FOCUS: Function Offloading from a Controller to Utilize Switch Power

Ji Yang[1]
Xi'an Jiaotong University
and Duke University

Zhenyu Zhou[1]
Duke University

Theophilus Benson
Duke University

Xiaowei Yang
Duke University

Xin Wu
Big Switch Networks

Chengchen Hu
Xi'an Jiaotong University

***Abstract*—Software Defined Networking (SDN) uses a logically centralized controller to replace the distributed control plane in a traditional network. One of the central challenges faced by the SDN paradigm is the scalability of the logical controller. As a network grows in size, the computational and communication demand faced by a controller may soon exceed the capabilities of a commodity server. In this work, we revisit the task division of labour between the controller and switches, and propose FOCUS, an architecture that offloads a specific subset of control functions, i.e., *stable local functions*, to the switches' software stack. We implemented a prototype of FOCUS and analyzed the benefits of converting several SDN applications. Due to space restrictions, we only present results for ARP, LLDP and elephant flow detection. Our initial results are promising and they show that FOCUS can reduce a controller's communication overhead by 50% to nearly 100%, and the computational overhead from 80% to 98%. Furthermore, we observe that FOCUS offloading to the switches saves switch CPU because FOCUS reduces the overheads for communication with the controller.**

I. Introduction

Software Defined Networking (SDN) is a new paradigm that replaces the distributed control plane in a traditional network with a logically centralized controller. The benefits of employing SDN over traditional networking include: ease of network management, ease of development, and ease of adoption of new protocols. Modern SDN controllers are used to run a variety of SDN applications ranging from topology management and host management protocols to traffic engineering, security, and cloud virtualization applications.

In addition to its proven benefits, SDNs introduce several critical challenges. Specifically, the logically centralized controller introduces scalability challenges as the controller is unable to scalably process events from large networks [32]. Moreover, [15], [16] show that the latency between control and data plane could not be ignored. This latency can be a major concern to the performance, or even robustness of SDN.

In this paper, we revisit the design space of offloading functionality from the controller to the switches. Rather than introducing new hardware primitives or limiting applications to legacy functions, we aim to develop a simple API that allows a dominant and crucial fraction of SDN applications to easily offload important functionality to the switches and design a framework that efficiently runs the offloaded functionality on the switch's CPU.

Packets in OpenFlow switches are often processed in switch fabrics, which contains ASICs such as forwarding chip, TCAM or QDR SRAM to enable line-rate forwarding and matching (generally considered as the fast path). The OpenFlow agent maintains a TCP connection to the controller, exchanges OpenFlow messages with the controller, and translates control messages into hardware instructions for the driver (known as the slow path). The CPU processes packets at about two to three orders slower than the ASIC. This gap in processing speed makes the CPU inadequate for line rate processing but sufficient for occasional packet processing.

To this end, we chose the following three representative SDN applications to discuss:

Address Resolution Protocol (ARP) (Default Gateway or GW): A host needs to recognized its default GW's MAC address by sending ARP request before communicating with WAN networks. In an SDN network, (Figure 1 (a)), the controller responses all the ARP requests instead of a real default GW. The IP and MAC of the defaut GW rarely change, and hosts sharing the same edge switch often share a same GW. Thus, the response to all ARPs for these hosts will be identical. From this, we observe that there is an opportunity to delegate the process of generating ARP replies. Other protocols, e.g. ICMP-Echo, ICMP-timeout, IGMP and DHCP relay running within SDN networks share a similar pattern.

Local Link Discovery Protocol (LLDP): The SDN controller sends unique LLDP packets for each switch port periodically to discover the physical links (Figure 1 (c)). Upon the message from the receiving ports, the controller learns of the existence of the physical connection between two devices. These periodic packets incur a tremendous amount of overhead and are used by other topology maintenance protocols including Link Aggregation Control Protocol (LACP) and Broadcast Domain Discovery Protocol (BDDP). The only time these periodic packets include any new information is when they are used to detect link failures. From this, we observe another opportunity to delegate functionality to the switches. Namely,

[1] These authors contributed equally to this work.

