Dynamic Switch Migration in Distributed Software-Defined Networks to Achieve Controller Load Balance

Yang Xu, *Member, IEEE*, Marco Cello, I-Chih Wang, Anwar Walid, *Fellow, IEEE*, Gordon Wilfong, Charles H.-P. Wen, *Member, IEEE*, Mario Marchese, *Senior Member, IEEE*, and H. Jonathan Chao, *Fellow, IEEE*

***Abstract*— Multiple distributed controllers have been used in software-defined networks (SDNs) to improve scalability and reliability, where each controller manages one static partition of the network. In this paper, we show that dynamic mapping between switches and controllers can improve efficiency in managing traffic load variations. In particular, we propose balanced controller (BalCon) and BalConPlus, two SDN switch migration schemes to achieve load balance among SDN controllers with small migration cost. BalCon is suitable for the scenarios where the network does not require a serial processing of switch requests. For other scenarios, BalConPlus is more suitable, as it is immune to the switch migration blackout and does not cause any service disruption. Simulations demonstrate that BalCon and BalConPlus significantly reduce the load imbalance among SDN controllers by migrating only a small number of switches with low computation overhead. We also build a prototype testbed based on the open-source SDN framework RYU to verify the practicality and effectiveness of BalCon and BalConPlus. Experiment confirms the results of the simulations. It also shows that BalConPlus is immune to switch migration blackout, an adverse effect in the baseline BalCon.**

***Index Terms*— Software-defined networking, distributed controllers, load balancing, switch migration.**

I. Introduction

SOFTWARE Defined Networking (SDN) is a promising networking technology that enables network innovation and provides network operators more control of the network infrastructure. It decouples the control plane logic from the data plane by moving the networking control functions from the forwarding devices (e.g., switches/routers) to the logically centralized controller, so that the network functions can be implemented by software. However, as the number of switches[1] in an SDN increases, the centralized controller may fail to process all the requests coming from the switches. Moreover, because of the single point of failure, malfunction of the SDN controller can bring down the whole network. Recent works have proposed using multiple physically distributed SDN controllers to improve system scalability and reliability, while preserving the simplicity of a logically centralized system [2]–[4].

One of the problems of existing multicontroller architectures is their static mapping between SDN switches and controllers which makes the control plane unable to adapt to traffic variation. As suggested in [5], real networks may exhibit huge variations in both temporal dimensions (traffic varies at different time of the day or even in a shorter time scale) and spatial dimensions (traffic varies at different locations of the network) [6]. If the SDN switch-controller mapping is static, the huge variations may result in imbalance among the controllers, i.e., some overloaded and some underutilized. An overloaded controller will response to switch requests with increased latency, deteriorating the quality of user experience. Therefore, dynamic mapping between the switches and the controllers can overcome imbalance and reduce the connection setup latency, by migrating some switches from an overloaded controller to other controllers with light load. However, the dynamic switch migration does incur some overhead due to the four-phase switch migration protocol [7] that causes service interruption (detailed in Section V).

Although some works have been proposed to address the switch migration issue among multiple controllers, there is a lack of systematic method to quantitatively identify which switches to be migrated for better controller load balance.

The contributions of this paper in theoretical, algorithmic and implementation aspects are summarized below:

- We show that dynamic mapping between SDN switches and controllers provides system elasticity and efficiency under varied traffic loads. Migration of switches among the controllers to achieve controller load balancing has

Manuscript received May 5, 2018; revised January 07, 2019; accepted January 11, 2019. Date of publication February 5, 2019; date of current version February 14, 2019. This paper was presented in part at the IEEE International Conference on Cloud Engineering 2017 [1].

Y. Xu and H. J. Chao are with the Department of Electrical and Computer Engineering, New York University, New York City, NY 11201 USA (e-mail: yang@nyu.edu; chao@nyu.edu).

M. Cello is with Rulex Inc., 16122 Genoa, Italy.

I.-C. Wang and C. H.-P. Wen are with the Department of Electrical and Computer Engineering, National Chiao Tung University, Hsinchu 30010, Taiwan.

A. Walid and G. Wilfong are with Nokia Bell Labs, Murray Hill, NJ 07974 USA.

M. Marchese is with the Department of Naval, Electrical, Electronic and Telecommunications Engineering, University of Genoa, 16126 Genoa, Italy.

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/JSAC.2019.2894237

[1]Unless specifically noted, we only consider switches as the forwarding devices in this paper. The conclusion made in the paper can be simply extended to scenarios where other devices (e.g., firewalls) exist.

0733-8716 © 2019 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

been modeled as an optimization problem and shown that it is NP-complete;
- Since the computational complexity of the optimal solution of the model is prohibitively high, we propose BalCon, a heuristic solution that is able to achieve load balancing among the controllers through switch migration;
- We analyze the overhead incurred in switch migration and discuss a service disruption problem during the migration, called switch migration blackout. BalConPlus, an improved version of BalCon, is proposed to eliminate the blackout by steering new arriving flows away from the switches that are being migrated. BalConPlus requires only minor changes from the baseline BalCon without incurring much additional implementation complexity;
- We implement BalCon and BalConPlus in Matlab. Simulation results show that load imbalance among controllers (expressed as variance of the load) is reduced by 40% and the load of the congested controller is reduced by 19% with a relatively low number of SDN switches migrated;
- We prototype a testbed based on RYU (a popular SDN controller written in python [8]) to verify the practicality and effectiveness of BalCon and BalConPlus. Through experiment results, we demonstrate that BalConPlus is immune to the blackout;
- The operation of BalCon and BalConPlus needs to know some system parameters in advance (e.g., the cost for computing a path and the cost for installing a flow entry to a switch) to predict migration results. We propose an automated parameter measurement and calculation framework to run BalCon and BalConPlus in our prototyped testbed.

BalCon and BalConPlus can be used in large scale data center networks, carrier networks, or enterprise networks that have a large number of network devices (such as switches, firewalls, and Intrusion Detection Systems (IDS)) controlled by multiple controllers to achieve controller load balance. In particular, BalCon can be used when the network does not require serial processing of switch requests, since such a network will not have the switch migration blackout problem (details are discussed in Section V). For networks that require serial processing of switch requests, BalConPlus is more suitable as it can steer new flows away from the switches that are being migrated to avoid service disruption.

The rest of the paper is organized as follows: Section II presents the motivations of our work. Section III presents the system model. Section IV presents the design and the details of BalCon. Section V discusses the switch migration blackout problem and presents BalConPlus. In Section VI we evaluate the performance of BalCon and BalConPlus using Matlab simulations. Section VII presents a prototype testbed with RYU controller and detailed experiment results. Section VIII reviews prior related works. Conclusions are in Section IX.

II. Motivations

An SDN network is composed of SDN switches and a logically centralized SDN controller. Each SDN switch processes and delivers packets according to rules stored in its flow table (forwarding state), whereas the SDN controller configures the forwarding state of each switch using a standard protocol (e.g., OpenFlow [9]). Traffic rules, representing the forwarding state, are installed in SDN switches when a new flow arrives.[2]

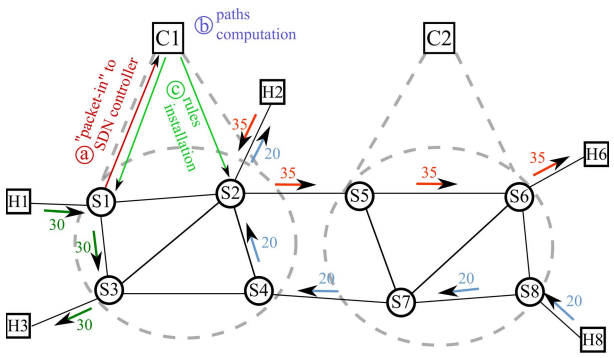


Fig. 1. SDN controller load imbalance scenario.

In order to overcome the scalability issues of a single centralized controller, several approaches have been proposed in the literature. One of the most effective methods is the use of distributed controllers. Existing distributed controller solutions still suffer from the static mapping between SDN switches and controllers, limiting the capability of dynamic load adaptation.

Let's briefly explain the reactive mode behavior in SDN using an example in Figure 1, where the network is divided into two domains and each of them is controlled by a controller. Assume that a new flow $f1$ generated by host H_1 arrives at switch S_1. S_1 doesn't have any rule associated with the flow and generates a "packet-in"[3] to controller C_1 (i.e., the first red arrow in step ⓐ). C_1 then computes the route (i.e., step ⓑ in blue) and installs the flow rules on SDN switches controlled by itself (i.e., the green arrows to S_1 and S_2 in step ⓒ by assuming that the forwarding path of the flow is $S1 \rightarrow S2 \rightarrow$ the second domain). When the flow arrives at S_5, the switch doesn't have any rule associated with the flow and, consequently, sends a packet-in request to C_2 that computes the flow's path and installs the flow rules on S_5 and S_6 (assuming that the forwarding path of the flow in the second domain is $S_5 \rightarrow S_6$).

