Inter-controller Traffic to Support Consistency in ONOS Clusters

Abubakar Siddique Muqaddas[†], Paolo Giaccone[†], Andrea Bianco[†], Guido Maier[‡]
[†] Dip. di Elettronica e Telecomunicazioni, Politecnico di Torino, Italy
[‡]Dip. di Elettronica, Informazione e Bioingegneria (DEIB), Politecnico di Milano, Italy
E-mails:{abubakar.muqaddas,paolo.giaccone,andrea.bianco}@polito.it, guido.maier@polimi.it

***Abstract*—In distributed SDN architectures, the network is controlled by a cluster of multiple controllers. This distributed approach permits to meet the scalability and reliability requirements of large operational networks. Despite that, a logical centralized view of the network state should be guaranteed, enabling the simple development of network applications. Achieving a consistent network state requires a consensus protocol, which generates control traffic among the controllers whose timely delivery is crucial for network performance.**

We focus on the state-of-art ONOS controller, designed to scale to large networks, based on a cluster of self-coordinating controllers. In particular, we study the inter-controller control traffic due to the adopted consistency protocols. Based on real traffic measurements and the analysis of the adopted consistency protocols, we develop some empirical models to quantify the traffic exchanged among the controllers, depending on the considered shared data structures, the current network state (e.g. topology) and the occurring network events (e.g. flow or host addition). Our models provide a formal tool to be integrated into the design and dimension the control network interconnecting the controllers. Our results are of paramount importance for the proper design of large SDN networks, in which the control plane is implemented in-band and cannot exploit dedicated network resources.

***Index Terms*—Software Defined Networking, ONOS controller, consistency protocols, inter-controller traffic measurements.**

I. Introduction

A naïve centralized approach for SDN is based on a single controller managing all network switches. Even if this simplifies the network management and the development of network applications, it poses severe limitations to network scalability and reliability. Indeed, a single centralized controller is a single point of failure. Moreover, a single controller may not be able to handle a large number of switches, because communication load and processing overhead for the controller increases with the number of switches. Finally, in very large networks (as in WANs), switches can be physically very far from the controller, and due to the propagation delays, flow modifications in switches can experience large latency.

Distributed SDN controllers face all the above impairments. Multiple instances of the controller manage the whole network, which is divided into different domains, each of them under the control of one controller instance. Distribution of the controller functions over multiple physical servers improves the robustness of the control plane, by providing backup control resources for each network node. Furthermore, large networks can be handled, because the switch control is distributed among the controllers and the processing load can be balanced. Finally, being the control servers also geographically distributed across the network area, they can reduce the switch-to-controller delay, thus improving the controller reactivity as perceived by the network nodes.

However, a logical centralized view of the network state must be guaranteed also with distributed controllers, to ease the development of advanced network applications. This transparent behavior for the network operator/programmer comes at the cost of keeping all the shared data structures synchronized among the controllers by means of some consensus protocol. For example, the same network topology must be known at each controller to take correct routing decisions. However, since each controller is responsible for a subset of switches, it is of paramount importance to distribute any data plane related event in a timely fashion to keep the same state among the controllers and avoid possible misbehaviors (e.g. routing loops, firewall leaks), as highlighted in [1].

In large SDN networks (as SDWANs), the control plane distributed among the controllers is implemented in-band, without the possibility of relying on a dedicated out-of-band high-performance network as the data center scenarios [2]. This poses technical challenges in designing the control network, which does not only interconnect the switches to controllers, but also supports the communication between controllers. Due to the complexity of the adopted consensus protocols, the reactivity of the controllers as perceived by the switches depends also on the bandwidth and delays experienced in the inter-controller communication. This fact advocates a proper design and plan of the network supporting the control traffic, in particular guaranteeing adequate bandwidth for the inter-controller traffic.

We focus on the control traffic exchanged among the controllers, which is often neglected in the literature. We consider the state-of-art ONOS controller [3], which is an open-source project developed by ON.Lab [4] and supported by a large community of network operators and vendors. Differently from the initial versions of well-known OpenDaylight project [5], ONOS has been designed specifically to cope with reliability and scalability issues arising in large ISP/WAN networks. It natively supports a distributed version of the controller, running on a cluster of servers.

A. *Our contributions*

We run an experimental testbed which includes a cluster of ONOS controllers and evaluate the amount of traffic exchanged among the controllers. Since the traffic depends on the specific updates committed on the shared data structure, we address our problem by analyzing the impact of each update for all the shared data structures (i.e. topology, flow and host stores) that manage the network state. Thanks to tailored experiments, we evaluate the exact amount of traffic in function of the specific event or change of state in the network and thus we develop some empirical models of the ONOS inter-controller traffic. Our results are general in terms of network topology and partition of the network into different controller domains. The adopted methodology is also general and provides experimental guidelines to extend our results to an arbitrary number of SDN controllers. A preliminary version of our work appeared in [6].

B. Organization of the paper

Sec. II introduces the general architecture of distributed SDN controllers and describes the two main consistency models adopted to synchronize the data structures. We concentrate on the specific distributed architecture of ONOS and describe the two main protocols to achieve the consensus on the data structures. We present in Sec. III the methodology we adopt to quantify the impact of network related events on the inter-controller traffic for a general distributed cluster of SDN controllers. The subsequent three sections are devoted to investigate the impact of updates occurring in different shared data structures. Indeed, in Sec. IV we concentrate on the store describing the network topology. The experimental data allows us to devise a set of empirical models to estimate the bandwidth for any network topology and any domain partition (Properties 1 and 2). In Sec. V we concentrate on the store describing the flow tables and investigate the impact of flow modifications in the switches. Finally, in Sec. VI we concentrate on the store recording the hosts attached to the network switches. In Sec. VII we discuss some related work, and finally in Sec. VIII we draw our conclusions.

II. Distributed SDN controllers

Fig. 1 shows a distributed SDN architecture with two controllers managing a single network divided in two domains. The traffic is exchanged directly among the controllers through the *east-west interface* [7], which is in addition to the *north-bound interface* (providing the APIs to interact with the controller at application level) and the *south-bound interface* (running a standard control protocol to manage the switching devices, as OpenFlow), both available in any SDN controller.

To understand the role of the traffic exchanged by the controllers, we start by describing an important result in the theory of distributed systems.

A. CAP theorem

Consistency of shared data in distributed systems is a well known and deeply investigated property. This property is achieved with quite complex protocols and algorithms [8]. The consistency dilemma is explained thoroughly using the famous CAP theorem [9] which states the impossibility of enjoying

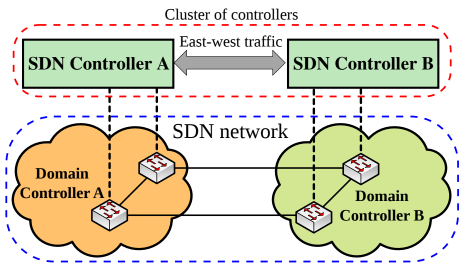


Fig. 1: Distributed SDN architecture with a single cluster of two controllers.

the following three properties at the same time: Consistency, i.e. all the data reads access the latest written version of the data; Availability, i.e. all data is accessible and can be updated; Partition, i.e. the system is tolerant to node partitions.

Even if the proof of the CAP theorem is complex, a convincing scenario to understand this property is a storage system with the data replicated locally in two servers connected through a communication link. If availability and consistency are required at the same time (CA case), i.e. each server should be able to update the local data and access the most recent version of it, network partitions are not allowed, since the two servers must always be able to communicate an update to the other. Similarly, if availability and tolerance to partitions is required (AP case), i.e. each server should be able to update the local copy of the data, then consistency cannot be guaranteed anymore when partitions occur. Finally, if consistency and tolerance to partitions is required (CP case), i.e. the servers must access the most recent version of the data even in the case of partitions, availability cannot be guaranteed since each server cannot update the local copy in case of partitions. Depending on the pair of required guarantees (CA, AP or CP) in a distributed system, a large number of consistency protocols and algorithms have been devised and implemented so far.

B. Consistency in distributed SDN controllers

In a distributed SDN scenario, consistency means that all the controllers view the same network state, e.g. have the same local copy of the network topology and of the node/link availability state in their shared data structures. Any change of state occurring on each controller (due to, for example, new flow setups, link failures) must be promptly propagated to the other controllers according to one consistency protocol. If the controllers have an inconsistent view, the network policies may not run correctly and this can lead to potential network misbehaviors (as routing loops, packet drops, firewall leaks). For example, consider Fig. 1: if the communication between the east-west interfaces is not available, the control network is partitioned. In this case if there is a change in topology in controller B's domain, then it will not be propagated to controller A. Consequently controller A could take routing decisions based on an older view of the network topology in controller B's domain that could lead to unexpected behaviors.

In the theory of distributed systems, many consistency models have been defined. We concentrate here on just two of them, which have a direct application in SDN networks.

1) Eventual consistency model: This model provides a weak form of consistency, in sense that data modifications on a certain controller will be *eventually* propagated on all the other controllers. This implies that, for some time, some controllers may read values different from the actual updated ones; but after some time, all the controllers will have the updated values, given that they are able to communicate. This model is typically employed in distributed systems requiring high availability. The anti-entropy protocol, implemented in ONOS and described in Sec. II-C, supports this consistency model.

