DeepContext: An OpenFlow-Compatible, Host-Based SDN for Enterprise Networks

Mohamed E. Najd and Craig A. Shue
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609
Email: {menajd, cshue}@cs.wpi.edu

***Abstract*—The software-defined networking (SDN) paradigm promises greater control and understanding of enterprise network activities, particularly for management applications that need awareness of network-wide behavior. However, the current focus on switch-based SDNs raises concerns about data-plane scalability, especially when using fine-grained flows. Further, these switch-centric approaches lack visibility into end-host and application behaviors, which are valuable when making access control decisions.**

In recent work, we proposed a host-based SDN in which we installed software on the end-hosts and used a centralized network control to manage the flows. This improve scalability and provided application information for use in network policy. However, that approach was not compatible with OpenFlow and had provided only conservative estimates of possible network performance.

In this work, we create a high performance host-based SDN that is compatible with the OpenFlow protocol. Our approach, *DeepContext*, provides details about the application context to the network controller, allowing enhanced decision-making. We evaluate the performance of DeepContext, comparing it to traditional networks and Open vSwitch deployments. We further characterize the completeness of the data provided by the system and the resulting benefits.

KEYWORDS

Software-defined networking; OpenFlow; host-based agents.

I. INTRODUCTION

Enterprise network operators typically lack a detailed understanding of the traffic that flows across their devices. While these operators may deploy middleboxes, such as intrusion detection systems (IDSes) or firewalls, these operators remain blind to intra-subnet traffic that traverses a switched path without reaching the middleboxes. Even for traffic that does traverse a middlebox, these devices can only make inferences about the type of traffic based on packet header information or deep packet inspection on the packet payload. While such inferences may be correct for legitimate traffic, malicious traffic may attempt to blend in with innocuous traffic to avoid detection.

The software-defined networking (SDN) paradigm has the potential to greatly enhance network understanding for network operators. Rather than using highly aggregated, coarse-grain rules that forward solely on destination address or prefix, the OpenFlow protocol allows network operators to specify fine-grain rules to direct traffic based on the standard network flow 5-tuple (IP_{src}, IP_{dest}, Transport Protocol, Port_{src}, Port_{dest}) and other header fields. When switches lack a rule that matches a packet, they will consult a logically-centralized controller, which makes a decision about the traffic and issues a set of rules to the SDN switches to forward the packet. Unfortunately, the memory available at physical OpenFlow switches is limited, leading to well-documented scalability concerns when using fine-grained flows [6], [19], [25]. As network aggregation points, these switches see a large number of distinct flows, but lack the capacity to maintain the detailed forwarding rules.

In our own previous work [22], called the TaylorSDN, we eliminate the data-plane scalability concern in SDN switches by moving the SDN agent into the communicating end-hosts for fine-grained flows. The physical network switches can be either traditional, non-OpenFlow switches or implement OpenFlow with coarse-grained rules, as suggested by Curtis *et al.* [6], to avoid the scalability concerns. The TaylorSDN further addressed the lack of host context by sending application information to the SDN controller. Unfortunately, the TaylorSDN was not OpenFlow-compatible and relied upon its own protocol. Further, the TaylorSDN used an on-demand data collection approach for host context which introduced delays in new flow processing.

Given the inherent limitations of switch-based SDNs and our initial host-based SDN approach, we ask two research questions: *1) How can we feasibly create a host-based SDN that integrates with the existing OpenFlow community while providing additional context? 2) What is the best design for such a host-based SDN from both a performance and data utility perspective?* While the detailed context from such a system would likely be useful for enhancing organizational security, we focus on the performance and the utility of the contextual data provided by the system. We consider the integration of our system with security tools to be outside the scope of this work.

We created a new host-based SDN system, which we call the *DeepContext* SDN, that is designed to provide host information to an SDN controller with minimal added latency. By using hosts to store rules, DeepContext avoids the data-plane scalability concerns in OpenFlow for fine-grained flows. In our design, we leverage a popular existing OpenFlow agent, called Open vSwitch [16], that we extend to provide host contextual

information that can aid controllers in making decisions.

In exploring this direction, we make the following contributions:

- **Integration of Host Context into the OpenFlow Protocol:** We have identified a straightforward means to include host-context information into the existing OpenFlow protocol in a way that can support existing OpenFlow controllers and applications without requiring modifications for each.
- **A New, OpenFlow-compatible Host-Based SDN:** Our DeepContext SDN agent provides detailed information about the process creating new flows, including the application's name, executable path, associated user ID, group ID, and even command line options. DeepContext uses a custom module on the popular Floodlight controller [2] to allow operators to make standard OpenFlow control decisions that leverage the additional host context.
- **Performance and Completeness Evaluation:** To provide context for the innovations in DeepContext, we compare the system with an unmodified Open vSwitch network. Further, we compare the DeepContext SDN with 1) our own previous host-based SDN [22] and 2) with `ident++` [15], a reactive host querying approach. We quantify the overheads of adding context and host-controller communication to the network communication.