Suppose now that due to the traffic variations, a large number of new flows arrive to the network and the current traffic pattern is depicted in Figure 1. In particular:

- host H_1 generates 30 new $flows/second$ to H_3, which are routed through $S_1 \rightarrow S_3 \rightarrow H_3$ (green arrows);
- host H_2 generates 35 new $flows/second$ to H_6, which are routed through $S_2 \rightarrow S_5 \rightarrow S_6 \rightarrow H_6$ (red arrows);

[2]This method is known as "reactive" mode. A less-used and less-effective method is "proactive" mode in which the controller installs rules beforehand.

[3]When a packet does not match any of the existing rules inside an SDN switch, the default policy is to send a copy of that packet up to the controller. This "packet sent to the controller" message is called, in OpenFlow-parlance, a packet-in [10].

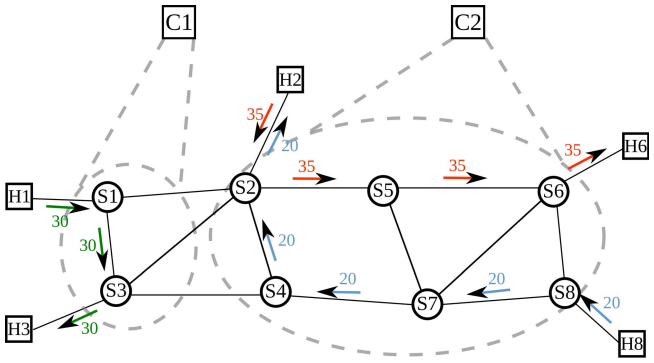


Fig. 2. Controller load balance is improved after switch migrations.

- host H_8 generates 20 new $flows/second$ to H_2, which are routed through $S_8 \rightarrow S_7 \rightarrow S_4 \rightarrow S_2 \rightarrow H_2$ (blue arrows).

At this point, we want to ask what are the computational burdens of controllers C_1 and C_2 due to the instantiation of the new flows. Suppose that the path computation for a single flow requires α units of load, whereas the rules installation of a single flow in a single switch requires β units of load. At controller C_1:

- the green flows generate 30α units for path computation and $(30+30)\beta$ units for rules installation at S_1 and S_3;
- the red flows generate 35α units for path computation and 35β units for rules installation at S_2;
- the blue flows generate 20α units for path computation and $(20+20)\beta$ units for rules installation at S_4 and S_2.

At controller C_2:

- the red flows generate 35α units for path computation and $(35+35)\beta$ units for rules installation at S_5 and S_6;
- the blue flows generate 20α units for path computation and $(20+20)\beta$ units for rules installation at S_8 and S_7.

If we assume $\alpha = 1$ and $\beta = 0.1$,[4] we obtain:

$$L_{C_1} = (30+35+20)\alpha + (30+55+30+20)\beta = 98.5\, units/s.$$

$$L_{C_2} = (35+20)\alpha + (35+35+20+20)\beta = 66\, units/s.$$

In the aforementioned example, the load between controllers C_1 and C_2 is highly unbalanced. If we have the capability to dynamically shrink or enlarge the SDN domains or partitions through a proper switch migration, we can obtain the new mapping between controllers and switches in Figure 2. S_2 and S_4 are now part of the second domain and controlled by C_2. The new controllers' load are now:

$$L_{C_1} = (30)\alpha + (30+30)\beta = 36\, units/s.$$

$$L_{C_2} = (35+20)\alpha + (55+20+35+35+20+20)\beta = 73.5\, units/s.$$

Therefore, we obtained a significant reduction of the controller load at C_1 (63%) compared to a relatively small increase of the controller load at C_2 (11%).

[4]Here we consider the path computation load, ten times larger than the rules installation load. In Section VII-C, an automated parameter measurement and calculation mechanism is presented.

As explained in [5], using real measurements of a production datacenter, Benson *et al.* [6] found that there are 1-2 orders of magnitude difference between peak and median flow arrival rates at the switch: peak flow arrival rate can be up to 300M/s with the median rate between 1.5M/s and 10M/s. Assuming that each controller can manage up to 2M/s as flow arrival rate, it requires only 1-5 controllers to process the median load, but 150 for peak load. If we use static mapping, each controller needs to have the capacity to process the peak flow arrival (worst-case situation). If we have a dynamic mapping, the capacity of each controller can be lowered, since the peak of different partitions (domains) usually will not occur at the same time due to multiplexing and sharing effect.

Motivated by the above observations, we seek to answer our key question: how to dynamically select and migrate switches from the domain of one controller to another to balance controller load? The answer will largely depend on the complexity and cost of the switch migration process.

We first develop optimal controller load balancing (OCLB) problem in SDN multicontroller scenarios, and prove, however, that it is an NP-Complete problem. We then model the OCLB problem as a graph partitioning problem and develop BalCon and BalConPlus: two effective algorithms for load adaptation among SDN controllers through SDN switch migrations.

III. Modeling of Controller Load Balancing Problem

A. System Model

The objective of this section is to find an appropriate model that takes into account the flow arrival dynamics at each SDN switch and relate them to the computational load at each SDN controller. We then formalize the Controller Load Balancing (CLB) problem into an optimization one.

An SDN scenario is composed of a set $\mathcal{S}$ of SDN switches, $S_i \in \mathcal{S}$, managed by a set $\mathcal{C}$ of SDN controllers, $C_m \in \mathcal{C}$. In accordance with prior works, we cannot assume predictable traffic or well-known traffic patterns among the SDN switches, but we can monitor the traffic load during runtime. Therefore, we indicate with f_{o,S_i} the current arrival rate of new flows at SDN switch S_i from outside the SDN network, with $f_{S_i,o}$ the current arrival rate of new flows that leave the SDN network from switch S_i, whereas with f_{S_i,S_j} we indicate the current arrival rate of new flows traversing the link between the two connected SDN switches S_i and S_j. In other words, f_{S_i,S_j} represents the current arrival rate of new flows at the SDN switch S_j coming from SDN switch S_i. Referring to Figure 1 we have: $f_{o,S_1} = 30$, $f_{o,S_2} = 35$, $f_{o,S_8} = 20$, $f_{S_3,o} = 30$, $f_{S_2,o} = 20$, $f_{S_6,o} = 35$, $f_{S_1,S_3} = 30$, $f_{S_2,S_5} = 35$, $f_{S_4,S_2} = 20$, $f_{S_5,S_6} = 35$, $f_{S_7,S_4} = 20$, $f_{S_8,S_7} = 20$.

As shown before, the load L_{C_m} at controller C_m is composed of three main components: the path computation load of new flows arriving from outside the SDN network (e.g., green arrow $H_1 \rightarrow S_1$ and red arrow $H_2 \rightarrow S_2$ in Figure 1); the path computation load of the flows arriving from another SDN domains (e.g., blue arrow $S_7 \rightarrow S_4$ in Figure 1); the rule

installation load at each switch controlled by C_m for all flows traversing the domain controlled by C_m.

Definition 1: - Path Computation Load for External Flows - When a batch of flows arrive at S_i from outside the network with a rate of f_{o,S_i}, they generate a computational load due to the *path computation* at the SDN controller of S_i equal to:

$$\mathcal{K}(f_{o,S_i}) \tag{1}$$

Definition 2: - Path Computation Load of flows from Other SDN Domains - When a batch of flows arrive at S_i from S_j, a switch controlled by another SDN controller, with a rate of f_{S_j,S_i}, they generate a computational load due to the *path computation* at the SDN controller of S_i equal to:

$$\mathcal{K}(f_{S_j,S_i}) \tag{2}$$

The computational load at SDN controller necessary to perform path computation is dependent on the arrival rate of flows through a function $\mathcal{K}$. The definition of the function $\mathcal{K}$ is not the objective of this work.

Definition 3: - Rules Installation Load - The computational load at the controller due to rules installation in switch S_i is equal to:

$$\sum_{S_j \in \mathcal{S}} \mathcal{G}(f_{S_i,S_j}) + \mathcal{G}(f_{S_i,o}) \tag{3}$$

Equation 3 expresses the amount of flows that are traversing S_i going to other switches or out of the SDN network. Function $\mathcal{G}$ maps the the flow arrival rate at S_i to the computational load at the SDN controller needed for rules installation.

Definition 4: The set of SDN switches controlled by SDN controller C_m is denoted by $\mathcal{P}_m$.

The set $\mathcal{S}$ is then partitioned in $|\mathcal{C}|$-partitions, with $\mathcal{P}_m \subset \mathcal{S}$, $\mathcal{P}_m \cap \mathcal{P}_n = \emptyset,\ n \neq m$.

Definition 5: The overall computational load at SDN Controller C_m (L_{C_m}) is computed as:

$$L_{C_m} \triangleq \sum_{S_i \in \mathcal{P}_m} \mathcal{K}(f_{o,S_i}) + \sum_{\substack{S_j \notin \mathcal{P}_m \\ S_i \in \mathcal{P}_m}} \mathcal{K}(f_{S_j,S_i}) + \sum_{\substack{S_i \in \mathcal{P}_m \\ S_j \in \mathcal{S}}} \mathcal{G}(f_{S_i,S_j}) + \sum_{S_i \in \mathcal{P}_m} \mathcal{G}(f_{S_i,o}) \tag{4}$$

Overloading the SDN controller reduces its responsiveness and causes a performance degradation since the flows will experience an unexpected latency.

