Modeling Control Traffic in Software-Defined Networks

Jesse Chen, Ananya Gopal, and Behnam Dezfouli
Internet of Things Research Lab, Department of Computer Science and Engineering, Santa Clara University, USA
jschen@scu.edu, agopal@scu.edu, bdezfouli@scu.edu

***Abstract*—The southbound control protocols used in Software Defined Networks (SDNs) allow for centralized control and management of the data plane. However, these protocols introduce additional traffic and delay between network controllers and switches. Despite the well understood capabilities of SDNs, current representations of control traffic overhead consist of approximations at best. In addition to high reactivity to incoming flows, the need for resource allocation and deterministic messaging delay necessitates a thorough understanding and modeling of the amount of control traffic and its effect on latency. In this work, we capture the network overhead of various switch configurations on a testbed and extract mathematical models to predict expected overhead for arbitrary switch configurations. We demonstrate that controller-switch traffic patterns are non-negligible and can be accurately modelled to compute the bandwidth utilization and latency of controller-switch communication.**

***Index Terms*—Edge, Fog, Cloud, Control Traffic, Resource Allocation, Determinism, OpenFlow, OVSDB.**

I. Introduction

Software Defined Networks (SDNs) facilitate network monitoring and management by removing the control plane from switches and placing it in a logically centralized controller. The two main components of an SDN architecture are the controller(s) and switches, which represent the control and data planes, respectively. The controller (e.g., OpenDayLight (ODL) [1], Ryu [2]) communicates with network applications through northbound interfaces and with switches through southbound interfaces. The SDN industry uses various southbound protocols, ranging from traditional protocols such as Simple Network Management Protocol (SNMP) to more sophisticated ones including OpenFlow [3], Open vSwitch Database Management Protocol (OVSDB) [4], and NETCONF [5].

In general, there are two categories of southbound protocols: *management*, and *control*. Management protocols such as OVSDB and NETCONF are used for tasks such as queue provisioning, port provisioning, and policy enforcement. On the other hand, control protocols such as OpenFlow, ForCES, and I2RS are used to operate on the resources available on the data plane. Among these protocols, OpenFlow and OVSDB provide a rich set of features and are supported by most SDN products.

Although OpenFlow and OVSDB are control-plane and management-plane southbound interfaces, respectively, we generalize and refer to all the traffic generated by southbound protocols as *control traffic*. Analysis and modeling of control traffic is essential from multiple perspectives. First, these southbound protocols can generate a large amount of overhead, depending on the network. It is particularly important to understand this overhead in edge and fog computing networks where the amount of bandwidth between nodes (e.g., when using wireless links [6], [7]) may be significantly limited compared to the bandwidth in cloud computing networks. Second, control traffic increases the processing and queuing delays on switches along the switch-to-controller path. Computing bandwidth usage and per-switch processing time requires information about packet size and rate. This is particularly important for applications where communication resources must be sliced based on application demands. For example, QoS-driven resource allocation in edge and fog networks requires the controller to perform data plane operations to handle incoming task requests [8]. Accurate modeling of controller-switch delay is essential to plan an appropriate network topology and configuration that satisfies application requirements. Fourth, from the security standpoint, unpredictable variations in controller-switch communication may prevent the controller from immediate reactions to security threats. Awareness of control traffic behavior reduces the spaces in which a network intruder can hide their presence, increasing the controller's ability to detect and respond to abnormalities in a network. As the network size and deployment complexity grows, computing the effect of control traffic on network performance becomes increasingly important. One example of complex networks is when Network Function Virtualization (NFV) is employed, where the interconnection of components naturally introduces significant complexity to the system.

Existing works either abstract the effect of control traffic [9]–[11], or rely on unrealistic and incomplete values [7], [12]–[14]. In particular, the impact of control traffic intensity is neglected in most of these works and there is a lack of mathematical models (derived from empirical analysis) and discussion of polling and configuration traffic between a switch and its controller [7], [13]. For example, the authors in [15] assume that the cost of statistic collection from each switch is always 40 KBps, and [16], [17] use 160-byte messages sent from switches to controller at the rate of 100 to 400 k/s. Another category of works on reducing the overhead of switch-controller delay enhances the autonomy of switches to handle flows [18]–[21]. These works also do not utilize realistic control traffic in their performance evaluation studies.

