Reducing the Latency of OpenFlow Rule Changes in Data Centre Networks

Jonathan Sherwin
Department of Computer Science
Cork Institute of Technology
Cork, Ireland
jonathan.sherwin@cit.ie

Cormac J. Sreenan
Department of Computer Science
University College Cork
Cork, Ireland
cjs@cs.ucc.ie

***Abstract*— The number and size of data centres being constructed continues to grow, in response to increasing demand for cloud-based services. In a Data Centre Network (DCN), maximizing performance is essential, and network management tools that leverage the advantages of Software-Defined Networking (SDN) are key to achieving that goal. DCN switches are now expected to support the OpenFlow protocol – however, performance issues have been observed with hardware switch OpenFlow implementations in relation to path establishment for new flows, and path modification for existing flows. In particular, concerns have been raised regarding the delays in effecting changes to hardware flow tables. We report on recent research that seeks to address this issue. Specifically, we investigate the approach of temporarily tunnelling packets for new flows while more efficient non-tunnelled paths are being established, and to which the flows are later migrated. Our early results show the potential and limitations of this approach.**

Keywords*— *Software-Defined Networking; Data Centre Networks; Network Management; OpenFlow

I. INTRODUCTION

Data Centre Network traffic is expected to increase to 15.3 Zettabytes per year by 2020, with cloud data centre traffic accounting for almost 92% of that figure, according to [1]. As data centres grow in both size and number to handle traffic demands, the task of managing the network to maximize use of resources increases in complexity. Software-Defined Networking – and the OpenFlow [2] protocol, in particular – has offered the flexibility of dynamically reconfiguring the network to meet the constantly changing traffic of a multi-tenant cloud data centre.

Most network switch manufacturers currently support OpenFlow on their devices, however their implementations suffer from some problems. Researchers have shown [2,3] that OpenFlow rule modification times vary from device to device, and can be significantly large, particularly if rule modification requests are received at a high rate, if the rules have different priorities, and if flow tables have high occupancy levels. Furthermore, if a switch indicates that a rule has been installed in its flow tables, the rule may still not affect the forwarding of packets in the data plane for some length of time. Such delays and uncertainty are unacceptable in a multi-tenant DCN for two reasons: 1) distributed application performance is degraded, and 2) security is compromised if the destination of packets cannot be fully controlled. In contrast to the situation with hardware switches, Open vSwitch [5], a widely-used open-source software switch, can handle much higher flow-modification request rates, without the uncertainty as to whether a rule has taken effect (see section III for verification).

Our prototype rule-management system masks the shortcomings of hardware switches outlined above by temporarily forwarding packets of a new flow through a tunnel between software switches, provided across hardware switches. Tunnel entry and exit rules are configured as individual flows arrive; flows persisting long enough or having special needs are migrated to reactively-configured non-tunnelled paths.

This approach allows new flows to be accepted by the network at a higher rate than that of which hardware switches may be capable. It provides the opportunity to then establish non-tunnelled paths at a flow-rule configuration rate that is within the hardware switches capabilities. Each tunnelled flow is only migrated to a non-tunnelled path when that path is verified as fully configured and active, avoiding packet loss or leakage to unexpected destinations. We plan to enhance this simple approach to prioritize the flows to be migrated, since during an extended period of higher flow-arrival rate, it will not be possible to migrate all flows within their lifespan.

II. RELATED WORK

One approach to dealing with flow-rule configuration delays is to pre-configure all possible paths before any data arrives. Since OpenFlow tables typically would not have the capacity for a rule for every path, PAST [6] implements pre-configured per-address spanning trees in the layer-2 forwarding tables of switches. Planck [7] adds monitoring and control to PAST, and demonstrates those features being used to provide redundant paths and rerouting capabilities. Both use switches that specifically support programmatic access to the layer-2 forwarding tables (rather than just to the TCAM tables used by OpenFlow). Also, both provide paths between all pairs of hosts, rather than only paths that match the security policies present in a multi-tenant DCN. PARIS [8] forwards traffic through pre-configured summary routes to the core, where Equal Cost Multipath (ECMP) routing then decides the route to edge switches that have a forwarding entry for every local virtual machine (VM). This requires a layer 3 addressing

Jonathan Sherwin is funded by Cork Institute of Technology.
Cormac J. Sreenan is funded by Science Foundation Ireland.

scheme that lends itself to summarisation – not a reasonable assumption in a multi-tenant DCN. Although PARIS describes an enhancement to support multi-tenancy, it depends on MPLS or VLAN tags – increasing the complexity and reducing the flexibility of the solution.

