Scalability of ONOS reactive forwarding applications in ISP networks

Scalability of ONOS reactive forwarding applications in ISP networks

Andrea Bianco[a], Paolo Giaccone[a], Reza Mashayekhi[a], Mario Ullio[b], Vinicio Vercellone[b]

[a] *Dip. di Elettronica e Telecomunicazioni, Politecnico di Torino, Italy*
[b] *Telecom Italia, Torino, Italy*

Abstract

Software Defined Networking (SDN) is a powerful approach that enhances network control and management, and provides a flexible way to develop network applications. However, scalability of SDN networks is an important concern for many network operators. The main peculiarities of SDN when applied to an Internet Service Provider (ISP) network are the large geographical extension and the need of in-band transmission of control traffic. Therefore, the control traffic exchanged between the SDN controller and the network nodes must be carefully evaluated for the network design and dimensioning.

We consider an ISP network controlled by the recently ONOS (Open Network Operating System) controller developed by ON.Lab. We devise a quantitative model to compute the exact number of exchanged OpenFlow messages and the corresponding bandwidth needed to install a traffic flow when running the default ONOS layer-2 forwarding applications. We compute also the exact number of flow rules installed in each switch. We show the general applicability of our models for a Point Of Presence (POP) network and for a large set of real nation-wide and world-wide ISP networks.

Our quantitative models can be used for a safe network planning also when the network applications are not fully reactive.

Keywords: Software Defined Networking, control traffic, reactive forwarding applications

1. Introduction

Software Defined Networking (SDN) is the leading paradigm to enable an efficient separation between data and control plane and allows the de-

velopment of advanced network applications by introducing a logically centralized controller. The controller has a complete view of the network and is responsible to compute the routing paths and to implement the desired forwarding and dropping rules. Therefore, the design of the switching devices can be simplified because they have to support only the data plane. As such, the SDN approach is considered an enabling technology for advanced traffic engineering.

The adoption of SDN in legacy networks of Internet Service Providers (ISP) is challenging both in terms of performance and reliability. The difficulty is exacerbated by the fact that the wide geographical extension imposes an in-band control plane, i.e. the control traffic is sharing the same network resources than the data traffic. To properly design the network and assess the scalability of the adopted SDN approach, a quantitative model is needed to investigate the bandwidth requirements for the control plane.

Motivated by this observation, in our paper we provide the following contributions. We propose a model to estimate the bandwidth consumption due to control traffic on the southbound interface of the SDN controller: we compute the number of OpenFlow control messages required to install traffic flows for fully reactive forwarding applications, in which the switch asks the controller for instructions anytime a new flow arrives. We have chosen such applications because they provide a reference approach to achieve maximum network programmability and flexibility. Furthermore, they constitute a worst-case scenario in terms of control traffic, since reactive forwarding provides an upper bound for the amount of control traffic with respect to other fully or partly proactive applications, which trade reduced control traffic for limited network programmability. We also evaluate the exact number of flow rules installed in the switches for each flow, by which it is possible to understand the maximum scalability of the reactive applications, given the maximum memory available to store the flow tables internally in the switches. Notably, the memory size for flow rules depends on the specific hardware/software architecture implementing the flow tables (e.g., hash tables, Patricia tries, TCAMs, etc.) and thus the feasibility of a network application depends actually on the deployed switches. In our work we are neglecting the additional flow setup delays due to the reactive approach, which could be another limiting factor to the scalability of reactive approaches.

In particular, we develop a model tailored to the two standard layer-2 forwarding applications available in the ONOS (Open Network Operating System) controller [1], developed by ON.Lab [2], which is one of the most practical relevant open-source controllers targeting ISP networks. Both for-

warding applications are reactive at the MAC/IP flow level, i.e., rules are installed in the switches on-the-fly, upon the identification of new flows at MAC/IP level. To show the general applicability of our model, we first apply it to a realistic network topology present in the Point Of Presence (POP) access network of a nation-wide ISP. Finally, we apply the model to a set of 242 real ISP network topologies and highlight the role of the internal structure of the graph describing the topology.

A similar analysis was presented in [3] for the OpenDaylight controller [4] running the default reactive forwarding application, in the scenario of a single POP access network. Even if the methodology is similar, the control traffic in OpenDaylight cannot be compared with the one considered in this paper. Indeed, the specific forwarding application considered in [3] is based on broadcasting any forwarding rule across all the network switches, independently on the actual switches involved in the forwarding process along the path between the source and the destination, thus by construction the control traffic is much higher than the one considered in our scenario. Nevertheless, also for the reactive applications in ONOS, we will show that the amount of OpenFlow control traffic can be relatively large, with respect to the data transferred by each flow.

We expect a practical relevance of our models also for other applications. Indeed, reactive applications provide the network designers with the worst-case control traffic due to OpenFlow protocol and with the maximum number of flow rules installed in each switch. This allows a safe design and plan of the control network, based on the specific topology of the network supporting the control plane and on the flow arrival pattern.

The paper is organized as follows. Sec. 2 discusses the related works. Sec. 3 briefly describes the SDN approach and the standard forwarding applications available in ONOS. In Sec. 4 we analyze the message exchange on the control plane for the considered network applications and propose the detailed model for its quantitative evaluation in terms of traffic and installed rules for the two considered applications. In Sec. 5 we introduce a realistic POP topology on which we evaluate the control traffic thanks to our model and in Sec. 6 we apply our quantitive model for a large set of ISP networks. Finally, in Sec. 7 we draw our conclusions.

2. Related Work

The scalability of SDN networks for large networks has been widely investigated during the last years. One approach to improve the scalability of SDN networks is to decrease the interaction on the control plane,

i.e. between the switches and the controller/s, by delegating some control capabilities to the network devices. A relevant contribution is [5], which proposed to add specific control functionalities on the switches in order to balance the flexibility to program the data plane and the scalability of the control plane. Some other works such as DevoFlow [6] and SDCs [7] also aimed at decreasing control plane overhead by delegating some forwarding decisions to the devices in the data plane. In all these approaches, the forwarding devices complement the controller's decisions and take the responsibility for some actions on specific flows. An example of such approach was proposed by [8], according to which the switch asks the controller for instructions selectively only on elephant flows. The approach is based on a flow classification scheme implemented through on a two-stage adaptive system. Another relevant work is control protocol named OpFlex [9]. Unlike OpenFlow, OpFlex distributes some parts of network control to forwarding devices in order to increase scalability of control plane. OpenState [10] is another recent control protocol exploiting stateful programming abstractions and introducing finite state machines implemented with each switch. This is an extension of the OpenFlow match/action concept and it is compatible with the standard TCAM-based implementation of OpenFlow flow tables.

Another approach to address the scalability is to improve the processing capabilities of the controllers. This is achieved by exploiting advanced and highly efficient computing techniques such as parallelism and pipelining. Examples of such approach are: Beacon [11] and Maestro [12] which use multi-threaded controllers, Kandoo [13] that utilizes a hierarchical control platform, and HyperFlow [14] and ONOS [1] (considered in our work) which exploit distributed controllers.