The remainder of this work is structured as follows. In Section II, we describe related work. In Section III, we describe extensions to the OpenFlow protocol design. In Section IV, we introduce our DeepContext SDN approach. In Section V, we evaluate each of the SDN approaches. We present discussion in Section VI and conclude in Section VII.

II. Related Work

We briefly describe OpenFlow, work related to fine-grained flow scalability, and research surrounding host-context in SDNs.

A. OpenFlow Fine-Grained Flow Scalability

OpenFlow switches maintain a local cache of rules to forward packets. While OpenFlow allows the specification of very fine-grain rules, the physical memory on the OpenFlow switches is limited. In particular, Curtis *et al.* [6] raised concerns that modern OpenFlow switches cannot handle the large number of fine-grained flows that a medium-sized organization would need. To overcome this, those authors encouraged coarse-grain rules with broad matching criteria to reduce the switch storage requirements. Unfortunately, since these broad rules will match multiple flows, they cannot be used to ensure each new connection will reach the network controller. Other work found that some switches could support only between 750 and 2,000 flows in hardware and that software tables may be needed for additional flows [12]. Even some high-end data center switches have a maximum of 97,000 OpenFlow rules [9]. This capacity issue in OpenFlow switches makes them a target for denial-of-service attacks [19].

Given these scalability concerns, Wang *et al.* [25] devised a tunneling strategy in which the physical OpenFlow switches use coarse-grained rules to divert traffic to host-based hypervisor virtual switches (using Open vSwitch) to provide fine-grained flow enforcement. Unfortunately, this inflates network paths by directing packets via hosts and adds latency. Our proposed approach also uses a host-based SDN, but ours runs within the host operating system itself, rather than in a hypervisor. This allows us to provide detailed host feedback while avoiding the need for virtualization or indirect paths.

B. Extracting Context from End-Hosts

The Ethane SDN [3] allows operators to write more detailed network policy including entities such as end-host machines and access points. However, Ethane, like OpenFlow, is switch-based and lacks the end-host instrumentation needed for more detailed specification.

The work in HoNe [8] correlates network traffic with processes, but it is not an SDN approach and lacks centralized coordination. Dixon *et al.* [7] allow network administrators to use Virtual Machines (VMs) and Trusted Platform Modules (TPMs) on the end-hosts to enforce network policy, but the approach lacks the network visibility present in OpenFlow-based SDNs. Similarly, Parno *et al.* [17] use end-host TPMs to allow the hosts to attest to network state, such as the number of packets sent, and use a set of network verifiers to query hosts and provide time-limited authorization tokens for the hosts to communicate. These approaches do not provide the proactive controls of OpenFlow on a network-wide basis.

In another work, also named HONE [21], the authors introduce a system that allows a network controller to query end-hosts to gather statistics and measurements for traffic engineering. Our approach likewise instruments the network functions in the kernel, similar to our approach, and supports SDN controllers. However, our approach focuses on fine-grained flows and provides application context to a network controller, which is not part of HONE's design.

As a proposed successor to the `ident` [11] protocol, Naous *et al.* [15] describe `ident++`, a protocol that would allow a remote system to query for details about the application and other information associated with a flow. In conjunction with OpenFlow, the `ident++` protocol would allow a controller to reactively query a host for additional information. While that position paper does not implement the approach or address the scalability concerns of OpenFlow switches, it adds valuable host-based context. To enable a direct comparison, we implement their proposed approach and evaluate its performance as part of our work.

Additional approaches have examined how to gather more detailed context about end-hosts, such as mouse-clicks and keyboard presses [5] and application-specific sensors (e.g., in a web browser [13], [27]). Such sensors could be used to increase the information available in our approach.

Finally, in our own prior work [22], we constructed a host-based SDN that provides host context for its network

communication. While that host-based SDN was not named, we will refer to the system as the *TaylorSDN* (named after the first author) for easier reference. The TaylorSDN used Python scripts to communicate with an SDN controller and used the Linux `netfilter_queue` library to intercept new flows. The approach took roughly 17ms to authorize each flow and did not use the OpenFlow standard, making its controller incompatible with the rest of the SDN community. The TaylorSDN demonstrated the approach's utility for security by evaluating how one could detect drive-by download malware and attacks from email attachments based on the application's process hierarchy. In this work, we propose a new approach to enhance performance and interoperability and then compare it with our prior work and that of other approaches.