978-1-5090-0933-6/16/$31.00

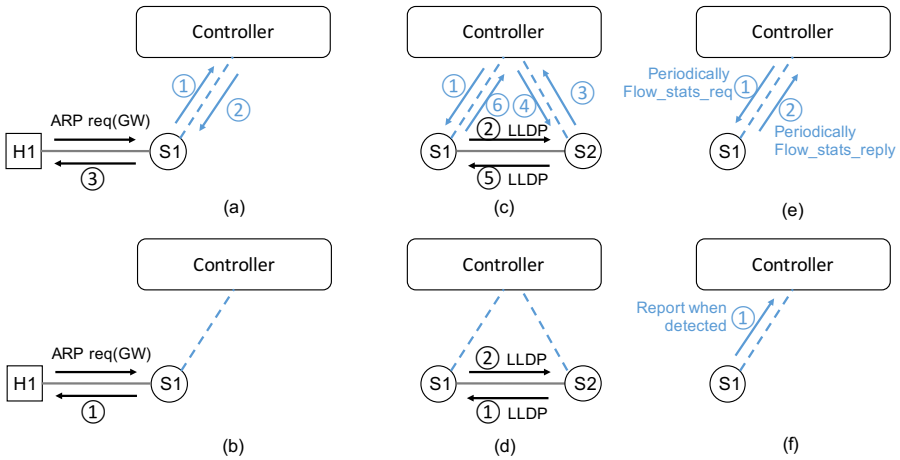


Fig. 1: **Comparison between OpenFlow (a, c, e) and FOCUS (b, d, f) handling events for ARP, LLDP, and Elephant Flow Detection respectively. The numbers show the sequence of messages exchanged between a controller and switches.**

the process of periodically generating packets.

Elephant Flow Detection: Many SDN applications [4]–[6], [11], [28], [31], [38] employ packet counters maintained by a switch to measure, monitor, secure and improve the performance of the network. Most of these applications, e.g. Hedera [4] (Figure 1 (e)), frequently query the network switches for the information. Similarly to the previous example, the elephant flow pattern incurs a tremendous amount of overheads, but the only time these periodic packets include any new information is when counters are over a certain threshold. From this, we observe out the last opportunity to delegate functionality: namely, periodically getting switch meta-data and evaluating it against a predefined threshold.

Motivated by these applications, we chose to delegate a subset of control functions called **stable local functions** (will be introduced in Section III). We propose the FOCUS (Function Offloading from a Controller to Utilize Switch power) architecture, where a controller delegates these stable local functions to switches. We are attracted to delegating stable local functions because we can offload a variety of those functions to switches with a simple set of APIs without introducing switch-to-switch communications and without introducing a code execution environment at switches.

As a proof of concept, we implemented and evaluated the controller's communication and computation workload before and after function offloading for three functions: ARP replies for the default gateway, LLDP's link discovery function, and elephant flow detection using the FOCUS architecture. Our experiments show that function offloading can reduce BOTH a controller's CPU utilization and the one of a switch, as well as the number of protocol messages. In addition to the three functions we implemented, we examined and analyzed several other commonly used SDN applications and showed tremendous savings with FOCUS (we omitted these applications due to space constraints and put them in our technical report [35]).

II. Related Work

Legacy Protocol (and OVS): OpenVSwitch (OVS) allows applications to turn on legacy functions and configure them using OVSDB [25]. However, each legacy function is uniquely developed for certain specific protocols and thus fails to scale. Instead, we show with FOCUS that with a small number of API functions we are able to achieve generality and scale to a large number applications and protocols.

Novel Switch Primitives: Unlike existing techniques on novel switch primitives [10], [23], [37], [39], FOCUS only delegates control plane functions to a switch's software stack without changes to the switch's hardware. This difference makes FOCUS immediately applicable to all Whitebox SDN Switches. Further, FOCUS supports delegation of a richer set of functionality than exist approaches [8], [10], [17], [30].

Function Delegation: While FOCUS aims to delegate control functions to the data plane, others, e.g. FRESCO [28], have explored orthogonal ideas of moving rich data plane functions, e.g. DPI, from middleboxes to the SDN controllers. Unlike these approaches to increase load on the controller, FOCUS aims to reduce load and processing on the controller.

Distributed Control Planes: Whereas distributed controller architectures [12], [14], [19], [20], [33], [36] partition visibility and control of network dynamics among controllers, in FOCUS, the controller retains visibility and total control.

Local Stable Functions: Identification of local stable functions is in principle similar to the concepts of local functions identified in prior works [7], [14], [20], [27]. Unlike orthogonal approaches [14], [20], [27], which define locality based on a set of switches, FOCUS focuses on locality relative to one switch and network configuration. Meanwhile, the definition of stable local functions could be complementary to previous work [30] and further reduce the traffic of control channel.

Novel APIs and Programming Models for SDNs: The introduction of a programming interface to support delegation of local functions, while similar to the interfaces provided by Kandoo [14], is fundamentally different. FOCUS limits the developer to a narrow and simple interface to ensure simplicity at the switch, whereas Kandoo [14] allows developers to develop arbitrarily rich offload messages due to the complexity supported by the local controllers. P4 [9] abstracts a pro-

grammable switch into match-action processing and Domino [29] introduces new programmable atoms to support a line-rate stateful packet processing. Both of them are solutions on the pacekt forwarding process. FOCUS will help the dataplane using hardware efficiency at the line speed forwarding.