This paper presents the following contributions. *First* (§III-B), we study the overhead of flow rule installation transactions and show that this overhead is dependent on the triggering event and the flow rule complexity. *Second* (§III-C and §III-E), we demonstrate that the overhead imposed by status update messages is directly correlated to the number of configured flow rules/queues and the complexity of said configurations. *Third* (§III-F), we present observed values for the overhead of one-time transactions as well as mathematical models for long-term status update overheads.

II. Background

OpenFlow. This protocol allows for remote management of flow rule tables on switches through the use of various types of messages, such as configuration request, version negotiation, and status update messages [22]. For example, the `multipart_request` and `multipart_reply` messages are used to query status information from the switch.

Open vSwitch Database Management Protocol (OVSDB). This protocol allows a network application to configure the non-flow components of the switch such as queues, QoS, and bridges [23]. OVSDB performs operations such as `insert`, `update`, and `delete`, as part of "transact" Remote Procedure Call (RPC) requests [24] to the switch. Bandwidth slicing is accomplished by configuring QoS on the switch with OVSDB. When combined with flow rules, this allows for the routing of data flows into queues that enforce QoS. The relevant tables in the switch database for queue management are `port`, `qos`, `interface`, and `queue`.

Open vSwitch (OVS). Open vSwitch (OVS) [25]–[27] is an open-source, production-quality switch implementation. Although OVS was originally built for connectivity in virtual environments, its evolution has extended its applications to areas such as the control stack of white-box switches. Furthermore merging paradigms such as forwarding plane programmability seek to further streamline the flow of control messages via FPGA-enabled and programmable NICs. OVS is extensively used in cloud computing infrastructures such as OpenStack [28] and Open-Nebula [29].

Open Daylight (ODL). ODL is a widely-used SDN controller that supports a variety of southbound protocols. Through a RESTCONF [30] north-bound interface, ODL provides access to the many necessary southbound protocols used for network configuration, such as OpenFlow and OVSDB.

III. Control Traffic Characterization

In this section, we evaluate and characterize the communication between switch and controller. We first present empirical evaluation of control traffic considering flow table and queue configurations. Then, we present mathematical modeling of control flow characteristics.

A. Methodology

We set up a testbed consisting of one machine running ODL Oxygen as the SDN controller, and another machine running OVS 2.9.5 as the software switch. OpenFlow packets exchanged between ODL and OVS are collected by tshark [31] and are characterized with the following metrics: traffic rate, packet size, and packet rate. These metrics were chosen because traffic rate characterizes bandwidth utilization, while packet size and packet rate are used to determine queuing and transmission delays. Packet size and rate are important variables for determining message delivery delay with scheduling algorithms such as Weighted Fair Queuing (WFQ) and Hierarchical Token Bucket (HTB). Packet size is especially significant because it is used to calculate the effective service rate of a queue. We analyze these metrics with regards to varying numbers of flow rules/queues and different configurations of flow rules. Table I presents a summary of the key notations.

TABLE I: Summary of key notations

Notation	Description
p_i	A packet
$s(p_i)$	Size of packet p_i
τ	Polling period (seconds)
$\mathcal{F}$	The set of flow rules on a switch
f_i	A flow rule
$s(f_i)$	Size of flow rule f_i
$\mathcal{Q}$	The set of queues on a switch
q_i	A queue configuration
$s(q_i)$	Size of queue configuration q_i
R_b	Switch-to-controller data rate (bps)
R_b^0	Data rate (bps) when no flow rules/queues are configured
R_p	Switch-to-controller packets per second (pps)
$S_{\bar{p}}$	Switch-to-controller average packet size (byte)
D_0	Amount of data (bytes) sent from switch to controller per polling period when no flow rules/queues are configured
P_0	Number of packets sent from switch to controller per polling period when no flow rules/queues are configured

B. Flow Rule Installation

Flow rule installation can be a reactive or proactive process. Although proactive installation of a flow rule onto a switch only requires one message, the normal scenario involves a reactive two-message exchange between the switch and controller. A `packet_in` message is generated by the switch and sent to the controller when a packet arrives at the switch and at least one of the following conditions holds: (i) the packet does not match any existing flow rules on the switch, (ii) the packet matches a flow rule that explicitly specifies the generation of a `packet_in` message, (iii) the packet's IP TTL value is invalid. In response to the `packet_in` message (p_{p_in}), a `flow_mod` message (p_{f_mod}) containing flow rule data is sent from the controller to the switch. The `packet_in` message structure contains two variable-size fields: match and data. The match field is used to specify the context of the packet that arrived at the switch and triggered the `packet_in` message, and the data field transfers the packet itself for further inspection by the controller.

