Controller Placement for Minimum Control Traffic in OpenDaylight Clustering

Marios Karatisoglou, Kostas Choumas and Thanasis Korakis
Dept. of ECE, University of Thessaly, Volos, Greece
Email: karatiso, kohoumas, korakis@uth.gr

Abstract— **Software-defined networking (SDN) decouples the control and data planes and moves the control logic to the SDN controllers. The controllers are servers handling the switches and directing their traffic with the use of open and non-proprietary protocols. Consequently, the controller is crucial for the network performance. OpenDaylight (ODL) is a widely recognised controller platform implementing a clustering mechanism, where multiple controllers are synchronized to behave as a single one, while also enabling load balancing and failure resilience. The purpose of this paper is to extract the communication patterns between the ODL controllers in a cluster, as well as between the controllers and the switches, in order to place the cluster controllers appropriately in a network. Our testbed experimentation reveals the bandwidth requirements of the control traffic in an ODL cluster, while a network model is introduced for the problem of determining the optimal controller placement for minimum control traffic.**

I. Introduction

Software-defined networking (SDN) has brought about radical changes in the way networks are designed. Some of the most important features that have been affected are the control and data plane. The control plane (how the network traffic is handled) and the data plane (how the traffic is forwarded according to the decisions made by the control plane) used to be embedded inside the networking devices (routers and switches), reducing the overall flexibility of the networking infrastructure. With a software-defined network architecture, the control plane is now separated from the data plane. Thus, a software control program, named SDN controller, can now control the functionality of many data-plane elements [1], named SDN switches. Still, a network with a single controller comes with several weaknesses, such as reliability and scalability, since the control plane is centralized. To address this issue, ODL [2] community has implemented a clustering mechanism, in which multiple controllers operate, achieving high availability and fault tolerance.

Running multiple controllers introduces new problems, due to the aim of operating as a single controller with a centralized logic. When different controllers are operating together, the overall state of the cluster is distributed among them. The state of each individual controller should not differ from the rest of the cluster controllers. Thus, a mechanism to synchronize the functionality of the cluster controllers is essential. ODL implements various techniques and algorithms to address these problems, which generate extra control traffic. The focus of this research is the controller placement [3] in the network aiming at minimizing the bandwidth required for the total control traffic, including both the controller-to-controller and the controller-to-switch traffic.

The minimization of the bandwidth required for the control traffic is the highest priority objective in networks with low capacity control plane, such as low speed wireless networks with in-band control [4]. Moreover, the less the control traffic, the less the extra energy consumed for non-data transmissions, which is very important for limited-energy sensor networks. Finally, the minimization of the time delays between the controllers and the switches is not essential in cases where the SDN control is proactive. For all these reasons, the minimum control traffic is very important e.g. in IoT networks exploited by massive machine-type communications (mMTC) in 5G. The application of SDN in these networks, especially regarding the controller placement models [5], is an open issue with a lot of ongoing research.

The paper is divided in the following sections. Section II presents work related to SDN controllers and controller clustering. Section III focuses on the ODL clustering mechanism. It describes how the data is distributed between the controllers. Section IV presents the network model, as it is derived by the description of the ODL clustering mechanism. Afterwards, Section V includes the results from monitoring the communication between the cluster nodes, along with the experimentation environment setup and tools that have been used during our research. Section VI concludes the paper and presents implications for future work.

II. Related Work

There are various studies that have been conducted to investigate the performance and scalability of SDN controllers. The research in [6] focuses on the comparison between three well-known controllers, called ONOS, Floodlight and POX, based on the way they communicate with the switches. However, this work does not cover the performance of those controllers as a part of a cluster. In [7], failover time, failback time and QoS issues in ODL clustering are addressed. Still, there is lack of research on how the ODL cluster scales, while switches are being connected to the network and flow rules are being installed. The impact of controller placement in clustering performance is also left unexamined.

The focus of this study is the controller placement problem, that was first introduced in [3]. In [8], a new dimension in this problem is introduced, since the controller-to-controller traffic is also considered, besides the controller-to-switch one. The optimization criterion is related to the reaction time perceived at the switches, that is dependent on the communication delays. In [9], a joint study of both types of control traffic overhead is presented, which are considered as two of the four costs that are included in the objective's weighted sum to be minimized. In [10], Pareto-optimal placements are derived aiming to solve a multi-objective optimization, where one of the objectives is the minimization of the control traffic. In [11], the objective of minimizing the overhead of software defined measurements is considered, assuming however fixed controller-to-controller costs. In [12], an approximation algorithm with a guaranteed performance bound is derived for a model similar to the one we consider in this paper, however the proof requires that the per-unit-load cost for the traffic between any two controllers is a constant, which is usually not the case in practice.