Definition 6: An SDN controller is overloaded or congested when its overall computational load is:

$$L_{C_m} > \boldsymbol{L} \tag{5}$$

where $\boldsymbol{L}$ that indicates the maximum computational load tolerated at each SDN controller.

When congestion occurs a migration procedure is needed to reduce overload. In particular, starting from a partition $(\mathcal{P}_1, \ldots, \mathcal{P}_{|\mathcal{C}|})$ for which at least one controller, C_m, the condition $L_{C_m} > \boldsymbol{L}$ holds, we need to find a new partition $(\mathcal{P}'_1, \ldots \mathcal{P}'_{|\mathcal{C}|})$ such that the SDN controller load $L_{C_m} \leq \boldsymbol{L},\ C_m \in \mathcal{C}$.

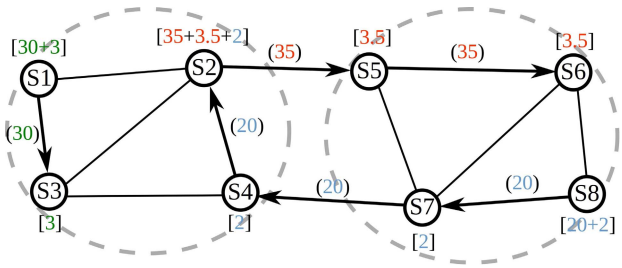


Fig. 3. The SDN network scenario of Figure 1 as graph partitioning problem.

The CLB problem can be expressed as a mathematical optimization problem which we call the Optimal CLB (OCLB) problem, and it is defined as follows:

Definition 7: - OCLB Problem.

$$\min_{\mathcal{P}_1,\ldots,\mathcal{P}_{|\mathcal{C}|}} \max_{C_m \in \mathcal{C}} L_{C_m}; \quad \text{subject to } \mathcal{P}_m \cap \mathcal{P}_m = \emptyset,\ m \neq n; \quad \bigcup \mathcal{P}_m = \mathcal{S}. \tag{6}$$

B. OCLB as Graph Partitioning Problem

The OCLB problem can be expressed as a partitioning problem on a graph and the computation of L_{C_m} can be induced directly on the graph. In particular, we represent the SDN network as a directed edge-weighted and vertex-weighted graph $G(\mathcal{S}, \mathcal{E})$ in which SDN switches are the vertices with weights $l(S_i), S_i \in \mathcal{S}$ and edges $\mathcal{E} = \{(S_i, S_j) : S_i, S_j \in \mathcal{S}, l(S_i, S_j) > 0\}$, are the connections among SDN switches. $l(S_i, S_j)$ is the edge weights of (S_i, S_j). That is

$$l(S_i) = \mathcal{K}(f_{o,S_i}) + \sum_{S_j \in \mathcal{S}} \mathcal{G}(f_{S_i,S_j}) + \mathcal{G}(f_{S_i,o}); \tag{7}$$

$$l(S_j, S_i) = \mathcal{K}(f_{S_j,S_i}). \tag{8}$$

The overall load at C_m, denoted by L_{C_m}, is then the sum of the weights of the vertices belonging to its partition plus the sum of weights of the edges directed to the partition of C_m. Specifically:

$$L_{C_m} = \sum_{S_i \in \mathcal{P}_m} l(S_i) + \sum_{\substack{S_j \notin \mathcal{P}_m \\ S_i \in \mathcal{P}_m}} l(S_j, S_i). \tag{9}$$

Note that Equation 9 is just another expression for Equation 4.

Figure 3 is a representation of Figure 1 as a graph partitioning problem. For example, the vertex weight of S_1 represent the computational load "brought" by S_1 to C_1. In particular $l(S_1) = 33$, which is the sum of $\mathcal{K}(f_{o,S_1}) = 30$ (30 $flows/s$) and the rule installation for the flows going to S_3 $\mathcal{G}(f_{S_1,S_3}) = 3$.[5]

[5]For simplicity here we consider the functions $\mathcal{K}$ and $\mathcal{G}$ as linear functions of the rate: $\mathcal{K}(rate) = rate$, $\mathcal{G}(rate) = rate/10$.

Refering to the same figure we get:

$$\begin{aligned} L_{C_1} &= l(S_1) + l(S_2) + l(S_3) + l(S_4) + l(S_7, S_4) \\ &= 33 + 40.5 + 3 + 2 + 20 = 98.5\ units/s. \\ L_{C_2} &= l(S_5) + l(S_6) + l(S_7) + l(S_8) + l(S_2, S_5) \\ &= 3.5 + 3.5 + 2 + 22 + 35 = 66\ units/s. \end{aligned}$$

C. NP-Completeness Proof

We have proved that the OCLB problem is an NP complete problem. Details of the proof are omitted due to space limitations. The complete proof can be found here: **NP-completeness Proof**.[6]

IV. BalCon Algorithm

An optimal SDN switch migration is impractical due to its computational complexity (i.e., OCLB problem is NP-complete) and could lead to undesirable excessive switch migrations. A more practical approach should involve incremental adjustment of the switch partitions, i.e., only a small number of SDN switches are migrated.

In this section, we propose *Balanced Controllers* (BalCon), an algorithmic solution designed to tackle and reduce the load imbalance among SDN controllers through a proper SDN switch migration. The key observation behind *BalCon* is that an effective switch migration can be based on analysis of the communication patterns of the SDN switches. The switch migration should be at the granularity of *clusters*: switches with strong connections[7] should always be assigned to the same controller.

BalCon is an heuristic algorithm which operates during the network runtime and is able to detect and solve congestion at the SDN controllers through proper SDN switch migrations. BalCon can be implemented as a northbound application of the SDN controller (more details are available in Section VII). BalCon consists of three phases, as summarized below:

1) *Monitoring and congestion detection*: During the network operation, BalCon continuously monitors the congestion level at each SDN controller. An SDN controller, C_m, is considered congested when L_{C_m} reaches a predetermined threshold. BalCon then computes a list of SDN switches that may be migrated. The list is ordered by a priority computed using a pre-determined metric. For example, the SDN switches that are observing a rapid increase of new flows could get high priority since they could rapidly overload the SDN controller with packet-ins.
2) *Clustering and migration evaluation*: Starting from the SDN switches in the priority list, BalCon analyzes the traffic pattern among SDN switches to find clusters of heavily connected switches (discussed below).
3) *Cluster migration*: When the best cluster is found and the migration is evaluated, the SDN switches belonging to the cluster are migrated to the new SDN controller.

[6]https://marcocello.github.io/pubs/IC2E2017-BalCon-Proof.pdf

[7]We consider the relative density of the cluster [11].

Algorithm 1 BalCon

Input: Edge- and node-weighted graphs $G(\mathcal{S}, \mathcal{E})$, congested SDN controller C_m;

1 $\mathcal{P}_m$: set of SDN switches controlled by the congested SDN controller C_m;
2 $\mathcal{A} = ComputeStartingSwitchesList(\mathcal{C}_m)$
3 **foreach** $S_i \in \mathcal{A}$ **do**
4 $\quad \mathcal{T} = \{S_i\}$;
5 $\quad alternatives = alternatives \cup ComputeMigrationAlternatives(\mathcal{T})$;
6 $\quad$ **while** *1* **do**
7 $\quad\quad new\mathcal{T} = IncreaseCluster(\mathcal{T})$;
8 $\quad\quad$ **if** $size(\mathcal{T}) > mcs \| new\mathcal{T} = \mathcal{T}$ **then**
9 $\quad\quad\quad$ break;
10 $\quad\quad \mathcal{T} = new\mathcal{T}$;
11 $\quad\quad alternatives = alternatives \cup ComputeMigrationAlternatives(\mathcal{T})$;
12 $[\mathcal{T}^0, \text{Target SDN controller}^o] \leftarrow EvaluateMigrationAlternatives(alternatives)$;

The algorithm we propose is substantially based on the iteration of three functions: $IncreaseCluster$ in which the cluster is expanded; $ComputeMigrationAlternatives$ in which the migrations to different target SDN controllers of the selected cluster are evaluated (producing the "migration alternatives" or simply called "alternatives" in the sequel); $Evaluate\text{-}BestMigrationAlternative$ in which given a list of alternatives, the best alternative (based on some criteria described in the following) is computed. The algorithm is shown in Algorithm 1.

From the set $\mathcal{P}_m$ (SDN switches controlled by the congested SDN controller C_m), the algorithm extracts a subset list $\mathcal{A}$ (*StartingSwitch List*) that contains the starting nodes used for the cluster construction (line 2). $\mathcal{A}$ could be computed, for example, by looking for the SDN switches that have a significant increase in flow arrival rate. The first SDN switch belonging to $\mathcal{A}$ is selected and inserted in the empty cluster $\mathcal{T}$ (Line 4). The migration alternatives of the SDN switches belonging to $\mathcal{T}$ are computed through $ComputeMigrationAlternatives$. The algorithm, subsequently, executes a while loop in which the cluster is continuously enlarged with the $IncreaseCluster$ function and evaluated with the function $ComputeMigration\text{-}Alternatives$. The algorithm halts when one of the two stop conditions are met: the cluster reaches a predetermined size mcs (max cluster size), i.e., $size(\mathcal{T}) > mcs$, or the increased cluster is equal to the old one ($new\mathcal{T} = \mathcal{T}$). The next switch in $\mathcal{A}$ is then selected and inserted in an empty cluster $\mathcal{T}$. When the $mssls$ (max starting switch list size) is reached, all the migration alternatives are evaluated using the $AlternativeEvaluation$ function. The best alternative composed by $\mathcal{T}^0$ (the cluster) and the target SDN controller (the controller that will receive $\mathcal{T}^0$) are chosen and the

migration can occur. In the following we will give a detailed explanation of the aforementioned functions.