Devoflow [9] originally identified the cause of flow-setup latency as a bottleneck between the data-plane and the control-plane, and proposed more autonomy in switches with aggressive use of wildcard OpenFlow rules, although security-sensitive flows must still be handled centrally. The approach of giving switches more responsibility is also taken in [10]. Both [9] and [10] require modified switch hardware.

Monocle [11] verifies that flow-rule modifications have been applied at the data plane, delaying confirmation from OpenFlow switches to the controller until data-plane probes have shown the modified rules to be active. It does not make any effort to reduce flow-setup time, rather it reflects the true flow-setup time.

Currently the most widely used commercially-available solution, VMware NSX [12], treats the physical DCN as an 'underlay', across which tunnels are configured to create an 'overlay' or virtualised network. The underlay network uses a traditional routing protocol (e.g. BGP, OSPF, IS-IS) to route VXLAN-tunnelled traffic. An SDN controller manages the VXLAN tunnels (the overlay network), allocates flows to tunnels, and ensures tunnel end-points are advertised through the traditional routing protocol. VXLAN carries a significant encapsulation overhead. No attempt is made to migrate flows to non-tunnelled paths, and underlay network devices are relegated to performing simple packet-forwarding.

III. Problem Verification

To verify that software switches could out-perform hardware switches at the control plane for flow-rule setup, we configured an experimental test-bed using Mininet [13], a network emulation tool. We use the Pox [14] OpenFlow controller, and Open vSwitch as a software switch (software that runs in a hypervisor or in a server operating system and that offers switching services to VMs or processes on that node). The authors of [3] kindly shared with us code they wrote to replicate hardware switch control plane delays (hereafter referred to as an emulated hardware switch), developed based on data from their tests of actual HP and Pica8 switches. We validated our own testing code by reproducing their results. We then tested Open vSwitch with the same parameters, to allow us to compare the control plane performance of a software switch with that of a hardware switch. Results are shown in Fig. 1 and summarised numerically in Table 1.

An experiment is made up of batches, each batch consisting of a controller-to-switch request to delete an existing flow rule, a request to add a new flow rule, and a barrier request to prompt the switch to confirm when it has previous requests complete. A batch starts when a barrier reply is received for the second previous batch, the aim being to keep the switch busy while avoiding overloading it with flow-rule modification requests. The figure shows timings from the start of the experiment, and illustrate that for each individual batch, the delay between the flow-mod request being acknowledged by the switch and the arrival of the first data packet forwarded following the newly added flow rule is significantly greater on a hardware switch as compared to the delay on a software switch. Also, there is a substantial periodic delay observed with the particular hardware switch being emulated, seen in Fig. 1(a) roughly every 18th batch, matching the results in [3]. We refer readers to that paper for explanation of causes of these delays.

IV. Proposed Approach

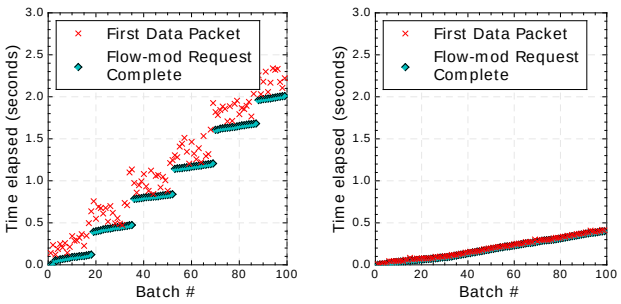
While installing flow rules in hardware switches is relatively slow, installing flow rules in software switches is fast, as illustrated by Fig. 1 and Table I. Our idea takes advantage of the fact that much of the traffic in a DCN is east-west [14,15] (between servers) and the first and last switches to handle packets of a particular flow are often software switches in the hypervisors of host servers, but could also be software switch processes running in a non-hypervisor operating system.

This allows us to leverage the software switches' flow rule installation speed to mask the slowness of the hardware switches, as illustrated in Fig. 2 and outlined here:

i. Pre-configured tunnels connect software switches across the hardware switches in the DCN. A mesh of tunnels between all servers is proposed – for simplicity, however, the figure shows just two servers with a single tunnel between their software switches.

ii. When a new flow arrives (or a current flow needs to be modified), install high priority rules on software switches to forward the flow across the appropriate tunnel. Then install normal priority rules on software and hardware switches to be ready to carry the flow on a non-tunnelled path. Flow rules are matched in order of priority, so while the high priority rules are present, all matching packets will be tunnelled.

iii. When the non-tunnelled path has been verified, remove the high priority rules so that subsequent packets of the flow follow the non-tunnelled path.