Recently, [15] has addressed SDN scalability in both data and control planes through hardware accelerators. They utilized the generic x86 CPU as forwarding engine and RAM to handle the chained flow tables and achieve forwarding capabilities of Gbps. All the mentioned works aim at increasing scalability of SDN control plane. Finally, [16] has proposed the response time perceived at the switches as metric of scalability for SDN control planes, and studied three typical SDN control plane structures (centralized, decentralized and hierarchical architectures).

Differently from all the previous works, we are not improving the scalability of SDN networks, but we are evaluating the amount of exchanged traffic in the control plane between the switches and the controller/s and the number of installed flow rules per flow, in order to provide some quantitive model to understand the maximum scalability of reactive network applications, given an available network topology and a flow arrival pattern.

3. Packet forwarding in SDN

We consider a physically centralized implementation of an SDN network, where a single controller is responsible for the control plane decisions in all the switching nodes. Note that practical implementations are based on distributed controllers (as better detailed in Sec. 3.1), which appear as a logically centralized controller to enable a consistent network view. Our results in terms of generated OpenFlow traffic are general and are independent from the distributed architecture of the controller.

We can identify two opposite operational modes, showing a different impact on the scalability and reactivity of the SDN control [17]. In the *reactive mode*, the forwarding decisions are taken whenever a new flow (i.e. a flow with no previously installed rule) arrives at a switch. Indeed, the switch sends a copy of the packet (or its header) to the controller and the controller installs a forwarding rule in the switch to route all the following packets in the flow. This mode enables the implementation of flexible online routing/dropping policies, without any preliminary traffic knowledge. Furthermore, flow entries are added only when needed, reducing the memory requirements in the switches. On the other side, an overhead in terms of control bandwidth and delay is experienced for each new flow. In the *proactive mode* the controller pre-populates flow entries on network switches. This mode avoids the initial flow overhead needed to dynamically install rules in the switch. However, it requires a preliminary knowledge of the routing rules applicable in the network.

In our work we focus only on the reactive forwarding scenario, because it represents the worst-case in terms of the amount of generated control traffic and flow rules installed in switches, while achieving the highest flexibility in network programmability. If we consider SDN networks with reactive applications exploiting OpenFlow (OF) [18], the control traffic is due to different OF messages types, devoted to the initial device configuration, the flow setup, and the keep-alive mechanisms. The flow setup phase has the strongest impact on the control traffic, because it requires the higher frequency of interaction between the switch and the controller when the traffic in the network is significant. Indeed, when the first packet of an unmatched flow reaches a switch, the switch sends a *pkt_in* message to the controller with a copy of the packet (or its header). The controller processes the message and sends back to the switch a *pkt_out* message with the action (mainly, "drop" or "forward to a specific output port") to handle the received packet. Furthermore, the controller sends a *flow_mod* message to install a new forwarding rule on the switch. Subsequent packets belonging to the same flow

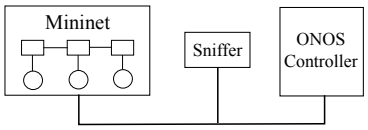

Figure 1: Experimental scenario to evaluate the control traffic in ONOS

match the forwarding rule and are directly processed in the switch, without any further interaction with the controller. Note that *barrier* messages can be sent after the *flow_mod* to guarantee a correct installation and processing of the forwarding rule for all the packets currently buffered in the switch.

3.1. Distributed Control Plane

As previously mentioned, many practical implementations are based on distributed controllers, due to their higher scalability and reliability. In such scenarios, two kinds of control traffic are present in the network. First, the southbound control traffic (typically based on OpenFlow protocol) between the controller/s and the switching devices. Second, the east-west control traffic, which is the inter-controller traffic to keep the shared data consistent among the controllers. Depending on the control platform and the level of consistency (e.g., strong/eventual) to provide on the data, ad-hoc or proprietary consensus algorithms and protocols are adopted [19].

In our work we focus just on the first kind of traffic, in particular OpenFlow. Note that this traffic is completely independent from the east-west traffic, since the distributed architecture of the controllers must operate transparently for the switch. Thus, our model is general and independent of the distributed architecture of the controller.

3.2. Reactive Forwarding in ONOS

Open Network Operating System (ONOS) is an open-source distributed SDN control platform, developed by the Open Networking Lab (ON.Lab) organization [1], and supported by some of the leading industries and academic institutions. Differently from OpenDaylight [4], the ONOS project specifically addresses ISP networks, by providing high availability and scalability, thanks to its distributed architecture. In December 2014 the first version of ONOS (1.0.0, named Avocet) was released and our results apply to this specific release. ONOS provides two default reactive forwarding applications: *simple reactive forwarding application* (denoted as "onos-app-fwd"), and *intent-based simple reactive forwarding application* ("onos-app-ifwd"), respectively named `fwd` and `ifwd` in our paper.

To compute the exact number of messages exchanged in the control plane for these applications, we adopt two complementary approaches. First, we use the SDN network emulator *mininet* [20] to create a test network topology. Following the scheme shown in Fig. 1, we run an instance of ONOS with `fwd` or `ifwd` applications, and generate traffic on mininet nodes. Through a network sniffer, we analyze the control traffic due to the OF messages exchanged between each emulated switch and ONOS. Second, to validate our understanding of the interaction on the control plane and generalize our findings to any network, we also analyze in details the ONOS source code. We evaluate the number of flow rules added in a switch through the number of received *flow-mod* packets. In theory, *flow-mod* messages could carry multiple flow rules, but we have never observed any rule bundling in our traces.

Both reactive applications in ONOS work at layer-2 and assume a single IP network. Each flow is identified by a source/destination address pair, both at MAC and IP level. Thus, one flow can comprise multiple sessions at higher protocol levels (e.g. TCP, UDP or RTP).

Within a switch, one port connected to another switch is denoted as *internal port*, whereas one port connected to a host (client and/or server) is denoted as *host port*. Note that a host port can be connected to many hosts (e.g., through non-OF switches/hubs). The topology interconnecting the switches (i.e. the internal ports and the communication links among them) is known in advance to the controller, thanks to the preliminary phase of topology discovery based on the LLDP [21] protocol, which is implemented by the controller by sending specific *pkt_out* messages to the switches. Instead, the *host location*, i.e. the port where the host is connected, is a-priori unknown and must be discovered in real time by the controller. Sec. 4 describes in detail the behavior of both `fwd` and `ifwd` applications in ONOS and the process followed to discover the host location.

4. ONOS control traffic

We consider the flow setup process to establish a bidirectional communication between a source host H_S and a destination host H_D, assuming that the corresponding host ports are unknown to the controller. Let P_{SD} be the number of switches along the shortest path from H_S to H_D. Let N be the total number of switches in the network, and let E be the number of inter-switch communication links. We evaluate the exact number of OF messages exchanged for a new flow setup between H_S and H_D, for the `fwd` Sec. 4.1) and the `ifwd` applications (Sec. 4.2). For simplicity, we will refer

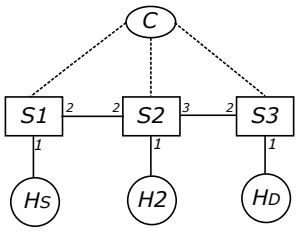

Figure 2: Example of network connecting 2 hosts H_S and H_D through a path of $P_{SD} = 3$ switches. In dashed the communication links with the ONOS controller C.

to the linear topology shown in Fig. 2 as a reference case, but our results apply in general to any given topology. We assume that the ARP tables of both H_S and H_D and the forwarding tables of all the switches are initially empty.