In [4], a theoretical and experimental study has been conducted, regarding the bandwidth usage of the control traffic in a cluster of Kandoo controllers [13]. Our paper can be seen as an extention of the results in [4] for the ODL controller, which is generally more widely used than Kandoo. To the best of our knowledge, this is the first study of ODL controller placement focusing exclusively on the minimization of the total required bandwidth for the control traffic. Furhtermore, this work is one of the few works that present experimentation results in a realistic environment offered by an SDN testbed.

III. OpenDaylight Clustering Overview

A. Akka Cluster

ODL clustering uses the fault-tolerant and peer-to-peer membership service of Akka cluster [14], which relies on a gossip protocol and an automatic failure detector. Every time a node starts with the intention to join an Akka cluster, it initiates a 4-way-handshake defined by the gossip protocol. It communicates with the seed nodes (preconfigured points of contact for new nodes) by sending *InitJoin* messages. The seed node confirms the *InitJoin* request by replying with a *InitJoinAck* message. The node now sends a *Join* message and it has successfully joined the cluster when it receives a *Welcome* message back from the seed node. The gossip protocol is not only responsible for replicating the current state of the cluster to all its nodes, but also it is crucial for the cluster's persistency. *Heartbeat* messages are sent among the cluster nodes periodically and expect a *HeartbeatRsp* back within a timeout to consider a node valid.

B. Distributed Datastore

ODL's distributed datastore is based on Akka and segmented in three shards (or chunks), called *inventory*, *topology* and *toaster*. Inventory shard contains the flow rules of the switches controlled by the cluster, while topology shard holds data about the network topology. Inventory or topology shard's size is increasing when extra flows or switches are added respectively. Toaster shard defines the Remote Procedure Calls (RPC) that can be executed in each shard, along with the actions that should be taken upon each call.

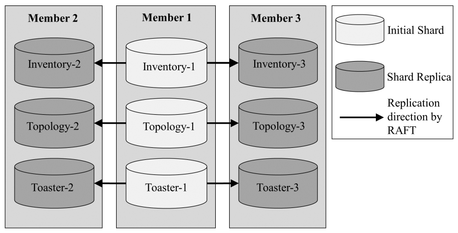

Fig. 1. Shard distribution and replication between the cluster nodes. The initial shards are replicated from member 1 to members 2 and 3.

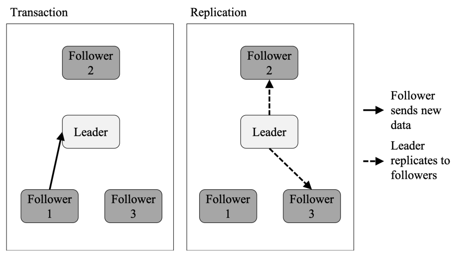

Fig. 2. At first, Follower 1 sends to the Leader its new data. Then, Leader replicates the new data to the rest of the followers. Notice that there is no direct replication between the followers.

Each shard is firstly assigned to a specific cluster node, which is not necessarily the same for all shards. Then, it is distributed across the cluster by being replicated to all other nodes, in order to avoid data loss when a node failure is encountered. All shard replicas in the cluster are always synchronized for achieving consistency. For this reason, ODL implements the Raft consensus algorithm [15], which aims at achieving consensus in a distributed system. Figure 1 shows the shard and replica distribution.

C. Raft Consensus Algorithm

For each shard, the Raft algorithm provides one of the three states, namely *leader*, *candidate* or *follower*, to each cluster node. Raft starts by initiating an election among the cluster nodes. The node which gets the most *RequestVote* messages becomes the shard leader. By default, ODL uses one node as the leader of all shards, usually the one that starts first, as it is demonstrated in [16]. The leader is responsible for discovering inconsistencies among the cluster nodes related to the shard and replicating the proper data by sending *AppendEntries* messages. Moreover, empty *AppendEntries* messages are periodically initiated by Raft, having the role of heartbeat messages. The leader must respond with a *HeartbeatRsp* message within a timeout, otherwise followers should change their state to candidate and a new election round begins.

In case that a follower has new information, e.g. a new switch connected to it, it sends a *CreateTransaction* message to the leader to notify about the new data. After the transaction is done, the leader forwards the new data to the other followers to keep them updated. The role of leader is very crucial, since all shard updates are forwarded to it, and it is the leader's responsibility to replicate them to the rest of the followers. Visualization of a transaction from a follower to the leader and then the replication from the leader to all followers is shown in Figure 2. *The overall communication of the cluster seems to be centralized towards the leader*, which is also validated by our experimentation results. In what follows there will be a description of the process of building a network model representing the total control traffic produced in an ODL cluster.

