RESEARCH-ARTICLE

AggFlow: Scalable and Efficient Network Address Virtualization on Software Defined Networking

BONG-YEOL YU, Korea University, Seoul, South Korea

GYEONGSIK YANG, Korea University, Seoul, South Korea

KYUNGWOON LEE, Korea University, Seoul, South Korea

CHUCK YOO, Korea University, Seoul, South Korea

Published: 12 December 2016

CoNEXT '16: The 12th International Conference on emerging Networking EXperiments and Technologies
December 12, 2016
California, Irvine, USA

Conference Sponsors:
SIGCOMM

CAN '16: Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking (December 2016)
https://doi.org/10.1145/3010079.3012012
ISBN: 9781450346733

AggFlow: Scalable and Efficient Network Address Virtualization on Software Defined Networking

Bong-yeol Yu, Gyeongsik Yang, Kyungwoon Lee, Chuck Yoo
Korea University, Seoul, Korea
koreagood13@gmail.com, {ksyang, kwlee, chuckyoo}@os.korea.ac.kr

ABSTRACT

In this paper, we propose AggFlow, a new address virtualization scheme for high scalability and resource efficiency. AggFlow is intended to provide complete address virtualization while incurring low overhead for physical switches and the control channel. To reduce overhead of address virtualization, we propose mapping-less address virtualization. We additionally introduce hop-by-hop-based forwarding, which aggregates flow rules installed in physical switches. This leads to efficient use of ternary content addressable memory and bandwidth of the control channel. Our evaluation using Mininet with simple linear and tree topologies, the flow table size of the core switch is reduced up to 1/16 for both topologies, and the control channel traffic decreases by 51% and 29% respectively. Furthermore, we measure the overhead incurred from AggFlow. Compared to OpenVirteX, AggFlow increases control plane delay by 0.1 ms, only 0.4% increase of CPU usage, and no change of throughput for UDP traffic.

Keywords

Network Virtualization; Software Defined Networking; Address virtualization

1. INTRODUCTION

Virtualization is a technique that enables the sharing of physical resources among multiple users. It thereby ensures isolation among virtualized entities. To date, virtualization has been widely studied, especially for computing resources, such as the CPU, memory, and storage. Virtualization technology has significantly advanced in recent years, which enables many types of new services. Network virtualization is used to create multiple virtual networks within a single physical network. To date, most research for network virtualization is performed based on tunneling techniques [4, 7]. However, tunneling incurs the packet processing overhead to encapsulate and de-capsulate packets. In addition, network managers cannot control their own virtual network topology or policy with tunneling because parts of the existing network cannot be respectively configured for each virtual network.

These drawbacks are due to the native characteristic of the existing network; that is, elements in the existing network are controlled in a distributed way. Therefore, abstracting each network element is difficult. Fortunately, use of software defined networking (SDN) [8] enables it by centralizing control of network resources with a global view of network states. Proxy-type network hypervisors [1, 3, 10] have been proposed to provide topology, address, and policy abstractions that leverage SDN benefits. In data center, where multiple tenants exist, each tenant requires its own network topology and policies. By using network virtualization with SDN, clouds can provide a dynamic configuration of each tenant's network, similar to cloud resources.

However, in applying technologies in actual practice, previous works have shown limitations. In OpenVirteX (OVX) [1], each IP address of hosts is mapped to the address of the physical network in a one-to-one manner. In the case of FlowN [3], it uses virtual local area network (VLAN) to contain tenant identifier (TID). Both have limitations in scalability and resource utilization. They install all flow rules of tenants without checking duplicate flow rules, which decreases resource efficiency. Thus, ternary content addressable memory (TCAM), which consists of flow tables in a physical switch, exhausts rapidly. TCAM is expensive and limited in terms of physical space and power [9]; it is significant overhead for network virtualization. Furthermore, whenever a new flow of a virtual network is generated, control messages are sent to physical switches, such as 'PacketIn', which requests a new flow rule, and 'FlowMod', which is a reply for PacketIn. Thus, as the number of virtual networks increases, a larger number of control messages are generated, which is not efficient for use of a control

CAN'16, December 12 2016, Irvine, CA, USA

DOI: http://dx.doi.org/10.1145/3010079.3012012

channel. Because control channels, the channel between the control plane and data plane, is critical in SDN, efficient control channel usage is important for accurate and fast network management.