Other related works [18], [22], [34], [36] propose higher level programming abstractions that simplify development and minimize errors. As part of future work, we intend to explore such high level programming languages and determine how these languages can enable automated detection and offloading of stable functions.

III. Design

A. Design Goals

Our observations are similar to those made by prior works [7], [8], [10], [37]. These problems persist because they propose a solution that explores interesting points in the design space that render them hard to adopt: namely, they require new hardware primitives. We argue that a practical and deployable design must satisfy the following constraints:

Global Visibility: Delegating control plane function introduces philosophical concerns: namely, delegation can minimize global visibility and reduce the efficiency of centralized control. Thus, we argue that the API should be designed such that the controller has identical visibility in our environment as it would in a traditional SDN environment.

Local Decisions: Many of the functions we aim to delegate to the switches require communication between several switches. For example, LLDP requires two switches to exchange LLDP packets and agree on the status of the switch. Naively designed APIs will introduce complexity and undermine the architecture. Instead, our abstraction should be designed to require and act solely on local information.

No Hardware Modification: We argue that rather than burdening the hardware, solutions should be implemented using switch software. [13] verifies that it is reasonable to introduce workload even on legacy switches and the switches do have the ability to handle a bunch of tasks. Moreover, given the rise of commodities Whitebox Switches that run Linux [3], we believe that developing a solution that runs on the switch CPU will dovetail with orthogonal efforts in industry.

B. Architecture

To support delegation, FOCUS exposes a set of narrow but expressive APIs that extend on the traditional OpenFlow dataplane API and support a large number of SDN applications. And to support these APIs, FOCUS redesigns the traditional SDN architecture presented in Figure 2, which differs from a traditional SDN environment in the following ways:

FOCUS extension: The controller includes a FOCUS extension that allows the SDN applications to leverage the FOCUS API – we call SDN applications using the FOCUS API "FOCUS enabled applications".

FOCUS agent: At least one switch in the network runs a FOCUS agent. The FOCUS agent runs on the switch CPU alongside the OpenFlow agent. The FOCUS agent is charged with setting up triggers and implementing the appropriate actions for each trigger. Further, the FOCUS agent sits between the OpenFlow agent and the switch OS, thus allowing it to intercept OF-events and perform offloaded actions before the OpenFlow agent can process the event.

The FOCUS design aims to offload **stable local functions** to switches. By stable, we mean the output of a function does not change with time as long as the network configuration does not change. By local, we mean that the function does not require input from other switches. This definition allows FOCUS to delegate proper functions.

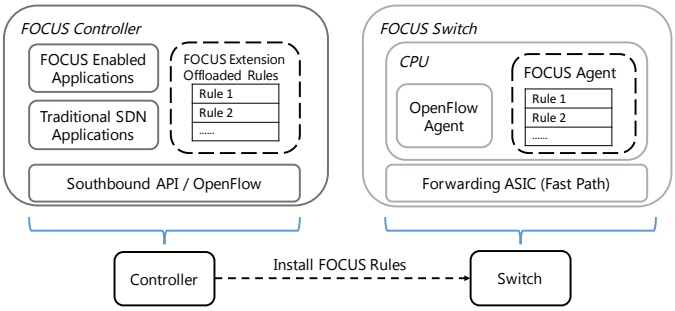


Fig. 2: **The FOCUS architecture.**

The result of the FOCUS architecture is that fewer control packets are exchanged between switches and the controller which results in CPU savings both on the controller and surprisingly on the switches (discussed in Section V). All control messages need to go to the controller whereas within FOCUS a large number of messages are handled locally by the FOCUS agent running on the switch.

We note two interesting features of this architecture. The architecture for FOCUS is backwards compatible: unmodified applications can run along side modified FOCUS enabled applications with minimal performance degradation (we discuss this in Section III-E). Further, FOCUS is incrementally deployable because each offloaded function is local to each switch and thus an application can simultaneously employ FOCUS on FOCUS-enabled switches and interact in a traditional manner with traditional switches.

Next we elaborate on the API for configuring FOCUS rules, then discuss challenges in designing the FOCUS architecture. These range from: (1) ensuring no loss in visibility or control (Section III-D); to (2) ensuring performance and security isolation between the FOCUS agent and traditional OpenFlow agent (Section III-E).

C. FOCUS API for Expressing FOCUS Rules

A FOCUS rule is composed of a *Trigger* and an *Actionlist*, similar in principle to OpenFlow's match and action primitive.

Triggers: Our trigger primitives subsume traditional OpenFlow match primitives and includes timer-based triggers in addition to the ability to match packet fields. Each FOCUS rule can be triggered by either, a timeout or recipient of a packet whose fields matches a pre-specified predicate. Next, we elaborate on each trigger.