We set up a scenario where a `packet_in` message is generated by a flow table miss on a switch. We observed that although there are four potential context fields, only one is included in the packet header, and the rest always use their default values. This results in a 108-byte header. When a flow table miss is caused by an input packet p_{in}, the `packet_in` message size can be represented by $s(p_{p\_in}) = 108 + s(p_{in})$ bytes, where $s(p_{in})$ refers to the size of p_{in}. The `flow_mod` message is then generated by the controller to add a new flow rule to the switch's tables. Similar to the `packet_in` message, the `flow_mod` message has two variable-size fields: match and instruction [22]. We focus on variations in match

TABLE II: Size of a `flow_mod` packet and `flow_stats` structure for common match configurations.

Flow Rule Configuration	Specified Match Fields	`flow_mod` Packet Size ($s(p_{f\_mod})$) (bytes)	`flow_stats` Struct Size ($s(f_i)$) (bytes)
(C1)	IPv4 src	162	96
(C2)	IPv4 src+dst	170	104
(C3)	IPv4 src+dst, switch ingress port	178	112
(C4)	IPv4 src+dst, switch ingress port, MAC src	186	120
(C5)	IPv4 src+dst, switch ingress port, MAC src+dst	194	128
(C6)	IPv4 src+dst, switch ingress port, MAC src+dst, TCP src+dst	226	160
(C7)	IPv4 src+dst, switch ingress port, MAC src+dst, UDP src+dst	226	160
(C8)	IPv4 src+dst, switch ingress port, MAC src+dst, ICMP src+dst	218	152

field configurations so we only include the default instruction of `output-to-port`. Table II (third column) lists the common match field configurations and the size of a `flow_mod` packet to configure a flow rule with these parameters on a switch. This table also shows the size of `flow_stats` structures, which we discuss in §III-C2.

C. Polling Flow Table

1) Variations in Flow Table Size: We install various flow rules on the switch to measure the overhead of poll messages between the controller and the switch. These poll messages are observed for all generalized switch configurations; we highlight specific interactions between switch configuration and control overheads. Each flow rule, denoted as f_i, contains a match for IPv4 EtherType, IPv4 source address, and IPv4 destination address. A flow rule with this structure is represented using 104 bytes in each `multipart_reply-flow` message. However, in §III-C2 we show that not all flow rules require the same amount of data to convey rule information. To measure the network load caused by status update polling, we vary the number of flow rules installed on the switch between 0 and 500 flow rules, and capture network traffic between the switch and controller.

Data Rate. Figure 1(a) shows the relationship between control traffic bandwidth and the number of flow rules in the flow table. The amount of utilized bandwidth increases linearly versus the number of flows. As this figure shows, the traffic primarily consists of `multipart_reply` messages from the switch to the controller to convey the current switch state. The slope of the data transfer rate from the switch to the controller is directly correlated to the amount of data required to convey a single flow rule's information and the controller's polling rate. On the other hand, the traffic rate from the controller to the

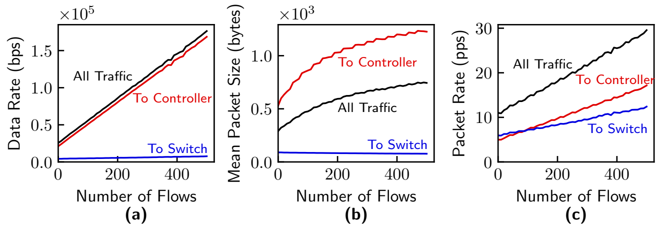

Fig. 1: The network overhead imposed by status update messages increases as the number of flow rules in the switch's flow table grows. The traffic is primarily from the switch to the controller to convey the current status of the switch.

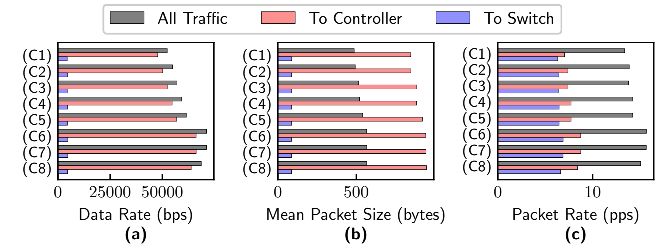

Fig. 2: This figure highlights the effect of flow rule match field configuration on control traffic overhead when there are 100 flow rules on the switch. Values in this figure are related to the slope of the lines in Figure 1.

switch only exhibits a slight increase and is not significantly affected by switch state.