III. Extending the OpenFlow Protocol

The OpenFlow protocol header does not support options capable of expressing arbitrary data. Accordingly, encoding host and application context data into existing OpenFlow messages in a backwards-compatible way presents a challenge.

In Figure 1, we depict the headers and structure of an OpenFlow `OF_PACKET_IN` message in the unshaded region. This unshaded region represents the message an OpenFlow controller receives when it must make a decision about a new flow. Accordingly, we want to modify this message to include additional host context. In our design, we simply append this information to the end of the existing OpenFlow message, as shown in the shaded region of the figure.

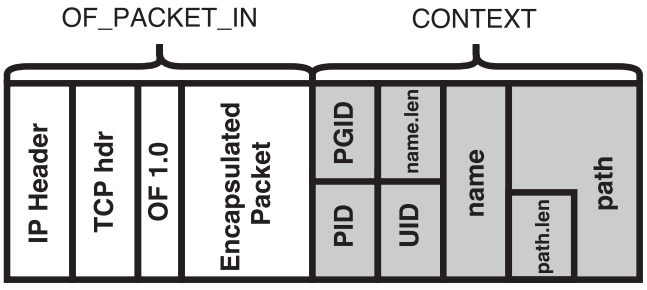

Fig. 1. The DeepContext `OFPT_PACKET_IN` modified packet with context appended to the trailer.

While appending the contextual information to the existing OpenFlow message is straightforward, it provides significant advantages. Some existing OpenFlow controllers will process the message and send it to applications without any negative effects. Frequently, controller applications look into the initial headers in the encapsulated packet to make their decisions, without examining the application's payload. Such applications do not examine the encapsulated packet's payload size or content and will not notice our additional contextual information. Thus, our approach is backwards-compatible with this type of controller application.

Applications that wish to consult the host context can examine the encapsulated packet's headers to determine the length of the packet payload and skip past it to begin processing our payload[1].

A final complication is the network's maximum transmission unit (MTU). Appending information to an OpenFlow message that is already at or near the boundary of the largest transmittable frame size may cause the packet to be too large for the underlying media. In this case, we would need to determine the size of the context and truncate the encapsulated packet to allow the context to fit. This is the same approach that OpenFlow takes when its own header would cause the packet to exceed the MTU. The only complication in our approach is that we must update the header lengths of the truncated encapsulated packet so that context-aware controller applications are able to correctly find the appended host context.

IV. DeepContext: Detailed Process Context

In building the DeepContext SDN, we have multiple design goals. We want to be OpenFlow-compatible to ensure the work appeals to the existing OpenFlow community. We further want to achieve high performance to ensure the approach is feasible in production networks. We want minimal modifications to the host's operating system, and avoid modifications to existing applications. Finally, we want to provide useful context about each process as it creates new flows to enable more detailed network policy.

To achieve these goals, we designed the DeepContext system to integrate with two existing SDN systems: Open vSwitch (OVS) and the Floodlight SDN controller. We provide a visual depiction of the system in Figure 2. We first describe the contextual information we acquire and why it is useful and then describe our modifications to OVS and Floodlight to leverage this context.

A. Context Tracking and Policy Impact

The end-host operating system must keep track of the information associated with each process. In the Linux operating system, this information is stored on a per-process basis in a kernel-level data structure called the `task_struct`. This data structure contains information about the user running the process, the executable path used to run the application, the command line options, the associated group, and even detailed information about process ancestry and threads. We call this task information the process's operating *context*.

The process context can be valuable information for a network controller. Network operators may wish to write policy on a per user, group, or application basis. For example, network operators may wish to write a policy such as "Only Alice may connect to 1.2.3.4 on port 80 and only if she uses `/usr/bin/chrome` with the `--disable-javascript` option." Such a policy can easily exclude any user-installed programs, unapproved browsers, and browsers with undesired features. Since our approach will provide this additional information when flows are elevated to the controller, the controller

[1]Naturally, the sending host will need to validate those packet sizes upon transmission to avoid applications being able to forge contextual information by manipulating the length field in headers.