2) Strong consistency model: This model ensures that each controller always reads the most updated version of the data. If certain data are not yet updated to all (or most of) the controllers, then they are not allowed to be read, thereby favoring consistency instead of availability. The RAFT consensus protocol, implemented in ONOS and described in Sec. II-C, supports this consistency model.

The controllers exchange some control traffic, denoted as *inter-controller traffic*, through their east-west interfaces, to synchronize their shared data structures. The adopted consistency model heavily affects the inter-controller traffic, whose evaluation and modeling is the main contribution of our work.

C. Distributed ONOS

We now focus on the specific distributed architecture of ONOS controller, which allows to achieve a large scalability and availability, thanks to a distributed cluster of controllers. Each controller in the cluster is responsible of managing the switches under its domain, and updating their state on the distributed data stores. Each switch can connect to multiple ONOS controllers for reliability, but only one will be its *master* with full control on it in terms of read/write capabilities on the switch forwarding tables. The other controllers are denoted as *slaves* and one of them takes the control of a switch whenever the master controller fails. Anytime a cluster of controllers is set up, each controller interacts with all the other controllers, thus the controllers are always logically connected in a full mesh according to a peer-to-peer approach, using a specific TCP port (9876) for their interaction. The controllers send and accept keep-alive messages to/from other controllers to monitor the other cluster members.

Two consistency protocols are implemented in the previous and current (Ibis - Dec. 2016) versions of ONOS to manage the distributed stores, each protocol tailored to guarantee a specific level of consistency.

1) Anti-Entropy Protocol: It is based on a simple gossip algorithm in which each controller chooses at random another controller in the cluster every 5 seconds and then sends a message, containing the timestamp of each entry, to compare the actual content of its store with the other one. After the synchronization messages are exchanged and the stores are updated based on the timestamp of each entry (i.e. more recent updates supersede the older ones), the two controllers become mutually consistent. This ensures that all the controllers achieve consensus according to an eventually consistent model. However, in parallel with the above scheme, whenever an update occurs in the store managed by a controller, this is immediately broadcasted to all the other controllers in the cluster.

2) RAFT Protocol: It is a recently proposed scheme [10] which provides strong consistency in ONOS. A RAFT implementation requires a cluster of nodes (i.e. controllers in our scenario) each having a database termed as the "log" which is replicated in all the nodes: each update is appended to this shared data structure. The consistency is coordinated by a leader node in the cluster, which is responsible for receiving update requests from all the other nodes and then relaying log updates to the other nodes. Once the majority of the followers have acknowledged the update, this is actually committed to the log. In the case of network partitions, only the side with the majority of the nodes is able to update the log, thus avoiding contemporary and conflicting updates in two different network partitions. All the updates on the distributed stores are tracked using logical timestamps, which allow to reconcile conflicts based on the most recent updates.

In ONOS, multiple instances of RAFT protocol run simultaneously. The data structures in the distributed stores are partitioned into shards, where each shard is managed by a different RAFT instance. Partitioning is aimed at improving scalability. The total number of partitions is $N+1$ where N is the number of controllers in an ONOS cluster. The partitions are termed as $p_0, p_1, \dots p_N$. Partition p_0 encompasses all the controllers in the cluster and is just for temporary storage, which is reset if the controller shuts down. For durable storage, the data is partitioned into N shards. The number of controllers that participate in each partition is $\min(3, N)$, i.e. each shard is shared among not more than 3 controllers. The partition p holding the value corresponding to a given key k within a data structure is chosen with a simple hash map $h(\cdot)$ as follows:

$$p = [h(k) \mod N] + 1 \tag{1}$$

Stores are the actual distributed data structures in ONOS. Each store is based on either the Anti-Entropy protocol, RAFT protocol or both. In particular, the main ONOS stores are:

- *Mastership store*, which keeps the mapping between each switch to its master. It is managed by RAFT protocol.
- *Network topology store*, which describes the network topology in terms of links and switches; consistency is achieved using the anti-entropy protocol.
- *Flow store*, which is responsible for backing flows of each switch from the master controller to the slave controller on detecting a change in the flow table. The details of the adopted consistency model are discussed in Sec. V.
- *Host store*, which maintains the list of the network hosts. It is managed by RAFT protocol.
- *Application store*, which manages the inventory of applications, and adopts the anti-entropy protocol.
- *Intent store*, which manages the inventory of intents using the anti-entropy protocol. Intents are part of the ONOS Intent framework used by applications to define which policy is operating on the network, without the details of how the data plane must be actually programmed.

- *Component configuration store*, which stores system-wide configurations for various software components in ONOS. It adopts the anti-entropy protocol.
- *Network configuration store*, which is used to store network configurations inserted into ONOS via the north-bound (e.g. REST API) or the south-bound API (e.g. OpenFlow). It adopts RAFT consensus algorithm.
- *Security mode store*, which manages permissions granted to applications using RAFT protocol. Instead, security violations are managed using the anti-entropy protocol.

Of all these distributed stores, the ones which are related to the data plane behavior are network topology store, flow store and host store, and each of them will be investigated in a dedicated section (Secs. IV-VI). The other distributed stores are specific of each application and are neglected as part of our experimental work in order to keep our results general.

III. Methodology for inter-controller traffic analysis

For prototyping and testing, a test setup based on a standalone Ubuntu 14.10 server machine is used. A cluster of ONOS version 1.4.1 controllers runs in a set of Linux containers (LXC) [11] hosted on the server machine as shown in Fig. 2. LXC was chosen since containers are lighter on the CPU than virtual machines and do not show undesired background traffic, thus allowing to easily identify all the traffic generated by each instance of the controller. Notably, the adopted choice of the operating system (OS) virtualization is transparent for the controller instances, and thus our results hold for any other virtualization system compatible with the considered ONOS distribution.

We adopt Mininet-2.2.1 to emulate a network topology consisting of OpenFlow compliant software switches. Each switch is associated to one master controller and all the other slave controllers.

As shown in Fig. 2, three logical network topologies are created using virtual bridges available in Linux: 1) *North-bound Virtual Bridge*, connecting the controllers to our test application through the north-bound interfaces of the controllers; 2) *South-bound Virtual Bridge*, connecting the network emulated with Mininet to the controllers; 3) *East-West Virtual Bridge*, connecting directly the controllers to each other. The use of separate virtual bridges simplifies traffic capture and management. We run Wireshark as a sniffer to capture the inter-controller traffic between any pair of controllers by capturing all the TCP traffic on the interface of a controller towards the other controller(s). ONOS uses port 9876 for the inter-controller communications, thus it is simple to identify such traffic. The total inter-controller traffic is sampled every $T_s = 0.1$ s to compute the consumed bandwidth. The bandwidth samples are averaged through a sliding window of $T_w = 10$ s.

In each experiment, we start the controllers and then we wait until the initial transient phase ends. Then, appropriate events are generated either on the south-bound interface through the terminal commands available in Mininet, or on the north-bound interface using our test application, which leverages

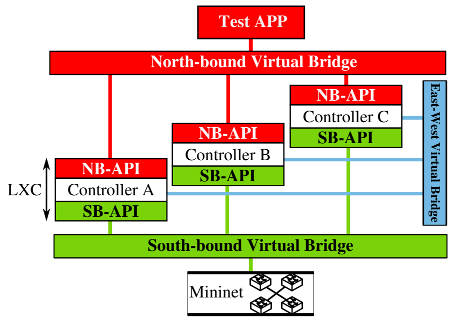


Fig. 2: Single-host testbed to investigate inter-controller traffic for distributed SDN controllers.

the APIs exposed by the controller. To repeat an experiment, we restart by rebooting the LXC container for each SDN controller, to avoid residual data due to previous experiments. The reboot procedure is necessary since affecting the "tombstone" inter-controller traffic, as explained further in Sec. IV.

A. Implementation approaches for consistency models

The specific implementation of the shared data structures across the controllers and the adopted consistency model have significant impact on the inter-controller traffic. We categorize the inter-controller traffic as a combination of the following types of updates:

- *incremental updates, or full updates*: this feature describes the actual information that is exchanged among controllers. In the case of incremental updates, only the differences with respect to the previous updates are exchanged. Since incremental updates must rely on a coherent update of the previous states, the approach is typically employed by a strong consistency model. Instead, in the case of full updates, the whole data structure is exchanged. Full updates are typically exchanged for eventually consistent data structures, due to the unreliable state coherence among data structures.
- *periodic updates, or event-driven updates*: this feature describes when the updates are issued. Periodic updates are generated periodically over the time, whereas event-driven updates are triggered by specific change of states or events.

All the four combinations of the two above features are possible in practice, as shown later in Secs. IV, V and VI.

The overall inter-controller traffic is due to the superposition of the synchronization of different data structures, each of them with a specific feature. Thus, to understand the traffic due to a specific data structure, we specifically modify the data on just a single data structure, generating carefully crafted events in the test application or in the Mininet topology.