IV. Network Model

An undirected connected graph $\mathcal{G} = (\mathcal{S}, \mathcal{L})$ is considered, representing the control plane of an SDN network, where $\mathcal{S}$ represents the set of SDN switches and $\mathcal{L}$ represents the set of network links. Let $S = |\mathcal{S}|$ and $L = |\mathcal{L}|$ be the number of the switches and the number of the links respectively. Without loss of generality, it is assumed that the routing algorithm used for the control traffic is the shortest path routing, the path connecting the couple of switches $s_1, s_2 \in \mathcal{S}$ is $p(s_1, s_2)$ and the number of links included in this path is $|p(s_1, s_2)|$. Finally, $\mathcal{C} \subseteq \mathcal{S}$ is the subset of switches where $C = |\mathcal{C}|$ controllers are placed and grouped in a cluster, while $c_l \in \mathcal{C}$ is the leader of this cluster. From now on, we may refer to $c \in \mathcal{C}$ as a controller or the switch hosting it, interchangeably. Let $c_s \in \mathcal{C}$ denote the controller that switch $s \in \mathcal{S}$ is assigned to. Vector $\mathbf{c} = (c_l \in \mathcal{C}, (c_s \in \mathcal{C} : s \in \mathcal{S}))$ describes a controller placement and switch assignment. The first vector's value indicates the leader controller. Each other vector's coordinate maps to a switch $s \in \mathcal{S}$ and the vector's value indicates the corresponding controller $c_s \in \mathcal{C}$.

The bandwidth usage for the controller-to-switch (*Ctr-Sw*) traffic from all network links is noted as

$$B^S \triangleq \sum_{s\in\mathcal{S}} \sum_{l\in p(s,c_s)} b^s = \sum_{s\in\mathcal{S}} w^s b^s, \tag{1}$$

where b^s is the bandwidth required for the *Ctr-Sw* traffic exchanged between switch s and controller c_s, while $w^s \triangleq |p(s, c_s)|$ is the length of the path connecting s to c_s. Similarly, the bandwidth usage for the controller-to-controller (*Ctr-Ctr*) traffic from all network links is noted as

$$B^C \triangleq \sum_{c\in\mathcal{C}-\{c_l\}} \sum_{l\in p(c,c_l)} b^c = \sum_{c\in\mathcal{C}-\{c_l\}} w^c\big(b^c_{\text{inv}} + b^c_{\text{topo}}\big), \tag{2}$$

where b^c is the bandwidth required for the *Ctr-Ctr* traffic exchanged between controller c and leader c_l, that is the sum of the b^c_{inv} traffic produced for the inventory shard and the b^c_{topo} traffic produced for the topology shard, while $w^c \triangleq |p(c, c_l)|$ is the length of the path connecting the two controllers c and c_l.

This work aims to formulate a mathematical problem that gives as solution the optimal controller placement $\mathcal{C}^*$, the optimal controller leader c_l^* and the optimal switch assignment, given in the vector $\mathbf{c}^* = (c_l^*, (c_s^* \in \mathcal{C}^* : s \in \mathcal{S}))$, which minimizes the total bandwidth required for the control traffic. This problem is expressed as

$$\mathbf{c}^* \triangleq \arg\min_{\mathbf{c}} \Big(\sum_{s\in\mathcal{S}} w^s b^s + \sum_{c\in\mathcal{C}-\{c_l\}} w^c\big(b^c_{\text{inv}} + b^c_{\text{topo}}\big)\Big). \tag{3}$$

At this point, we make the following remarks that are validated by our experimentation with ODL controllers. More details about our experimentation results will be provided later in Section V.

Remark 1: *The required bandwidth for the Ctr-Sw traffic exchanged between a switch and its controller is proportional to the number of flows existing in this switch.*

Let β^s denote the required bandwidth for each flow. If f^s is the number of flows existing in $s \in \mathcal{S}$, then $b^s = f^s\beta^s,\ \forall s \in \mathcal{S}$.

Remark 2: *The required bandwidth for the Ctr-Ctr traffic exchanged between a follower and the leader controller, due to the inventory shard, is proportional to the number of flows existing to the switches assigned to the follower. It is also proportional to the number of flows existing to the switches assigned to all other controllers (including also the leader).*

Let $\beta^{\text{int}}_{\text{inv}}$ denote the required bandwidth for the connection between a controller and the leader, for each flow configured by this controller, and $\beta^{\text{ext}}_{\text{inv}}$ denote the corresponding bandwidth, for each flow configured by all other controllers (including the leader). If $f^c \triangleq \sum_{s\in\mathcal{S}:c_s=c} f^s$ is the number of flows existing at the switches assigned to controller $c \in \mathcal{C}$ and $F \triangleq \sum_{s\in\mathcal{S}} f^s$ is the number of all existing flows, then $b^c_{\text{inv}} = f^c\beta^{\text{int}}_{\text{inv}} + (F - f^c)\beta^{\text{ext}}_{\text{inv}},\ \forall c \in \mathcal{C}$.