Timer-based: A timer-based trigger enables FOCUS to support periodic functions. For example, constantly polling a

switch's resources to determine if certain conditions are met (e.g. Elephant Flow detection) or constantly sending heart-beat messages to other devices in the network (e.g. LLDP). The resolution of the timer depends on the hardware characteristics and the switch OS: ideally timer resolutions are empirically defined to minimize CPU overheads.

Predicate (Packet-matching): In addition to timers, FOCUS rules may be triggered by the receipt of packets that match certain conditions. Recall, FOCUS agent is inserted between the OpenFlow agent and the switch OS, thus allowing the FOCUS agent to receive all packets that the data plane sends to software. FOCUS supports a more expressive set of predicates than OpenFlow. Whereas OpenFlow V1.5 [2] supports 44 predicates in its match primitives, FOCUS utilizes the Type-Length-Value (TLV) scheme [26] which allows FOCUS to support an arbitrary number of predicates. Recall, TLV allows the controller to map the bits in a packet to arbitrary 'types' and then define predicates based on these types.

Action-List: FOCUS supports a radically different set of actions than the traditional OpenFlow primitives; the differences in actions underscore the fact that OpenFlow's actions are optimized to support data-plane functionality, whereas FOCUS's actions are optimized to enable delegation of control plane functionality to the switches. At its core, FOCUS supports three groups of actions :

Packet operations: These operations are generally used in conjunction with the predicate trigger and allow FOCUS to access fields of the input packet, and generate an output packet.

Flow entry operation: This action can be used with either trigger and allows FOCUS to access the meta-data associated with flow table entries within the datapath. Currently, we support a single action: `rate_with_thresh()`, which examines packet counters and returns information based on pre-specified threshold.

Message operations: This action manages and delivers the output packets created by the previously described actions.

The primitive APIs form a series of operations. Every operation can get the return value of its direct previous operation and the global static fields. A generated packet template waiting to be filled is a typical example of global static fields: it can be touched by a series of operations to form a complete output packet. If a value *EOF* is returned, the operation series will terminate directly, which means the previous operation meets errors or already wraps up the whole process.

D. Control and Visibility

A key benefit of SDN is global visibility and centralized control. FOCUS maintains this visibility and central control by limiting the functionality that is delegated to the set of control plane functionality that requires information local to the switch and that requires the switch to make no independent decisions outside of that pre-specified by the controller (we refer to these function as being stable). When conditions change at the switch, FOCUS's reports and timeout values enable the FOCUS agent on the switch to alert the controller of changes.

FOCUS Timeouts: Each FOCUS rule has a timeout associated with it: the timeout expires if the rule is not used within a pre-specified amount of time. Using this information, the FOCUS agent can eliminate stale FOCUS rules and inform the application of changes in network conditions. For example, if a switch stops receiving packets that match LLDP, the FOCUS agent can inform the controller through the FOCUS extension of a change in network conditions – the controller can treat this as a link failure and react accordingly.

FOCUS Reports: The FOCUS rules and actions are defined to use flows or ports local to a switch. When the status of these ports or flows changes, e.g. port down or flow removed, the FOCUS agent informs the controller and thus allows the controller to react appropriately.

To support these reports, FOCUS includes special control messages that enables the FOCUS agent to update the information at the FOCUS extension and thus the controller.

E. Isolation and Resource Contention

A key challenge in running a FOCUS agent on the switch lies in its contention of resources with the traditional OpenFlow agent. Fortunately, many switches employ traditional Linux OS, e.g. Cumulus and Open Networking Linux employs Debian [3], thus allowing us to leverage traditional OS isolation and containment techniques. In our architecture, the FOCUS agent runs as an independent process, communicating with the OpenFlow agent and the controller through two independent TCP connections. This design choice allows us to reuse process level techniques provided by traditional OSs.

IV. FOCUS Examples

In this section, we use concrete examples to show how an application can use the simple set of FOCUS APIs to delegate certain functions to switches. When an application delegates a function to switches, it reduces both its computational overhead and its communication overhead with the switches.

An incoming packet that matches certain patterns could trigger an outgoing packet. If the function is local and stable, i.e., the same matched patterns trigger the same outgoing packets using switch-local information, a controller can delegate this function to switches.

We show a concrete example using ARP. Other examples include the ICMP Time Exceeded function, the MAC address learning function, and the DHCP relay function.

ARP Reply for Default Gateway: Recall that we describe in Section I, without FOCUS, an SDN controller must reply an ARP request to the default gateway from every host. With FOCUS, a controller can offload this function to a switch by installing a new FOCUS rule and a set of action items. It can either proactively offload this function at all edge switches or reactively install this information at an edge switch after it receives the first OF_Packet_In packet triggered by an ARP query received by an edge switch.