In this paper, we present a new address virtualization scheme, AggFlow, to increase scalability and improve resource efficiency. To improve scalability, we introduce mapping-less address virtualization, which enables virtual networks to manage a large number of hosts regardless of the network size while incurring low latency for packet processing. To improve resource efficiency, we propose flow rule aggregation using hop-by-hop forwarding. This paper is organized as follows. Section 2 explains our motivation. The mapping-less address virtualization and flow rule aggregation of AggFlow are presented in Section 3. Our evaluation results are in Section 4 and this paper concludes in Section 5.

2. MOTIVATION

In network virtualization with SDN, a network hypervisor receives flow rules from tenants with virtual addresses. A virtual address is a pair of source and destination addresses in the form of MAC or IP addresses. To forward a packet in a virtual network, address virtualization (e.g., address translation) in the network hypervisor is essential because each virtual network has a different address space. For address virtualization, VLAN-based technique [3] and one-to-one address mapping [1] have been proposed. The VLAN-based technique uses tunneling, which wraps packets with VLAN headers that contain a TID in VLAN ID fields. It is used to forward packets in heterogeneous networks. The one-to-one address mapping divides the physical address space and allocate the each division to virtual networks. In OpenVirteX, the eight bits of IPv4 address is used to identify the tenant. Then, each virtual IP is mapped on a physical IP in one-to-one manner and ingress edge switches rewrite virtual address, and egress switches reinstate the original address.

The VLAN-based technique and one-to-one address mapping both successfully virtualize network addresses. However, they have limitations in terms of scalability and resource efficiency. In terms of scalability, the ID field of VLAN is only 12 bits, which means that only 4,096 tenants can be identified. Also, one-to-one address mapping in OVX uses eight bits to identify tenants. So both are not scalable. In terms of resource efficiency, both methods require a considerable amount of network resources because all virtual flow rules are installed in the physical network with a one-to-one manner. For example, all flow rules generated by a tenant are installed in physical switches, which requires an immense amount of TCAM entries as the number of virtual network increases. Furthermore, the flow rules installed in the physical switches cannot be aggregated because each rule contains matching fields up to end-to-end identifying fields. And this leads to generating redundant control traffic for requesting and creating rules, and so the bandwidth of the control channel increases.

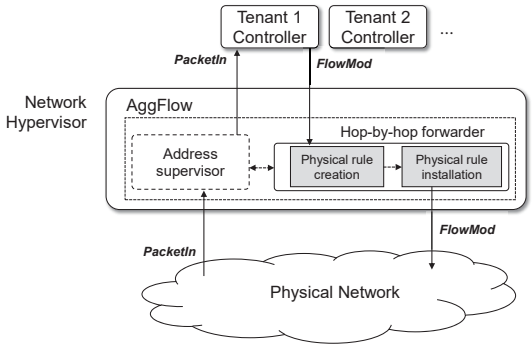


Figure 1: AggFlow design.

TID	Virtual MAC	Virtual IP	MAC of connected switch
1	01:00:00:00:00:01	192.168.10.1	A0:00:00:00:00:01
2	01:00:00:00:00:02	192.168.10.3	A0:00:00:00:00:01
2	01:00:00:00:00:03	192.168.10.2	A0:00:00:00:00:03
1	01:00:00:00:00:03	192.168.10.2	A0:00:00:00:00:02

Figure 2: Table of Address supervisor.

3. DESIGN

In this section, we outline the design of AggFlow, which is intended to improve the scalability and resource efficiency of network address virtualization. First, we present its overall design. We then detail mapping-less address virtualization for high scalability and hop-by-hop forwarding to minimize the TCAM usage and network bandwidth of the control channel.

3.1 AggFlow Structure

As shown in Figure 1, AggFlow consists of Address supervisor and Hop-by-hop forwarder. Address supervisor contains the address information of each host and provides information for enabling virtualization without address mapping (will be explained in Section 3.2). Hop-by-hop forwarder is used for translating a virtual rule into a physical rule based on hop-by-hop forwarding. It decreases the control channel traffic by checking the duplicate rules (details will be covered in 3.3).

As Figure 1 indicates, when a PacketIn message—a request for a new rule—arrives at a network hypervisor, Address supervisor receives the message and updates the table in Figure 2. And Address supervisor delivers it to the tenant controller. Then, the tenant controller sends the FlowMod message, which is a reply to the PacketIn message. After receiving the message, Hop-by-hop forwarder devirtualizes the address and installs the reconfigured rule to the switch.