Remark 3: *The required bandwidth for the Ctr-Ctr traffic exchanged between a follower and the leader, due to the topology shard, is proportional to the number of switches assigned to the follower. It is also proportional to the number of switches assigned to all other controllers (including the leader).*

Let $\beta^{\text{int}}_{\text{topo}}$ denote the required bandwidth for the connection between a controller and the leader, for each switch assigned to this controller, and $\beta^{\text{ext}}_{\text{topo}}$ denote the corresponding bandwidth, for each switch assigned to all other controllers (including the leader). If $y^c \triangleq \sum_{s\in\mathcal{S}:c_s=c} 1$ is the number of switches assigned to controller $c \in \mathcal{C}$, then $b^c_{\text{topo}} = y^c\beta^{\text{int}}_{\text{topo}} + (S - y^c)\beta^{\text{ext}}_{\text{topo}},\ \forall c \in \mathcal{C}$.

Based on these Remarks, the problem presented in Equation 3 changes to

$$\begin{aligned}\mathbf{c}^* = \arg\min_{\mathbf{c}} \Big(&\sum_{s\in\mathcal{S}} w^s f^s \beta^s + \\ &\sum_{c\in\mathcal{C}} w^c\big(f^c\beta^{\text{int}}_{\text{inv}} + (F - f^c)\beta^{\text{ext}}_{\text{inv}}\big) + \\ &\sum_{c\in\mathcal{C}} w^c\big(y^c\beta^{\text{int}}_{\text{topo}} + (S - y^c)\beta^{\text{ext}}_{\text{topo}}\big)\Big) \Rightarrow \end{aligned} \tag{4}$$

$$\mathbf{c}^* = \arg\min_{\mathbf{c}} \Big(\sum_{s\in\mathcal{S}} w^s f^s \beta^s + \sum_{c\in\mathcal{C}} w^c \big(f^c \beta^c_{\text{inv}} + y^c \beta^c_{\text{topo}} \big) \Big). \tag{5}$$

Problem 5 is equivalent to Problem 4, since the quantities $F\beta^{\text{ext}}_{\text{inv}}$ and $S\beta^{\text{ext}}_{\text{topo}}$ are constants for a given network, $\beta^c_{\text{inv}} \triangleq \beta^{\text{int}}_{\text{inv}} - \beta^{\text{ext}}_{\text{inv}}$ and $\beta^c_{\text{topo}} \triangleq \beta^{\text{int}}_{\text{topo}} - \beta^{\text{ext}}_{\text{topo}}$.

As we did in our previous work [4], we make the simplifying assumption that all switches feature the same number of flows f, thus $f^c = y^c f$ and Problem 5 is equivalent to the following Integer Quadratic Programming (IQP) problem

$$\min_{\mathbf{x},\mathbf{y},\mathbf{z}} f\beta^s \sum_{i,j=1}^{S} w_{ij}x_{ij} + (f\beta^c_{\text{inv}} + \beta^c_{\text{topo}}) \sum_{i,j=1}^{S} w_{ij}y_i r_j \tag{6}$$

$$\begin{aligned}
\text{s.t.} \quad & \sum_{j=1}^{S} x_{ij} = 1, && \forall i = 1,\dots,S,\\
& \sum_{i=1}^{S} x_{ij} = y_j, && \forall j = 1,\dots,S,\\
& r_i \le y_i, && \forall i = 1,\dots,S,\\
& \sum_{i=1}^{S} r_i = 1, && \\
& x_{ij}, r_i \in \{0,1\}, y_i \in \mathbb{N} && \forall i,j = 1,\dots,S.
\end{aligned} \tag{7}$$

This problem has S^2 binary variables x_{ij}, S nonnegative integer variables y_i and S binary variables r_j. Indexes i and j take integer values from 1 to S, each value corresponding to a switch $s \in \mathcal{S}$. If $x_{ij} = 1$, then switch s_i has to be assigned to controller c_j (the controller collocated with switch s_j). Nonnegative integer y_i is the number y^{c_i} of switches assigned to controller c_i. Binary $r_i = 1$ if and only if the leader controller is placed at switch s_i. Finally, w_{ij} is the length of the path connecting switch s_i (or controller c_i) and controller c_j. The first and second sum terms of Equation 6 correspond to B^S and B^C respectively. The first term is the sum of the lengths of the paths connecting all switches to their controllers, multiplied by the bandwidth $f\beta^s$. The second term is the sum over all controller to leader pairs of the products between the length of the path connecting them and the number of switches assigned to the controller, scaled by the bandwidth $f\beta^c_{\text{inv}} + \beta^c_{\text{topo}}$.