1 **function** $ComputeMigrationAlternatives$ ($\mathcal{T}$);
2 **foreach** *SDN controller* C_i **do**
3 "virtual" migrate cluster $\mathcal{T}$ to SDN controller C_i;
4 **if** $L_{C_i} < \boldsymbol{L}$ **then**
5 compute $L_{C_n}, \forall C_n \in \mathcal{C}$;
6 compute $migrationSize$ for this new configuration;
7 save them in $lastAlternatives$
8 return $lastAlternatives$

$ComputeMigrationAlternatives$ "virtual" migrates cluster $\mathcal{T}$ to different SDN controller destinations. For each controller, it computes the controller load and the migration size. Table I shows a possible output of $ComputeMigration$-$Alternatives$ routine in a scenario with 60 switches and 5 controllers, when $\mathcal{T} = \{S_1,S_2,S_{56}\}$. For SDN controller C_i, the function migrates $\mathcal{T}$ to SDN controller C_i (Line 3), computing the new computational load at each SDN controller (Line 5) and the migration cost $migrationSize$ (Line 6) defined as the number of switches that need to be migrated.

1 **function** $IncreaseCluster(\mathcal{T})$;
2 $neighbors\mathcal{T} = ComputeNeighborsOfCluster(\mathcal{T})$;
3 **foreach** $S_i \in neighbors\mathcal{T}$ **do**
4 $new\mathcal{T} = \mathcal{T} \cup S_i$;
5 $savedDensities$ = [$savedDensities$; S_i, $Density(new\mathcal{T})$];
6 $S_i^o = \text{argmax}_{savedDensities} Density(new\mathcal{T})$;
7 return $\mathcal{T} \cup S_i^o$;

Starting from the cluster $\mathcal{T}$, the function constructs the set $neighbors\mathcal{T}$ composed of all SDN switches that are neighbors to $\mathcal{T}$. An SDN switch S_i is a neighbor of $\mathcal{T}$ if $\exists S_j \in \mathcal{T}$: $l(S_i, S_j) \neq 0, l(S_j, S_i) \neq 0$. The function then selects the neighbor that maximizes the relative density $Density$ [11] of the newly created cluster. The rationale behind this relative density maximization is that only SDN switches with strong connections should be grouped into the same cluster. The cluster will then be migrated between controllers as a whole to reduce the overall computation complexity of controllers.

Definition 8: Relative density is the ratio of the internal degree to the number of incident edges, i.e.,

$$Density(\mathcal{T}) = \frac{\sum\limits_{\substack{S_i,S_j \in \mathcal{T} \\ S_i \neq S_j}} l(S_i, S_j)}{\sum\limits_{\substack{S_i,S_j \in \mathcal{T} \\ S_i \neq S_j}} l(S_i, S_j) + \sum\limits_{\substack{S_i \in \mathcal{T} \\ S_j \in \mathcal{S} \setminus \mathcal{T}}} l(S_i, S_j)} \quad (10)$$

Given the $alternatives$ vector, *EvaluateMigration Alternatives* chooses the best alternative ([$\mathcal{T}^o$,Target SDN controller^o]) among them, that optimizes one of the following $Evaluation$-$Method$:

TABLE I
EXAMPLE OF *Alternatives* CARRIED OUT BY BALCON ALGORITHM IN A TOPOLOGY WITH 60 SWITCHES, 5 CONTROLLERS AND A CLUSTER $\mathcal{T} = \{S_1, S_2, S_{56}\}$

$\mathcal{T}$	Target SDN Controller	$[L_{C_1}, \ldots, L_{C_{\lvert\mathcal{C}\rvert}}]$	$migration\ size$
$\{S_1,S_2,S_{56}\}$	C_1	[90, 9, 6, 10, 8]	0
$\{S_1,S_2,S_{56}\}$	C_2	[86, 51, 6, 10, 8]	3
$\{S_1,S_2,S_{56}\}$	C_3	[70, 9, 48, 10, 8]	3
$\{S_1,S_2,S_{56}\}$	C_4	[80, 9, 6, 51, 8]	3
$\{S_1,S_2,S_{56}\}$	C_5	[96, 9, 6, 10, 50]	3

minMax - Minimize the maximum controllers load:

$$\underset{alternatives}{\text{argmin}} \left(\max\ [L_{C_1}, \ldots, L_{C_{\lvert\mathcal{C}\rvert}}]\right) \quad (11)$$

minSum - Minimize the sum of controllers load:

$$\underset{alternatives}{\text{argmin}} \sum_{C_m \in \mathcal{C}} L_{C_m} \quad (12)$$

integral - Maximize the distance from the controllers load configuration in case of congestion:

$$\underset{alternatives}{\text{argmax}}\ \mathcal{D}([L_{C_1}, \ldots, L_{C_{\lvert\mathcal{C}\rvert}}], [\widehat{L}_{C_1}, \ldots, \widehat{L}_{C_{\lvert\mathcal{C}\rvert}}]) \quad (13)$$

with $[\widehat{L}_{C_1}, \ldots, \widehat{L}_{C_{\lvert\mathcal{C}\rvert}}]$ the vector of controllers load when congestion appears just before BalCon, and function $\mathcal{D}(\mathbf{u}, \mathbf{v})$ defined as follow:

$$\mathcal{D}(\mathbf{u}, \mathbf{v}) = \sum_i \int_{u_i}^{v_i} x^2 dx \quad (14)$$

V. MIGRATION BLACKOUT AND BALCONPLUS

A. Migration Blackout

Migrating switches among controllers dynamically based on the controllers' load can balance their loads so as to relief congestion. However, during the migration, some switches may not be able to handle new connections timely, which is called the migration blackout [5], [7].

Dixit *et al.* [5], [7] presented a switch migration protocol that can safely migrate switches between two controllers without violating the *liveness*, *safety*, and *serializability* properties.

The migration protocol is explained in Figure 4, where a switch is migrated from controller 1 to controller 2 in four phases.

In phase 1, controller 1 sends a *Start Migration* message to controller 2, which upon receiving the message will change its role to *equal*, meaning that it can now receive messages from the switch, but can not process them. Controller 2 will then immediately send a *Ready for migration* to controller 1, which completes phase 1.

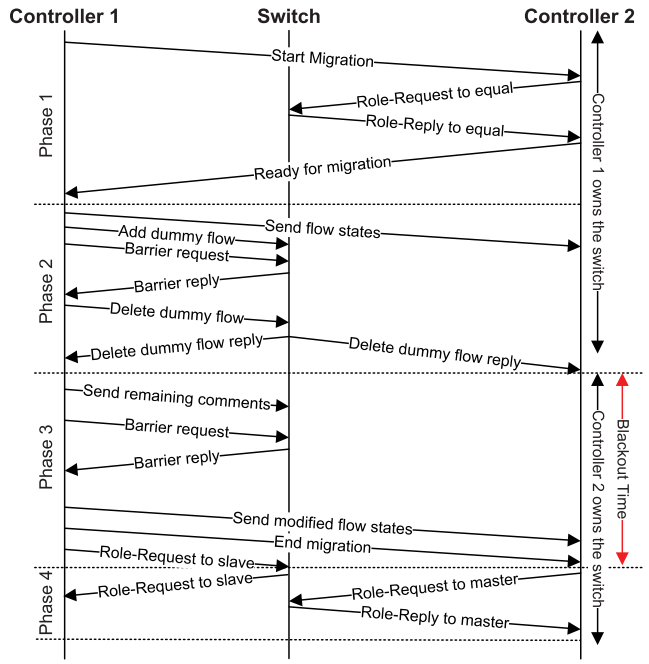


Fig. 4. Protocol for migrating a switch from a controller to another.

In phase 2, controller 1 will first send state info of the switch to controller 2 to enable it to take over from where controller 1 left after the migration. At the same time (in parallel to the state info transmission), controller 1 will install an dummy flow entry to the switch and delete it afterwards. The purpose of doing this is to trigger a dummy flow deletion reply from the switch, which is sent to both controllers to signal them a migration event. After this event, all processing and decision-making will be the responsibility of controller 2 while controller 1 will ignore any messages from the switch. This concludes phase 2.

In phase 3, although controller 2 possesses the control of the switch, it cannot install any flow entries to the switch yet. This is because there might be outstanding tasks being processed by the controller 1. Controller 2 needs to wait until the completion of these outstanding tasks and the installation of corresponding flow entries before it can install flow entries to the switch. Meanwhile, all messages received by controller 2 will be buffered. Once finishing all outstanding tasks, controller 1 will ensure that the corresponding flow entries are successfully installed in the switch by sending a barrier request to the switch to flush all outstanding flow-mod messages. After receiving the barrier reply from the switch, controller 1 needs to send all modified state info since the beginning of phase 2 to controller 2 and ends the entire migration procedure by sending out an *End migration* message.