(a) Emulated H/W switch (b) S/W switch

Fig 1. Flow-rule installation delay

TABLE I. Flow-rule Installation Statistics

Switch Type	Mean Delay (ms)	Std. Deviation (ms)
Emulated Hardware	218.3	132
Software	11.7	5.5

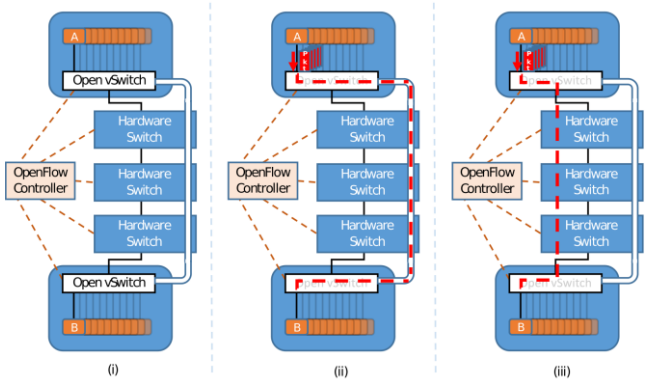


Fig. 2. Temporary tunnelling of new flow

While flows could be left to complete on the tunnelled connections (essentially a virtual network solution), there are reasons not to: 1) this would lead to an imbalance in traffic across the DCN, the architecture of which normally includes multiple paths between servers; 2) the ability to have granular treatment of flows at intermediate switches (e.g. for QoS, security) is impacted; 3) tunnelling imposes a per-packet overhead of marking packets by, for example, adding fields, consuming bandwidth and processing power, and adding latency. In addition, the ability to migrate flows to alternative paths can be useful to proactively resolve congestion.

With our system, there could be several tunnelled paths available between any pair of software switches (and this is planned for future work), for redundancy in case of switch or link failure, and for traffic load distribution. The number of tunnels could increase or decrease dynamically according to requirements, but should be kept small to control the quantity of flow-rules required in hardware switch tables for tunnels, especially in a large DCN network. However, all paths through the network are available for non-tunnelled flows. Currently, our prototype implementation uses one path for tunnelled traffic, and a different path, if available, for non-tunnelled.

Flows that require granular treatment at intermediate switches could potentially not be tunnelled at all, or could be prioritized for migration to non-tunnelled paths over flows that do not require such treatment.

Our system incurs the overhead of tunnelling to some extent. For flows of short duration, all packets may end up being tunnelled. For long flows consisting of packets with large payloads (e.g. jumbo frames), the overhead of tunnelling is not so significant. If long-lived flows of packets with small payloads can be identified (planned future work), de-tunnelling these will yield the most benefit and so should be prioritized.

V. Experimental Test-Bed and Implementation

We had the use of a single DCN hardware switch in our physical test-bed – an Edgecore AS5712-54X running PicOS 2.6.3. We used a Cisco C200 M2 server and Dell OptiPlex 760, both running Ubuntu Linux Server 14.04, with Open vSwitch 2.7.3 installed. Both had a management interface through which the OpenFlow controller exchanges control-plane traffic with Open vSwitch, and a Mellanox 10G fibre interface connected to the hardware switch to carry data-plane traffic.

Debian 8 Linux VMs were hosted on the server and PC with VirtualBox [17] virtualization software, and were run in headless mode for experiments. Each VM had a control network interface used for automated control during experiments, and a data network interface through which TCP connections could be initiated with other VMs. The control interface was linked to the physical management interface on the host machine, as experiments were controlled from the machine on which the controller was located. The data network interface was connected to an Open vSwitch bridge through a Linux tap interface. The bridge was in turn connected to the physical fibre network interface on the host machine.

Our SDN controller ran in an Ubuntu VM on an Apple MacBook Pro. The MacBook was connected to an 8-port 100Mb/s 3Com switch, to which the management interfaces of the server, PC and hardware switch were also connected.