4.1. ONOS `fwd` *application*

In Tab. 1 we report the detailed sequence of packets exchanged in the data and control planes for the `fwd` application.

During phase 1, the first generated packet from H_S is an ARP request (ARP-REQ). This message reaches the local switch (denoted as *root switch*) to which the source host is connected. This ARP packet is sent to the ONOS controller through a *pkt_in* message, thus allowing the controller to learn the source host port. Now, the ARP-REQ is flooded across all the switches in the network. Note that the flooding is interrupted as soon as the destination is reached. In more detail, the controller sends a *pkt_out* message to the root switch, encapsulating the same ARP-REQ packet with the action to flood it to all the ports except the one from which the original packet was received. After receiving this message, the root switch floods the ARP-REQ to $S2$. Upon the reception of this packet, $S2$ sends a *pkt_in* to the controller, which replies with a *pkt_out* to flood the ARP-REQ to $H2$ and $S3$. This procedure continues until the ARP-REQ reaches the destination H_D.

In phase 2, H_D sends an ARP reply (ARP-REP) to the attached switch, but this packet is not flooded any more like in phase 1, because it is forwarded to H_S through the shortest path. Indeed, H_D's switch sends the *pkt_in* message to the controller. Now the controller learns the host port of H_D. Then, it computes the shortest path from $S3$ to $S1$, at which H_D is attached and issues a *pkt_out* to send the ARP-REP to $S2$. The procedure continues until the ARP reaches H_S. Note that, so far, no *flow_mod* has been sent by the controller and so all the forwarding tables inside the switches are empty.

Table 1: Sequence of packets exchanged during flow setup $H_S \rightarrow H_D$ for ONOS `fwd` application, for the network of Fig. 2.

Phase	Direction	Packet
1	$H_S \rightarrow$ S1	ARP-REQ:H_D?
	S1 $\rightarrow$ C	*pkt_in* (ARP-REQ:H_D?)
	C $\rightarrow$ S1	*pkt_out* (ARP-REQ:H_D?):flood
	S1 $\rightarrow$ S2	ARP-REQ:H_D?
	S2 $\rightarrow$ C	*pkt_in* (ARP-REQ:H_D?)
	C $\rightarrow$ S2	*pkt_out* (ARP-REQ:H_D?):flood
	S2 $\rightarrow$ H2	ARP-REQ:H_D?
	S2 $\rightarrow$ S3	ARP-REQ:H_D?
	S3 $\rightarrow$ C	*pkt_in* (ARP-REQ:H_D?)
	C $\rightarrow$ S3	*pkt_out* (ARP-REQ:H_D?):flood
	S3 $\rightarrow H_D$	ARP-REQ:H_D?
2	$H_D \rightarrow$ S3	ARP-REP:H_D
	S3 $\rightarrow$ C	*pkt_in* (ARP-REP:H_D)
	C $\rightarrow$ S3	*pkt_out* (ARP-REP:H_D):output port 2
	S3 $\rightarrow$ S2	ARP-REP:H_D
	S2 $\rightarrow$ C	*pkt_in* (ARP-REP:H_D)
	C $\rightarrow$ S2	*pkt_out* (ARP-REP:H_D):output port 2
	S2 $\rightarrow$ S1	ARP-REP:H_D
	S1 $\rightarrow$ C	*pkt_in* (ARP-REP:H_D)
	C $\rightarrow$ S1	*pkt_out* (ARP-REP:H_D):output port 1
	S1 $\rightarrow H_S$	ARP-REP:H_D
3	C$\rightarrow$S3, C$\rightarrow$ S2, C$\rightarrow$S1	*flow_mod*($H_D \rightarrow H_S$, EthType=ARP)
	C$\rightarrow$S3, C$\rightarrow$ S2, C$\rightarrow$S1	*barrier_request*
	S3$\rightarrow$C, S2$\rightarrow$C, S1$\rightarrow$C	*barrier_reply*
4	$H_S \rightarrow$ S1	IP $H_S \rightarrow H_D$
	S1 $\rightarrow$ C	*pkt_in* (IP $H_S \rightarrow H_D$)
	C $\rightarrow$ S1	*pkt_out* (IP $H_S \rightarrow H_D$):output port 2
	S1 $\rightarrow$ S2	IP $H_S \rightarrow H_D$
	S2 $\rightarrow$ C	*pkt_in* (IP $H_S \rightarrow H_D$)
	C $\rightarrow$ S2	*pkt_out* (IP $H_S \rightarrow H_D$):output port 3
	S2 $\rightarrow$ S3	IP $H_S \rightarrow H_D$
	S3 $\rightarrow$ C	*pkt_in* (IP $H_S \rightarrow H_D$)
	C $\rightarrow$ S3	*pkt_out* (IP $H_S \rightarrow H_D$):output port 1
	S3 $\rightarrow H_D$	IP $H_S \rightarrow H_D$
5	C$\rightarrow$S1, C$\rightarrow$ S2, C$\rightarrow$S3	*flow_mod*($H_S \rightarrow H_D$, EthType=IP)
	C$\rightarrow$S1, C$\rightarrow$ S2, C$\rightarrow$S3	*barrier_request*
	S1$\rightarrow$C, S2$\rightarrow$C, S3$\rightarrow$C	*barrier_reply*
6	*Usual IP forwarding from H_S to H_D*	
7	Same as phase 4	IP $H_D \rightarrow H_S$
8	C$\rightarrow$S1, C$\rightarrow$ S2, C$\rightarrow$S3	*flow_mod*($H_D \rightarrow H_S$, EthType=IP)
	C$\rightarrow$S1, C$\rightarrow$ S2, C$\rightarrow$S3	*barrier_request*
	S1$\rightarrow$C, S2$\rightarrow$C, S3$\rightarrow$C	*barrier_reply*
9	*Usual IP forwarding from H_D to H_S*	

In phase 3, after the ARP reply has reached H_S, the controller installs the forwarding rule to allow any unicast ARP packet from H_D to H_S to reach the source, on all the P_{SD} switches in the shortest path from H_D to H_S. Flow establishment messages include: a *flow_mod* message and a *barrier_request* message sent from the controller to each switch, and a *barrier_reply* message sent from the switch to the controller.

During phase 4, the first IP packet is sent from H_S to H_D. When the packet reaches the root switch, it triggers a *pkt_in*, because the forwarding table has been set only for ARP packets. The controller now instructs the root switch to send the packet on the port corresponding to the shortest path towards H_D. This procedure keeps repeating until the IP packet reaches H_D.

In phase 5, similarly to phase 3, the controller installs a layer-2 rule for all IP packets from H_S to H_D in all the switches along the shortest path. From now on, all the IP packets from H_S to H_D do not trigger any interaction with the controller, thanks to the installed forwarding rules (phase 6).