We show in Table I how to implement this function using the FOCUS API. A controller installs a packet match rule in a switch's FOCUS rule table. If a packet matches the type "ARP" and the ARP query is for the default gateway IP, then the FOCUS agent will execute the installed actions. The first action is to generate an ARP reply packet template

using the `pkt_compose()` function. Then a sequence of `get_field()` and `set_field()` actions set the corresponding fields in the outgoing packet from the incoming packet. For example, the destination MAC address in the outgoing packet is set from the source MAC address of the incoming packet. After the corresponding fields in the packet are set, the last action sends out the outgoing ARP packet.

In addition to ARP replies to the default gateway address, it is possible to offload ARP replies to all host IP addresses to switches as done in [8]. It is a design decision an SDN operator can make. We do not show the examples here for clarity. Interested readers can find more examples in the technical report version of this paper [35].

Trigger	Actions
ARP target_IP=GW_IP	*pkt_compose*(ARP)
	get_field(src_MAC)
	set_field(dst_MAC, ret[1])
	set_field(target_MAC, ret)
	get_field(src_IP)
	set_field(target_IP, ret)
	pkt_output(in_port)

[1] Return value of the last operation.

TABLE I: **Delegating ARP reply.**

V. Implementation and Evaluation

We implemented our prototype of FOCUS in 1000 lines of Java code (at Floodlight Controller [1]) and 700 lines of C code (at Open vSwitch [24]). At controller side, FOCUS interface to the applications is added and a hash table called "Offloaded Function List" is maintained to manage the offloaded functions. At switch side, we alter the processing pipeline of OVS to send all packets from the data-plane to the FOCUS agent before the OpenFlow agent. The FOCUS agent implements the offloading table, described in Section III, as two tables: trigger table and extended action table.

In evaluating FOCUS, we aim to answer the following questions: What are the benefits of employing FOCUS? What are the costs of offloading functionality to the switches? How do the benefits and overheads of FOCUS vary across the different API calls (and applications)?

A. Evaluation Setup

We evaluate FOCUS using the Mininet [21] emulator. All our experiments are run on a Dell R620 server with 8G RAM and 2.80GHz Quad Core Intel CPU running Ubuntu 14.04 LTS. Recall, we developed our applications on the Floodlight controller, thus we will be evaluating performance of the Floodlight controller with and without FOCUS extensions.

Topology We evaluate FOCUS on three topologies: a simple topology for the ARP, a complex mesh-like topology for the LLDP, and a linear topology for the elephant flow detection. These different topologies allows us to explore controller performance as a function of network devices and links.

Metrics To understand the costs and benefits of applying FOCUS, we analyze the *CPU utilization* (for controller and switch) and the *total number of control messages exchanged* (this serves as a proxy for control plane bandwidth).

B. ARP

We begin, in Figure 3(a), by analyzing the impact of FOCUS on the ARP application. Specifically, the controller's and switch's CPU utilization as a function of the number of ARP requests generated. We observe that without FOCUS, the controller's CPU utilization is a linear function of the number of ARP request whereas with FOCUS, the CPU utilization at the controller is constant. As expected, the CPU saving is nearly 100% because the switch takes over all the tasks.

As for switch's CPU utilization, given that FOCUS minimizes controller's CPU utilization by overloading to the switch. Interestingly, we observe that FOCUS also reduces CPU utilization at the switches. To understand this phenomenon, we analyze the OpenFlow agents (OS processes) running the switches and observe that FOCUS eliminates the generation of additional Packet-In to the controller which results in a significant savings. To confirm this, in Figure 4(a) we present the number of control messages generated as a function of packets sent. From this figure it is apparent that FOCUS virtually eliminates all communicate between the controller and switches, thus eliminating the overheads for generating and processing messages to/from the controller.

Finally, in Figure 4(b), we compare the ARP resolution times. We observe that by offloading to the switches and eliminating the need to involve the controller in ARP resolution, FOCUS improves ARP response times by more than 20 ms.

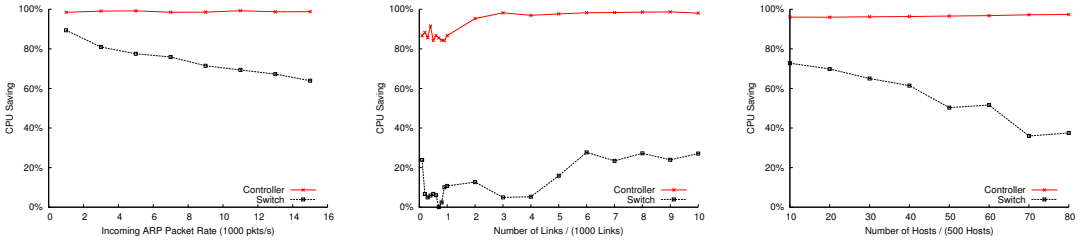
C. LLDP

Next, in Figure 3(b), we examine the performance of the LLDP application. We observe, similar trends in the LLDP experiments as with the ARP experiments with one difference: the CPU savings significantly increases after 4000 links. This increase occurs because after the incoming links exceeds 4000, the switches are constantly context switching to process the different controller messages – where as in FOCUS there is no need to process these controller messages.