We measure the traffic due to each update event in terms of amount of data or bandwidth. For the first case, we measure the additional traffic generated during the transient phase. For the second case, we just evaluate the derivative of the cumulative amount of exchanged traffic. Notably, the transient phase is identified as included between two periods of steady-state bandwidth values, as shown in Fig. 3. Interestingly, as shown

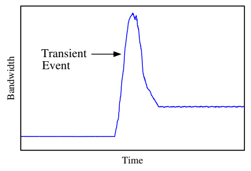


Fig. 3: Transient phase detection

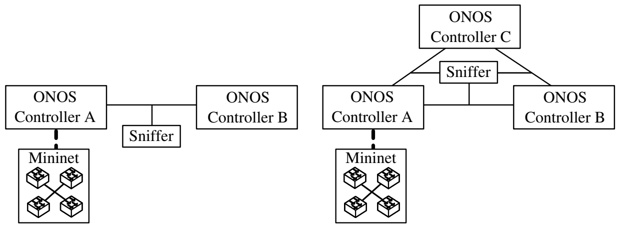


Fig. 4: Experimental testbed for 2 and 3 ONOS controllers

in Sec. IV, the bandwidth after the transient phase may be different from the initial one.

IV. Distributed Topology Store

We evaluate the traffic exchanged among the ONOS controllers due to the network topology store. Our results are general since they apply to *any network topology* and *any partition* of the network into controller domains. To highlight the role of the topology, we adopt a time-variant topology in which the number of active switches and active edges changes with the time. By measuring the variation of the inter-controller traffic, we are able to understand the detailed effect of modifications in the topology store.

We consider two main scenarios, both shown in Fig. 4: the first one with 2 controllers and the second one with 3 controllers belonging to the same cluster. We denote with A, B and C the instances of the controller, running within the same controller cluster. Let S be the total number of switches in the network topology and L be the corresponding number of (bidirectional) links. We adopt some simple test network topologies to highlight the individual contribution of each network element (switch or link) and thus obtain general results holding for *any* topology. In the *isolated topology* we have S isolated switches without links ($L = 0$). In the *linear topology*, S switches are connected linearly ($L = S - 1$). In the *star topology*, $S - 1$ switches are connected to the same central switch ($L = S - 1$). We repeat all the experiments 20 times and compute the 98% confidence intervals.

The inter-controller traffic generated among the controllers due to the topology store is *periodic with full updates*. This is due to the adoption of anti-entropy protocol for maintaining consistency in network topology store. Another contribution is *periodic with incremental updates* and it is due to the LLDP packets sent for the topology discovery received on the southbound interface, as later discussed in Sec. IV-B.

A. Transient behavior in the linear topology with 2 controllers

In Fig. 5 we show the communication bandwidth from controller A to controller B for a linear topology with $S = 10$, with all the switches managed by controller A. We start with no topology ($S = 0$, $L = 0$) added to controller A; at time 120 s the linear topology is added ($S = 10$, $L = 9$); at time 240 s the linear topology is removed. At the beginning of the experiment we observe an initial communication of 63 kbps (denoted as *Zero Bandwidth 1*). When the linear topology is added, after a short transient phase, the traffic reaches 88 kbps. When the network is removed, the bandwidth reaches 78 kbps (denoted as *Zero Bandwidth 2*). This value is different from the initial one at the beginning of the experiment, and it is due to the exchanged "tombstone" traffic. Tombstone traffic is due the anti-entropy protocol and refer to devices, links and hosts which have been removed from the active topology. The reason for it is to react faster to network partitions. Indeed, in the case of temporary network partitions, keeping tombstones minimizes the variation in the internal topology store, and thus the allocation/deallocation of memory for the internal data structures. Notably, after each experiment, the LXC container is rebooted so that no tombstone traffic persists in the observed traffic.

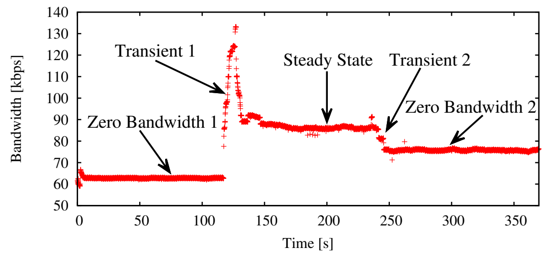


Fig. 5: Traffic from controller A to B by adding and removing a linear topology $S = 10$, $L = 9$

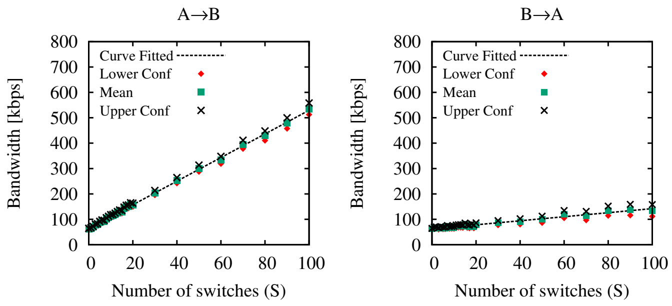


Fig. 6: Isolated topology associated to controller A as master in the scenario with 2 controllers

B. Scenario with 2 controllers

We investigate the traffic exchanged by controllers A and B in steady state for different sizes of the topology, in which all the switches are under A's control. Fig. 6 shows the bandwidth from A to B and vice versa, when an isolated topology is added to controller A. We show also the confidence intervals and one linear curve fitting the experimental measurements. Similarly, Fig. 7 shows the bandwidth when a linear topology is added to controller A. Both graphs show that the bandwidth is increasing linearly in both communication directions. This is coherent with the linear growth of the internal data structures,

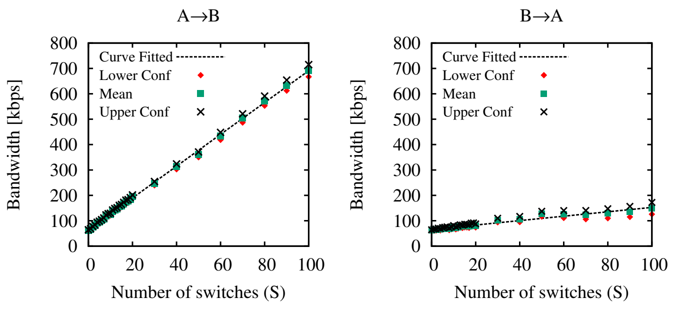


Fig. 7: Linear topology associated to controller A as master in the scenario with 2 controllers

based on hash tables. Moreover, the bandwidth for A → B is larger than B → A. If we consider that the topology store is distributed with the anti-entropy protocol, we should expect a symmetric behavior. Instead at controller A, the topology is periodically refreshed (even if not changing) through the LLDP packets received on the south-bound interface for topology discovery. This causes an update on the topology store, which generates additional traffic from A to B and causes the asymmetry. In addition, port and flow statistics gathered periodically by controller A are also sent to B.

Due to the internal data structures, whose memory occupancy grows linearly with the number of elements (nodes and links), we can assume that the exchanged traffic B in each direction is proportional to the size of the topology store:

$$B = S \cdot b^s + L \cdot b^l + b^0 \tag{2}$$

where we used the notation in Table I. By applying (2) to the linear topology ($L = S - 1$) and to the isolated topology ($L = 0$) considered in our experiments, we can write the following system of equations, assuming that A is master controller of all the switches in the network:

$$\begin{cases} B^L_{\text{A}\to\text{B}} = S \cdot b^s_{\text{A}\to\text{B}} + (S-1) \cdot b^l_{\text{A}\to\text{B}} + b^0 \\ B^L_{\text{B}\to\text{A}} = S \cdot b^s_{\text{B}\to\text{A}} + (S-1) \cdot b^l_{\text{B}\to\text{A}} + b^0 \\ B^I_{\text{A}\to\text{B}} = S \cdot b^s_{\text{A}\to\text{B}} + b^0 \\ B^I_{\text{B}\to\text{A}} = S \cdot b^s_{\text{B}\to\text{A}} + b^0 \end{cases} \tag{3}$$

This system can be solved by measuring $B^L_{x\to y}$, $B^I_{x\to y}$, for any $x, y \in \{\text{A}, \text{B}\}$ ($x \neq y$) and b^0 and thus estimating the remaining unknown values of per-link and per-switch bandwidth.

Based on our measurements, we observe always a constant value of zero bandwidth equal to $b^0 = 62.46$ kbps (obtained with 3.6% accuracy at 96% confidence level) for the both directions, given our measurements. Thus, solving the system in (3), we obtain experimentally:

$$b^l_{\text{A}\to\text{B}} = 1.63 \text{ kbps} \qquad b^l_{\text{B}\to\text{A}} = 0.11 \text{ kbps} \tag{4}$$

$$b^s_{\text{A}\to\text{B}} = 4.65 \text{ kbps} \qquad b^s_{\text{B}\to\text{A}} = 0.80 \text{ kbps} \tag{5}$$

So far all the network switches have been associated to the same controller. In order to extend our model to any topology, arbitrarily partitioned among two controller domains, we need to evaluate the effect of inter-domain links, i.e. connecting one switch in one domain with another in the other domain.