Our controller implementation was written in Python, and based on Pox. A simple client/server Java application to create a configurable number of new TCP connections per second as required was deployed on the Debian VMs. A TCP connection comprises two uni-directional flows, requiring flow rules to be configured for both. Our controller configures the rules for the two flows as soon as a packet_in event for a new TCP connection is received. Flow-rule addition requests are sent for the inbound flow first, and then the outbound flow, to reduce the likelihood of packet_in events for the same flow being generated by switches other than the first one on the path. The controller keeps track of all active flows, using the information to ignore duplicate packet_in events, to migrate flows to non-tunnelled paths as necessary, and to verify a non-tunnelled path before migrating a flow to it. For now, since we are focused on control-plane performance, TCP connections are established and remain open until the end of the experiment, but no data is sent through the connections.

The controller verifies non-tunnelled paths through the use of probe packets. When a flow-rule is added to the last switch on a non-tunnelled path, a higher priority rule with the same match criteria is added to redirect matched packets to the controller. The controller sends probe packets through the first switch on the path, and when it successfully receives a probe back from the last switch, the higher priority rule is removed.

Experiments were automated via a Python script interacting with all components through the management switch. The script prepared the Debian VMs (added static ARP entries for each other VM, started packet captures), started the SDN controller and a packet capture on the MacBook, and started the client/server TCP connection generator. At the end of the experiment, the same script collected the packet captures and metrics recorded by the controller, processed and analysed the data and generated graphs.

For tunnelling, we use MPLS labels. We chose MPLS because the overhead it introduces is low compared to other tunnelling methods, it is supported by all versions of OpenFlow, and requires no special NIC or switch support. When the switches initially connect to the controller, the hardware switches are configured with flow rules to forward packets between ports through which software switches are reachable, based on MPLS label values. When the controller

learns of a new flow, it configures rules on the software switches to label packets of that flow before forwarding to a hardware switch, and remove labels from packets of the flow received from a hardware switch. Although we just had a single hardware switch for our experiments, the controller code is designed for a spine-leaf DCN topology, and has been functionally tested with that topology in Mininet.

Our system specifies a mesh of tunnels between servers, however a set of tunnels that terminate at the same point can share the same MPLS label. Therefore, for *n* servers, a leaf switch requires *n* rules: $n - 1$ rules to forward packets towards all remote servers, but only 1 rule to deliver packets from all remote servers to the directly connected server. Packets are forwarded by leaf switches such that all labelled packets a single spine switch receives share the same tunnel end-point – requiring only 1 rule on the spine switch for forwarding.

VI. Experimental Testing and Results

Each experiment ran for 20 seconds, long enough to verify whether our approach could maintain deterministic results for a given flow arrival rate beyond the initial start-up. Flow tables were cleared before the experiment started. Experiments were run with a range of TCP connection arrival rates. Note that one TCP connection consists of two unidirectional flows, and is handled as follows. When the first packet of a TCP connection arrives at a software switch, since there are no matching rules it is forwarded to the controller. The controller then configures rules on all switches on both the outgoing and return paths to allow the connection to be established as quickly as possible.

To evaluate our approach, we ran experiments where flows were initially tunnelled, and later migrated to non-tunnelled paths. For comparison, we ran experiments with the same test-bed setup with the only change being that flows were not tunnelled at all, and more experiments where the flows were tunnelled but never migrated from the tunnels. Fig. 4 shows TCP connection setup delay with rates of 30, 70 and 90 new connections per second, for the three scenarios: flows not tunnelled at all, flows tunnelled for their lifetime, flows tunnelled initially, but later migrated to a non-tunnelled path. To measure connection setup delay, we record the time at which the client sends a TCP SYN packet, and the time that it receives a SYN/ACK in return. While this measure disregards the time required for the final ACK of the TCP 3-way handshake, it captures the time taken for the flow-rules for outbound and inbound paths to be configured.

In the three graphs of Fig. 4, the TCP connection delay rises dramatically (note that the y-axis is plotted on a log scale) for non-tunnelled operation – i.e. when flow-rules must be added to the hardware switch for each arriving connection. Even for a relatively low arrival rate of 30 connections per second, the delay incurred as the hardware switch queues up requests to add flow rules causes the TCP connection setup delay to rise exponentially. As expected, the situation only deteriorates with higher arrival rates of 70 and 90 TCP connections per second.

In contrast, the performance for 'tunnelled without migration' is maintained at a reasonably stable delay of around 10ms at an arrival rate of up to 70 TCP connections per second.