Finally, we analyze the traffic load for LLDP in Figure 4(c). We observe that FOCUS reduces control plane traffic by 50% on average – this is because the controller no longer needs to poll the switches, the switches actively report changes to the topology. However, unlike ARP, in LLDP there is still a significant amount of control traffic even with FOCUS because in ARP the switch no longer needs to communicate with the controller, whereas in LLDP the switches still need to periodically report changes to the controller.

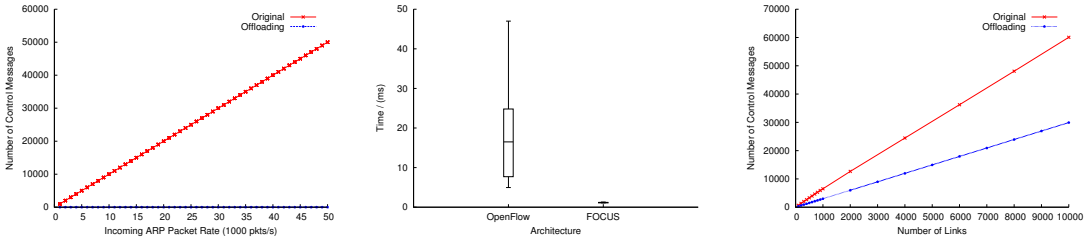
D. Elephant Flow Detection

Our last application implements elephant flow detection. Unlike the previous two applications, this application employs the `rate_with_thresh()` API. In Figure 3(c), we present the results of our experiment, from which we can find that FOCUS similarly decreases CPU utilization on both the switch and the controller for similar reasons as with ARP – with more flows the CPU on the switch does more work. We also



(a) ARP (b) LLDP (c) Elephant Flow Detection

Fig. 3: **The CPU saving for different SDN applications: (a) ARP, (b) LLDP, and (c) Elephant Flow Detection.**



(a) ARP Packet Traffic (b) ARP Response Time (c) LLDP Packet Traffic

Fig. 4: **Control channel overheads for ARP (a) and LLDP (c). ARP response times (b).**

analyzed network traffic, omitted here due to space, and found its pattern to be similar to the LLDP.

E. Memory Limitations

In OpenFlow switches, flow table entries and memory are crucial. Motivated by this fact, next, we examine the memory overheads of FOCUS. In our experiment, we observed that FOCUS scaled linearly with the number of actions and that FOCUS never required more than 4MBs of memory.

VI. Discussion

SDN Philosophy and FOCUS: At first glance, FOCUS may appear at odds with the SDN philosophy – separation of control and data plane. However, we believe that FOCUS presents a logical next step to the SDN paradigm. FOCUS's API presents programmers with a way to cache functionality at the switch's CPU – in a similar way that flow table entries cache forwarding policies in the TCAM. Similar to TCAM entries, FOCUS's rules are a local cache of functionality at the controller they do not hamper or obscure the controller's ability to create a global view of network state.

Running FOCUS on Hardware Switches: In this paper we demonstrated the benefits of running FOCUS on software switches. Yet, we believe that switches extend beyond software switches to hardware switches. In fact, we believe that given the limited CPU on switches the savings gain from reduced packet-in/packet-out and their corresponding context switches will significantly improve the switches' performance. Moreover, while we have not run FOCUS on software switches others have shown that modern Whitebox Switches are easily capable of running novel agents. For example, Cumulus, BigSwitch and OFX (from UPenn) all demonstrated the feasibility to run custom agents on commodity switches. We are currently extending BigSwitch's Open Networking Linux to implement FOCUS.

Security, Isolation, and Availability: The notion of offloading functionality into the switches introduces a number of interesting research challenges we plan to explore in the future. These include security, isolation, and fault tolerance.

Generally, opening up the switch OS to third party code introduces novel avenues for attacks. Yet, with FOCUS we avoid this by opening introducing a small and limited set of API calls rather than allowing arbitrary code. These calls maintain a similar attack surface as the current OpenFlow API – which similarly exposes a limited set of API calls supported by an agent on the switch. Furthermore, the offloading rules still obey the priority given by the controller application, which keeps the same security level as the current OpenFlow.