Average Packet Size. The average packet size function is an asymptotic function with regards to the number of flow rules on the switch. As Figure 1(b) shows, the average packet size of the two traffic directions are significantly different. This is because the size of most data packets in the switch-to-controller direction is the Maximum Transmission Unit (MTU), while traffic in the other direction mostly consists of small `multipart_request` messages. The average packet size curves at both directions are asymptotic (the controller to switch curve is very flat), with the average packet size of traffic in the controller-to-switch direction approaching the size of a TCP ACK packet, 66 bytes, and the average packet size of traffic in the switch-to-controller direction approaching the 1500-byte MTU.

2) Variations in Flow Rule Match Field Configurations: In this section, we study the effect of different match field configurations in the flow rules. We vary the flow rules pushed to the switch in the specificity of their match fields. The variations in specified match fields range from only matching a source IP address to matching specific IP protocols from specific machines. We use a total of eight common types of match field configurations. We present the results in Figure 2. Naturally, increasing the number of match fields in each flow rule increases the amount of data needed to transfer state information from the switch to the controller via `flow_stats` structures. This highlights the benefits of having simpler flow rules that require less data to transfer between switch and controller.

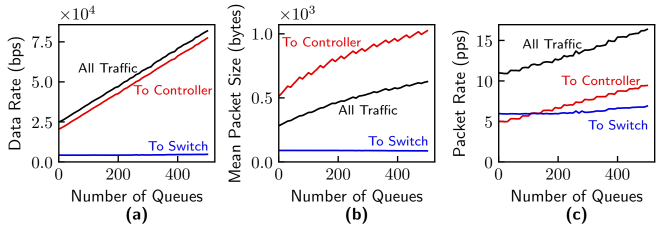

Fig. 3: The network overhead imposed by status update messages increases as the number of queues in the switch database increases. These figures show the same patterns as Figure 1; however, since representing each queue requires less number of bytes compared to a flow rule, the rate of increase in each figure is lower.

D. Queue Configuration

In an ODL-OVS system, flow rules are configured through the OpenFlow protocol, and everything else in the switch database is configured through OVSDB, including queues and queuing disciplines. Assuming that queuing disciplines have already been configured on each relevant port, adding a queue to a switch involves a three-message exchange between the controller and the switch. There are two messages from the controller to the switch: one message to add the queue to the switch database, and another message to add the new queue to a queuing discipline. Both messages from the controller to the switch are `insert` actions as part of a "transact" RPC request. After these messages have been received by the switch, the switch responds with an "update" RPC request to inform the controller about the changes made to its database. Unlike with flow rules, the switch and controller do not periodically exchange status updates through OVSDB after the queue has been configured; any status updates are handled through OpenFlow messages.

E. Polling Queue Status

1) Variations in Number of Queues: In our testbed, each queue pushed to the switch is constructed using an identical structure. A total of 500 queues are created and pushed to the switch in increments of 10 queues. Each of our specified queues is represented with 36 bytes in a `multipart_reply` message. The packet capture method for queues is similar to the packet capture method for flow rules in §III-C1: control traffic data is captured via tshark for each state where the switch has {0, 10, 20, ..., 500} queues in its database.

Date Rate. Figure 3(a) presents the relationship between the number of queues in the switch database and the overhead. Similar to the observations in §III-C1, the data rate follows a linear trend, with most of the data traveling in the switch-to-controller direction. However, since a queue status update requires fewer bytes than a flow rule, the slope of the linear trend is lower. The network traffic overhead of conveying 500 queues' status information is less than the traffic overhead of conveying 500 flow rules' information: 81.8 Kbps vs 176.3 Kbps.

Average Packet Size. The average packet size function of controller-to-switch control traffic is similar to that in §III-C1—asymptotic. However, since the amount of information associated with a queue is less than that of a flow, Figure 3(b) approaches the asymptote less aggressively than Figure 1(b). Comparing 500 queues to 500 flow rules, the average packet size of transmitting information about 500 queues is 1024 bytes/packet while the average packet size of transmitting information about 500 flow rules is 1225 bytes/packet. Due to the asymptotic nature of the average packet size function, greater amounts of flow rules and queues in the switch have increasingly lesser effects on average packet size.