TABLE I: Notation for traffic in the scenario with 2 controllers x and y

Symbol	Meaning
B	generic unidirectional bandwidth
b^0	generic zero bandwidth
b^s	average unidirectional bandwidth per switch
b^l	average unidirectional bandwidth per intra-domain link
b^d	average unidirectional bandwidth per inter-domain link, (shared also by target controller for 3 controller scenario)
b^e	average unidirectional bandwidth per inter-domain link, (external to target controller for 3 controller scenario)
$B^I_{x\to y}$	bandwidth from x to y in isolated topology
$B^L_{x\to y}$	bandwidth from x to y in linear topology
$b^s_{x\to y}$	average bandwidth from x to y per switch
$b^l_{x\to y}$	average bandwidth from x to y per intra-domain link

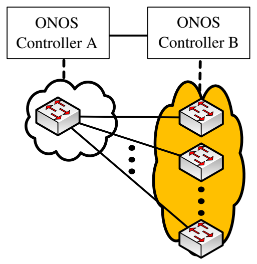


Fig. 8: Scenario with a star topology

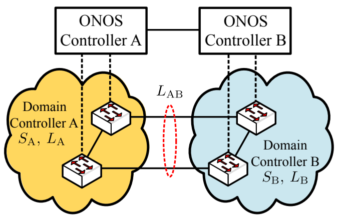

ONOS Controller A
ONOS Controller B
L_{AB}
Domain Controller A $S_{\text{A}}, L_{\text{A}}$
Domain Controller B $S_{\text{B}}, L_{\text{B}}$

Fig. 9: Notation depicting the network topology in the scenario with 2 controllers

We consider the star topology in Fig. 8, in which we vary the number of switches and consequently the number of inter-domain links. Now the observed bandwidth in one direction is obtained by summing the following contributions: B^I for 1 switch to model the switch in controller A's domain; B^I for $S-1$ switches to model the $S-1$ switches in B's domain; $S-1$ times the average bandwidth per inter-domain link b^d. Using the same methodology before and exploiting the estimated values obtained so far, we estimate that the average bandwidth per inter-domain link is

$$b^d = 0.63 \text{ kbps} \tag{6}$$

By combining the results so far and the estimated bandwidths in (4), (5) and (6), we can claim the following, by referring to the notation in Table II:

Property 1: In an arbitrary network managed by a ONOS cluster of 2 controllers A and B, the traffic exchanged from controller x to controller y is:

$$B_{x\to y} = 62.46 + 4.65 \cdot S_x + 1.63 \cdot L_x + 0.80 \cdot S_y + 0.11 \cdot L_y + 0.63 \cdot L_{xy} \quad [\text{kbps}] \tag{7}$$

for $x = \text{A}$ and $y = \text{B}$, or for $x = \text{B}$ and $y = \text{A}$.

TABLE II: Notation describing the network topology in the scenario with 2 controllers. Let x be a controller, with $x \in \{\mathrm{A}, \mathrm{B}\}$.

Symbol	Meaning
S_x	number of switches in controller x's domain
S	total number of switches in the network
L_x	number of intra-domain links in controller x's domain
L_{AB}	number of inter-domain links
L	total number of links in the network

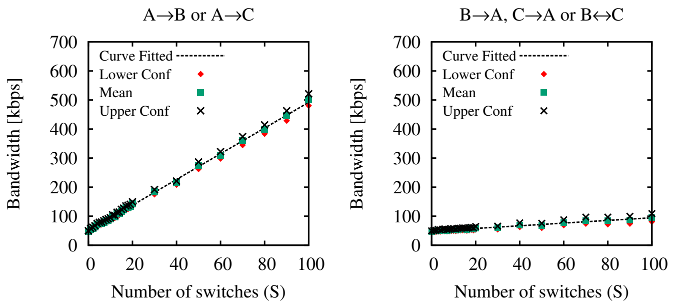


Fig. 10: Isolated topology added to controller A in the scenario with 3 controllers

Let $B_{TOT} = B_{\mathrm{A}\to\mathrm{B}} + B_{\mathrm{B}\to\mathrm{A}}$ be the total exchanged traffic among the two controllers. Referring to the notation in Table II and depicted in Fig. 9, we claim:

Corollary 1: In an arbitrary network managed by a ONOS cluster of 2 controllers A and B, it holds

$$B_{TOT} = 124.92 + 5.45 \cdot S + 1.74 \cdot L - 0.48 \cdot L_{AB} \quad \text{[kbps]}$$

Thus, the total inter-controller traffic grows linearly with respect to the number of switches and links in the topology.

We validated the above formulas by considering multiple scenarios, including full mesh topologies, ring topologies, irregular topologies. All the experimental results have been always compatible with the model prediction of Property 1 within 98% confidence interval. The details are omitted for the sake of space.

C. Scenario with 3 controllers

The methodology adopted in the previous scenario is now extended to the 3 controllers scenario using the configuration shown in Fig. 4. We start by adding the topology to controller A. For symmetry, the bandwidth $B_{\mathrm{A}\to\mathrm{B}} = B_{\mathrm{A}\to\mathrm{C}}$; as no topology is added to controllers B and C, similarly $B_{\mathrm{B}\to\mathrm{A}} = B_{\mathrm{B}\to\mathrm{C}} = B_{\mathrm{C}\to\mathrm{A}} = B_{\mathrm{C}\to\mathrm{B}}$. Fig. 10 shows the bandwidth from A to B and vice versa, when an isolated topology is added to controller A.

Fig. 11 shows the bandwidth when a linear topology is added. As compared to Fig. 6 and 7, the bandwidth values in the 3-controller case are lower than the 2-controller case. This is due to the anti-entropy protocol: periodically, each controller randomly selects another controller to synchronize the network topology. Say the synchronization rate for each controller is λ. Thus the average contribution of this process on each link is $3\lambda/6 = \lambda/2$, since 6 possible links are present with 3 controllers. Instead, in the case of 2 controllers, the average contribution was $2\lambda/2 = \lambda$. Thus, a reduction

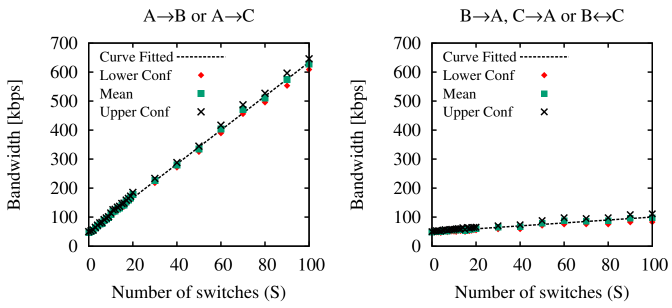


Fig. 11: Linear topology added to controller A in the scenario with 3 controllers

of a factor 2 in the bandwidth due to the anti-entropy is expected. In the figure, the reduction is much lower, due to the lower impact of this protocol with respect to the topology updates sent by the master controller of a switch and caused by LLDP packets, as explained in Sec. IV-B. Globally, the bandwidth exchanged in each direction is still proportional to the size of the topology store. Following the assumption while deriving (2) and considering that the topology is only added to controller A, the following system of equations can be written with the notation in Table I:

$$\begin{cases} B^L_{\mathrm{A}\to\mathrm{B}} = S \cdot b^s_{\mathrm{A}\to\mathrm{B}} + (S-1) \cdot b^l_{\mathrm{A}\to\mathrm{B}} + b^0 \\ B^L_{\mathrm{B}\to\mathrm{A}} = S \cdot b^s_{\mathrm{B}\to\mathrm{A}} + (S-1) \cdot b^l_{\mathrm{B}\to\mathrm{A}} + b^0 \\ B^I_{\mathrm{A}\to\mathrm{B}} = S \cdot b^s_{\mathrm{A}\to\mathrm{B}} + b^0 \\ B^I_{\mathrm{B}\to\mathrm{A}} = S \cdot b^s_{\mathrm{B}\to\mathrm{A}} + b^0 \end{cases}$$

which can be solved numerically. The zero bandwidth between any two controllers here is $b^0 = 47.81$ kbps (obtained with 4.15% accuracy at 98% confidence level).

Thus, we obtain:

$$b^l_{\mathrm{A}\to\mathrm{B}} = b^l_{\mathrm{A}\to\mathrm{C}} = 1.43 \text{ kbps} \tag{8}$$

$$b^l_{\mathrm{B}\to\mathrm{A}} = b^l_{\mathrm{B}\to\mathrm{C}} = b^l_{\mathrm{C}\to\mathrm{A}} = b^l_{\mathrm{C}\to\mathrm{B}} = 0.06 \text{ kbps} \tag{9}$$

$$b^s_{\mathrm{A}\to\mathrm{B}} = b^s_{\mathrm{A}\to\mathrm{C}} = 4.43 \text{ kbps} \tag{10}$$

$$b^s_{\mathrm{B}\to\mathrm{A}} = b^s_{\mathrm{B}\to\mathrm{C}} = b^s_{\mathrm{C}\to\mathrm{A}} = b^s_{\mathrm{C}\to\mathrm{B}} = 0.46 \text{ kbps} \tag{11}$$

To extend our empirical model to any topology, arbitrarily partitioned among the two controller domains, the star topology in Fig. 8 is considered albeit with 3 controllers and no switch added to controller C, in which we vary the number of switches and thus the inter-domain links. In this scenario, the bandwidth originating from each controller to the other two controllers is different, since different number of switches are added to each controller. Hence, here $B_{\mathrm{B}\to\mathrm{A}} = B_{\mathrm{B}\to\mathrm{C}} \neq B_{\mathrm{C}\to\mathrm{A}} = B_{\mathrm{C}\to\mathrm{B}}$. Furthermore, the average unidirectional bandwidth per inter-domain link in this case is b^d for controllers A and B, but it is b^e for controller C, since the links are external to it but of inter-domain type. Now the observed bandwidth in one direction is obtained by summing the following contributions: B^I for 1 switch to model the switch in A's domain; B^I for $S-1$ switches to model the $S-1$ switch in B's domain; $S-1$ times the average bandwidth per inter-domain link b^d and $S-1$ times the average bandwidth per external inter-domain link b^e.