When H_D sends back an IP packet to H_S (phase 7), this packet is transferred along the shortest path with a similar sequence of *pkt_in* and *pkt_out* messages observed in phase 4. Only after reaching H_S, in phase 8 a *flow_mod* is used to install a layer-2 rule on all the switches along the shortest path for all the IP packets from H_D to H_S. From now on, also the IP packets from H_D to H_S are forwarded directly without interaction with the controller (unless the rule timer expires).

At high level, we can summarize the whole message exchange by the following properties:

- P1: the host positions are discovered by the ARP packets sent by the source and destination hosts, allowing the controller to compute the shortest path among the two hosts.

- P2: the application mimics the flooding occurring at layer 2 in a single broadcast domain for the ARP request packet, but the flooding is interrupted as soon as the destination host is reached; each ARP request is sent to the controller by the traversed switches.

- P3: the ARP reply is sent to the source host without flooding; each ARP reply is sent to the controller by the traversed switches.

- P4: the first IP packet for each direction is sent hop-by-hop to the destination; each IP packet is sent to the controller by the traversed switches.

Table 2: OpenFlow messages for a new flow in `fwd` application

Phase	m_{pi}	m_{po}	m_{fm}	m_{bq}	m_{bp}
1	$(P_{SD}, 2E]$	$[P_{SD}, N]$	-	-	-
2, 4, 6	P_{SD}	P_{SD}	-	-	-
3, 5, 7	-	-	P_{SD}	P_{SD}	P_{SD}
Total	$(4P_{SD}, 2E + 3P_{SD}]$	$[4P_{SD}, N + 3P_{SD}]$	$3P_{SD}$	$3P_{SD}$	$3P_{SD}$

- P5: the installed forwarding rules are defined at layer-2 and refer specifically to the unicast ARP reply packets and the IP packets, in both directions. The rules are installed along all the switches occurring in the shortest path, after the first end-to-end packet exchange has been successful. Thus, a total of 3 flow rules are added in each switch along the shortest path for each new flow.

Focusing on the OF messages shown in Tab. 1, it is possible to generalize and compute the total number of exchanged OF messages in *any* topology with N nodes, for a single flow traversing P_{SD} switches. The number is evaluated in terms of number of *flow_mod* messages (denoted as m_{fm}), *pkt_in* messages (denoted as m_{pi}), *pkt_out* messages (denoted as m_{po}), *barrier_request* messages (denoted as m_{bq}), and *barrier_reply* messages (denoted as m_{bp}). The number of flow rules installed in each switch is given by m_{fm}/P_{SD}. Let d_s be the number of internal ports for switch s, let also $d_{\min} = \min_s d_s$ be the minimum number of internal ports, and let E be the total number of links connecting all the switches. By construction, $E = \sum_s d_s/2$. Finally, let Ω_{SD} be the set of all the switches along the shortest path from H_S to H_D.

Considering the phases identified in Tab. 1, for a generic topology we have the total number of messages shown in Tab. 2. The total number of *pkt_in* and *pkt_out* generated during phase 1 depends on the number of flooded ARP requests. We provide an upper and a lower bound for it. The minimum value of switches involved in the flooding is P_{SD}, whenever the flooding is interrupted as soon as all the switches along the shortest path have received the ARP request. In this case, as lower bound we can claim that $m_{po} \geq P_{SD}$ and $m_{pi} > P_{SD}$ since

$$m_{pi} = P_{SD} + \sum_{s \in \Omega_{SD}} d_s \geq P_{SD} + P_{SD} d_{\min} = P_{SD}(1 + d_{\min})$$

The maximum values of switches involved in the flooding is N, each of

them receiving one *pkt_out* from the controller for the flooding, and sending a copy of the ARP to all the neighboring switches, which will send the corresponding *pkt_out* to the controller. Thus, in the worst case, all the inter-switches links are involved in the flooding by sending *pkt_in*. In conclusion, the upper bounds to the number of OF messages are $m_{po} \leq N$ and $m_{pi} \leq 2E$.

We can summarize the above results as follows:

Property 1. *In Simple Reactive Forwarding Application (*`fwd`*) of ONOS, for each new flow traversing P_{SD} switches, the following relations hold for the number of OF messages:*

$$m_{pi} \in (4P_{SD}, 2E + 3P_{SD}]$$
$$m_{po} \in [4P_{SD}, N + 3P_{SD}]$$
$$m_{fm} = m_{bq} = m_{bp} = 3P_{SD}$$

The number of rules installed per flow in each switch along the routing path is 3.

4.2. ONOS `ifwd` *application*

This application is very similar to `fwd` but it reduces the number of OF messages. The main idea is to use the control plane (i.e. the controller) to forward some packets from the source to the destination, thus "shortcutting" the path on the data plane. In the following, we explain in detail its behavior.

Tab. 3 shows the sequence of packets exchanged for the same toy network of Fig. 2. The first difference between `ifwd` and `fwd` occurs in phase 2, when the ARP reply from H_D is received by the controller. Instead of mimicking the standard routing of the ARP reply packet along the shortest path, as for `fwd`, the controller directly sends the ARP to the root switch $S1$. The installation of the flow rules for the ARP (as in phase 3 of `fwd`) does not occur anymore. Furthermore, the first IP packet (i.e. ICMP request) is directly forwarded through the controller from S1 to S3 (phase 3) and also the answer IP packet in the reverse direction (i.e. ICMP reply) is directly forwarded through the controller (phase 4). Thus, in phases 2, 3, 4 the controller is shortcutting the routing path from/to H_S to/from H_D, acting as "software switch". Now the controller installs all the forwarding rules for the couple H_S and H_D of MAC addresses along the path; in phase 5 for the direction from H_S to H_D, and in phase 6 in the reverse direction. Finally (phase 7), thanks to the installed forwarding rules at MAC level, all

Table 3: Sequence of packets exchanged during flow setup $H_S \rightarrow H_D$ for ONOS ifwd application, for the network of Fig. 2.