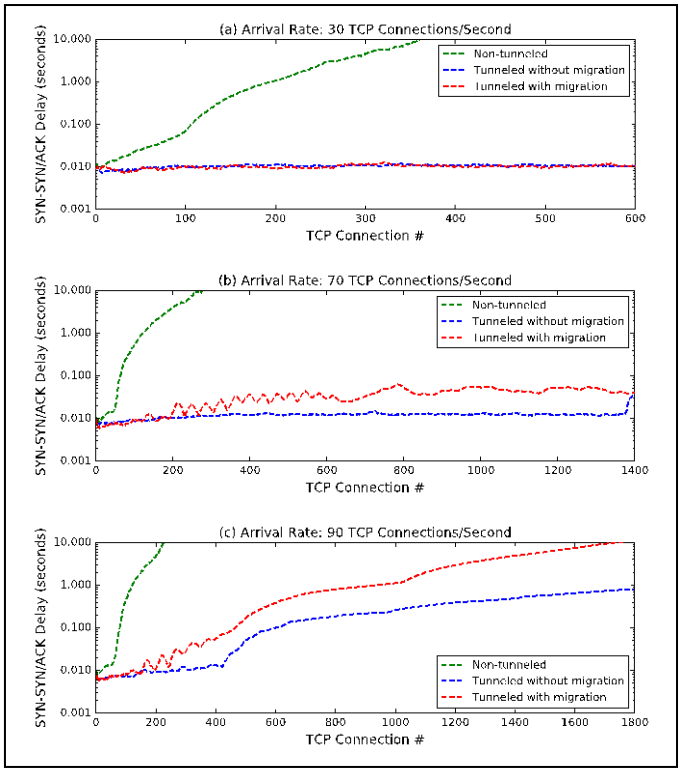


Fig 4. Rolling mean TCP connection setup delay at various arrival rates

With this approach, only the software switches are required to add flow rules before the SYN packet can reach the server, and the SYN/ACK response can get to the client. In Fig. 4(c), there is a blip in the results for 'tunnelled without migration' near the end of the experiment, coming up to TCP connection #1400. The raw data indicates that this was a short-lived anomaly, and the connection delay appears to be returning to the levels observed before the blip, but it is too close to the end of the experiment to be definitive. We conjecture that the blip is due to another process running on the server or PC causing our controller to be briefly starved of resources.

Our 'tunnelled with migration' approach fares well at an arrival rate of 30 TCP connections per second, with delay comparable to 'tunnelled without migration' in Fig. 4(a). The cost of migrating flows from tunnelled to non-tunnelled paths starts to weigh in Fig 4(b), but note that the TCP connection setup latency is maintained at less than 100ms for the full experiment for 'tunnelled with migration'. While migrating flows incurs a cost for TCP connection setup, clearly that cost can be balanced out over the lifetime of a connection through utilisation of redundant paths and by the elimination of tunnelling overhead after a flow is migrated.

The results in Fig. 4(c) for an arrival rate of 90 connections per second, with TCP connection setup delay rising without an upper limit for all scenarios, show that the capabilities of our implementation are exceeded. Having profiled our controller code, we can say that this is because as new flows arrive, our data structure tracking current flows grows, and takes longer to search to make sure that a packet received as part of packet_in

message doesn't belong to an existing flow. With re-designed code, we are confident we can reduce the search time. There will still be a point at which the number of concurrent flows limits the TCP connection arrival rate that can be supported.

For comparison, the authors of [16] found a per-switch arrival rate of up to 60 flows per second (equivalent to 30 TCP connections per second) in DCNs they analysed – our approach has already surpassed that. More recently, a figure of up to 500 flows per second at a single DCN switch has been quoted in [15] by a hyperscale data-centre operator, giving us a target to aim for. As Open vSwitch is quoted as supporting a flow-rule update rate of 42,000 per second [18], we are optimistic about our system achieving a DCN-appropriate level of performance.

VII. Conclusions and Future Work

Our results show clearly the benefit of reducing the dependence on hardware switches' capability to handle flow-rule additions, and we provide a mechanism for doing so that leverages the capabilities of software switches. The results we present here represent initial work, and will improve with refinement of our implementation. More than that, they indicate to us the potential of extending our approach to accommodate flow-rule removals and modifications, in an integrated flow-rule management system intended to meet demanding performance requirements. The approach of initially tunnelling a flow will allow flows to be accepted as they arrive at rates seen in very large data centres, giving space in time for the flow migration function to selectively queue up and prioritize flow-rule requests for hardware switches at the controller to deliver them at a rate that matches the switches capabilities, minimizing the latency of request completion.