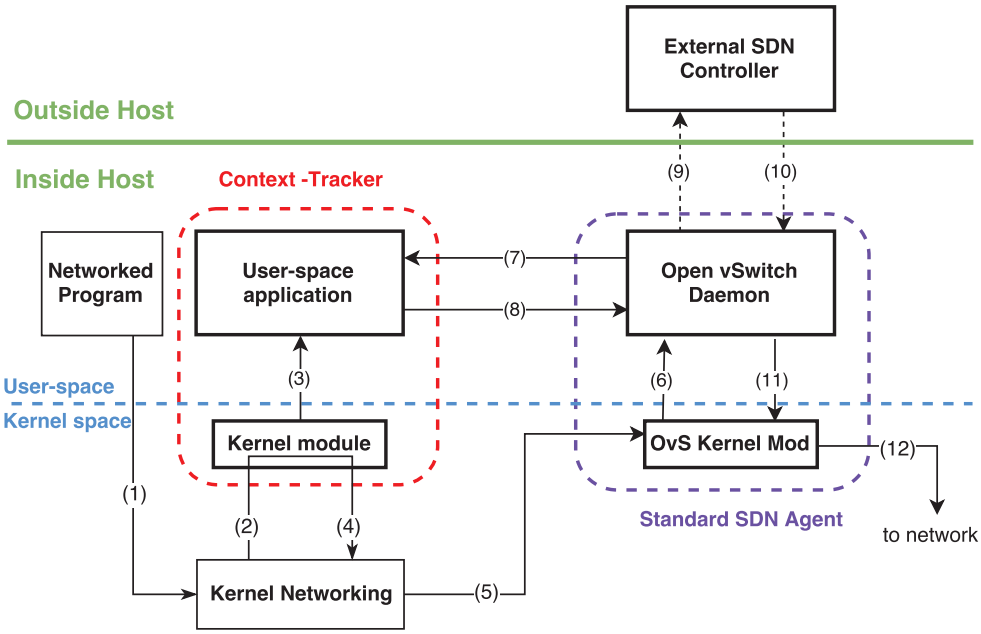

Fig. 2. When a program issues a network system call (1), the DeepContext system hooks network system calls using a kernel module (2) and gathers this information in a user-space application (3) then releases the hooks (4). Once the packet is intercepted by the Open vSwitch Kernel Module (5) and Elevated to the Open vSwitch (OvS) Daemon (6), the daemon queries the user-space application for context (7). The OvS Daemon receives the context (8), appends it to the trailer of the packet and sends it to the controller (9). The controller replies with the corresponding rule (10) that gets relayed to the kernel module (11). If the controller rule allows it, the packet gets released to the network (12); else, it's dropped.

can leverage this information when authorizing or denying a network flow.

Our context tracking application, shown in the red ring in Figure 2, is responsible for collecting, storing, and providing this information upon demand. The tracker is essentially a multithreaded user-space application that manages data stored in an internal map data structure. To populate this data structure, we use a kernel module to hook each of the Linux networking functions. Whenever a process calls a network function, our special function hooks are executed in kernel mode. We then extract relevant information from the process's `task_struct`, including the process's PID, group ID, associated user ID, the name of the process, and the executable path associated with the process. We then use a `netlink` socket to transmit this information, along with the full network flow-tuple, to our user-space context tracker for storage.

When our modified OVS needs to elevate a flow to the controller, it queries the context tracking system via a Unix socket. OVS provides the network flow tuple (i.e., IP_{src}, IP_{dest}, Transport Protocol, Port_{src}, Port_{dest}) for the flow in question. The the context tracker looks up the appropriate context in its map data structure and returns it to OVS. OVS then sends the context data and the packet to the controller machine, as explained in the following subsection and Figure 1.

B. Modifications to Open vSwitch

While Open vSwitch (OVS) is often used in a hypervisor between virtual machines, it can also be installed without virtualization as a kernel module on a Linux system. In this configuration, the OVS essentially acts as an OpenFlow agent for a single system. However, even in this mode, OVS does not provide any details about the host context when elevating requests to the controller.

We extend the host-based installation of OVS, shown in the purple ring in Figure 2, with our own functionality. In particular, we have modified existing OVS functions to include additional context on `OFPT_PACKET_IN` messages before OVS transmits those messages to the OpenFlow controller. The altered functions use a Unix socket to request information from the context tracker application. Once the context tracker responds with context, the OVS functions appends the supplied context to the end of the `OFPT_PACKET_IN` before it is transmitted, as shown in Figure 1. At that point, the message is sent to the OpenFlow controller for a decision. The OVS system then implements any orders it receives from the controller.

By leveraging Open vSwitch to implement DeepContext, we inherit many of its features. We can transmit the OpenFlow packets, including the modified `OF_PACKET_IN`, over encrypted channels. DeepContext can bind to any network interface (e.g. WLAN, Ethernet), making it topology agnostic. Furthermore, our flow tables can contain around a million flow rules [18].

C. Controller Module Customizations

To leverage the additional context from the end-host, we created an SDN controller module. For our solution, we have chosen to use the Floodlight controller due to its solid performance and common use in the community. We modified

the default Forwarding module in Floodlight, which simply implements packet forwarding. We altered the module to include code to parse the trailer information in the `OFPT_PACKET_IN` message to extract the additional context we encoded in OVS. That controller module then considers the standard network flow-tuple and the context in making decisions using preconfigured policy rules. As a proof-of-concept, our module approves all network traffic unless the application's absolute path begins with `/home` or `/tmp`, in which case the policy dictates that the flow be dropped. While just an example, other policies could focus on other elements of context to provide particularly detailed control.