Phase	direction	Packet
1	$H_S \rightarrow$ S1	ARP-REQ:H_D?
	S1 $\rightarrow$ C	*pkt_in* (ARP-REQ:H_D?)
	C $\rightarrow$ S1	*pkt_out* (ARP-REQ:H_D?):flood
	S1 $\rightarrow$ S2	ARP-REQ:H_D?
	S2 $\rightarrow$ C	*pkt_in* (ARP-REQ:H_D?)
	C $\rightarrow$ S2	*pkt_out* (ARP-REQ:H_D?):flood
	S2 $\rightarrow$ H2	ARP-REQ:H_D?
	S2 $\rightarrow$ S3	ARP-REQ:H_D?
	S3 $\rightarrow$ C	*pkt_in* (ARP-REQ:H_D?)
	C $\rightarrow$ S3	*pkt_out* (ARP-REQ:H_D?):flood
	S3 $\rightarrow H_D$	ARP-REQ:H_D?
2	$H_D \rightarrow$ S3	ARP-REP:H_D
	S3 $\rightarrow$ C	*pkt_in* (ARP-REP:H_D)
	C $\rightarrow$ S1	*pkt_out* (ARP-REP:H_D):output port 1
	S1 $\rightarrow H_S$	ARP-REP:H_D
3	$H_S \rightarrow$ S1	IP $H_S \rightarrow H_D$
	S1 $\rightarrow$ C	*pkt_in* (IP $H_S \rightarrow H_D$)
	C $\rightarrow$ S3	*pkt_out* (IP $H_S \rightarrow H_D$):output port 1
	S3 $\rightarrow H_D$	IP $H_S \rightarrow H_D$
4	$H_D \rightarrow$ S3	IP $H_D \rightarrow H_S$
	S3 $\rightarrow$ C	*pkt_in* (IP $H_D \rightarrow H_S$)
	C $\rightarrow$ S1	*pkt_out* (IP $H_D \rightarrow H_S$):output port 1
	S1 $\rightarrow H_S$	IP $H_D \rightarrow H_S$
5	C$\rightarrow$S1, C$\rightarrow$ S2, C$\rightarrow$S3	*flow_mod*($H_S \rightarrow H_D$)
	C$\rightarrow$S1, C$\rightarrow$ S2, C$\rightarrow$S3	*barrier_request*
	S1$\rightarrow$C, S2$\rightarrow$C, S3$\rightarrow$C	*barrier_reply*
6	C$\rightarrow$S3, C$\rightarrow$ S2, C$\rightarrow$S1	*flow_mod*($H_D \rightarrow H_S$)
	C$\rightarrow$S3, C$\rightarrow$ S2, C$\rightarrow$S1	*barrier_request*
	S3$\rightarrow$C, S2$\rightarrow$C, S1$\rightarrow$C	*barrier_reply*
7	*usual MAC forwarding from/to H_S to/from H_D*	

Table 4: OpenFlow messages for a new flow in `ifwd` application

Phase	m_{pi}	m_{po}	m_{fm}	m_{bq}	m_{bp}
1	$(P_{SD}, 2E]$	$[P_{SD}, N]$	-	-	-
2, 3, 6	1	1	-	-	-
4, 7	-	-	P_{SD}	P_{SD}	P_{SD}
Total	$(P_{SD}+3, 2E+3]$	$[P_{SD}+3, N+3]$	$2P_{SD}$	$2P_{SD}$	$2P_{SD}$

the packets between the pair H_S and H_D are routed without the interaction with the controller.

If we consider the list of properties of the `fwd` application, P1 and P2 still hold, whereas P3, P4 and P5 should be modified as below:

- P3': the ARP reply is sent to the source host directly by the controller.
- P4': the first IP packet for each direction is sent from the source to the destination directly by the controller.
- P5': the forwarding rules are installed per source and destination pair at MAC level, and are installed after the first end-to-end IP packet exchange has been successful.

Using the same methodology adopted for `fwd`, we can compute the total number of OF messages for ifwd as shown in Tab. 4 and summarize our findings as follows:

Property 2. *In Intent Based Simple Reactive Forwarding Application (`ifwd`) of ONOS, for each new flow traversing P_{SD} switches, the following relations hold for the number of OF messages:*

$$m_{pi} \in (P_{SD}+3, 2E+3]$$
$$m_{po} \in [P_{SD}+3, N+3]$$
$$m_{fm} = m_{bq} = m_{bp} = 2P_{SD}$$

The number of rules installed per flow in each switch along the routing path is 2.

Table 5: Total number of OpenFlow messages and installed flow rules in the switches along the routing path

Application	Openflow messages		Installed flow rules per flow in each switch
	Lower bound	Upper bound	
`fwd`	$17P_{SD}$	$2E + N + 15P_{SD}$	3
`ifwd`	$8P_{SD} + 6$	$2E + N + 6P_{SD} + 6$	2

4.3. Summary of the results

By combining the results of Properties 1 and 2, in Table 5 we show the number of OF messages, independently from their kind, exchanged in the control plane for the two applications, and the corresponding number of flow rules installed for each switch along the routing path connecting the source to the destination. Note that this evaluation holds for any network topology.

5. OpenFlow control traffic in a POP network

To show the broad applicability of our results, we start by considering an ISP network, composed of a backbone network interconnecting P access networks, denoted as POPs (Points Of Presence). Our aim is to understand the effect on OpenFlow traffic of different sizes of the POPs by which a given population of users is divided. We do not make any assumption about the position of the SDN controller within the POP.

We assume the reference topology in the POP depicted in Fig. 3, in which all the switching devices present in the POP are assumed to be OF switches managed by a single ONOS controller, responsible for the routing internal within the POP, for the case in which both communication endpoints are internal to the POP and for the case in which only one endpoint is internal (in this case, the routing is directed to the POP access router). Coherently with the considered layer-2 applications, the whole network is a single IP network. The POP is composed of B OF access routers (denoted as "AR"), each of them acting as a Broadband Remote Access Server (BRAS) and connected to D DSLAMs (Digital Subscriber Line Access Multiplexers) [22]. The corresponding D ports are configured as host ports. Each access router is connected to D DSLAMs and provides the access to K users, distributed across the DSLAMs. Two intermediate switches (denoted as "IS") are connected to a CDN ingress server through a host port and to two routers acting as border routers (denoted as "BR"). Note that the

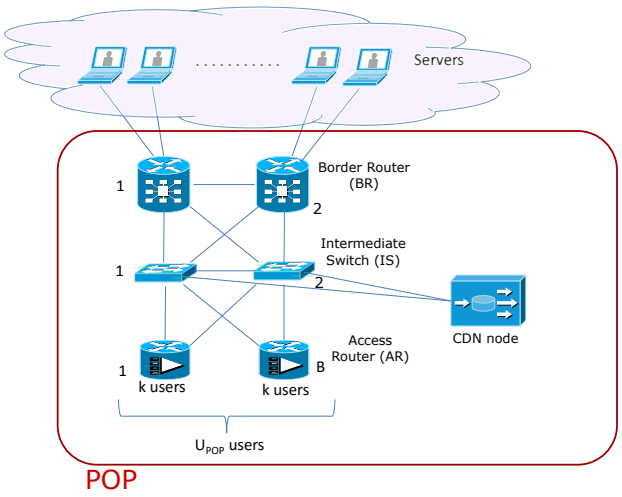

Figure 3: Reference topology for a POP network.

duplication of the IS and BR devices is due to load balancing and reliability reasons.

The overall number of users U_{POP} managed in the POP is $U_{POP} = KB$, whereas the total number of users supported by the ISP is $U_{ISP} = PU_{POP}$ assuming that all the POPs have the same size, for simplicity. Thus, $B = U_{ISP}/(PK)$. With all the above formulas, it is possible to compute the number of internal ports for each switch involved in the topology. Note that this scenario can be easily extended to other realistic topologies.

Coming back to Fig. 3, two OF switches are responsible to provide load balancing and redundant paths between the access routers and two OF border routers connected to the backbone. The ports of the border router towards the backbone network are configured as host ports to keep the controller aware only of the internal POP topology and to route the data across the POP obliviously of the backbone routing policies. In total, the number of OF switches within the POP is: $N = 4 + B = 4 + U_{ISP}/(PK)$ and the number of inter-switch links is $E = 6 + 2B = 6 + 2U_{ISP}/(PK)$.