3.2 Mapping-less Address Virtualization

The key of network virtualization is how to manage physical addresses and virtual addresses. If we allocate each virtual network address space with division of physical address space, as was done in previous re-

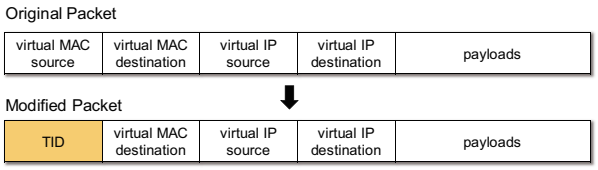


Figure 3: Resolution of overlapping addresses.

search, each virtual network address space cannot guarantee the full size of the address space (2^{48} for MAC and 2^{32} for IPv4). Furthermore, one-to-one mapping causes all virtual flow rules to be installed in physical switches, which means that the flow rule entries increase as the number of hosts increases. Therefore, AggFlow dynamically aggregates flow rules, rather than allocating physical addresses for virtual addresses in OVX.

Figure 2 shows the structure maintained by Address supervisor. It contains the TID, each host's MAC address, virtual IP address and MAC address of the connected edge switch. With this table, AggFlow reinstate the translated packet addresses into virtual addresses at egress switches.

The information for the table is collected as follows. When a virtual network is created, information on the hosts and the tenant where each host belongs is provided to network hypervisor (typically by network manager). Then, when a host sends packets that belong to a new flow, the edge switch receiving the first packet sends PacketIn to the network hypervisor. The message contains requests of flow rules and the first packet's header or entire packet. With that information, the Address supervisor updates the 'Virtual IP' field and 'MAC of connected switch' field. Because the IP address of hosts and connected edge switch vary, the Address supervisor updates the table with PacketIn.

To deliver packets to the destination, the minimum information that a packet should have is the location of its destination, which is the destination IP address. However, in a virtualized environment, the IP addresses can overlap across virtual networks. Thus, a packet should contain a TID to classify the packets of each tenant. It can be implemented using tunneling. However, tunneling without hardware offloading significantly increases the packet processing delay [5]. Moreover, OpenFlow, a de facto standard southbound interface, does not provide flexible tunneling actions; it only supports VLAN and multiprotocol label switching (MPLS).

To reduce the packet processing overhead, AggFlow borrows the idea of 'set' action from OpenFlow. OVX also uses 'set' action but in a different way – to translate virtual IP to physical IP address in one-to-one mapping. However, AggFlow does not use mapping at all – it put TID into the source MAC field at the ingress switch, as shown in Figure 3. This is mapping-less virtualization. In the packet processing, switches can recognize the previous hop of packets with 'in-port' field which is the port the packet came from. Therefore, we can use the source MAC field to contain TID. At the point where the destination information is needed in the forwarding path, this field needs to be matched. At the destination, the source MAC address is restored in the egress switch by the flow rule installed from network hypervisor. With this design, AggFlow supports both Layer 2 and Layer 3 address virtualization.

3.3 Hop-by-hop forwarder with flow rule aggregation

For the minimum number of flow rules in physical switches, we present a Hop-by-hop forwarder design. A key idea is to modify flow rules created by the tenant controller into flow rules that match only MAC addresses and in-port. With the modified flow rules, hop-by-hop forwarding is done. However, there is some cases in which the final destination should be identified. At the edge switches, IP addresses are needed for delivering packets to the end. Also, at the core switches, flows from the same previous hop can be forwarded to different ports according to the location of end hosts. To distinguish the flows, IP addresses should be matched.

First, AggFlow distinguishes the rules to be installed in edge or core switches. In edge switches, the flow rule that sets the TID and MAC addresses is installed. For core switches, Hop-by-hop forwarder modifies matching fields and actions of a rule to perform Layer 2 based forwarding. Both cases are handled in Algorithm 1 below (physical rule creation).