Importantly, when the Floodlight controller sends a `OFPT_FLOW_MOD` message to the end-host OVS, it authorizes or denies the full network flow tuple. It does not specify any of the extra host context in this message since the full flow information is sufficient to uniquely identify the flow. As a result, the flow state in the host's OVS is the same as an unmodified OVS.

V. Evaluating Host-Based SDNs

We begin by evaluating the DeepContext SDN approach's performance in a virtual network setup and then evaluate it in a physical LAN. We evaluate the scalability, performance, and the completeness of the enhancements of our approach.

A. Virtual Network Experiment Setup

We performed our initial virtual network performance evaluation using a single VM hosting server running Ubuntu 14.04 with a 64 bit kernel and a Kernel-Based Virtual Machine (KVM) hypervisor. That server has 16 cores, each running at 2.8 GHz. The server has 64 GB of RAM. Each of the VMs, excluding the network controller, runs with a single core and 1024 MB of RAM using Ubuntu 14.04 with Linux kernel version 3.13.0-24. The controller runs with the same specifications, but has an additional core assigned to avoid performance bottlenecks on the controller. In this experiment, we pin each VM core to a specific physical core to avoid overprovisioning.

B. Performance Evaluation of DeepContext

We evaluate DeepContext in a virtual network and compare it to Open vSwitch and previous work that leverages application context. We then evaluate DeepContext in a physical LAN.

To better understand the performance of our SDN technique, we compare it with an unmodified Open vSwitch implementation to obtain a baseline. However, OVS does not actually gather host context that can be used for detailed-oriented decisions, such as access control decisions. Accordingly, we look at two other approaches to provide host context: `ident++` and the TaylorSDN.

The `ident++` approach [15] essentially uses traditional OpenFlow network switches in combination with a user agent on each system that can be queried by an SDN controller for more details about the packet. This system requires an existing OpenFlow network and requires the controller to queue the packets while awaiting a response from the end-host agent.

Our own work in the TaylorSDN took a similar approach to DeepContext, pushing the SDN agent from switches to end-hosts and instrumenting those hosts for context. However, it was not OpenFlow-compatible and was written in a scripting language, which may affect performance and hinder its deployment. Rather than repeat the experiments, we directly use the results from that work's publication [22].

We begin by measuring the performance characteristics of an unmodified OVS implementation. We perform two experiments. In the first experiment, we generate 1,000 new flows sequentially using a script running on the host. This allows us to measure the elevation latency between when a packet is queued at the kernel and when it is placed on the wire. This captures the flow elevation and controller decision-making time for each of the SDN approaches. For the round-trip time (RTT), we measure the amount of time between when the packet is first queued and when the reset response from the responder is received. In each network type, the responder is a traditional network host without any SDN functionality.

In the second experiment, we evaluate the number of flows each host can create sequentially in a given time window. We created a program that creates as many sequential flows per second as it can in a 300 second period. Essentially, once the program receives the reset response from the receiver, the program closes the socket and opens a new one. We then divide the new flow count by the duration to obtain a base flow-per-second rate.

TABLE I
Performance comparison of each network type. The elevation latency and RTT metrics are the median of 1,000 trials while the new flow rate was determined over a 5 minute period.

Metric	Open vSwitch	TaylorSDN	`ident++`	DeepContext
Median Elevation (ms)	1.98	16.72	4.25	2.739
Median RTT (ms)	6.25	34	10.17	7.390
New Flows/sec	103.19	27.4	48.19	86.80

We show the results of these experiments in Table I. The Open vSwitch system has the best performance of the SDN techniques, since it does not spend any time gathering host context. The TaylorSDN is significantly worse than the Open vSwitch approach, likely due to its Python implementation and its polling-based approach for gathering host context. For both the DeepContext and the `ident++` approaches, we use the same kernel modifications, Open vSwitch daemon, and context-gathering agent. However, in the `ident++` approach, the host agent provides the entries reactively when prompted by the controller. In essence, the `ident++` gains the benefit of DeepContext's performance optimizations, but still performs worse due to the extra round-trip time from the controller.

When examining the results in Table I, we see that the host context instrumentation present in DeepContext adds 0.76 milliseconds to the median elevation time (the difference of DeepContext's 2.74ms elevation with the 1.98ms elevation in Open vSwitch).

TABLE II
CPU AND MEMORY USAGE OF AN UNMODIFIED OPEN VSWITCH SYSTEM AND THE MODIFICATIONS RELATED TO THE DEEPCONTEXT COMPONENTS.