We only focus on the control traffic generated/received to/from the SDN controller, due to flow setup by POP users, i.e. the ISP customers. We assume that each user establishes one distinct flow at IP level, during some observation window T, and we evaluate the actual number of messages (and the corresponding amount of bytes) observed in the control plane. The actual bandwidth (in byte/s) can be directly computed by dividing the ex-

Table 6: Size of OpenFlow messages

Message	Size
pkt_in	98 bytes + payload
pkt_out	104 bytes + payload
flow_mod	146 bytes
barrier_request	74 bytes
barrier_reply	74 bytes

Table 7: POP scenario setting

Parameter	Symbol	Value
Total users	U_{ISP}	16M
Users per access router	K	32k
DSLAMs per access router	D	32
Number of POPs	P	16-256
Users per POP	U_{POP}	65k-1M

changed bytes by T.

The traffic flows are generated according to two scenarios. In *Local traffic*, all the POP users' traffic is destined towards the internal CDN node. Thus, $P_{SD} = 2$. In *External traffic*, each user connects with a distinct server, located externally to the POP, as shown in Fig. 3. Thus, $P_{SD} = 3$. Since the routing is occurring at layer-2, in this case the IP destination will be one of the BR interfaces (i.e. the POP default gateway).

5.1. Evaluation of OpenFlow control traffic

We investigate the number of OF messages and the required bandwidth for the POP reference topology. We evaluate the amount of OF messages exchanged in each POP for each flow setup, based on the results of Properties 1 and 2. The considered scenario is a nation-wide ISP providing service to $U_{ISP} = 16$ million active users, divided into P homogeneous POPs, with POP configuration parameters as reported in Tab. 7. We investigate the control traffic as a function of the number of available POPs in which the user population is divided. Since the flow is defined at MAC/IP level, the average number of messages per flow does not depend neither on the number of higher level flows (defined at the transport layer), nor on the order of flows arrivals.

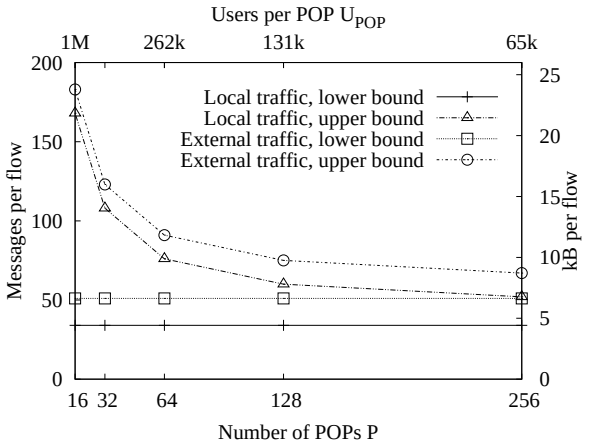

Figure 4: OpenFlow control traffic for the ONOS `fwd` application

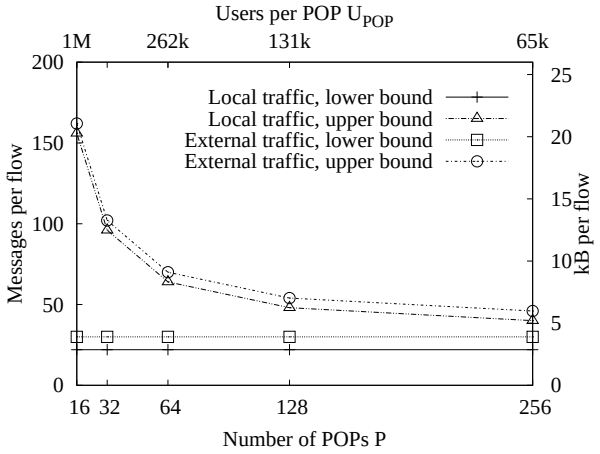

Figure 5: OpenFlow control traffic for the ONOS `ifwd` application

Figs. 4 and 5 show the upper and lower bounds for the number of OF messages as a function of the POP size for the ONOS `fwd` and `ifwd` applications respectively. We report also the actual bandwidth per flow, computed with the size of the OF packets shown in Tab. 6, and neglecting the packet copy carried inside the *pkt_in* and *pkt_out* payload (i.e. ARP or IP packet).

The two applications behave similarly, coherently with the results in Tab. 5, because the bounds for the number of messages in both `fwd` and

`ifwd` differ by a constant factor equal to $9P_{SD} - 6$.

In the case of local traffic, the lower bound on the number of OF messages is independent of the number of hosts and switches, but depends only on the constant $P_{SD} = 2$ between hosts and the CDN node. However, the upper-bound strongly depends on the POP size, and it is proportional to the total number of links E. This bound is mainly due to the worst-case scenario for the flooding of ARP requests and, in relative terms, is only 4 times larger than the lower bound for large POPs, and only 20% larger for small POPs.

In the case of external traffic, $P_{SD} = 3$ and the lower bound is constant, as expected. The upper bound has the same behavior than local traffic. Note that the bandwidth increase in external traffic with respect to local traffic for `fwd` is larger than for `ifwd`, due to the flooding of the ARP requests.

The overall bandwidth varies between 5 and 25 kB per flow, independently of the application and the traffic scenario. These values permit to properly plan the control network (for a given flow arrival rate) and to quantitatively evaluate the conditions under which the traffic overhead due to SDN can be considered negligible with respect to the data carried by the flow. If we consider an average flow size of 10-20 kB for the data as reported in [23, 24], the overall control bandwidth is surprisingly large relatively to the amount of exchanged data and is mainly due to the reactive nature of the two applications. These results suggest that reactive applications in large POP networks should be avoided and provide an upper bound on the maximum amount of control traffic that any real application (which is supposed to act in a hybrid way between reactive and proactive behaviors) would generate.

5.2. Evaluation of installed flow rules

Leveraging the results of Tab. 5, we can evaluate the number of installed flows per switch in the two traffic scenarios (local and external). We assume that a fraction α, with $\alpha \in [0, 1]$, of users are active at the same time in the POP. Thus, the number of flows traversing each access router is αk on average and the ones traversing each intermediate switch will be $\alpha U_{POP}/2$ (assuming a perfect load balancing), in both traffic scenarios. For the border routers, in the case of local traffic no flow will be present, but in the case of external traffic the average number of flows will be $\alpha U_{POP}/2$. Now Table 8 reports the expected number of flow rules installed for each device, for the two reactive applications, and Fig. 6 shows them for the different network devices. Note that the messages for BR are by construction zeros in the case of local traffic. The results have been obtained by setting $\alpha = 0.09$ since it

Table 8: Average number of flow rules installed per device

Device	`fwd`	`ifwd`	Note
Access Router (AR)	$3\alpha k$	$2\alpha k$	-
Intermediate Switch (IS)	$1.5\alpha U_{POP}$	αU_{POP}	-
Border Router (BR)	$1.5\alpha U_{POP}$	αU_{POP}	Only for external traffic

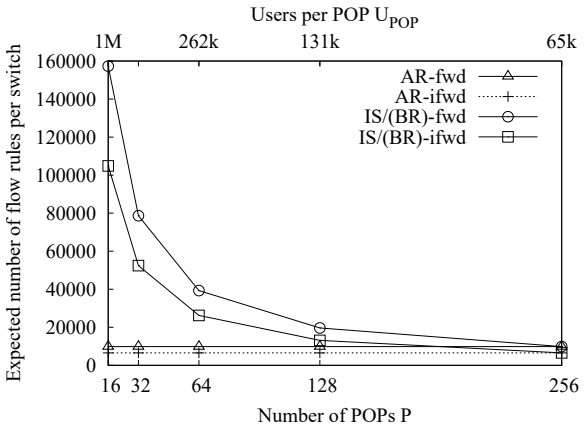

Figure 6: Installed flow rules for each network device assuming 9% of active users in the POP.

has been shown by [25] (and other works referred in [25]) that the fraction of active ADSL users is usually less than 9%, even during peak-hours.