In phase 4, controller 1 changes its role to slave and controller 2 changes its to master. All messages buffered at controller 2 in phase 3 can now be processed.

Based on the above migration procedure, we can see that there is a migration blackout period equal to the length of phase 3. In this period, packet-in messages from the switch cannot be immediately processed by controller 2, which may defer connection setup for new flows. Our prototype shows the blackout period can be as large as 370ms (details are in Section VII-E), which is larger than the value 50 $\sim$ 100ms reported in [5] and [7]. This larger blackout period is due to the fact that the controller that is releasing the control of a switch is severely overloaded. Thus, the processing is slower as opposed to an idle controller.

The 370ms blackout time is apparently too large, especially for applications in datacenters, which usually require latency in tens of μs. However, as pointed out in [5] and [7], if serializability property is not required (i.e., messages from the switch can be processed out-of-order), this blackout period can be removed and the setup of new connections will not experience extra latency. BalCon is good for such cases.

If the network requires the serializability property, switch migration based on BalCon may cause temporary service disruption. To address this issue, we propose an improved version of BalCon, named BalConPlus, to avoid service disruption during the migration.

B. BalConPlus

The main idea of BalConPlus is to temporarily steer newly arriving flows away from switches that are to be migrated so their flow setup will not be affected by the migration. To achieve this goal, BalConPlus makes two changes to the baseline BalCon.

(1) The first change applies when BalCon selects which switch(es) to migrate. In order to ensure that there is always an alternative path bypassing the switches to be migrated, BalConPlus adds a new constraint to the selection of migrating switches: the (hypothetical) removal of the selected switches should not break network connectivity. This change only requires slight modification of $ComputeStartingSwitchesList$ and $IncreaseCluster$ in Algorithm 1. When preparing a list of individual SDN switches that could be migrated in $ComputeStartingSwitchesList$, we exclude those that could cause the network disjointed if they were removed from the network. Consider the network in Figure 1, all switches could be considered in $ComputeStartingSwitchesList$ because removing any single one of them will not break network connectivity. When we gradually augment the set of candidate migrating switches in $IncreaseCluster$, we exclude those that could break the network if they were removed. Consider the same example in Figure 1, S5 and S7 cannot be selected at the same time for migration, because removing them from the network will divide the network into two parts. However, S5 or S7 can be migrated individually because removing either one of them will not break the network.

(2) The second change applies when BalCon conducts routing computation for new flows arrived in the middle of a migration. BalConPlus will steer new flows to the paths bypassing the migrating switches. This may slightly increase the hop count of the forwarding path of some flows; but as compared to the hundreds of ms migration blackout period, the slightly larger forwarding delay due to a longer path becomes insignificant. Once the migration completes, new

TABLE II
COMPARISON OF BALCON AND BALCONPLUS

	BalCon	BalConPlus
Good for Application requiring serializability property?	No	Yes
Good for Application not requiring serializability property?	Yes	Yes
Migrating Edge Switches?	Yes	No

flows will be routed on their best paths. Consider the example in Figure 1, if we want to migrate S5 from controller C2 to C1, new flows from H2 to H6 during the switch migration should be routed to a different path (such as $H2 \rightarrow S2 \rightarrow S4 \rightarrow S7 \rightarrow S6 \rightarrow H6$) to bypass S5.

It is noted that based on the above two changes, BalConPlus will not select edge switches for migration (i.e., those directly connected to hosts). Because if an edge switch is selected for migration, its connected hosts will inevitably be disconnected from the network and lead to temporary service disruption. With the example in Figure 1, switch S2 should not be selected for migration, because it will temporarily disconnect host H2 from the network during the migration blackout period.

Here we call switches that are not directly connected to hosts *core switches*. In many network topologies, such as FatTree topology in datacenter networks, we usually have more core switches than edge switches in the scale-out structure. Thus, migrating core switches provides enough flexibility to adjust workload among the controllers.

C. BalCon Vs. BalConPlus

Table II provides a comparison between BalCon and BalConPlus.

VI. PERFORMANCE EVALUATION WITH SIMULATIONS

BalCon and BalConPlus have been implemented using Matlab R2015a 64bit for Linux. The simulations has been carried out using a PC equipped with an Intel Core i5-3340@3.10 GHz with 8 GB of 1600 MHz DDR3 RAM and an OS Linux Mint 17. Both schemes have similar simulation results and due to lack of space we only present the simulation results for BalCon. The performance of BalConPlus and comparison between BalCon and BalConPlus will be presented in Section VII based on prototype testbed we build.

A. Dynamic Scenario–Effectiveness of BalCon

Here we fix BalCon parameters (mcs, $mssls$ and $EvaluationMethod$) and evolve the network over time in order to show the effectiveness of BalCon during a (simulated) runtime network operation. We simulated 4 different network topologies shown in Table III, varying the degree in which edge-core (dEC) and core-core (dCC) nodes are connected. In particular, dEC represents the number of connections that each edge node has towards core nodes, while each dCC represents the number of connections each core node has towards other core nodes. To perform Dynamic Scenario simulations we implemented a routine that generates flow arrivals and departures at edge nodes following a Poisson process. For each topology presented, we run 200 different simulations with different seeds of the Poisson process generator. Each run simulates $2000s$ of network runtime operation. BalCon has been setup using a starting switch list size $mssls = 20$ and a maximum cluster size $mcs = 20$ using Equation 13 (Integral) as $EvaluationMethod$ in $EvaluateMigrationAlternatives$.

TABLE III
TOPOLOGIES SIMULATED FOR PERFORMANCE ANALYSIS

Name	# edge	# core	dEC	dCC	# controllers
$Topology1$	50	40	2	full mesh	5
$Topology2$	50	40	5	full mesh	5
$Topology3$	50	40	2	$\frac{\#\text{ core nodes}}{5}$	5
$Topology4$	50	40	5	$\frac{\#\text{ core nodes}}{5}$	5

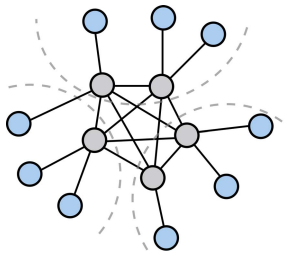


Fig. 5. Example of network topology with 9 edge nodes (in blue), 5 core nodes (in gray), 3 controllers, $dEC = 1$ and $dCC =$ full mesh.

Figure 5 shows a topology composed of 9 edge nodes (in blue), 5 core nodes (in gray), and 3 controllers. $dEC = 1$ indicates that each edge node is connected to a single core node, while $dCC =$ full mesh since the core nodes form a full mesh network.

Figure 6 shows the computational load of 5 controllers (0 means no congestion at all, while 100 indicates overload) during the simulation of $Topology1$. The green line represents the congestion level of controller C_5. As soon as it reaches the threshold $\boldsymbol{L} = 90$, BalCon is triggered using the starting switch list size $swlsm = 20$ and the maximum cluster size

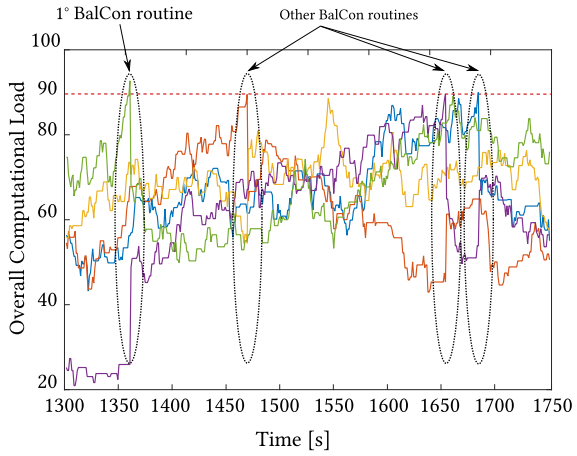


Fig. 6. Computational load of 5 controllers during Dynamic Scenario and the effect of BalCon algorithm in simulations with $Topology1$ and $seed = 1$. The blue line is L_{C_1}, the red line is L_{C_2}, the yellow line is L_{C_3}, the violet line is L_{C_4} and the green line is L_{C_5}.

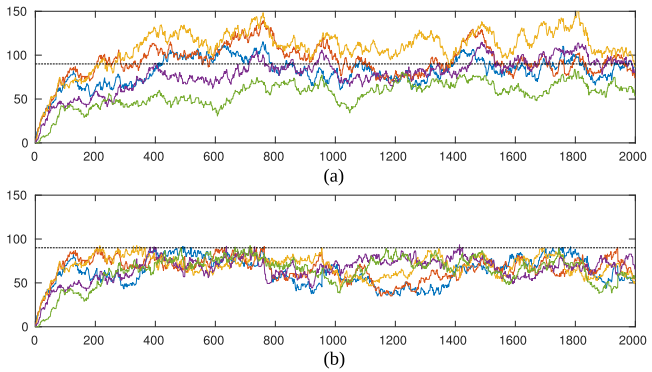


Fig. 7. Comparison of the computational load between a static assignement (a) and BalCon (b) in Dynamic scenario with $Topology3$.

