integrity of operational flows is compromised as a network service is circumvented. Our single trial also counterintuitively depicts the OS cache flushing test scenario as having overall faster DNS resolution times than the non-OS cache flushing scenario. We identified two contributing factors (a) anomalous DNS response times sometimes orders of magnitude greater and (b) variable number of DNS queries/responses affecting the average calculation. At the 50th percentile of all sites, the difference in DNS resolution time between scenarios (c) and (d) is less than 0.02 seconds.

V. Conclusion and Future Work

In this paper, we characterize the performance of the HPE VAN controller through page load time and DNS resolution from our Ubuntu desktop instance in GNS3. We emulate a naïve flooding attack that delivers OFPT_PACKET_IN messages from an address space of 10.0.0.0/8 and port range of 1-65535 (a stateful firewall would most likely identify this behavior as a DoS event). We tested across four unique scenarios, collecting metrics using tools like Wireshark, iPerf, sysstat, and the Page Load Timer browser extension. However, our results are limited due to the lack of trials and other local variables that must be controlled for optimal empirical accuracy.

Given the scope of experimentation, controller fault tolerance or scalability is not considered in our testbed. Techniques to improve controller latency (like controller clustering to improve the controller placement over WAN issue), redundancy and scalability have been explored in other works.

The OpenFlow channel in our testbed operates over plain TCP without any provision for TLS to protect the confidentiality/integrity of messages. Therefore, its impact to page load time and DNS resolution cannot be determined without additional work. Although the original OpenFlow specification states that "The switch and controller communicate through a TLS connection," our setup is near stock and does not provision TLS [4]. Moreover, the version one specification does not specify the set of TLS protocols allowed or how to reconcile interoperability. The lack of TLS implies that an adversary can impersonate another switch (as we have done) or controller or view the contents of an OpenFlow message.

We constructed a simplification of a production network—an organization opting to use SDN will have many operational flows in place to harmonize network functions like DNS, load balancing, and firewalling.

References

[1] Lee, S., Yoon, C., Lee, C., Shin, S., Yegneswaran, V., & Porras, P. A. (2017, February). DELTA: A Security Assessment Framework for Software-Defined Networks. In NDSS.

[2] Taylor, C. R., Guo, T., Shue, C. A., & Najd, M. E. (2017, November). On the feasibility of cloud-based SDN controllers for residential networks. In 2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN) (pp. 1-6). IEEE.

[3] Badotra, S., & Panda, S. N. (2019). Evaluation and comparison of OpenDayLight and open networking operating system in software-defined networking. Cluster Computing, 1-11.

[4] OpenFlow Switch Specification (2009): https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf. Accessed 13 August 2020

[5] Zhu, L., Karim, M. M., Sharif, K., Li, F., Du, X., & Guizani, M. (2019). SDN controllers: Benchmarking & performance evaluation. arXiv preprint arXiv:1902.04491.

[6] Mininet: http://mininet.org/overview/. Accessed 13 August 2020

[7] Wang, R., Jia, Z., & Ju, L. (2015, August). An entropy-based distributed DDoS detection mechanism in software-defined networking. In 2015 IEEE Trustcom/BigDataSE/ISPA (Vol. 1, pp. 310-317). IEEE.

[8] Yu, M., Jose, L., & Miao, R. (2013). Software Defined Traffic Measurement with OpenSketch. In Presented as part of the 10th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 13) (pp. 29-42).

[9] Miao, R., Yu, M., & Jain, N. (2014). NIMBUS: cloud-scale attack detection and mitigation. Acm sigcomm computer communication review, 44(4), 121-122.