	CPU				RAM (of 1024 MB Total)			
Component	Median	Mean	Max	Std. Dev.	Median	Mean	Max	Std. Dev.
Context Tracking System	1.00%	0.97%	1.30%	0.19%	0.10%	0.10%	0.10%	0.00%
Unmodified Open vSwitch	44.00%	43.40%	47.30%	4.65%	0.40%	0.39%	0.40%	0.01%
Open vSwitch with Context	48.30%	47.40%	49.00%	3.86%	0.90%	0.89%	0.90%	0.01%

We next test the DeepContext system on a physical local area network. In this experiment, we deploy DeepContext on two VMs, each running on a separate physical VM hosting server. Each VM machine has a 2.8 GHz processor and 1024 MB of RAM each. Both hosts run Ubuntu 14.04 with Linux Kernel 3.13.0-24. The controller runs in a VM on a third physical machine and has two 2.8 GHz Cores and 1024 MB of RAM. All the VMs use a bridged network interface and connect to our organization's production network via a gigabit Ethernet network switch. Table III shows the results we obtained by generating 1,000 new flows sequentially in the same fashion as in the virtual network experiment. The elevation time is consistent with the virtual environment while the RTT has a slightly higher median RTT (9ms versus 7.4ms), likely due to the forwarding in the physical hardware's switch and network interface cards.

TABLE III
EVALUATION OF DEEPCONTEXT IN A PHYSICAL LAN. ALL METRICS ARE IN MILLISECONDS AND ARE THE MEDIANS OF 1,000 TRIALS.

Context Retrieval Time	Elevation	Median RTT
0.468	2.259	8.901

Finally, we did a quick confirmation test with two types of applications: a daemon that periodically probes the network, the `avahi-daemon` which uses Multicast DNS, and that of a user-initiated web application, `wget`. The controller could easily confirm the application responsible for each type of message based on the context. In the case of `wget`, the controller could link the application's DNS request with its HTTP interaction with the web server.

C. Understanding the Performance of the DeepContext System

To characterize the performance of the DeepContext components, we track the resource usage for the processes that run on the end-host, such as the modified Open vSwitch daemon and the context tracking system. Using the same testing environment described in Section V-A, we use a script to continuously generate traffic. We monitor the resulting flows per second and the resources used via the `top` command.

In Table II, we show the CPU and memory requirements of the components. We can see that much of the CPU usage is associated with the Open vSwitch component, as shown with the unmodified Open vSwitch results. The additional context processing adds roughly 4.3% CPU usage. The RAM is also low for both, amounting to less than 1% of the memory of the system. That memory usage is roughly divided between the Open vSwitch process and our tracking of context.

Even in this stress test scenario, we use less than half the CPU at the end-host. We ultimately stored 85,135 flows, which greatly exceeds the number of flows most end-hosts simultaneously manage. In that scenario, the memory usage was still less than 1% of the 1024MB RAM available. In normal daily usage, we do not expect a noticeable performance impact in typical client system usage.

D. Impact of Context Processing at the Controller

While OpenFlow controller scalability is its own separate topic of research, we are primarily concerned with whether our processing of host context in a controller application significantly affects the controller's scalability. To measure this impact, we use `cbench` [20] tool to measure the number of responses per second the controller can handle. In using `cbench` on a single client, we vary number of simulated switches and measure the response time. We compared the Floodlight controller running our DeepContext module to a Floodlight controller running the unmodified Forwarding module, which does not need to consult host context for its decisions.

In Table IV, we show our results. We find that our controller module generates $5,455.38$ response per second as a maximum average, which occurs when 50 switches are simulated. In comparison, the unmodified Forwarding module can handle $6,692.82$ responses per second in the same scenario. In general, the DeepContext module's flows per second range from 77% to 88% of the unmodified Forwarding module's rate in each scenario. This may be acceptable in some networks. However, in busy networks, additional controllers may need to be provisioned to compensate for the additional processing at the controller.

TABLE IV
FLOW RATE OF THE DEEPCONTEXT SYSTEM COMPARED TO AN UNMODIFIED FORWARDING CONTROLLER APPLICATION

cbench	DeepContext Controller		Forwarding Controller	
# of Switches	Avg. flows/s	Std. Dev.	Avg. flows/s	Std. Dev.
1	1,450.12	429.66	1,728.93	516.57
2	2,189.75	739.56	2,812.86	634.07
4	3,536.06	842.05	4,213.70	942.02
8	4,773.87	1,034.28	5,405.35	1,137.86
16	5,230.06	1,338.50	6,264.34	1,160.08
32	5,213.97	1,699.68	6,352.14	1,886.39
50	5,455.38	1,439.98	6,692.82	1,684.47

E. Completeness Assessment of DeepContext

In this section we evaluate how comprehensive DeepContext's contextual benefits are. We do so by examine two classes

of attacks that the context may aid in detecting: malicious software running from the user's home directory and software that attempts to inject commands into vulnerable applications.