$msc = 20$. The different routines of BalCon are indicated with black dotted ellipse.

BalCon performs well: the maximum computational load during the 4 BalCon instances is reduced on average by 15%, with an average of 2.4 switches migrated in each routine. The computational time is 0.69s. The variance of the computational load is reduced at each routine on average by 66%. In this case BalCon can effectively balance the computational load and solve the overloading problem at the controller with few switch migrations.

Figure 7 clearly shows the performance advantage of BalCon algorithm compared to the static assignment of the switches to the controller using the same traffic pattern. Figure 7(a) shows the computational load of the 5 controllers without load balancing, i.e., static assignment, while Figure 7(b) is the case in which BalCon is implemented. As we observe, BalCon maintains the controllers' load below the threshold during runtime, whereas in the static assignment case the congestion load exceeds the threshold (90) by 50%. Other settings with different topologies in Table III show similar results as Figure 7.

B. Static Scenario

In Static Scenario simulations set we fix the time instant (when congestion occurs) and we vary BalCon parameters in order to show how the parameters affect BalCon's performance. We varied $mssls$, mcs and the method for $EvaluateMigrationAlternatives$ function. We simulated 4 different network topologies shown in Table III. For each topology we synthetically generated 500 different "congestion traffic configurations" in which one controllers is congested. For each congestion traffic configuration we run several instances of BalCon algorithm varying $mssls = \{3, 5, 10, 20\}$ and $mcs = \{3, 5, 10, 20\}$.

For each simulation, we evaluated different performance indicators. Let $\boldsymbol{L_C} = [L_{C_1}, \dots]$ the vector denote the controllers' load, $\boldsymbol{L_C}^{con}$ the controllers' load when congestion appears just before the application of BalCon and $\boldsymbol{L_C}^{bal}$ the loads after BalCon routine.

Definition 9: Let the congested controller $C_m^* = \text{argmax } \boldsymbol{L_C}^{con}$ and the congested controller load $\boldsymbol{L_C}^{con}(C_m^*)$. We define the Reduction Congested Controller Load (%) as:

$$\frac{\boldsymbol{L_C}^{bal}(C_m^*) - \boldsymbol{L_C}^{con}(C_m^*)}{\boldsymbol{L_C}^{con}(C_m^*)} \cdot 100. \tag{15}$$

Definition 10: Reduction Max Controller Load (%)

$$\frac{\max \boldsymbol{L_C}^{bal} - \max \boldsymbol{L_C}^{con}}{\max \boldsymbol{L_C}^{con}} \cdot 100 \tag{16}$$

Definition 11: Reduction Sum Controller Load (%)

$$\frac{\sum \boldsymbol{L_C}^{bal} - \sum \boldsymbol{L_C}^{con}}{\sum \boldsymbol{L_C}^{con}} \cdot 100 \tag{17}$$

Definition 12: Reduction Variance Load (%)

$$\frac{Var(\boldsymbol{L_C}^{bal}) - Var(\boldsymbol{L_C}^{con})}{Var(\boldsymbol{L_C}^{con})} \cdot 100 \tag{18}$$

Figure 8 shows the performance of different versions of BalCon by varying $mssls$ and mcs using $Topology1$ and $minMax$ as $EvaluationMethod$. In the first instance, we consider the black bars, representing the choice of parameters $[mssls, mcs] = [3, 3]$. We observe a reduction of the congested controller load by 12.55% (Figure 8(a)), a reduction of the max controllers load by 11.32% (Figure 8(b)), an almost negligible reduction of the sum of the controllers load (Figure 8(c)), a 47.10% of the reduction of the variance (Figure 8(d)). We also observe that we obtain an average migration size of 1.37 switches (Figure 8(e)) and an average BalCon computation time of 0.13s (Figure 8(f)). Considering now the other bars, we note that the performance is highly dependent on the parameters. If we have a larger $mssls$ and mcs, we can increase the search space of the possible solutions of BalCon. This translates to better performance. In fact, if we consider the case $[mssls, mcs] = [20, 20]$, we observe a significant increase of the performance indicators described before.

With large values of $mssls$ and mcs, we can observe a small increase of the migration size (from 1.37 to 2.05). BalCon is

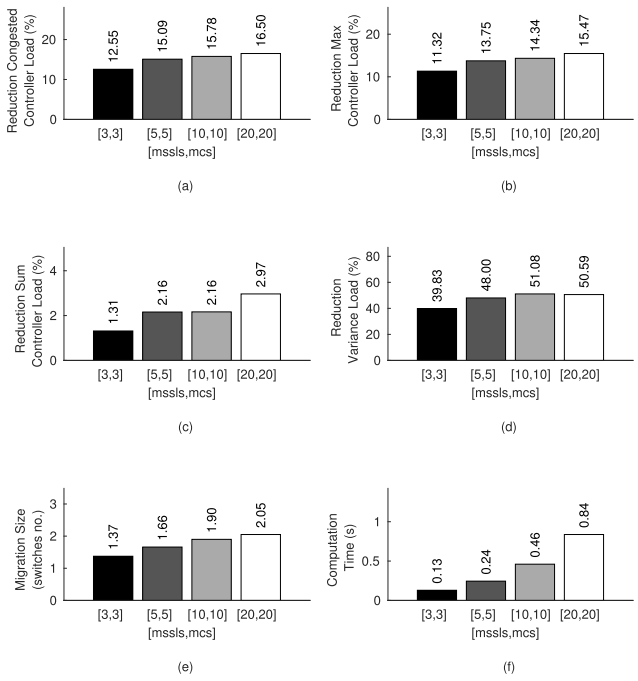


Fig. 8. Performance of different version of BalCon varying $mssls$ and mcs using $Topology1$.

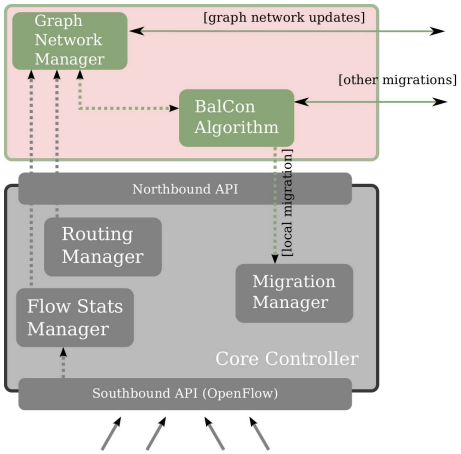


Fig. 9. BalConController architecture.

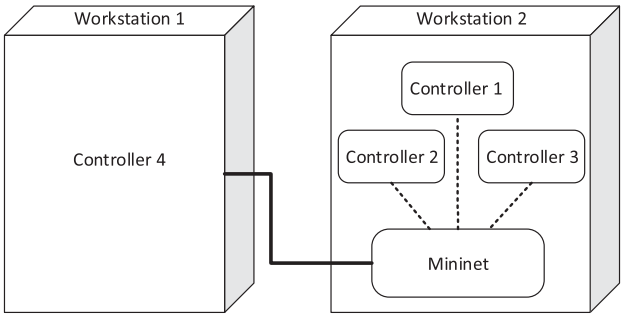


Fig. 10. Architecture of the experimental environment.

quite fast, in fact the computation time is lower than $1s$ ($0.84s$) with higher values of $mssls$, and mcs. As we observe, BalCon is highly efficient with low computation time and few switch migrations needed.

VII. Prototype of BalCon and BalConPlus and Experimental Results

In this section, we present further details on how we designed and implemented BalConController by modifying and adding components to RYU controller [8].

A. Design

BalConController architecture can be implemented through a NorthBound application of the SDN controller and run in a distributed fashion: only the congested controller will activate the BalCon routine based on an updated map of the network. In particular Figure 9 shows the modules involved in the BalConController and their relationship with existing modules in an SDN controller.

Graph Network Manager is the entity that gathers both flow arrival statistics from *Flow Stats Manager* entity and routing decisions from *Routing Manager* entity in order to construct and update the local version of the graph representation $G(\mathcal{S}, \mathcal{E})$. $G(\mathcal{S}, \mathcal{E})$ is then continuously updated (*[graph network updates]*) with the other SDN controllers. BalCon, using the updated information in the *local graph*, computes the computational load and the migration cluster in case of congestion through the *BalCon Algorithm* entity. In case of migration *BalCon Algorithm* informs *Migration Manager* entity for the local migrations and other controllers for the other migrations.

BalConController extends RYU functionalities, by supporting the multicontroller features: it can run on multiple instances on different hosts/networks (each controller has an IP address) and each instance manages a portion of the entire network. It also implements a homemade inter-controller messaging through UDP sockets and a custom application protocol in Python. The inter-controller messaging permits the controllers to exchange themselves different kind of information like among *Graph Network Manager* entities (e.g., traffic updates) and *Migration Manager* entities (e.g., switches to be migrated). A more reliable solution could be the use of distributed data store like Zookeeper or Hazelcast [7]. *Migration Manager* module implements the switch migration procedure proposed in [5] that guarantees liveness and safety for each switch migration. Finally, BalConController fully implements the BalCon algorithm that can run indipendently in each SDN controller based on the unified view of the entire network continuously updated.