Although the API restricts the attack surface, bugs in it can introduce fault tolerance and availability problems that can potentially cascade and render the OpenFlow agent on a switch in-operative. We plan to address this by isolating the FOCUS agent from the OpenFlow agent and furthermore by isolating the threads processing information for different application – thus bugs triggered by an application are limited to that specific application. These isolation primitives will provide further isolation by enforcing fine-grained resource limits.

VII. Conclusion

In this paper, we study how to improve the scalability of an SDN controller by enabling a controller to delegate certain control functions to switches. We study the tradeoffs between the flexibility of delegation and the cost of delegation. We choose a design point where a controller can use a simple set of APIs to delegate a variety of functions, which we call stable local functions. These functions remain stable over time

as long as the network configuration does not change and they only require input local to a switch to compute. We describe the FOCUS APIs and use concrete examples (including ARP replies, LLDP, and elephant flow detection) to show how a controller can use such simple APIs to delegate packet-match triggered functions and periodic timer triggered functions.

Finally, we implement ARP replies, LLDP, and elephant flow detection functions using the FOCUS API, and show that the FOCUS design can significantly reduce a controller's as well as a switch's communication and computational overhead. This result shows FOCUS does not introduce prohibitive complexity to switches. In our experiments, FOCUS can reduce a controller's communication overhead by 50% to nearly 100%, and the computational overhead by 80% to 98%. Because FOCUS reduces the communication overhead between a switch and a controller, it can reduce a switch's overall computational overhead by 60% to 90% for ARP replies, around 35% for large-scaled LLDP and 40% to 80% for elephant flow detection, even though it adds additional functions to a switch.

REFERENCES

[1] Floodlight. http://www.projectfloodlight.org/floodlight/.

[2] OpenFlow Switch Specification V1.5.0. https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.5.0.noipr.pdf.

[3] What's Inside Cumulus Linux for Networking? http://www.enterprisenetworkingplanet.com/netos/whats-inside-cumulus-linux-for-networking.html.

[4] AL-FARES, M., RADHAKRISHNAN, S., RAGHAVAN, B., HUANG, N., AND VAHDAT, A. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *NSDI* (2010).

[5] BALLARD, J. R., RAE, I., AND AKELLA, A. Extensible and Scalable Network Monitoring Using OpenSAFE. *Proc. INM/WREN* (2010).

[6] BENSON, T., ANAND, A., AKELLA, A., AND ZHANG, M. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *CoNEXT* (2011).

[7] BIANCHI, G., BONOLA, M., CAPONE, A., AND CASCONE, C. OpenState: Programming Platform-independent Stateful Openflow Applications Inside the Switch. *ACM SIGCOMM Computer Communication Review* (2014).

[8] BIFULCON, R., BOITE, J., BOUET, M., AND SCHNEIDER, F. Improveing SDN with InSPired Switches. *ACM SIGCOMM SOSR* (2016).

[9] BOSSHART, P., DALY, D., GIBB, G., IZZARD, M., MCKEOWN, N., REXFORD, J., SCHLESINGER, C., TALAYCO, D., VAHDAT, A., VARGHESE, G., ET AL. P4: Programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review 44*, 3 (2014), 87–95.

[10] CURTIS, A. R., MOGUL, J. C., TOURRILHES, J., YALAGANDULA, P., SHARMA, P., AND BANERJEE, S. DevoFlow: Scaling Flow Management for High-Performance Networks. In *ACM SIGCOMM Computer Communication Review* (2011).

[11] DAS, A., LUMEZANU, C., ZHANG, Y., SINGH, V., JIANG, G., AND YU, C. Transparent and Flexible Network Management for Big Data Processing in the Cloud. In *HotCloud* (2013).

[12] DIXIT, A., HAO, F., MUKHERJEE, S., LAKSHMAN, T., AND KOMPELLA, R. Towards an Elastic Distributed SDN Controller. In *ACM SIGCOMM Computer Communication Review* (2013).

[13] HAND, R., AND KELLER, E. ClosedFlow: Openflow-like Control over Proprietary Devices. In *HotSDN* (2014).

[14] HASSAS YEGANEH, S., AND GANJALI, Y. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In *HotSDN* (2012).

[15] HE, K., KHALID, J., DAS, S., GEMBER-JACOBSON, A., PRAKASH, C., AKELLA, A., LI, L. E., AND THOTTAN, M. Latency in Software Defined Networks: Measurements and Mitigation Techniques. In *ACM SIGMETRICS* (2015).

[16] HE, K., KHALID, J., GEMBER-JACOBSON, A., DAS, S., PRAKASH, C., AKELLA, A., LI, L. E., AND THOTTAN, M. Measuring Control Plane Latency in SDN-enabled Switches. In *SOSR* (2015).