Algorithm 1 Pseudocode of creating a physical rule

```
1: procedure ConfiguringFlowMod
2:     VRule ← Rule from tenant controller
3:     PSwitch ← Switch where VRule to be installed
4:     PFlowTable ← Flow table structure of PSwitch
5:     if VRule.Match.InPort is the edge port then
6:         Insert set action of VRule.Actions
             {set source MAC as TID}
7:     end if
8:     if VRule.Actions.Output is the edge port then
9:         Insert set action of VRule.Actions
             {set MAC as (source's MAC,
             destination's MAC) and IP as (source's IP,
             destination's IP)}
10:    else
11:        Modify VRule.Match as
             {in-port = Previous Switch,
             destination MAC = PSwitch's MAC}
12:        Insert the set action into VRule.Actions
             {set destination MAC as Next Switch's
             MAC}
13:    end if
14:    Set VRule.Actions.Output as the physical one
15: end procedure
```

Physical rule creation: Algorithm 1 shows the creation of the physical rule based on the virtual rule created by the tenant.

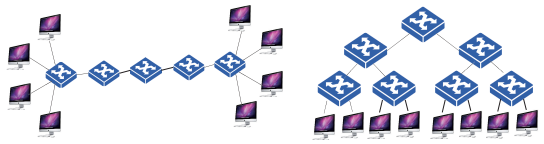
The rules are divided into two cases: those for the edge, and those for the core. First, in edge switches, Hop-by-hop forwarder adds the action of setting the source MAC address as the tenant number in the ingress node to distinguish final destination. For the egress node, it adds the reverse action that changes the tenant number of the source MAC field into the virtual MAC of a host. On the other hand, core switches change the matching fields and values into a in-port and destination MAC address pair, which is comprised of the previous switch information as the in-port and the current switch's MAC as the destination MAC. It furthermore adds an action that modifies the packet's destination MAC with the subsequent switch's MAC. At the end, it updates the port number of the output action, which is a virtual one, into a corresponding physical one.

To minimize the number of flow entries in a physical switch, AggFlow uses aggregation. If the matching field contains host-to-host information or virtual network information, the flow rules cannot be aggregated, as in existing IP networks. Therefore, flow rules in edge switches cannot be aggregated. On the other hand, core switches handling multiple virtual networks can have duplicate flow rules. AggFlow aggregates flow rules with same action as described in Algorithm 2 to minimize the number of flow rules installed in core switches.

Algorithm 2 Pseudocode for installing a physical rule

```
1: procedure InstallingFlowMod
2:     VRule ← Configured flow rule
3:     PSwitch ← Switch where VRule to be installed
4:     PFlowTable ← Flow table structure of PSwitch
5:     if VRule.Actions.Output is the edge port then
6:         Install VRule into PSwitch
7:     else
8:         for PRule in PFlowTable do
9:             if pRule.Match = vRule.Match then
10:                if pRule.Actions = vRule.Actions then
11:                    end procedure
12:                else
13:                    Extend VRule.Match to cover
                        MAC and IP address fields
14:                    Install VRule into PSwitch
15:                    Store VRule to PFlowTable
16:                    break loop
17:                end if
18:            end if
19:        end for
20:    end if
21:    Install VRule into pSwitch
22: end procedure
```

Physical rule installation: Core rules that result in the same action should be installed only once. To that end, AggFlow maintains the installed flow rules for each physical switches. Whenever it is requested to install a rule, it checks that the same rule has been installed, as illustrated in Algorithm 2.



(a) Linear topology (b) Tree topology

Figure 4: Topology.

First, if the rule is for edges, it is installed. Otherwise, the system checks the rule with the same matching field against a new rule to determine if it exists. If it exists, the system checks if the action fields are identical. If they are identical, the system omits the rule installation to avoid redundant flow installation. Because flow rules are installed using physical MAC address of switches, virtual networks having the same previous and next hop can have identical matching fields. However, if the action is not the same in spite of the same matching field, which means the flow has different end destination, longer matching is needed to identify them. Therefore, AggFlow extends the matching field to the MAC and IP address fields and installs the rule at the physical switch. If the rule was not previously installed, meaning that the matches or actions were not identical, the rule is installed. With this process, we can reduce the control channel traffic by reducing the number of control messages such as PacketIn and FlowMod, while increasing the CPU usage and flow rule installation delay. We evaluate the amount of overhead incurred from AggFlow in the next Section.