The maximum number of active flow rules that can be stored in a switch depends on the actual implementation of OF switches. If the tables were completely implemented in TCAMs, current switch implementations would accept around 10,000 rules, thus limiting the applicability of reactive applications to small POPs. Instead, if the tables were completely implemented in RAM (e.g., through Patricia tree or hash tables) much larger flow tables could be accepted, at the cost of a performance degradation. Only focusing on a particular implementation and knowing the exact fraction of active users, it would be possible to provide a definitive answer to the maximum scalability of the considered reactive applications.

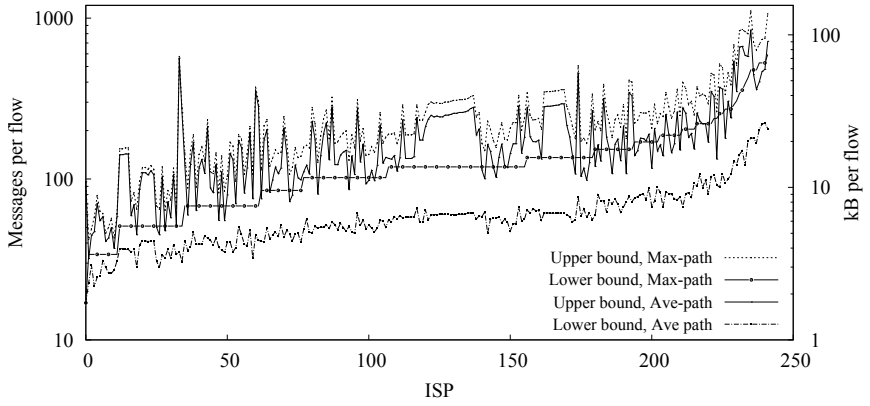

Figure 7: Overall OpenFlow control traffic for ONOS `fwd` application for the maximum length path and for an average length path

6. OpenFlow control traffic in real ISP networks

After evaluating the OF control traffic for the specific topology of a POP network, we extend our analysis to 242 different ISP network topologies, obtained from the Internet Topology Zoo project [26]. We do not compute the installed flow rules per switch, since it is not possible to know how the population, and thus the flows, are distributed across the nodes present in such large networks, some of them covering one or more countries/continents. Instead, we investigate the effect of the topology on the control traffic. We report the lower and upper bound of OF traffic obtained according to Properties 1 and 2, in two distinct cases. In the first case, denoted "Max-path", the distance P_{SD} between the source and the destination is maximum and corresponds to the network diameter. In the second case, denoted "Ave-path", the distance P_{SD} is the average distance in the topology. In the following discussion, when referring to the names of specific ISPs, we will refer to the identifiers used in [27], where the interested reader can view the corresponding topology.

Figs. 7-8 show the control traffic for the `fwd` and `ifwd` applications, respectively, for each ISP network.For the sake of readability, the ISPs have been sorted in increasing values of network diameter and consequently, according to Table 5, in increasing values of lower bound. As a reference to understand the correlation between control traffic and the features of the considered network topologies, Fig. 9 also shows the number of nodes, links, and the diameter of each network for the same sequence of ISPs used in

Figs. 7-8.

We now consider the behavior of `fwd` application in Fig. 7. If we compare the upper bounds, the flows with average length path ("Upper bound, Ave-path") show almost the same number of messages per flow with respect to ones with the longest paths ("Upper bound, Max-path"), since both upper bounds depend mainly on the flooding of the ARP-Request messages across all the nodes of the topology. Indeed, the two bounds show the same behavior as the number of edges and nodes in Fig. 9, independently of the path length and the specific considered flow.

If we now compare the upper bound with the lower bound for the longest path ("Upper bound, Max-path" vs. "Lower bound, Max-path") in Fig. 7, they coincide for linear topologies (e.g. *Sago* ISP in USA) since the routing along the shortest path between the extreme nodes involves all the other nodes in the middle. Instead, the upper bound can be much larger than the lower bound (up to 12×) for symmetric large topologies (e.g. *TATA* ISP in India), where the nodes involved in the flooding are much higher than the ones along the path connecting the farthest nodes. Thus, the topology has a strong impact on the lower bound when considering flows with long paths.

If we now compare the lower bounds obtained for longest paths ("Lower bound, Max-path") and for average paths ("Lower bound, Ave-path") in Fig. 7, they can be identical or very similar whenever the topology is either fully connected (e.g. *GlobalCenter* ISP in USA) or a star (e.g. *Itnet* ISP in Ireland) since the average distance is the same or very close to the diameter of the topology. Instead, when the diameter is much larger than the average distance (e.g. in *ITC-Deltacom* ISP in USA) then the lower bound for Max-path can be also 3.2× the one for Ave-path. As a conclusion, the topology and the length of the path affect strongly the lower bounds on the number of OF messages in the control plane.

Consider now the number of messages for `ifwd` application in Fig. 8. We can observe that the qualitative behavior of all the bounds is identical to the one shown in Fig. 7 for `fwd`, thus the same effects of the topology discussed above for `fwd` hold also in this case. Notably, both lower bounds for Max-path in both applications follow the behavior of the diameter, because they are both proportional to P_{SD}, as shown in Fig. 9. The difference between the two applications is equal to $9P_{SD} - 6$, which grows with the network diameter. Thus, for large networks the `ifwd` application is expected to be more efficient with respect to `fwd` in terms of control traffic.

The actual bandwidth for each flow can vary between 10 and 150 kB for each new flow, depending on the size of the network. As already observed in the previous section, these values are large relatively to the average flow

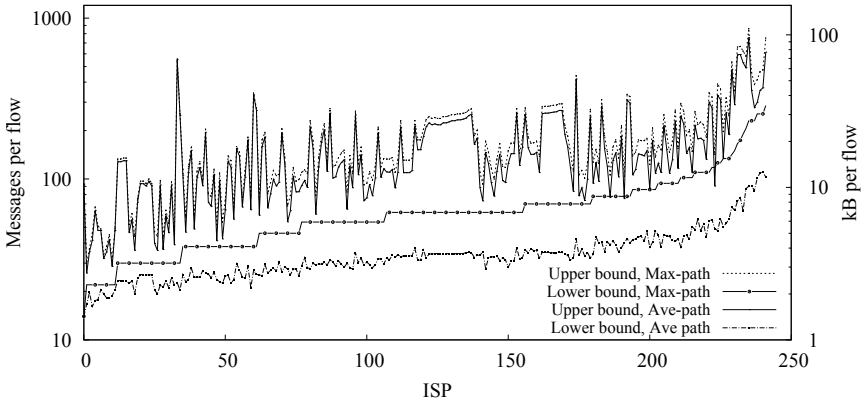

Figure 8: Overall OpenFlow control traffic for ONOS `ifwd` application for the maximum length path and for an average length path