We begin by focusing on applications running from user directories. Network oriented code running from user-writeable directories is often suspicious [26] since most applications are installed in administrator-controlled directories, such as `/usr/bin` or `/usr/local/bin`. We evaluate our system's ability to detect and block such applications by installing a rule that denies access to any application whose absolute path begins with `/home`. We then create a program that issues a request to an HTTP server. The SDN agent elevates the request to the controller, the controller application observes the application's path and applies the policy to drop all packets from that application. The controller sends the drop rule and our program is indeed denied network access.

Our second class of attacks is one in which the attacker tries to execute an unauthorized command based on how a vulnerable application works. As an example, the popular image manipulation library ImageMagick was vulnerable to a command injection attack [4], [14]. An attacker could create a special MVG file that would manipulate ImageMagick into executing arbitrary commands via a system shell. Attackers could manipulate programs using ImageMagick to download and execute malware.

In our experiment, we focus on our context tracking system's ability to obtain the parent process associated with any network-using application. For a GUI-less server, we created a controller policy that blocked applications unless the parent process was `init` (which spawns daemons) or the `bash` shell, which is used used by SSH or local console users. Using our own application that is vulnerable to command injection, we inject the `wget` command to get an image file. The SDN agent again elevated the network request to the controller. The controller applied its policy about process ancestry and denied the request. The SDN agent then dropped the traffic, rendering `wget` unable to download the image file.

While these two examples are relatively simple policies, they highlight the power of the system. A broad class of attacks can be thwarted with simple policies. Unlike with firewall rules or policies that use ports, organizations can create whitelist policies for specific legitimate applications in our approach by specifying paths without being concerned about creating openings for malicious applications.

VI. Discussion

In this work, we have focused on the contextual benefits, performance, and data-plane scalability of the SDN agents associated with an OpenFlow network. We have addressed the inherent scalability issues of physical OpenFlow switches by moving the SDN component into the end-host, allowing more rules per host. In doing so, we have moved from a switch-based SDN that can only store a few thousand flow rules in its flow table [18] and can manage roughly 100 new flows/second [1] overall to a host-based system in which each host can store roughly 1 million rules in its flow table [18] and can manage roughly 87 new flows/second on each host.

The contextual benifits and performance of OpenFlow controllers is being actively investigated by other researchers in the field [10], [24]. Scalable controllers [6] and distributed controllers [23] are both possible mechanisms to ensure that network controllers can keep up with the demands of the devices on the network. This is particularly important for controller applications that must perform additional processing, such as examining host context for decision making. While essential, we consider such work to be orthogonal to our own.

Host-based SDNs have limited influence over physical switches and the ports they use for forwarding traffic. Enterprise networks, unlike data center networks, often have star topologies with limited forwarding options. Future work may explore tunneling or encapsulation options, like MPLS, to enable more powerful traffic engineering.

The use of a host-based SDN necessarily requires changes at each of the end-hosts. Our paper shows how to do so in the Linux operating system. Other popular operating systems, both for traditional operating systems and for mobile OSes, would additionally require instrumentation to ensure coverage of all the end-user systems at an organization. In proprietary operating systems, the kernel-based functionality may need to be implemented as a kernel driver. Further, for large organizations, automated software distribution tools may help install the software organization-wide.

A final concern focuses on network policy. While the host-based SDN approaches can provide detailed context for network operators, these operators would need to specify policy that leverages these features to make decisions. The approach for enhancing network policy may be organization-specific. However, future work may explore the potential for a suite of template best practice policies for organizations.

VII. Conclusion

In this work, we introduced a new host-based SDN approach, called DeepContext, and compared it with existing approaches to enhance SDNs with host context. We found that the DeepContext system offers the best performance of the host context systems with only modest performance overheads as compared to the Open vSwitch system. Further, we found the DeepContext system provides each of the context building properties we analyzed.

Our work has found that proactively providing context on flow elevation requests can yield valuable management insights and performance benefits while providing compatibility with the existing OpenFlow community and tools.

Acknowledgements

The authors would like to thank Curtis Taylor and Tian Guo for their feedback on the system and valuable discussions.

This material is based upon work supported by the National Science Foundation under Grant No. 1422180. Any opinions, findings, and conclusions or recommendations expressed in this

material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Openflow performance testing. http://www.spirent.com/~/media/whitepapers/broadband/pab/openflow_performance_testing_whitepaper.pdf.

[2] Floodlight OpenFlow Controller - Project Floodlight. http://www.projectfloodlight.org/floodlight/, April 2016.