4. EVALUATION

In this section, we discuss the evaluation and results in terms of resource usage and overhead. We compare the results against those of OVX, a representative open-source network hypervisor that uses one-to-one mapping method. First, we evaluate the AggFlow resource usage in terms of the number of flow entries in the physical network and the number of messages for the control channel for a new flow. We additionally test the overhead in terms of processing latency of the network hypervisor and CPU utilization. We use Mininet [6] as an SDN network testbed. AggFlow is implemented on OVX. An ONOS controller [2] is used as tenant controller. Mininet, AggFlow and ONOS controller run on separated machines, they are connected each other with a 10-GbE switch. Even though OVX is used for implementation of AggFlow, AggFlow can be implemented on other network hypervisors. Also, other SDN controllers such as Floodlight, NOX, and OpenDaylight can be used as tenant controller.

For topology, we evaluate AggFlow in linear and tree topologies, as shown in Figure 4. The linear topology consisted of five switches and four hosts at each end.

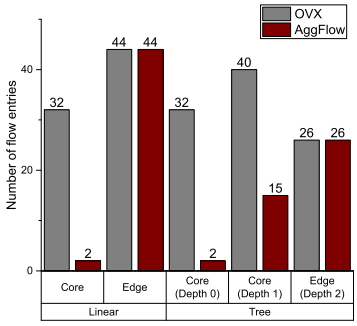


Figure 5: Number of flow rule entries.

The tree topology has a depth of three, and each edge node has two hosts. For both topologies, we create one virtual network. Its topology is the same as that of the physical network. Traffic is generated using pings, and each host creates ping traffic to all the other hosts, with a total of 56 flows.

4.1 Resource Efficiency

For resource usage, we evaluate the number of flow entries in a physical switch that shows the aggregation power of AggFlow. We additionally measure the network bandwidth for FlowMod messages to show the control channel usage.

TCAM usage: We show the results of switch resource usage in Figure 5. Compared to OVX, in the core, the number of switch flow entries dramatically decreases (up to 16 times) because flow entries are aggregated by hop-by-hop forwarding. However, the number of edge switch rules does not decrease because the edge switches should perform the actions of setting and recovering the physical and virtual addresses. Namely, AggFlow does more flow aggregation in core switches than edge switches.

Control channel usage: We measure the number of FlowMod messages generated for all-pair ping traffic. OVX creates 139 messages in a linear topology and 210 messages in a tree topology. AggFlow creates 94 and 178 messages, respectively, which is 32% and 15% lower than OVX. We also measure the average network bandwidth used for the FlowMod messages. In the linear topology, OVX and AggFlow show 11,380 bytes per second (BPS) and 5,535 BPS, respectively. For the tree topology, each shows 12,660 BPS and 8,981 BPS. That is, AggFlow reduce the average network bandwidth by 51% and 29% for each topology. The decreased usage of the control channel is due to the flow rule aggregation and redundancy checking in the network hypervisor. This improves the scalability of the control range of the physical network, although the additional process adds some overhead to the control plane.

4.2 Overhead

To evaluate the overhead incurred by AggFlow, we compare AggFlow with OVX that only changes address fields. AggFlow reconfigures the entire matching and action fields to aggregate flow entries. First, we measure the delay and CPU utilization for configuring the flow rules in the network hypervisor. Next, we describe the packet processing delay in the switch, which shows the virtualization overhead of the control plane.

Table 1: Delay for translating FlowMod messages (ms).

Topology	OVX	AggFlow	Increased delay
Linear	4.633	4.728	0.095
Tree	4.679	4.780	0.101

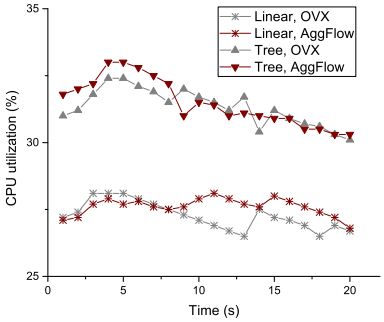


Figure 6: CPU utilization of the network hypervisor.

Control plane delay: Table 1 shows the average delay for translating a flow rule from the tenant. Compared to one-to-one mapping, AggFlow incurs an additional delay for checking the redundant flow rules. However, the increased delay is 0.095 ms and 0.101ms for the linear and tree topologies, respectively. These results are 2.05% and 2.16% of the one-to-one mapping address translating delay, which is not significant.

CPU usage: Figure 6 shows the CPU utilization of the network hypervisor for a period of processing FlowMod messages. On average, one-to-one mapping and AggFlow show 31.33% and 31.495%, respectively, of CPU utilization in the tree topology. In the linear topology, they show 27.27% and 27.61%, respectively. Therefore, AggFlow uses almost the same CPU time.