V. Experimentation and Results

A. Experiment Setup

In order to test the performance of the communication among the cluster's nodes, we use the NITOS experimentation testbed [17]. A cluster of ODL controllers is used for the control plane, as well as the Mininet emulator [18] for extending the physical data plane of NITOS. Each controller of the cluster is running on a separate physical node in the testbed. A total of three controllers is used to set up the ODL cluster, each of them using the OpenFlow protocol and the features of *L2 Switch* project for the flow configuration. Finally, we use *iftop*, a free software command-line system monitor tool that produces a frequently updated list of network connections, to monitor the communication among the cluster's nodes.

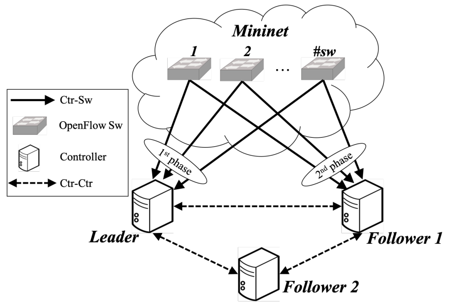

Fig. 3. *#sw* switches connected to either Leader (1st phase) or Follower 1 (2st phase), for measuring the *Ctr-Ctr* traffic of the topology shard.

TABLE I
TOPOLOGY SHARD TEST RESULTS: BANDWIDTH USAGE FOR VARIOUS NUMBER OF SWITCHES AT EACH CONTROLLER

#sw		***bandwidth usage (Mbps)***							
c_l	c_{f_1}	$c_l \to c_{f_1}$	$c_{f_1} \to c_l$	$c_l \leftrightarrow c_{f_1}$	avg_1	$c_l \to c_{f_2}$	$c_{f_2} \to c_l$	$c_l \leftrightarrow c_{f_2}$	avg_2
0	*0*	0.09	0.09	0.18	–	0.09	0.09	0.19	–
1	*0*	0.34	0.11	0.44	0.26	0.34	0.10	0.44	0.25
2	*0*	0.53	0.11	0.64	0.23	0.53	0.11	0.64	0.23
3	*0*	0.73	0.11	0.84	0.22	0.72	0.11	0.83	0.21
5	*0*	1.04	0.12	1.16	0.20	1.05	0.11	1.16	0.20
10	*0*	1.65	0.12	1.77	0.16	1.65	0.12	1.77	0.16
20	*0*	3.21	0.14	3.35	0.16	3.21	0.14	3.35	0.16
50	*0*	7.89	0.18	8.07	0.16	7.91	0.19	8.10	0.16
100	*0*	15.70	0.32	16.02	$\boxed{0.16}$	15.70	0.34	16.04	$\boxed{0.16}$
0	*1*	0.51	1.09	1.60	1.42	0.33	0.10	0.43	0.25
0	*2*	0.84	2.09	2.93	1.37	0.45	0.11	0.56	0.19
0	*3*	1.15	3.08	4.23	1.35	0.66	0.12	0.78	0.20
0	*5*	1.83	5.08	6.91	1.35	0.97	0.14	1.10	0.18
0	*10*	3.45	10.05	13.50	1.33	1.81	0.17	1.98	0.17
0	*20*	6.24	19.60	25.84	1.28	3.46	0.22	3.68	0.16
0	*50*	14.40	48.50	62.90	1.25	7.78	0.41	8.19	0.16
0	*100*	28.50	97.12	125.62	$\boxed{1.25}$	15.70	0.73	16.43	$\boxed{0.16}$

B. Topology Shard Tests

As described in Section III, topology shard holds data regarding the network topology. When a topology shard in a cluster member updates its data, the new data is replicated to the other members. For this purpose, we measure the changes in the used bandwidth for the synchronization of all topology shard replicas, when various numbers of switches are connected to each controller. The experiment topology is depicted in Figure 3.

Table I summarizes the results of these measurements. The leftmost column of the table represents the number of switches (*#sw*) connected to each controller, where c_l and c_{f_1} stand for Leader and Follower 1 controllers respectively. Follower 2 (c_{f_2}) never has any switches. The columns labeled as $c' \to c''$ give the bandwidth usage in Mbps of the traffic sent from c' to c'', where c' and c'' are either c_l, c_{f_1} or c_{f_2}, while columns $c_l \leftrightarrow c'$ give the total bandwidth usage for both $c_l \to c'$ and $c' \to c_l$. Finally, columns avg_1 and avg_2 give the average increase per added switch of the bandwidth usage, for both directions, for the communications between $c_l \leftrightarrow c_{f_1}$ and $c_l \leftrightarrow c_{f_2}$ respectively. For example, when 2 and 3 switches are controlled by c_l, the values in column avg_1 are $0.23 = (0.64-0.18)/2$ and $0.22 = (0.84-0.18)/3$ respectively.