Our next step is to improve the design of our code to use more efficient data structures and to make the performance of the event-driven code that handles packet_in events more independent of the periodically-run code that configures and verifies non-tunnelled paths before migrating tunnelled flows to them. Then we intend to investigate how to select flows for migration, mechanisms for prioritising the migration, and develop an algorithm for delivery of flow rule modification requests at a controlled rate within the capabilities of the receiving switches. We will expand our testing to situations where switch flow tables are pre-populated, we will use DCN packet traces, and we hope to access more hardware switches.

Acknowledgment

We thank Dr Fatima Gunning of the Tyndall National Institute under SFI research grant 13/CDA/2103 for access to testbed facilities; Łukasz Łukowski and Mark Basham from Edgecore Networks for technical assistance with the Edgecore Networks AS5712-54X switch; Tom Sheffield from Pica8 networks for the trial license of the PicOS 2.6.3 software; and BT TSO for the loan of the Edgecore Network Switches.

References

[1] Cisco Global Cloud Index: Forecast and Methodology, 2015–2020 White Paper. Available: http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, "OpenFlow: enabling innovation in campus networks," SIGCOMM Computer Communications Review, vol. 38, no. 2, pp. 69-74, 2008.

[3] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about SDN flow tables," from Passive and Active Measurement Conference 2015, in Lecture Notes in Computer Science, vol. 8995, 2015, pp. 347-359.

[4] K. He, J. Khalid, A. Gember-Jacobson, S. Das, C. Prakash, A. Akella, L. E. Li, M. Thottan, "Measuring control plane latency in SDN-enabled switches," presented at the Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, Santa Clara, California, 2015.

[5] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, M. Casado, "The design and implementation of Open vSwitch," in Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2015, 2015, pp. 117-130.

[6] B. Stephens, A. Cox, W. Felter, C. Dixon, and J. Carter, "Past: Scalable ethernet for data centers," in CoNEXT 2012 - Proceedings of the 2012 ACM Conference on Emerging Networking Experiments and Technologies, 2012, pp. 49-60.

[7] J. Rasley, B. Stephens, C. Dixon, E. Rozner, W. Felter, K. Agarwal, J. Carter, R. Fonseca, "Planck: Millisecond-scale monitoring and control for commodity networks," in SIGCOMM 2014 - Proceedings of the 2014 ACM Conference on Special Interest Group on Data Communication, 2014, pp. 407-418.

[8] D. Arora, T. Benson, and J. Rexford, "ProActive routing in scalable data centers with PARIS," in DCC 2014 - Proceedings of the ACM SIGCOMM 2014 Workshop on Distributed Cloud Computing, 2014, pp. 5-10.

[9] 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, 2011, pp. 254-265.

[10] H. Mekky, F. Hao, S. Mukherjee, Z. L. Zhang, and T. V. Lakshman, "Application-aware data plane processing in SDN," in HotSDN 2014 - Proceedings of the ACM SIGCOMM 2014 Workshop on Hot Topics in Software Defined Networking, 2014, pp. 13-18.

[11] P. Perešíni, M. Kuźniar, and D. Kostić, "Monocle: Dynamic, Fine-Grained Data Plane Monitoring," in CoNEXT'15 - Proceedings of the 2015 ACM International Conference on Emerging Network Experiments and Technologies, 2015.

[12] VMware, "VMware NSX Network Virtualization and Security Platform". Available: https://www.vmware.com/products/nsx

[13] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in Proceedings of the 9th ACM Workshop on Hot Topics in Networks, Hotnets-9, 2010.

[14] M. McCauley. (2013). The POX Controller. Available: https://www.github.com/noxrepo/pox

[15] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the Social Network's (Datacenter) Network," in Internet Security & Encryption Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, London, United Kingdom, 2015.

[16] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in Proceedings of the ACM SIGCOMM Internet Measurement Conference, IMC, 2010, pp. 267-280.

[17] Virtualbox.org, "Oracle VM VirtualBox". Available: https://www.virtualbox.org

[18] A. L. Aliyu, P. Bull, and A. Abdallah, "Performance implication and analysis of the OpenFlow SDN protocol," in Proceedings - 31st IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2017, 2017, pp. 391-396.