Packet processing delay in data plane: To observe the packet processing delay for AggFlow in physical switches, we compare the UDP throughput between the end hosts of AggFlow in physical switches with those of OVX using Iperf [11] in the tree topology. Each method shows identical throughput of 1.048 Mbits/sec, which means that the two methods have identical effects for long-time traffic. Then, to measure the effect on each packet, we measure the average round trip time (RTT) for each iteration of ping traffic.

Figure 7 shows the results of the two methods for the tree topology, and we perform the experiment on two cases. The first case is "with a pre-rule," whereby the flow traversing the same path of the experiment traffic already flowed. It means that the rules for the path already exist. In the second case, "without a pre-rule," no rule exists. For the latter case, both methods show almost the same RTT for each ping iteration, which means they incur the same per-flow processing overhead in the data plane. However, for the former case, the

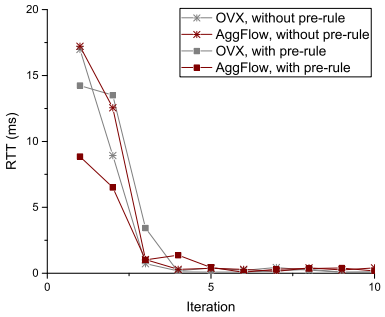


Figure 7: Round trip time of each ping iteration.

two methods work differently at the beginning phase. AggFlow shows a lower RTT in the beginning period because flows traversing the same path of existing flow can be forwarded by the already installed rules, rather than requiring new rules.

5. CONCLUSION

SDN-based network virtualization enables dynamic virtual network configuration, which is a significant advance for cloud centers. However, existing methods have limitations in supporting high scalability and efficient resource management. In this paper, we therefore propose an address virtualization technique, AggFlow, to reduce the required TCAM entries in the physical switches and also the required messages for the control channel. In the meantime, it supports high scalability of the address space. To achieve our AggFlow design goals, we introduce two techniques: mapping-less address virtualization and hop-by-hop forwarding. Our evaluation results show that AggFlow reduces the flow entries installed in the core network and the traffic for installing the flow rules. Overhead for AggFlow is very small in terms of the delay, CPU usage, and packet throughput. Moreover, AggFlow achieves a fast RTT at the beginning phase of the flow transmission.

In future work, we plan to individually support quality of service (QoS) or service-level agreements in virtual networks. We will extend AggFlow into a flow virtualization framework that dynamically maps physical flows and virtual flows, depending on the QoS requests and requirements of the virtual flow processing.

6. ACKNOWLEDGMENTS

The authors appreciate anonymous reviewers for their valuable comments. This work was partly supported by Institute for Information & communications Technology Promotion (IITP) grant funded by the Korea government (MSIP) (No.B0126-16-1046, Research of Network Virtualization Platform and Service for SDN 2.0 Realization) and (No.R0126-16-1066,(SW Starlab) Next generation cloud infra-software toward the guarantee of performance and security SLA).

7. REFERENCES

[1] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow. Openvirtex: Make your virtual sdns programmable. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 25–30. ACM, 2014.

[2] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6. ACM, 2014.

[3] D. Drutskoy, E. Keller, and J. Rexford. Scalable network virtualization in software-defined networks. *IEEE Internet Computing*, 17(2):20–27, 2013.

[4] D. Farinacci, T. Li, S. Hanks, D. Meyer, and P. Traina. Generic routing encapsulation (gre). Technical report, 2000.

[5] S. Guenender, K. Barabash, Y. Ben-Itzhak, A. Levin, E. Raichstein, and L. Schour. Noencap: overlay network virtualization with no encapsulation overheads. In *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, page 9. ACM, 2015.

[6] B. Lantz, B. Heller, and N. McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 19. ACM, 2010.

[7] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. Virtual extensible local area network (vxlan): A framework for overlaying virtualized layer 2 networks over layer 3 networks. Technical report, 2014.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[9] J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, A. R. Curtis, and S. Banerjee. Devoflow: Cost-effective flow management for high performance enterprise networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, page 1. ACM, 2010.

[10] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar. Flowvisor: A network virtualization layer. *OpenFlow Switch Consortium, Tech. Rep*, pages 1–13, 2009.

[11] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, and K. Gibbs. Iperf: The tcp/udp bandwidth measurement tool. *htt p://dast. nlanr. net/Projects*, 2005.