For the first phase of our experimentation, in order to explore how the data is replicated by Leader to the followers, we increase the number of switches connected with Leader. The results of this experimentation phase are depicted in the upper half of Table I, including all rows having non-zero values below c_l. It was found that bandwidth usage has been increasing in communication from the side of Leader to the followers. No bandwidth increase was observed in the communication from the followers to Leader. This occurs because the followers do not hold any new data in their topology shard that needs to be replicated. The average bandwidth increase per added switch converges to 0.16 Mbps, either for the $c_l \leftrightarrow c_{f_1}$ or $c_l \leftrightarrow c_{f_2}$ communication, which is the marked value in the middle of both avg_1 and avg_2 columns.

For the second phase, the number of switches connected to Follower 1 is increased. The relative results are grouped in the bottom half of Table I, including all rows having non-zero values below c_{f_1}. In this case, Follower 1 communicates with Leader to inform it about the new topology data that have appeared in the network, and afterwards Leader proceeds with the replication of the new data to the outdated topology shards. Indeed, as we can see in Table I, there is a bandwidth increase in the communication channel from Follower 1 to Leader. Moreover, increase in bandwidth usage is present in the communication from Leader to Follower 1 and Follower 2 (due to data being replicated). In this case, the average bandwidth increase per added switch, as it is illustrated in the avg_1 and avg_2 columns, is 1.25 Mbps and 0.16 Mbps for the $c_l \leftrightarrow c_{f_1}$ and $c_l \leftrightarrow c_{f_2}$ communication channels respectively.

Based on these results, it is safe to assume that Remark 3 is confirmed with $\beta^{\text{int}}_{\text{topo}} \simeq 1.25$Mbps, $\beta^{\text{ext}}_{\text{topo}} \simeq 0.16$Mbps and $\beta^{\text{c}}_{\text{topo}} \simeq 1.25 - 0.16 = 1.09$Mbps. Finally, there are no columns depicting the communication between the two followers, since it has been constantly stable and negligible in terms of bandwidth usage (almost 0.055 Mbps), compared to the other communications. This is due to the shard replication and how it is performed. When a replica shard of a follower has new data, it communicates with the leader's initial shard and the second one is now responsible for the replication to the other followers. This means that there is no direct communication concerning the replication among the followers. The only interaction among them mainly regards periodic messages sent by the Akka, the gossip and Raft protocol, such as heartbeat messages, which require negligible bandwidth.

C. Inventory Shard Tests

In this series of experiments, the cluster behavior is tested when new information appears in the inventory shard. Inventory shard contains the flow rules that are installed to the cluster. Three switches are kept, each one connected to a single controller. Figure 4 depicts the topology of these experiments. The flows are installed by using the *ovs-ofctl* command line tool, which is generally used for monitoring and administering OpenFlow switches.

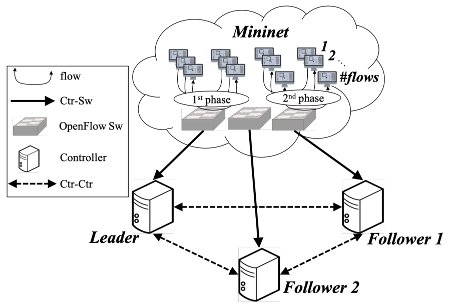

Fig. 4. *#flows* flows configured to the switch of either Leader (phase 1) or Follower 1 (phase 2), for measuring the *Ctr-Sw* and *Ctr-Ctr* traffic of the inventory shard.