[17] HE, K., ROZNER, E., AGARWAL, K., FELTER, W., CARTER, J., AND AKELLA, A. Presto: Edge-based Load Balancing for Fast Datacenter Networks. In *SIGCOMM* (2015).

[18] KIM, H., REICH, J., GUPTA, A., SHAHBAZ, M., FEAMSTER, N., AND CLARK, R. Kinetic: Verifiable Dynamic Network Control. In *NSDI* (2015).

[19] KOPONEN, T., CASADO, M., GUDE, N., STRIBLING, J., POUTIEVSKI, L., ZHU, M., RAMANATHAN, R., IWATA, Y., INOUE, H., HAMA, T., ET AL. Onix: A Distributed Control Platform for Large-scale Production Networks. In *OSDI* (2010).

[20] KRISHNAMURTHY, A., CHANDRABOSE, S. P., AND GEMBER-JACOBSON, A. Pratyaastha: An Efficient Elastic Distributed SDN Control Plane. In *HotSDN* (2014).

[21] LANTZ, B., HELLER, B., AND MCKEOWN, N. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In *HotNets* (2010).

[22] MONSANTO, C., REICH, J., FOSTER, N., REXFORD, J., WALKER, D., ET AL. Composing Software Defined Networks. In *NSDI* (2013).

[23] MOSHREF, M., BHARGAVA, A., GUPTA, A., YU, M., AND GOVINDAN, R. Flow-level State Transition as a New Switch Primitive for SDN. In *HotSDN* (2014).

[24] NICIRA NETWORKS. Open vSwitch, An Open Virtual Switch. *http://openvswitch.org/* (2010).

[25] PFAFF, B., AND DAVIE, B. The Open vSwitch Database Management Protocol. *RFC 7047* (2013).

[26] PRZYGIENDA, T. Reserved Type, Length and Value (TLV) Codepoints in Intermediate System to Intermediate System. *RFC 3359* (2002).

[27] SCHMID, S., AND SUOMELA, J. Exploiting Locality in Distributed SDN Control. In *HotSDN* (2013).

[28] SHIN, S., PORRAS, P. A., YEGNESWARAN, V., FONG, M. W., GU, G., AND TYSON, M. FRESCO: Modular Composable Security Services for Software-Defined Networks. In *NDSS* (2013).

[29] SIVARAMAN, A., CHEUNG, A., BUDIU, M., KIM, C., ALIZADEH, M., BALAKRISHNAN, H., VARGHESE, G., MCKEOWN, N., AND LICKING, S. Packet transactions: High-level programming for line-rate switches. In *Proceedings of the 2016 Conference on ACM SIGCOMM 2016 Conference* (New York, NY, USA, 2016), SIGCOMM '16, ACM, pp. 15–28.

[30] SONCHACK, J., AVIV, A. J., KELLER, E., AND SMITH, J. M. OFX: Enabling OpenFlow Extensions for Switch-Level Security Applications. In *CCS* (2015).

[31] SUH, J., KWON, T. T., DIXON, C., FELTER, W., AND CARTER, J. OpenSample: A Low-Latency, Sampling-Based Measurement Platform for Commodity SDN. In *ICDCS* (2014).

[32] TAVAKOLI, A., CASADO, M., KOPONEN, T., AND SHENKER, S. Applying NOX to the Datacenter. In *HotNets* (2009).

[33] TOOTOONCHIAN, A., AND GANJALI, Y. HyperFlow: A Distributed Control Plane for OpenFlow. In *INM/WREN* (2010).

[34] VOELLMY, A., WANG, J., YANG, Y. R., FORD, B., AND HUDAK, P. Maple: Simplifying SDN Programming Using Algorithmic Policies. In *ACM SIGCOMM Computer Communication Review* (2013).

[35] YANG, J., ZHOU, Z., BENSON, T., YANG, X., WU, X., AND HU, C. Technical Report CS-TR-2016.001. Tech. rep., Duke University, Department of Computer Science, 2016.

[36] YEGANEH, S. H., AND GANJALI, Y. Beehive: Towards a Simple Abstraction for Scalable Software-Defined Networking. In *HotNets* (2014).

[37] YU, M., REXFORD, J., FREEDMAN, M. J., AND WANG, J. Scalable Flow-Based Networking with DIFANE. *ACM SIGCOMM Computer Communication Review* (2011).

[38] ZAALOUK, A., KHONDOKER, R., MARX, R., AND BAYAROU, K. OrchSec: An Orchestrator-Based Architecture for Enhancing Network-Security Using Network Monitoring and SDN Control Functions. In *NOMS* (2014).

[39] ZHU, S., BI, J., SUN, C., WU, C., AND HU, H. SDPA: Enhancing Stateful Forwarding for Software-Defined Networking. In *ICNP* (2015).