B. Experiment Setting

We use two workstations to setup our testbed for experiments: Workstation 1 and Workstation 2, as shown in Figure 10. Workstation 1 with Intel Xeon Processor X5650 and CentOS 7 is mainly for measuring the hardware performance for Controller 4. Developed as a multi-thread Ryu application, Controller 4 executes directly on Workstation 1 for performance measurement. By separating it on an independent

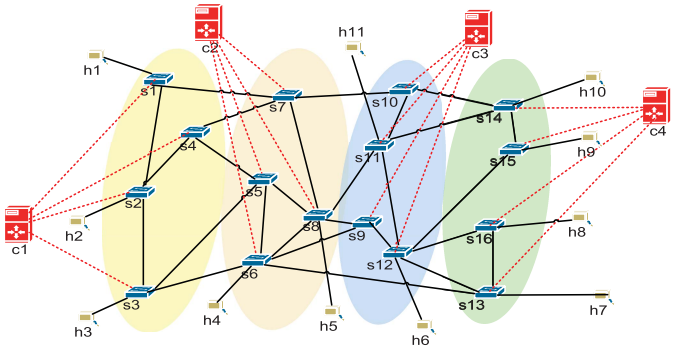


Fig. 11. Logical network topology used in the experiment.

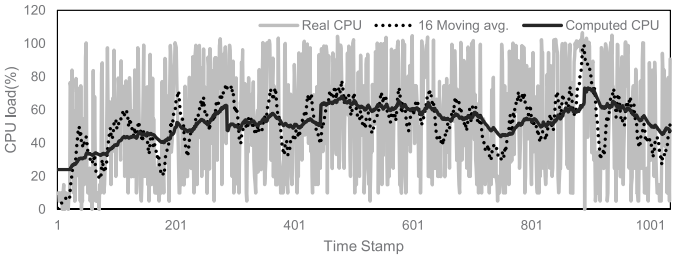


Fig. 12. Evaluation of controller load model accuracy.

machine, the overhead of virtualization [12] and scheduling of operation system can be eliminated. Therefore, the performance, including CPU loads and path-calculating time, can be accurately and directly measured on the hardware.

On the other hand, Workstation 2 with Intel Xeon CPU E5-2620 v4 and Windows Server 2016 Datacenter is mainly for emulating the other three controllers and the data plane, which runs in Mininet. The three controllers and Mininet run on four independent virtual machines (VMs). The VM of the mininet is set to have 16GB memory and 12 processors, while VMs of three controllers each has 8GB memory and 4 processors. These VMs run Ubuntu 16.04 and are bridged together with the physical network interface to connect to Controller 4.

The logical network topology used in our experiment is shown in Figure 11, which has 11 hosts and 16 switches emulated in Mininet. Each controller initially controls four switches. For example, Controller 2 controls S5, S6, S7 and S8 in the beginning. We will observe the migration of these switches between controllers to evaluate the functionality correctness of BalCon and BalConPlus. In the following experiments, the threshold used at each controller to trigger switch migration is set as 60% of CPU load.

C. Automated Parameter Measurement

The operation of BalCon and BalConPlus requires knowing values of α and β, based on which expected controller load for each candidate of migration can be calculated. Here we propose an automated method to measure α and β.

The load of a controller can be described by the following equation.

$$L_c = packet\_in\_rate * \alpha + rule\_installation\_rate * \beta + \delta \quad (19)$$

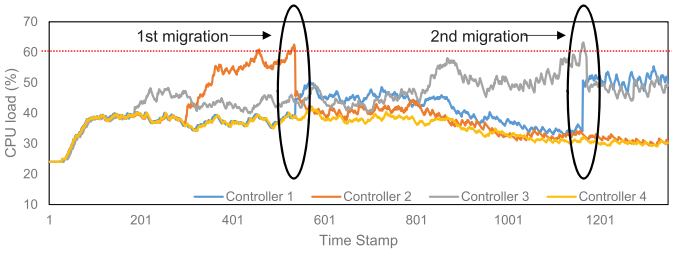


Fig. 13. Control load balancing achieved by BalConPlus under synthetic traffic.

where δ is the base (background) workload running in the controller. In the equation, L_c, $packet\_in\_rate$ and $rule\_installation\_rate$ can be measured in real-time by the controller. The three variables, α, β, and δ can be calculated if we can get three instances of this equation. In our experiment, we collect three combinations of L_c, $packet\_in\_rate$ and $rule\_installation\_rate$ at three random selected moments and solve α, β, and δ. These calculated paramenters are applied in BalCon and BalConPlus for controller load modeling and prediction.

Figure 12 shows the measured controller load (i.e., directly pull the load from CPU) vs. calculated load (i.e., calculated load based on solved α, β, δ and measured $packet\_in\_rate$ and $rule\_installation$ using Equation 19). In the figure, the gray curve represents the measured instantaneous controller load, which frustrates greatly. To get stable α, β and δ values, we use the 16-sample moving average load to solve them. The measured moving average load is shown in the dotted black curve. The calculated load is shown in solid black curve. We can see that even though the real CPU load does not exactly match our model due to the possible effects mentioned in [13], the trend is close enough for BalCon and BalConPlus to predict the controller load. In our experiment, the values of α, β, and δ are recalculated periodically since they may vary over time.

D. Controller Load Balance

Firstly, we observe the CPU load balancing achieved by BalConPlus (BalCon achieves similar balance results, and due to lack of space we only present the experiment results of BalConPlus). We conduct the experiments using both synthetic traffic and real-life data center traffic traces.

1) Synthetic Traffic: We first generate new flows at constant rate from host h1 to h10 on path ($h1 \rightarrow s1 \rightarrow s7 \rightarrow s10 \rightarrow s14 \rightarrow h10$). It is easy to see that the loads on the four controllers are roughly similar. Then, we gradually increase the flow generating rate from host h4 to h5 from 0 to a point that Controller 2 will start to be congested. Then switch 7 will be migrated by BalConPlus (BalCon as well) to either Controller 1 or Controller 3.

The experiment result is shown in Figure 13. From timestamp 273, we start to increase the flow generating rate between h4 and h5. At timestamp 536, BalConPlus is triggered, and switch s7 is migrated from Controller 2 to Controller 1. The load at Controller 2 drops significantly, while the load of Controller 1 only increases slightly. This shows

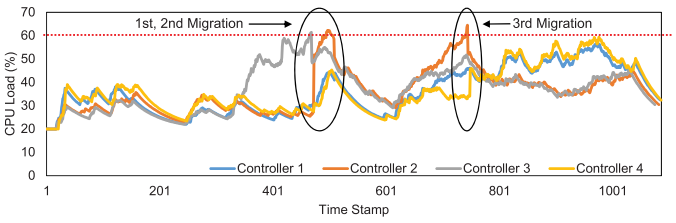


Fig. 14. Control load balancing achieved by BalConPlus under real-life data center traffic.

that the overall controller load can be decreased due to the switch clustering effect based on our model. The loads of four controllers are shown to be balanced.

Then we gradually increase the flow generating rate from host h6 to h11 from 0 to a point that Controller 3 starts to be congested. In the experiment, switch 10 will be migrated by BalConPlus (BalCon as well) to Controller 1. At time stamp 1168 in Figure 13, BalConPlus is triggered, and CPU load of Controller 3 is reduced to below the threshold. The reason why CPU load are not perfectly balanced among the four controllers is because of the potential increment of the overall CPU load if doing so. In this case, BalConPlus chooses the migration candidate that will not increase too much of the overall CPU load while reducing the load of Controller 3.

2) Real-Life Data Center Traffic Trace: We have collected real-life traffic trace on a backbone link from a data center operated by New York City Department of Education. We randomly select three trunks from the trace with each trunk lasting for 10 minutes and feed them to the three host pairs ($h1 \rightarrow h10$, $h4 \rightarrow h5$, and $h6 \rightarrow h11$) in the topology in Figure 11.

The experiment result is shown in Figure 14. At timestamp 466 (1st migration), BalConPlus is triggered, and switch 11 is migrated from Controller 3 to Controller 2. The load at Controller 3 drops and the load at Controller 2 increases. At timestamp 506 (2nd migration), BalConPlus is triggered again, and switch 6 is migrated from Controller 2 to Controller 1. The load at Controller 2 drops significantly and the load at Controller 1 increases slightly. This shows that the overall controller load can be decreased due to the switch clustering effect based on our model. The loads at the four controllers start to grow at timestamp 607. At timestamp 743 (3rd migration), BalConPlus is triggered, and switch 8 is migrated from Controller 2 to Controller 4. The load at Controller 2 drops and the load at Controller 4 increases. The loads of four controllers are shown to be balanced.

E. BalCon Vs BalConPlus on Packet-In Response Time

As compared to the baseline BalCon, the response delay caused by switch migration blackout is eliminated in BalConPlus, which will find an alternative path without passing through the migrating switches. To compare the packet-in response time in BalCon and BalConPlus, we measure the delay from the moment that a packet-in is received by a controller to the moment that corresponding flow-mod is sent out to the switch.