TABLE II
INVENTORY SHARD TEST RESULTS: BANDWIDTH USAGE FOR VARIOUS NUMBER OF FLOWS AT EACH SWITCH

#flows		*bandwidth usage (Mbps)*							
c_l	c_{f_1}	$c_l \rightarrow c_{f_1}$	$c_{f_1} \rightarrow c_l$	$c_l \leftrightarrow c_{f_1}$	avg_1	$c_l \rightarrow c_{f_2}$	$c_{f_2} \rightarrow c_l$	$c_l \leftrightarrow c_{f_2}$	avg_2
0	*0*	1.00	1.10	2.10	–	1.00	1.10	2.10	–
30	*0*	1.04	1.10	2.14	0.001	1.02	1.10	2.12	0.001
60	*0*	1.08	1.10	2.18	0.001	1.08	1.10	2.18	0.001
120	*0*	1.19	1.10	2.29	0.002	1.19	1.10	2.29	0.002
240	*0*	1.45	1.10	2.55	$\boxed{0.002}$	1.45	1.10	2.55	$\boxed{0.002}$
0	*30*	1.09	1.25	2.34	0.003	1.09	1.10	2.19	0.003
0	*60*	1.21	1.35	2.56	0.004	1.09	1.10	2.19	0.002
0	*120*	1.46	1.60	3.06	0.004	1.20	1.10	2.30	0.002
0	*240*	1.85	2.03	3.88	$\boxed{0.004}$	1.39	1.10	2.49	$\boxed{0.002}$

At first, the number of flows installed in the switch connected to Leader are increased. Similarly to the topology shard case, when inserting new data in the leader inventory shard, an increase in the bandwidth usage is noticed from the leader's side to the follower nodes, while the communication from the followers' side remains stable. The bandwidth monitored during this experiment is demonstrated in Table II, where the first column now is labelled as *#flows* and shows the number of flows at the switch of each controller. The avg_1 and avg_2 columns show the average bandwidth increase per added flow at the switch controlled by c_{f_1} and c_{f_2} respectively. Both avg_1 and avg_2 converge to 0.002Mbps, as we can see at the bottom marked values of the upper half of this table.

For the second phase, flow rules are installed in the switch connected to Follower 1. Once again, the major role of Leader is obvious in the replication of the new data to the followers. Table II makes it evident that there is an increase in the bandwidth from Follower 1 to Leader, while Follower 1 is forwarding the new data to Leader. Leader immediately replicates the new information to the followers, which results in the increase of the bandwidth usage noted in columns $c_l \rightarrow c_{f_1}$ and $c_l \rightarrow c_{f_2}$. The bottom marked values of the avg_1 and avg_2 columns converge to 0.004Mbps and 0.002Mbps respectively. As follows, Remark 2 is confirmed, resulting in $\beta^{\text{int}}_{\text{inv}} \simeq 0.004$Mbps, $\beta^{\text{ext}}_{\text{inv}} \simeq 0.002$Mbps and $\beta^{\text{c}}_{\text{inv}} \simeq 0.004 - 0.002 = 0.002$Mbps.

D. Communication between Controller and Switch

In this series of experiments, we examine the *Ctr-Sw* traffic. The results of these experiments are given in Table III,

TABLE III

CTR-SW TEST RESULTS: BANDWIDTH USAGE FOR VARIOUS NUMBER OF FLOWS AT THE SWITCH.

#flows	*bandwidth usage (Kbps)*			
	sw $\rightarrow$ ctr	*ctr $\rightarrow$ sw*	*sw $\leftrightarrow$ ctr*	*avg*
0	18.10	1.88	19.98	–
10	19.80	1.92	21.72	0.17
20	21.53	1.95	23.48	0.18
50	26.73	2.04	28.77	0.18
100	36.00	2.19	38.19	0.18
200	53.90	2.50	56.40	$\boxed{0.18}$

where columns *sw* $\rightarrow$ *ctr* and *ctr* $\rightarrow$ *sw* refer to the traffic sent from the switch to the controller and the opposite. In this set of experiments, the first phase of the previous experiments is repeated, as it is depicted in Figure 4, increasing the number of flows installed in the switch connected to Leader and monitoring the *sw* $\rightarrow$ *ctr* and *ctr* $\rightarrow$ *sw* traffic. Obviously, the average increase per added flow, as it is presented in the *avg* column, is 0.18Kbps. Thus, Remark 1 is confirmed and $\beta^s \simeq 0.18$Kbps.

At this point, it is worth mentioning that β^s is the traffic produced for each flow entry because of the statistics kept by the controller for this entry. This is the long term average control traffic produced for each flow entry, that may be much lower than the traffic peak produced when the flow entry is configured in the switch. Moreover, the switches can be connected to multiple controllers, belonging to the cluster, increasing in this way their resilience to controller failures. In this case, for each switch, one of the controllers suggested by the solution of our problem is the *master*, while the rest ones are called *slaves*. Our experimentation results show that the control traffic exchanged between the switch and the slave controllers is much lower, and negligible compared to the other control traffic.