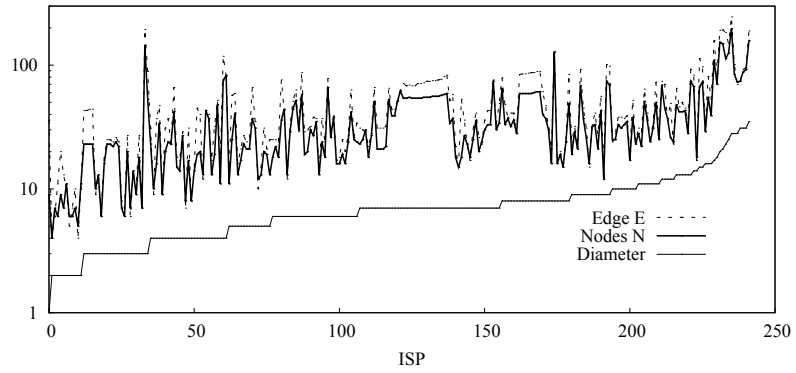

Figure 9: ISP network features

size and advocate a careful adoption of reactive applications in large ISP networks.

7. Conclusions

We investigated the amount of control traffic generated by the default reactive forwarding applications (`fwd` and `ifwd`), available in the ONOS controller. We developed a detailed quantitative model to estimate the messages exchanged on the control plane between the switches and the controller, due

to the flow setup phase, through the analysis of the interaction between the controller and a sample network implemented in mininet. We were also able to estimate the number of flow rules installed in each switch along the routing path from the source and the destination.

Our model was shown to have a wide applicability. Indeed, we applied the model to a realistic network topology representing a POP network and to a large set of ISP network topologies. We evaluated precisely the OpenFlow traffic generated on the control plane and the average number of flow rules installed in each switch, useful to plan the network resources to allocate for the in-band control plane and understand the maximum scalability of the considered reactive applications. When comparing the different ISP topologies, we were also able to highlight the important role of the structure of the topology in the amount of control traffic. As general result, our quantitative analysis confirms the expected poor scalability of fully reactive approaches at layer 2, when the number of flows becomes very large, since the amount of control information can become comparable with the amount of transferred data.

Notably, our models provide a worst-case scenario in terms of control traffic and installed rules per switch, thus it can be used for the correct plan of the transport network carrying the control plane between the switches and the SDN controller/s. If needed, our methodology can be extended to other network applications and SDN controllers, and the results can be tailored to the specific scenario considered by the network designer.

References

[1] ONOS controller.
URL https://www.onosproject.org

[2] Open Networking Lab.
URL http://onlab.us/

[3] A. Bianco, P. Giaccone, A. Mahmood, M. Ullio, V. Vercellone, Evaluating the SDN control traffic in large ISP networks, in: IEEE ICC, London, UK, 2015.

[4] OpenDaylight controller, Available at http://www.opendaylight.org.

[5] Z. Qingyun, C. Ming, D. Ke, X. Bo, On generality of the data plane and scalability of the control plane in software-defined networking, in: IEEE China Communications, 2014.

[6] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, S. Banerjee, Devoflow: Scaling flow management for high-performance networks, SIGCOMM Comput. Commun. Rev. 41 (4) (2011) 254–265.

[7] J. C. Mogul, P. Congdon, Hey, you darned counters! Get off my ASIC!, in: Hot Topics Softw. Defined Netw., 2012.

[8] C. Bi, X. Luo, T. Ye, Y. Jin, On precision and scalability of elephant flow detection in data center with SDN, in: IEEE Globecom, 2013.

[9] M. e. a. Smith, OpFlex control protocol, in: Internet Engineering Task Force, 2014.
URL http:tools.ietf.org/html/draft-smith-opflex-00

[10] G. Bianchi, M. Bonola, A. Capone, C. Cascone, OpenState: Programming platform-independent stateful OpenFlowapplications inside the switch, in: ACM SIGCOMM Comput. Commun. Rev., 2014.

[11] D. Erickson, The Beacon OpenFlow controller, in: SIGCOMM Softw. Defined Netw., 2013.

[12] Z. Cai, A. L. Cox, T. S. E. Ng, Maestro: A system for scalable OpenFlow control, in: Rice Univ., Houston, TX, USA, Tech. Rep., 2011.

[13] S. Hassas Yeganeh, Y. Ganjali, Kandoo: A framework for efficient and scalable offloading of control applications, in: HotSDN, 2012.

[14] A. Tootoonchian, Y. Ganjali, HyperFlow: A distributed control plane for OpenFlow, in: Internet Netw. Manage. Conf. Res. Enterprise Netw., 2010.

[15] A. Kondel, A. Ganpati, Evaluating system performance for handling scalability challenge in SDN, in: Green Computing and Internet of Things (ICGCIoT), 2015.

[16] j. Hu, C. Lin, X. Li, J. Huang, Scalability of control planes for software defined networks: Modeling and evaluation, in: IEEE IWQoS, 2014, pp. 147–152.

[17] M. P. Fernandez, Comparing OpenFlow controller paradigms scalability: reactive and proactive, in: IEEE International Conference on Advanced Information Networking and Applications (AINA), 2013, pp. 1009–1016.

[18] OpenFlow switch specification 1.4.0, Available at https://www.opennetworking.org.

[19] A. Muqaddas, A. Bianco, P. Giaccone, G. Maier, Inter-controller traffic in ONOS clusters for SDN networks, in: IEEE ICC, Kuala Lumpur, Malaysia, 2016.
URL http://www.tlc-networks.polito.it/public/faculty/paolo-giaccone/publications

[20] Mininet network emulator, Available at http://mininet.org.

[21] LLDP link layer discovery protocol, IEEE standard 802.1AB.

[22] White paper: Understanding DSLAM and BRAS access devices, Available at http://cp.literature.agilent.com/litweb/pdf/5989-4766EN.pdf.

[23] A. Bianco, V. Krishnamoorthi, N. Li, L. Giraudo, OpenFlow driven Ethernet traffic analysis, in: IEEE ICC, 2014, pp. 1765–1775.

[24] M.-S. Kim, Y. J. Won, H.-J. Lee, J. W. Hong, R. Boutaba, Flow-based characteristic analysis of Internet application traffic, in: E2EMON, 2004, pp. 62–67.

[25] E. Goma, M. Canini, A. Lopez Toledo, N. Laoutaris, D. Kostić, P. Rodriguez, R. Stanojević, P. Yagüe Valentin, Insomnia in the access: Or how to curb access network related energy consumption, SIGCOMM Comput. Commun. Rev. 41 (4) (2011) 338–349.

[26] S. Knight, H. Nguyen, N. Falkner, R. Bowden, M. Roughan, The Internet topology zoo, IEEE JSAC 29 (9) (2011) 1765–1775.

[27] The Internet Topology Zoo.
URL http://www.topology-zoo.org/dataset.html