TABLE III: Notation describing the network topology in the scenario with 3 controllers. Let x, y be two distinct controllers, with $x, y \in \{\text{A,B,C}\}$.

Symbol	Meaning
S_x	number of switches in controller x's domain
S	total number of switches in the network
L_x	number of intra-domain links in controller x's domain
L_{xy}	number of inter domain links between x and y
L	total number of links in the network
L_{ID}	total number of inter-domain links in the network

Using the same methodology before, and exploiting the estimated values obtained so far, we are able to estimate that the average bandwidth per inter-domain link as:

$$b^d = 0.77 \text{ kbps} \quad b^e = 0.15 \text{ kbps} \tag{12}$$

By combining the results so far and the estimated bandwidths in (8), (9), (10), (11) and (12), we can claim the following:

Property 2: In an arbitrary network managed by a ONOS cluster of 3 controllers A, B and C, the traffic exchanged from controller x to controller y is:

$$B_{x\to y} = 47.81 + 4.43 \cdot S_x + 1.43 \cdot L_x + 0.46 \cdot (S_y + S_z) + 0.06 \cdot (L_y + L_z) + 0.77 \cdot (L_{xy} + L_{xz}) + 0.15 \cdot L_{yz} \quad [\text{kbps}] \tag{13}$$

for any selection of distinct controllers $x, y, z \in \{\text{A,B,C}\}$ (i.e. such that $x \neq y$, $x \neq z$ and $y \neq z$).
Let $B_{TOT} = B_{\text{A}\to\text{B}} + B_{\text{A}\to\text{C}} + B_{\text{B}\to\text{A}} + B_{\text{B}\to\text{C}} + B_{\text{C}\to\text{A}} + B_{\text{C}\to\text{B}}$ be the total exchanged traffic among the 3 controllers. Referring to the notation in Table III, we can claim:

Corollary 2: In an arbitrary network managed by a ONOS cluster of 3 controllers, the total traffic exchanged among the 3 controllers is

$$B_{TOT} = 286.86 + 10.7 \cdot S + 3.10 \cdot L + 0.28 \cdot L_{ID} \quad [\text{kbps}]$$

where $L_{ID} = L_{\text{AB}} + L_{\text{BC}} + L_{\text{AC}}$. Thus, also in this scenario, the total traffic appears to be proportional to the number of switches and the number of edges in the topology.

D. Inter-controller traffic in real ISP topologies

To prove the wide applicability of our approach, we apply the empirical models of Property 1 (for 2 controllers) and Property 2 (for 3 controllers) to 262 real ISP network topologies obtained from the Internet Topology Zoo [12] to obtain the inter-controller traffic in case of a distributed ONOS cluster managing the whole ISP network. The number of nodes and edges in each ISP is shown in Fig. 12 and show a high variety, even if in most of the cases the topology graph is not dense. This is reasonable, since for a large (in term of geographical distance) ISP, dense graphs are expensive and this fact advocates a careful design of the in-band communication network to support inter-controller traffic.

In order to obtain results regarding the inter-controller traffic that are independent from the controller chosen as the master for each switch, we evaluate the maximum and minimum value of the inter-controller traffic by assuming (without loss of

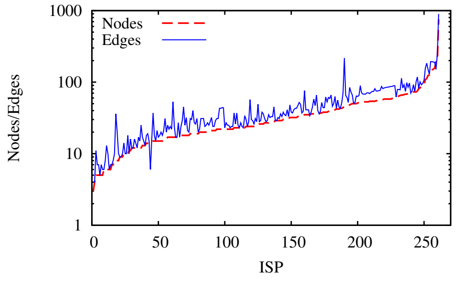


Fig. 12: Size of the network topologies considered for the inter-controller traffic in real ISP topologies

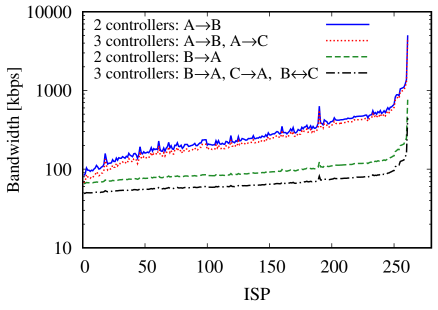


Fig. 13: Estimated inter-controller traffic, between pairs of controllers, due to the topology store for realistic ISPs

generality) that each switch in the ISP topology is connected to a single controller denoted as A. This is because the contribution to the inter-controller traffic for a controller is maximum when all the switches are in controller A's domain (i.e. $S_{\text{A}} = S$, $L_{\text{A}} = L$, $S_{\text{B}} = L_{\text{B}} = 0$), as when applying (7), the coefficients for S_{A} and L_{A} are larger. Conversely, this assumption minimizes the traffic generated by controller B towards A. A similar argument can be used in (13) to show that the upper and lower bounds can still be obtained by associating all the switches to controller A.

Fig. 13 shows the average amount of inter-controller traffic for each ISP, exchanged between pairs of controllers. In the case of 2 controllers, traffic A $\to$ B and B $\to$ A give the maximum and minimum values. In the case of 3 controllers, traffic A $\to$ B or A $\to$ C provide the maximum values, whereas B $\to$ A, C $\to$ A or B $\leftrightarrow$ C provide the minimum. According to our experiments, the maximum inter-controller traffic in the 2-controllers scenario for the 261st ISP is $B_{\text{A}\to\text{B}} = 5029$ kbps and the minimum is $B_{\text{B}\to\text{A}} = 763.75$ kbps. Both values are practically relevant, since for a generic partitioning of the network in two controller domains, a bandwidth of about 1-10 Mbps must be guaranteed among the pair of controllers, just to synchronise the topology store.

Similarly in the 3-controller case, the maximum inter-controller traffic for the 261st ISP is $B_{\text{A}\to\text{B}} = 4668.8$ kbps and the minimum is $B_{\text{B}\to\text{A}} = 447.73$ kbps. Also in this case the actual traffic is practically relevant, since about 1-10 Mbps is again required to support the communication among any pair of controllers. By comparing the results in Fig. 13 referred to different number of controllers, the traffic reduction is evident for 3 controllers with respect to 2 controllers case, as already

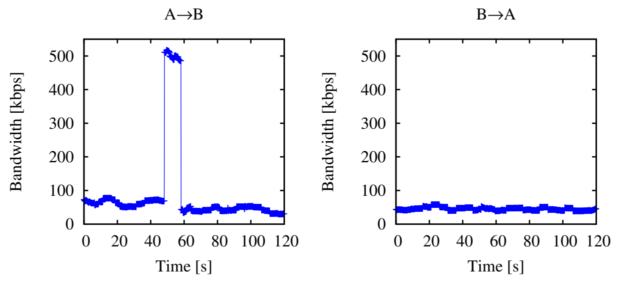


Fig. 14: Effect of a single flow addition the occurring at time 50 s on the synchronization of the flow store

observed in Sec. IV-C.

V. Distributed Flow Store

In ONOS, a copy of each switch's flow table is maintained in the flow store by its respective master controller and by the first slave controller, i.e. the new master if the current master fails. We investigate the impact of modifying the switches' flow tables by OpenFlow `flow-mod` commands.

To synchronize the flow stores within the ONOS cluster, according to the code available in [13], the following process occurs: every 2 seconds, the master controller checks for any change in the flow table of each switch under its domain since the last backup to the slave controller. The change detection is based on comparing the time when the flows changed in a switch and the time of the last backup to the slave controller. Interestingly, in the case of a single modification of a flow, the *whole* flow store of the corresponding switch is copied to the slave controller.

As an example, consider the bandwidth measurement shown in Fig. 14, referring to the scenario in which a switch is connected to master controller A and slave controller B. Initially, the switch has 5004 flows installed in it. At around time 50s, one additional flow is installed on the switch, which causes the controller A to backup the whole flow table (5005 flows) to controller B. This results in a transient traffic increase, just for the traffic A $\rightarrow$ B.

A. Experimental methodology

We describe here the adopted methodology to calculate data exchanged per flow, i.e. taking into account the contribution of each individual flow. According to the previous observations, the flow store backup from the master controller to one slave controller is a transient phenomenon, thus the inter-controller traffic is *event-driven with full updates*, according to the classification in Sec. III-A. By observing the traffic with the sniffer, we discover that ONOS adds the string "flow-backup" in its packets when backing the switch flow table. Thanks to this observation, we can easily isolate the traffic due to the flow table backup.