E. Control Traffic and Network Resilience

The network model of Section IV gives the controller placement for minimum control traffic. The model parameters β^s, $\beta^{\text{c}}_{\text{inv}}$ and $\beta^{\text{c}}_{\text{topo}}$ are given by the measurements that were taken in the previous experiments. As we have already shown in our previous work [4], *Ctr-Sw* traffic B^S decreases with the utilization of a distributed set of multiple controllers, while *Ctr-Ctr* traffic B^C decreases with the utilization of few controllers. In contrast to our previous work, where only B^S increases with f, now both B^S and B^C increase with f, and more specifically, B^C is faster-growing because $\beta^{\text{c}}_{\text{inv}} > \beta^s$. As follows, in contrast to our previous results for the Kandoo controllers, where the optimal number of controllers depends on f, now the minimum control traffic is always achieved when one single controller is used. However, operating a network with one controller implies zero resilience in controller failures. A minimum of three ODL controllers is required in order to create a fault-tolerant controller cluster, or even more controllers for increased network resilience.

VI. CONCLUSIONS

In this paper, we discuss the effects of having an ODL cluster set up in a network, instead of a single controller. First of all, we review the concept behind the distribution of the data among the cluster's controllers.Secondly, a network model is built that describes the total control traffic produced in an ODL cluster, based on the experimentation results. During the experiments, we measure the overhead that is introduced to the network's bandwidth usage due to the intercommunication of the cluster controllers. We monitor the changes in the bandwidth usage inside the cluster as a result of installing new flows as well as connecting new switches to the controllers. Concluding, the total control traffic is minimum when few controllers are used, as few as possible, given the network resilience requirement.

ACKNOWLEDGMENT

The research leading to these results has received funding by the European Horizon 2020 Programme for research, technological development and demonstration under Grant Agreement Number 857201 (H2020 5G-VICTORI).

REFERENCES

[1] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The Road to SDN: An Intellectual History of Programmable Networks. *SIGCOMM Comput. Commun. Rev.*, 44(2):87–98, April 2014.

[2] J. Medved, R. Varga, A. Tkacik, and K. Gray. OpenDaylight: Towards a Model-Driven SDN Controller architecture. In *Proc. IEEE WoWMoM*, 2014.

[3] B. Heller, R. Sherwood, and N. McKeown. The Controller Placement Problem. In *Proc. HotSDN*, 2012.

[4] P. Flegkas K. Choumas, D. Giatsios and T. Korakis. The SDN Control Plane Challenge for Minimum Control traffic: Distributed or Centralized? In *Proc. CCNC*, 2019.

[5] T. M. C. Nguyen, D. B. Hoang, and Z. Chaczko. Can SDN Technology Be Transported to Software-Defined WSN/IoT? In *iThings-GreenCom-CPSCom-SmartData*, 2016.

[6] B. Yu, G. Yang, and C. Yoo. Comprehensive Prediction Models of Control Traffic for SDN Controllers. In *Proc. IEEE NetSoft*, 2018.

[7] Andre Rizki Dewo Nugraha, Ridha Negara, and Danu Dwi Sanjoyo. High Availability Performance on OpenDayLight SDN Controller Platform (OSCP) Clustering and OpenDayLight with Heartbeat-Distributed Replicated Block Device (DRBD). *Journal Infotel*, 10:149, 08 2018.

[8] T. Zhang, A. Bianco, and P. Giaccone. The role of inter-controller traffic in SDN controllers placement. In *Proc. IEEE Conference on NFV and SDN (NFV-SDN)*, 2016.

[9] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba. Dynamic Controller Provisioning in Software Defined Networks. In *Proc. International Conference on Network and Service Management (CNSM)*, 2013.

[10] S. Lange, S. Gebert, J. Spoerhase, P. Rygielski, T. Zinner, S. Kounev, and P. Tran-Gia. Specialized Heuristics for the Controller Placement Problem in Large Scale SDN Networks. In *Proc. International Teletraffic Congress (ITC)*, 2015.

[11] Zhiyang Su and Mounir Hamdi. MDCP: Measurement-aware distributed controller placement for software defined networks. In *Proc. IEEE ICPADS*, 2015.

[12] Q. Qin, K. Poularakis, G. Iosifidis, and L. Tassiulas. SDN Controller Placement at the Edge: Optimizing Delay and Overheads. In *Proc. IEEE INFOCOM*, 2018.

[13] S. Hassas Yeganeh and Y. Ganjali. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In *Proc. HotSDN*, 2012.

[14] Akka Cluster. https://doc.akka.io/docs/akka/2.5/common/cluster.html.

[15] Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm. In *Proc. USENIX Annual Technical Conference*, 2014.

[16] T. Kim, S. Choi, J. Myung, and C. Lim. Load balancing on distributed datastore in opendaylight SDN controller cluster. In *Proc. IEEE NetSoft*, 2017.

[17] Network Implementation Testbed using Open Source platforms (NITOS). https://nitlab.inf.uth.gr/NITlab/nitos.

[18] Mininet: An Instant Virtual Network on your Laptop. http://mininet.org.