TABLE III: Baseline variables

Variable	Description	Value
τ	Polling period	3 s
M	Maximum Transmission Unit	1514 bytes
R_b^0	See Table I	19610.67 bps
D_0	See Table I	7354 bytes
P_0	See Table I	12 packets
H_f^0	`multipart_reply-flow` header size	82 bytes
H_q^0	`multipart_reply-queue` header size	122 bytes

F. Empirical Models

In this section, we leverage the observations made in §III-C and §III-E to develop mathematical models of switch-controller overhead. We create a set of generalized models based on empirical analysis of a switch-controller system that can be used to predict realistic control traffic overheads. These models include variables obtained through empirical observation, and these variable values are presented in Table III.

We observed that switch status update messages occur at regular intervals (3 seconds in our experiments), and that the size of `multipart_reply-flow` messages varies depending on the number and configurations of flow rules on the switch. However, the size of `multipart_reply-queue` messages only vary depending on the number of queues on the switch. We combine these observations with those of Figures 1, 2, and 3 to present a generalized form of the rate of control traffic (bps) from the switch to the controller:

$$R_b(\mathcal{F},\mathcal{Q}) = \frac{\sum_{\forall f_i\in\mathcal{F}} s(f_i) + \sum_{\forall q_j\in\mathcal{Q}} s(q_j)}{\tau} + R_b^0 \tag{1}$$

where $\mathcal{F}$ is the set of flow rules on the switch, $\mathcal{Q}$ is the set of queues on the switch, $s(f_i)$ is the size of flow rule f_i, $s(q_i)$ is the size of queue q_i, R_b^0 is the baseline transfer rate when there are no configured flow rules or queues on the switch, and τ is the polling period by the controller. Note that $s(f_i)$ depends on the configuration of the flow table entry.

Next, the average packet size of the control traffic from the switch to the controller is represented as:

$$S_{\bar{p}}(\mathcal{F},\mathcal{Q}) = \frac{\sum_{\forall f_i\in\mathcal{F}} s(f_i) + \sum_{\forall q_j\in\mathcal{Q}} s(q_j) + D_0}{\lceil\frac{(\sum_{\forall f_i\in\mathcal{F}} s(f_i))+H_f^0}{M}\rceil + \lceil\frac{(\sum_{\forall q_j\in\mathcal{Q}} s(q_j))+H_q^0}{M}\rceil + (P_0-2)} \tag{2}$$

where D_0 is the baseline amount of data transmitted per polling period, P_0 is the number of transmitted packets that are not `multipart_reply-flow` or `multipart_reply-queue` packets, H_f^0 is the `multipart_reply-flow` header size, H_q^0 is the `multipart_reply-queue` header size, and M is the MTU. Equation (2) was derived using a similar method as Equation (1): with the observation that control traffic data is periodic, we extrapolate that average packet size over a single burst is equivalent to the average packet size of the entire flow. From there, extracting an expression for average packet size consists of calculating the amount of data per burst and the number of packets per burst, and then combining the two to express overall average packet size.

Thus, Equations (1) and (2) provide a framework to predict control traffic from a switch to its controller given any arbitrary flow rule or queue configurations. These equations accurately predict control traffic overhead for each switch in the network and are applicable for multi-switch topologies.

IV. Conclusion

Accurate modeling of control traffic is essential for resource allocation in edge, fog, and cloud computing systems. In this paper, we first studied control traffic overhead when switches include varying flow table states and queue configurations. Analysis of this overhead revealed that the total throughput and average packet rate of control traffic are directly correlated with the number of active flow rules and queues. We studied these factors to highlight the non-negligible impact of control traffic congestion on resource management and effective network design. The methodology of this paper can be applied for modeling and analysis of various southbound protocols.

The results presented in this work lay the ground toward designing networks with reduced and predictable control traffic delay. Specifically, by carefully controlling network topology based on switch-controller communication path, we can reduce the amount of control data. Also, in any given network topology, control traffic delay can be predicted by combining the control traffic overhead models proposed in this paper with analysis of queuing strategies employed by switches. Developing deterministic-delay resource allocation methods is left as a future work.

References

[1] The Linux Foundation, "OpenDaylight." [Online]. Available: https://www.opendaylight.org/

[2] "Ryu Controller." [Online]. Available: https://ryu-sdn.org

[3] Open Network Foundation, "Openflow, version 1.3.0." [Online]. Available: https://opennetworking.org/wp-content/uploads/2014/10/openflow-spec-v1.3.0.pdf

[4] B. Pfaff and B. Davie, "The Open vSwitch Database Management Protocol," Dec 2013. [Online]. Available: https://tools.ietf.org/html/rfc7047

[5] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network Configuration Protocol (NETCONF)," RFC 6241, 2011.