We adopt the testbed illustrated in Fig. 15. The topology consists of one isolated switch connected to one master and one slave controller. The flows to be installed are configured on the master controller through the ONOS north-bound REST APIs (step 1). As a consequence, the flows are installed

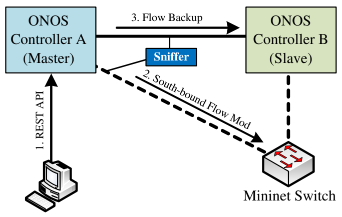


Fig. 15: Methodology to investigate the effect of flow modifications in the inter-controller traffic

on the switch via the controller south-bound interface using OpenFlow (step 2) and then the controller backups the flow table to the slave controller (step 3).

To obtain general results, we test different types of flows with different versions of OpenFlow, while varying the number of flows, in order to evaluate the minimum and the maximum amount of data exchanged per flow. Let F be number of new added flows whose effect must be analyzed. We start by installing $F-1$ flows. After the traffic has reach a steady state, we install a single additional flow, in order to avoid multiple backups. Thanks to the traffic trace, we calculate the data exchanged per flow by computing the ratio of observed data on the network (in terms of Ethernet packet size) by the number of flows in the table.

The amount of per-flow data depends on the adopted "match" and "actions" fields adopted in the experiments, which in turn, depend also on the specific version of OpenFlow. ONOS version 1.4.1 supports two versions of OpenFlow: 1.0 [14] and 1.3 [15]. In order to get general results, we devise two types of flow definitions to be added in the table. *Type-1* is forged to be the flow definition with the minimum size, corresponding to the smallest `flow-mod` packet (on the south-bound interface) and thus the minimum inter-controller traffic. *Type-2* is instead forged to be the flow definition with the maximum size. In order to induce a constant synchronization traffic among the controllers, we generate a new flow by changing the value of just one matching field. For OpenFlow 1.0, we configure Type-1 flow definitions by just setting the EtherType matching field and a basic forward action, as shown in Table IVa, and we vary just the EtherType field for each new flow. Instead, we configure Type-2 flow definitions by setting all the 10 matching fields available in the REST APIs exposed by ONOS for OpenFlow 1.0 and all the 8 allowed actions, as shown in Table IVb. Similarly, for OpenFlow 1.3, we use the same Type-1 definitions as before, shown in Table Va. Instead, for Type-2 definitions we set all 16 match fields and all the 11 available actions in Table Vb. Notably, we exploit IPv6 fields since they require a larger number of bits for their definitions. In all Type-2 definitions, we vary the TCP source port field for each new flow.

B. Experimental results

Fig. 16 shows the amount of data exchanged per flow averaged on 10 experiments, for all possible 4 possible cases, combining Type-1 and Type-2 flow definitions with the two considered OpenFlow versions. As a reminder, this data includes all the packet overheads starting from the Ethernet

TABLE IV: Types used for OpenFlow 1.0 experiments

(a) Type-1

	Field
Match	EtherType
Action	Output to Controller

(b) Type-2

	Field
Match	Input Port Ethernet Source/Destination Ethernet Type IPv4 Source / Destination IP Protocol Type / DSCP TCP Source/Destination Port
Action	Output to Controller Change VLAN ID / PCP POP VLAN Change Ethernet Source / Destination Change IPv4 Source / Destination

TABLE V: Types used for OpenFlow 1.3 experiments

(a) Type-1

	Field
Match	EtherType
Action	Output to Controller

(b) Type-2

	Field
Match	Input Port Metadata Tunnel ID VLAN ID / PCP Ethernet Source / Destination / Type IPv6 Source / Destination / Flow label IP Protocol Type / DSCP / ECN TCP Source / Destination
Action	Output to Controller Change VLAN ID / PCP POP VLAN Change Ethernet Source / Destination Change Tunnel ID Change IPv6 Source / Destination Change TCP Source / Destination

TABLE VI: Experimental results due to the modification of flow store

OpenFlow Version	Flow Type	Per-flow exchanged data [bytes]	
		OpenFlow packet	Inter-controller data
1.0	Type-1	146	110
	Type-2	218	304
1.3	Type-1	154	110
	Type-2	458	409

TABLE VII: Maximum inter-controller traffic generated in some commercial OpenFlow switches due to flow store updates

Commercial Switch	Maximum flow rules	Maximum inter-controller traffic [Mbps]
Dell PowerConnect 8132F	750	0.91
HP ProCurve 5406zl	1500	1.83
Pica8 P-3290	2000	2.43

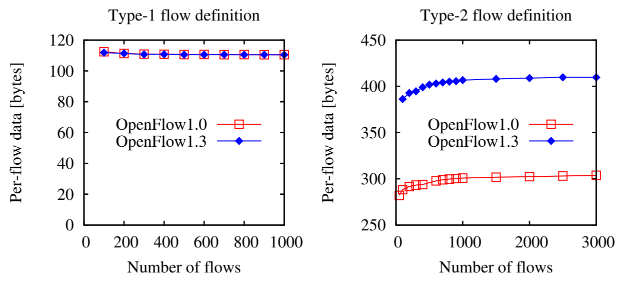


Fig. 16: Inter-controller data exchanged for each flow

PDU. All the graphs show that per-flow data converge to a fixed value, which can be evaluated for a large enough number of flows. Table VI compares the inter-controller data for each flow, evaluated for the fixed value obtained in Fig. 16 with the value obtained by observing the size of the corresponding `flow-mod` packet (including the Ethernet PDU for fair comparison). As the "match" and "action" fields increase from Type-1 to Type-2, the size of per-flow data increases for both OpenFlow versions. By comparing the two rightmost columns, the inter-controller data exchanged for each flow appears comparable with the size of the corresponding flow-mod packet. The difference is due to the different internal format[1] and the packet overheads. Type-1 always corresponds to 110 bytes for each flow in the inter-controller traffic due to the same internal representation in ONOS, whereas Type-2 shows a different size depending on the OpenFlow version due to the different match and action fields that are exploited. Thanks to the larger number of available fields, each flow can require up to 409 bytes to be synchronized across the other controllers.

To understand the practical impact of the above experimental results, we observe that the adopted full update scheme in the flow store may generate large synchronization traffic among the controllers, especially when the flow table is large. Thus, we now evaluate the maximum traffic generated in some commercial switches assuming that (i) the flow table is full, (ii) at least one flow modification occurs every 2 seconds, (iii) OpenFlow 1.0 Type-2 flow definitions are adopted in each flow update. Thus, a table update is triggered every 2 seconds, requiring the exchange of the full flow table, and each flow entry corresponds to 304 bytes, based on the results of Table VI. We consider the physical OpenFlow 1.0 switches analyzed in [16] with the maximum number of flow rules specified in Table VII, where we also show the numerical results for the worst-case inter-controller traffic due to flow updates evaluated based on the previous assumptions. Notably, the bandwidth required to backup the flow table is in the order of Mbps which is relevant, due to just one flow update every 2 seconds.

The above experimental results can be used to compute the inter-controller traffic due to the changes in a flow table for a network, with an arbitrary number of controllers and domains. For this purpose, the master and the first slave controller of each switch must be known, along with the information regarding the existing flows in the switch.

VI. Distributed Host Store

We describe the impact of the presence of hosts in the network on the inter-controller traffic. Events generated due to hosts in data plane have a transient effect on the inter-controller traffic. This is due to the fact that host information is exchanged among the controllers in a strongly consistent manner backed by RAFT consensus protocol; thus we classify this traffic as *event-driven with incremental updates.*

A. Methodology

The event according to which a host is added to a switch controlled by ONOS impacts on two data structures. First, an additional port is added to the data structure representing the switch in the topology store. Second, the information about the new host is recorded in the host store. Thus, the inter-controller traffic is affected by two different protocols: the anti-entropy for the topology store, generating periodic and full updates, and RAFT for the host store, generating event-driven and incremental updates. In order to distinguish between the two contributions, we adopt the following methodology.

[1]OpenFlow adopts Extensible Match (OXM) representation [15] to allow variable "match" field in the south-bound.

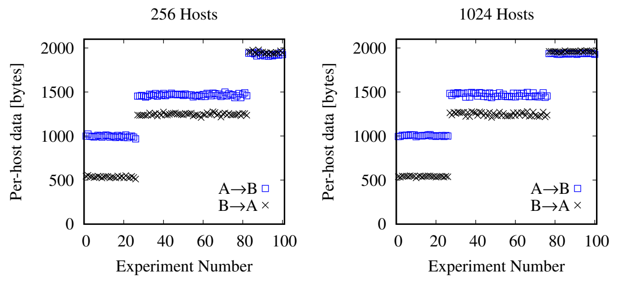


Fig. 17: Scenario with 2 controllers and all hosts added to the switch whose master controller is A

We exploit the north-bound REST APIs to connect multiple hosts (distinguished by different MAC addresses) to the same port of the switch. In such a way, we avoid adding a new port to the switch for each new host. We actually define a dummy switch at which all the hosts are connected, and in this way we avoid to use Mininet as network emulator. We proceed by simultaneously adding a batch of hosts to the same dummy switch. By evaluating the traffic increment due the transient phase induced by the hosts addition, we evaluate the average amount of data exchanged between the controllers for each new added host. The experiments are carried out for 2 and 3 controllers. Each experiment is repeated 100 times.