[3] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.*, 37(4):1–12, Aug. 2007.

[4] D. Cid. Imagemagick remote command execution vulnerability, May 2016.

[5] W. Cui, R. H. Katz, and W.-t. Tan. Design and implementation of an extrusion-based break-in detector for personal computers. In *Computer Security Applications Conference, 21st Annual*, pages 10–pp. IEEE, 2005.

[6] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: Scaling flow management for high-performance networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, SIGCOMM '11, pages 254–265, New York, NY, USA, 2011. ACM.

[7] C. Dixon, H. Uppal, V. Brajkovic, D. Brandon, T. Anderson, and A. Krishnamurthy. Ettm: A scalable fault tolerant network manager. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 85–98, Berkeley, CA, USA, 2011. USENIX Association.

[8] G. A. Fink, V. Duggirala, R. Correa, and C. North. Bridging the host-network divide: Survey, taxonomy, and solution. In *Proceedings of the 20th Conference on Large Installation System Administration*, LISA '06, pages 20–20, Berkeley, CA, USA, 2006. USENIX Association.

[9] GEANT. Technology investigation of openflow and testing. GEANT Whitepaper DJ1-2.1. [Online] http://geant3.archive.geant.net/Media_Centre/Media_Library/Media%20Library/GN3-13-003_DJ1-2-1_Technology-Investigation-of-OpenFlow-and-Testing.pdf, July 2013.

[10] S. Hassas Yeganeh and Y. Ganjali. Kandoo: a framework for efficient and scalable offloading of control applications. In *Hot Topics in Software Defined Networks*, pages 19–24. ACM, 2012.

[11] M. S. Johns. Identification protocol. IETF RFC 1413, February 1993.

[12] M. Kuzniar, P. Peresini, and D. Kostic. What you need to know about SDN flow tables. In *Lecture Notes in Computer Science (LNCS)*, number EPFL-CONF-204742, 2015.

[13] L. Lu, V. Yegneswaran, P. Porras, and W. Lee. Blade: an attack-agnostic approach for preventing drive-by malware infections. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 440–450. ACM, 2010.

[14] MITRE Corporation. Cve-2012-1823. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-1823, March 2012.

[15] J. Naous, R. Stutsman, D. Mazieres, N. McKeown, and N. Zeldovich. Delegating network security with more information. In *ACM Workshop on Research on Enterprise Networking*, pages 19–26. ACM, 2009.

[16] Open vSwitch Developers. Open vSwitch. http://openvswitch.org/, April 2016.

[17] B. Parno, Z. Zhou, and A. Perrig. Using trustworthy host-based information in the network. In *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing*, STC '12, pages 33–44, New York, NY, USA, 2012. ACM.

[18] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore. Oflops: An open framework for openflow switch evaluation. In *International Conference on Passive and Active Network Measurement*, pages 85–95. Springer, 2012.

[19] S. Scott-Hayward, G. O'Callaghan, and S. Sezer. SDN security: A survey. In *IEEE SDN for Future Networks and Services (SDN4FNS)*, pages 1–7, Nov 2013.

[20] R. Sherwood. Cbench: an open-flow controller benchmarker, 2010.

[21] P. Sun, M. Yu, M. J. Freedman, J. Rexford, and D. Walker. HONE: joint host-network traffic management in software-defined networks. *Journal of Network and Systems Management*, 23(2):374–399, 2015.

[22] C. R. Taylor, D. C. MacFarland, D. R. Smestad, and C. A. Shue. Contextual, flow-based access control with scalable host-based SDN techniques. In *IEEE INFOCOM Conference*, April 2016.

[23] A. Tootoonchian and Y. Ganjali. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 Internet Network Management Conference on Research on Enterprise Networking*, INM/WREN'10, pages 3–3, Berkeley, CA, USA, 2010. USENIX Association.

[24] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood. On controller performance in software-defined networks. In *USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Hot-ICE'12, pages 10–10, Berkeley, CA, USA, 2012. USENIX Association.

[25] A. Wang, Y. Guo, F. Hao, T. Lakshman, and S. Chen. Scotch: Elastically scaling up SDN control-plane using vswitch based overlay. In *ACM International on Conference on Emerging Networking Experiments and Technologies*, pages 403–414. ACM, 2014.

[26] J. Yonts. Digging for malware: Suspicious filesystem geography. http://www.malicious-streams.com/resources/articles/dgmw1_suspicious_fs_geography.html, Aug 2015.

[27] H. Zhang, W. Banick, D. Yao, and N. Ramakrishnan. User intention-based traffic dependence analysis for anomaly detection. In *Security and Privacy Workshops (SPW), 2012 IEEE Symposium on*, pages 104–112. IEEE, 2012.