[6] Q. Qin, K. Poularakis, G. Iosifidis, and L. Tassiulas, "SDN Controller Placement at the Edge: Optimizing Delay and Overheads," *IEEE Conference on Computer Communications (INFOCOM)*, pp. 684–692, 2018.

[7] M. J. Abdel-Rahman, E. A. Mazied, A. MacKenzie, S. Midkiff, M. R. Rizk, and M. El-Nainay, "On stochastic controller placement in software-defined wireless networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2017, pp. 1–6.

[8] C. Powell, C. Desiniotis, and B. Dezfouli, "The Fog Development Kit: A Platform for the Development and Management of Fog Systems," *IEEE Internet of Things Journal*, vol. 7, no. 4, pp. 3198–3213, 2020.

[9] S. Bera, S. Misra, and N. Saha, "Traffic-aware Dynamic Controller Assignment in SDN," *IEEE Transactions on Communications*, 2020.

[10] D. Zeng, C. Teng, L. Gu, H. Yao, and Q. Liang, "Flow setup time aware minimum cost switch-controller association in software-defined networks," in *11th International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness (QSHINE)*. IEEE, 2015, pp. 259–264.

[11] T. Y. Cheng, M. Wang, and X. Jia, "QoS-guaranteed controller placement in SDN," in *IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2015, pp. 1–6.

[12] G. Ishigaki and N. Shinomiya, "Controller placement algorithm to alleviate burdens on communication nodes," in *International Conference on Computing, Networking and Communications (ICNC)*. IEEE, 2016, pp. 1–5.

[13] M. J. Abdel-Rahman, E. A. Mazied, K. Teague, A. B. MacKenzie, and S. F. Midkiff, "Robust controller placement and assignment in software-defined cellular networks," in *26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017, pp. 1–9.

[14] A. Jalili, M. Keshtgari, and R. Akbari, "A new set covering controller placement problem model for large scale SDNs," *Information Systems & Telecommunication*, vol. 25, 2018.

[15] M. Obadia, M. Bouet, J.-L. Rougier, and L. Iannone, "A greedy approach for minimizing SDN control overhead," in *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2015, pp. 1–5.

[16] B. P. R. Killi and S. V. Rao, "Link failure aware capacitated controller placement in software defined networks," in *International Conference on Information Networking (ICOIN)*. IEEE, 2018, pp. 292–297.

[17] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 18, no. 8, pp. 1339–1342, 2014.

[18] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, A. Vahdat *et al.*, "Hedera: dynamic flow scheduling for data center networks." in *NSDI*, vol. 10, no. 8, 2010, pp. 89–92.

[19] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," *ACM SIGCOMM*, vol. 40, no. 4, pp. 351–362, 2010.

[20] A. R. Curtis, W. Kim, and P. Yalagandula, "Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection," in *IEEE INFOCOM*. IEEE, 2011, pp. 1629–1637.

[21] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *ACM SIGCOMM*, 2011, pp. 254–265.

[22] OpenFlow Message Layer. [Online]. Available: http://flowgrammable.org/sdn/openflow/message-layer/

[23] OVSDB User Guide. [Online]. Available: https://docs.opendaylight.org/projects/ovsdb/en/latest/ovsdb-user-guide.html

[24] RFC 7047 - The Open vSwitch Database Management Protocol. [Online]. Available: https://tools.ietf.org/html/rfc7047

[25] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending networking into the virtualization layer," in *Hotnets*, 2009.

[26] J. Pettit, J. Gross, B. Pfaff, M. Casado, and S. Crosby, "Virtual switching in an era of advanced edges," 2010.

[27] B. Pfaff, J. Pettit, T. Koponen, E. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar *et al.*, "The design and implementation of open vSwitch," in *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, 2015, pp. 117–130.

[28] O. Sefraoui, M. Aissaoui, and M. Eleuldj, "OpenStack: toward an open-source solution for cloud computing," *International Journal of Computer Applications*, vol. 55, no. 3, pp. 38–42, 2012.

[29] D. Milojičić, I. M. Llorente, and R. S. Montero, "OpenNebula: A cloud management tool," *IEEE Internet Computing*, vol. 15, no. 2, pp. 11–14, 2011.

[30] A. Bierman, M. Bjorklund, and K. Watsen, "RESTCONF Protocol," Jan 2017. [Online]. Available: https://tools.ietf.org/html/rfc8040

[31] tshark - dump and analyze network traffic. [Online]. Available: https://www.wireshark.org/docs/man-pages/tshark.html