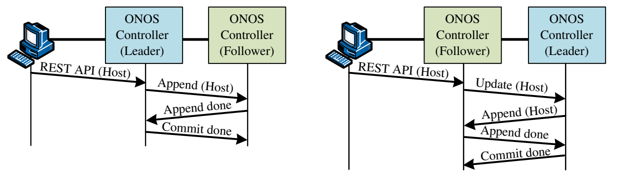
B. Experimental results for 2 controllers

We start by considering the scenario with 2 controllers. Fig. 17 shows the result of inter-controller data per host, with the hosts added to controller A. The results depend on the specific role of the controller (leader or follower of the corresponding shard) that acts as master of the switch at which the hosts are added. Notably, this role cannot be set a-priori and changes randomly for each experiment. By changing the number of added hosts in a batch the results are the same, thus our numerical results appear to be reliable. By observing the inter-controller data for each host (denoted as D) for different number of hosts that are added in batch, we identify three different behaviors which depend on the roles of the controllers in managing the shards:

- Case 1: Controller A is the leader of all shards of the host store. From the graphs, $D_{\mathrm{A}\to\mathrm{B}} \approx 1000$ bytes and $D_{\mathrm{B}\to\mathrm{A}} \approx 500$ bytes.
- Case 2: Controller B is the leader of all shards of the host store. Both $D_{\mathrm{A}\to\mathrm{B}}$ and $D_{\mathrm{B}\to\mathrm{A}} \approx 2000$ bytes.
- Case 3: Each controller is the leader of at least one shard. $D_{\mathrm{A}\to\mathrm{B}} \approx 1500$ bytes and $D_{\mathrm{B}\to\mathrm{A}} \approx 1200$ bytes.

Recall that controller A is always master of the switch to which the hosts are added. The different data values obtained in the 3 scenarios are explained in the following paragraphs. The results in Fig. 17 are grouped based on the above cases.

Case 1 and 2 are the most interesting as they give an upper and lower bound respectively to the per-host data. In case 1, controller A is master and manages directly the host updates received from the switch. Controller A is also the leader for all the shards and thus directly updates the follower controller B, which corresponds to the minimum amount of exchanged



(a) All hosts added to the leader controller
(b) All hosts added to the follower controller

Fig. 18: Scenario with 2 controllers for distributed host store

data. This is clear from the protocol behavior as shown in Fig. 18a: once the leader receives information about the host from REST API, it sends this to the follower. The follower adds this instruction to its log and sends a message to the leader that it is updated. The leader then sends a “commit done” message to the follower to end this transaction.

Instead in case 2, controller B is the leader of all the shards. Thus when a host is added to the switch whose master is controller A, acting as follower for the host store, then A must update the leader B first before anything is committed, as shown in the protocol diagram in Fig. 18b. After B is notified, the same sequence of messages is observed as in Fig. 18a. The additional messages exchanged in case 2 explain the larger traffic with respect to case 1. Observe now that the actual experimental values are not consistent if only a single message was added in case 2 (denoted “Update host” in Fig. 18b). For example in case 1, the follower sends around 500 bytes per host to the leader while in case 2, it sends around 2000 bytes. This can be explained as the Network Configuration Subsystem (i.e. a ONOS internal module) is involved whenever a host is added to a controller. When a controller receives a host to be updated, the Network Configuration Subsystem does a read operation on a strongly consistent data structure backed by RAFT. This read operation is done on the leader. If the controller is itself the leader, the read operation is served locally as in case 1. Instead, the read operation is served remotely by the leader in case 2. This adds extra messages to the inter-controller traffic, which amounts to extra data exchanged per host.

C. Experimental results for 3 controllers

We now consider the scenario with 3 controllers. Since the results in the previous scenario with 2 controllers depend heavily on the role of the controllers, we adopt the following methodology, in order to just find an upper and lower bound on the inter-controller traffic.

In each experiment, we start all the containers with ONOS controllers and then check if one specific controller is by chance the leader of all partitions of all the data structures, by following the logs of the leader election phase in RAFT consensus protocol; otherwise the containers are rebooted. The 3-controller cluster includes a leader and two follower controllers F_1 and F_2. Two specific cases are adopted to achieve a lower and upper bound on amount of data exchanged per host:

- Case 1: All hosts are added to the switch whose master is the leader controller.

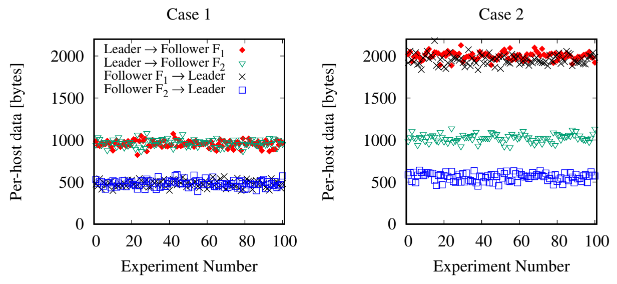


Fig. 19: Scenario with 3 controllers and all hosts added to the leader (case 1) or to one follower (case 2)

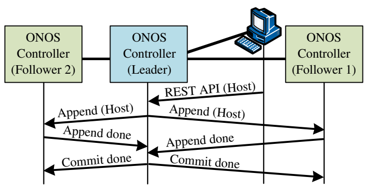


Fig. 20: Scenario with 3 controllers and all hosts added to the leader controller

- Case 2: All hosts are added to the switch whose master is one follower controller (assume F_1).

Similarly to the scenario with 2 controllers, case 1 produces the minimum amount of inter-controller traffic due to host addition, whereas case 2 the maximum one.

The data exchanged for each host update is shown in Fig. 19. The results show that, regardless of the role of the controllers, the minimum amount of data for each flow is around 500 bytes and the maximum one is around 2000 bytes, coherently with the previous scenario. To understand the actual values, we observe the protocol diagrams for the RAFT messages for the two cases, as reported in Figs. 20 and 21. Case 1 for 3 controllers achieves the same amount of data for each flow than in the 2 controllers case. This is because the data structure is updated in the same manner. Case 2 on the other hand has a different behavior. Fig. 19 shows that the data between the leader and F_1 is same as that of case 2 for 2 controllers; on the contrary, the data between leader and F_2 is equivalent to add a host to the leader. This is due to the fact that the Network Configuration Subsystem comes into play in the communication between leader and F_1, since F_1 does a read operation while accessing the leader. This read operation is not done on the communication between the leader and F_2. The traffic between the two followers F_1 and F_2 in all cases does not vary as all the read and write operations in RAFT are done through the leader.

Notably, in the RAFT implementation of ONOS, no more than 3 controllers constitute a partition. Thus the lower and upper bound of data exchanged per host within a partition computed in the previous section are expected to hold in general, independently from the number of controllers in the cluster.

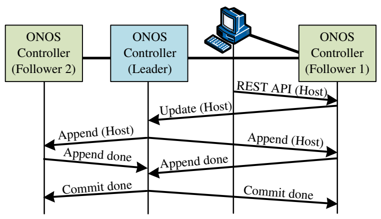


Fig. 21: Scenario with 3 controllers and all hosts added to the follower controller F_1

VII. Related work

The need of a distributed SDN architecture has been thoroughly advocated in literature, since it provides resiliency and scalability as compared to a centralized single controller implementation. Onix [17] is a type of distributed SDN controller which uses partitioning to introduce scalability in distributed SDN. In Onix, although the network topology information known as Network Information Base (NIB) is fully replicated on all the cluster nodes, in a way similar to the topology store in ONOS, yet the information such as forwarding state of devices and link utilization levels are partitioned among controllers, similarly to what RAFT does in ONOS.

Consistency issues in distributed SDN data plane have been highlighted in [1], which extended the classic CAP theorem for distributed systems. [1] discusses examples of network policies operating under network partitions and highlights the advantages of an in-band control plane in distributed SDN controllers. Indeed, out-of-band control information among the controllers may provide less resilience than in-band control one. The intuitive idea is that, for pure in-band control, only in case of data plane partitions the controllers are actually partitioned. Instead, in case of out-of-band control plane, it may happen that the data plane is fully working whereas the control plane is partitioned, creating inconsistency problems. Our work is strongly motivated by the need to understanding deeply the bandwidth required to implement an efficient transportation of in-band control information, and this is crucial for network planning purposes.

Recently, [18] proposed a centralized in-band synchronization approach to achieve a consistent behavior across distributed controllers. Coherently with the motivations of our work, the authors advocate the use of in-band control signaling and highlight the importance of achieving consistency among the controllers. They propose a new set of atomic primitives to ensure consistency, and our proposed experimental methodology could be applied also to their consistency system to evaluate the actual inter-controller traffic due to in-band synchronization. Such evaluation is currently neglected in their work.

The importance of preserving consistency in shared data structures across SDN controllers was highlighted by [19]. In contrast to ONOS as well as Onix, only a global strongly consistent key-value based data store is employed, since it can provide acceptable performance as well as fault-tolerance. The data store is based on replicas which employ state machine replication using a combination of Paxos (a well known consistency algorithm) and Viewstamped Replication (VR). Similar to RAFT, all operations are coordinated using a leader acting as the primary replica which handles all the read and write operations. While comparing the performance, the authors state that existing strongly-consistent data stores implementing the Paxos/VR protocols can perform as good as an eventually consistent data store in Onix for some applications, but the cost of latency is inevitable as a strongly consistent data store is involved.

DISCO (Distributed Multi-domain SDN Controllers) [20] discusses specifically the inter-controller traffic, which is given by two contributions: (i) delegating functions among various agents such as controller reachability, monitoring and relaying inter-controller link health, new controller domain discovery or reservation of inter-domain flow setup and teardown (ii) operating a *Messenger* module for inter-controller communication based on an Advanced Queuing Messaging Protocol (AQMP) used by the agents. An interesting mechanism in DISCO reconfigures the inter-controller links to abandon congested or slow links and use other controllers as relays for inter-controller communications, which is absent in ONOS.

Notably, [19] and [20] do not evaluate the cost in terms of bandwidth needed to support the proposed consistency schemes, and our methodology can be adapted to address such issue in both scenarios.

The work in [21] investigated the synchronization cost due to the exchange of inter-controller traffic among controllers by analyzing the synchronization delay. This delay consists of the time taken by a controller to detect an event in its domain till the time a different controller becomes aware of it. There exists a trade-off between the synchronization delay and the amount of synchronization data. Different network applications may require faster coordination among controllers at the expense of higher synchronization data exchange rate and vice versa. In contrast, our work focuses on the inter-controller traffic due to network events, while neglecting the delay to achieve consistency. Nevertheless, our empirical models enable a proper planning of the network supporting the control plane, and thus allow to control the corresponding delay performance.

Finally, our approach is perfectly complementary to [22], since the latter work focused on the OpenFlow traffic exchanged by ONOS controller with the switches on the southbound interface. Thus, by combining the results in [22] with the results of our work (i.e. the control traffic exchanged among the controllers), it is possible to properly plan and design the whole transport network supporting the overall control plane in a cluster of ONOS controllers.

VIII. Conclusions

We considered a distributed SDN architecture in which a cluster of ONOS 1.4 controllers, manages all network devices. We focused our investigations on the traffic exchanged between the controllers, which is mainly due to the consensus protocols enabling a consistent view of the network state.

We adopted an experimental testbed based on a cluster of 2 and 3 ONOS controllers and evaluated experimentally the inter-controller traffic due to different shared data structures and to different network configurations and events. We investigated specifically all the distributed stores that describe the network state (i.e. topology, host and flow stores) and derived some quantitive models to estimate the inter-controller traffic under very general conditions. Even if the results are specific of the considered version of the controller, our methodology is general and can be applied to other versions of ONOS and to different SDN controllers from ONOS. As a future work, we plan to apply our methodology to a larger number of controllers adopting the latest version of ONOS.

Thanks to our experimental results, a network designer can design and plan carefully the network infrastructure that support the inter-controller data plane. This is of paramount importance for network operators running large SDN networks, like SDWANs, where the control data is typically in-band and share the same resources devoted to the customers.

References

[1] A. Panday, C. Scotty, A. Ghodsiy, T. Koponen, and S. Shenker, "CAP for networks," in *HotSDN*. ACM, 2013, pp. 91–96.

[2] S. Vissicchio, L. Vanbever, and O. Bonaventure, "Opportunities and research challenges of hybrid software defined networks," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 2, pp. 70–75, Apr. 2014.

[3] "ONOS 1.4 Wiki," https://wiki.onosproject.org/display/ONOS14/Wiki+Home.

[4] "On.Lab website," http://onlab.us/.

[5] "OpenDaylight website," https://www.opendaylight.org/.

[6] A. S. Muqaddas, A. Bianco, P. Giaccone, and G. Maier, "Inter-controller traffic in ONOS clusters for SDN networks," in *IEEE International Conference on Communications (ICC)*, Kuala Lumpur, Malaysia, May 2016, pp. 1–6.

[7] J. Xie, D. Guo, Z. Hu, T. Qu, and P. Lv, "Control plane of software defined networks: A survey," *Computer Communications*, vol. 67, pp. 1 – 10, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0140366415002200

[8] P. Bailis and A. Ghodsi, "Eventual consistency today: limitations, extensions, and beyond," *Communications of the ACM*, vol. 56, no. 5, pp. 55–63, May 2013.

[9] E. Brewer, "CAP twelve years later: How the "rules" have changed," *IEEE Computer*, vol. 45, no. 2, pp. 2–13, March 2012.

[10] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Annual Technical Conference*, 2014, pp. 305–320.

[11] "Linux Containers," https://linuxcontainers.org/.

[12] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[13] "ONOS Distributed Flow Rule Store," https://github.com/opennetworkinglab/onos/blob/onos-1.8/core/store/dist/src/main/java/org/onosproject/store/flow/impl/DistributedFlowRuleStore.java.

[14] "OpenFlow 1.0 (Wire Protocol 0x04) specification," http://archive.openflow.org/documents/openflow-spec-v1.0.0.pdf.

[15] "OpenFlow 1.3 (Wire Protocol 0x04) specification," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.3.0.pdf.

[16] M. Kuźniar, P. Perešíni, and D. Kostić, "What you need to know about SDN flow tables," in *Passive and Active Measurement: 16th International Conference*. Springer International Publishing, 2015, pp. 347–359.

[17] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievskiy, M. Zhuy, R. Ramanathany, Y. Iwataz, H. Inouez, T. Hamaz, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proc. USENIX conference on Operating systems design and implementation* , Berkeley, CA, USA, 2010.

[18] L. Schiff, S. Schmid, and P. Kuznetsov, "In-band synchronization for distributed SDN control planes," *SIGCOMM Comput. Commun. Rev.*, vol. 46, no. 1, pp. 37–43, Jan. 2016.

[19] F. A. Botelho, F. M. V. Ramos, D. Kreutz, and A. N. Bessani, “On the feasibility of a consistent and fault-tolerant data store for SDNs,” in *2013 Second European Workshop on Software Defined Networks*, Oct 2013, pp. 38–43.

[20] K. Phemius, M. Bouet, and J. Leguay, “DISCO: Distributed multi-domain SDN controllers,” in *IEEE Network Operations and Management Symposium*. IEEE, 2014, pp. 1–4.

[21] F. Benamrane, F. J. Ros, and M. B. Mamoun, “Synchronisation cost of multi-controller deployments in software-defined networks,” *Int. J. High Performance Computing and Networking*, vol. 9, no. 4, pp. 291–298, 2016.

[22] A. Bianco, P. Giaccone, R. Mashayekhi, M. Ullio, and V. Vercellone, “Scalability of ONOS reactive forwarding applications in ISP networks,” *Computer Communications*, 2016.

Abubakar Siddique Muqaddas received the B.E. in Electrical (Telecommunications) Engineering from NUST, Rawalpindi, Pakistan and M.Sc. in Telecommunications Engineering from Politecnico di Torino, Italy, in 2011 and 2015 respectively. Currently he is a Ph.D. candidate in the Electrical, Electronics and Telecommunications Engineering program in Politecnico di Torino. He is a Cisco Certified Network Associate (CCNA) and a Cisco Certified Network Professional (CCNP). His current interests are in assessing distributed SDN controller architectures, software-defined optical network operations and management of state in SDN.

Paolo Giaccone is an Associate Professor in the Department of Electronics, Politecnico di Torino. During the summer of 1998, he was with the High Speed Networks Research Group, Lucent Technology-Bell Labs, Holmdel, NJ. During 2000-2001 and in 2002 he was with the Information Systems Networking Lab, Electrical Engineering Dept., Stanford University, Stanford, CA. His main area of interest is the design of network algorithms, in particular for the control of SDN networks and of cloud computing systems. He is an IEEE Senior Member.

Andrea Bianco is Full Professor and Department Head of the Dipartimento di Elettronica e Telecomunicazioni of Politecnico di Torino, Italy. He has co-authored over 200 papers published in international journals and presented in leading international conferences in the area of telecommunication networks. He is Area Editor for the IEEE JLT (Journal of Lightwave Technology) and of the Elsevier Computer Communications journal. His current research interests are in the fields of protocols and architectures of all-optical networks, switch architectures for high-speed networks, SDN networks and software routers. Andrea Bianco is an IEEE Senior Member.

Guido Maier is Associate Professor at Politecnico di Milano (Italy). His main areas of interest are: optical network modeling, design and optimization; SDN orchestration and control-plane architectures; WAN optimization. He is author of more than 100 papers in the area of Optical Networks and Networking published in international journals and conference proceedings (h-index 17) and 6 patents. He is currently involved in industrial and European research projects. He is editor of the journal Optical Switching and Routing and TPC member in many international conferences. He is a Senior Member of the IEEE Communications Society.