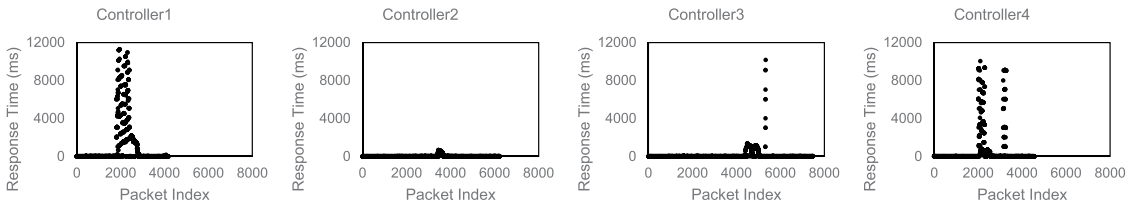
We first measure the response time of Controller 4 during a migration when new flows' route passes the migrating switch. Given Controller 4 is running on a dedicated server, we are able to measure very accurate delay, which is around 370 ms. The measure on Controller $1 \sim 3$, however, is difficult, because these controllers are running in VMs that share the same physical workstation with the VM running Mininet. The virtualization of VMs and Mininet and scheduling of OS introduce huge disturbance on the measurement delay by as much as 3 seconds, which is too large and drowns the delay of migration blackout. The similar delay disturbance of Mininet has also been observed and reported in [14].

In order to show the effect of migration blackout on Controllers $1 \sim 3$ with disturbance introduced by virtualization and OS scheduling, we purposely enlarge the length of switch migration blackout (i.e., phase 3 in Figure 4) to 10 seconds. The result is shown in Figure 15, where the x-axis is the index of each packet-in, and the y-axis is the response time of the packet-in request. Figure 15(a) shows that many packet-in requests in BalCon suffer from large response delay due to two migrations occurred at Controller 1 and Controller 4, and one migration occurred at Controller 3. In Figure 15(a), Controller 2 has lower packet-in arrival rate, so less packet-in requests suffer from the extra delay comparing to controllers 1 or 3. On the other hand, Figure 15(b) shows the packet-in response time of BalConPlus. We can see that none of packet-in requests suffers from the response delay even though we have two migrations occurred at Controller 1 and three migrations occurred at Controller 4 during the experiment.

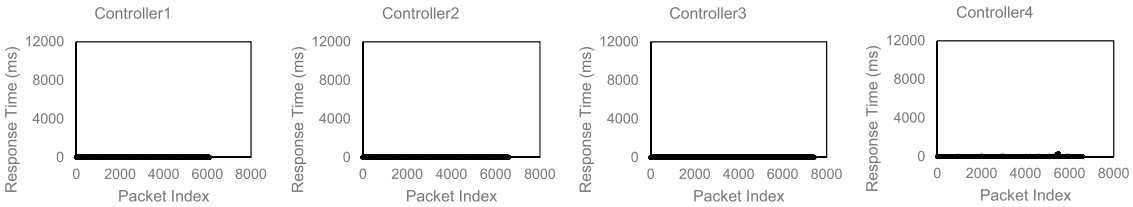
VIII. Related Works

References [15]–[18] propose multi-threaded design and parallelization techniques of OS processes in the SDN controller. Mallon *et al.* [19] propose a rethinking of the design of the SDN controllers into a lower level software that leverages both operating system optimizations and modern hardware features. Renart *et al.* [20] mitigate the scalability problem of the SDN controller by offloading all the packet inspection and creation to the GPU. References [21]–[27] study the controller placement and QoS enforcement for SDN in 5G and carrier networks.

Other works have also explored the implementation of distributed controllers through the using of multiple hosts: with different roles [28]–[30] or with equal roles [2]–[4]. The main focus of these papers is to address the state consistency issue across distributed controller instances, while preserving good performance. Whereas [31]–[33] focus on the controller placement problem minimizing the communication delay between controllers and switches. Current existing distributed controller solutions still suffer from the static mapping between SDN switches and controllers, limiting the capability of dynamic load adaptation. Dixit *et al.* [5], [7] propose an elastic distributed controller architecture able to force migration of SDN switches to different controllers using the existing OpenFlow standard, whereas Bari *et al.* [34] try to model the problem of



(a) Four controllers with BalCon



(b) Four controllers with BalConPlus

Fig. 15. Packet-in Response time of BalCon and BalConPlus when there is switch migration blackout.

switch-controller assignment, minimizing the communication cost (in terms of hops) among controllers and switches.

Shah *et al.* [35] propose an SDN controller framework named Cuttlefish that can adaptively offload a portion of the application state to local controllers to achieve higher throughput and lower latency on control plane. Wang *et al.* [36] propose a new routing scheme to achieve both controller load balance and link load balancing in an SDN. Wang *et al.* [37], [38] propose a dynamic SDN controller assignment scheme in data center networks with a goal to balance the controller load while keeping the control traffic overhead low. However, their model only considers the controller load caused by flow request processing but ignores the load for handling rule installation. They also don't consider the overhead incurred in switch migration. To overcome the switch migration overhead, Huang *et al.* [39] propose BLAC, a scheduling layer, between switches and controllers. BLAC intercepts flow requests from switches and dispatches them to different controllers to achieve controller balance. Unfortunately, the scheme doesn't consider the impact of switch/controller location to the performance and the new scheduling layer introduced will increase the communication latency between switches and controllers. Muthanna *et al.* [40] present a dynamic clustering algorithm to balance the load among the distributed controllers in the SDN network. However, the scheme doesn't consider the overhead involved in the switch migration and the evaluation is solely based on Matlab simulation.

IX. Conclusions

In this paper, we presented BalCon and BalConPlus, two SDN switch migration schemes to achieve load balance among SDN controllers with small migration cost. BalCon is suitable for scenarios where the network does not require serial processing of switch requests. For other scenarios, BalConPlus is more suitable, as it is immune to the switch migration blackout and does not cause any service disruption. Both schemes have been thoroughly evaluated with simulations and experiments. The results demonstrate the practicality and effectiveness of both schemes to achieve SDN controller load balance. In our future work, we plan to extend the implementation of BalCon and BalConPlus to other SDN controller platforms such as OpenDayLight to study the impact of controller platforms to the performance of BalCon and BalConPlus.

References

[1] M. Cello, Y. Xu, A. Walid, G. Wilfong, H. J. Chao, and M. Marchese, "Balcon: A distributed elastic SDN control via efficient switch migration," in *Proc. IEEE Int. Conf. Cloud Eng. (IC2E)*, Apr. 2017, pp. 40–50.

[2] T. Koponen *et al.*, "Onix: A distributed control platform for large-scale production networks," in *Proc. 9th USENIX Conf. Operating Syst. Design Implement. (OSDI)*, Berkeley, CA, USA, Oct. 2010, pp. 1–6.

[3] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: State distribution trade-offs in software defined networks," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, New York, NY, USA, 2012, pp. 1–6.

[4] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proc. Internet Netw. Manage. Conf. Res. Enterprise Netw. (INM/WREN)*, Berkeley, CA, USA, 2010, p. 3.

[5] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, New York, NY, USA, 2013, pp. 7–12.

[6] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. 10th ACM SIGCOMM Conf. Internet Meas. (IMC)*, New York, NY, USA, 2010, pp. 267–280.

[7] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. R. Kompella, "ElastiCon; An elastic distributed SDN controller," in *Proc. ACM/IEEE Symp. Archit. Netw. Commun. Syst. (ANCS)*, Oct. 2014, pp. 17–27.

[8] *RYU Controller*. [Online]. Available: https://osrg.github.io/ryu/

[9] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[10] *OpenFlow Switch Specification*, Open Netw. Found., Menlo Park, CA, USA, Mar. 2014. [Online]. Available: https://www.opennetworking.org

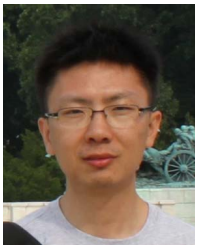
[11] S. E. Schaeffer, "Survey: Graph clustering," *Comput. Sci. Rev.*, vol. 1, no. 1, pp. 27–64, Aug. 2007.

[12] S. Sudevalayam and P. Kulkarni, "Affinity-aware modeling of CPU usage for provisioning virtualized applications," in *Proc. IEEE 4th Int. Conf. Cloud Comput. (CLOUD)*, Jul. 2011, pp. 139–146.

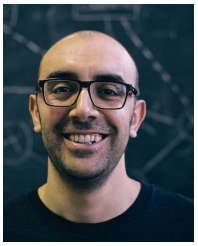
[13] A. Shalimov, D. Zuikov, D. Zimarina, V. Pashkov, and R. Smeliansky, "Advanced study of SDN/openflow controllers," in *Proc. 9th Central Eastern Eur. Softw. Eng. Conf. Russia*, Oct. 2013, p. 1.

[14] S.-Y. Wang, C.-L. Chou, and C.-M. Yang, "EstiNet openflow network simulator and emulator," *IEEE Commun. Mag.*, vol. 51, no. 9, pp. 110–117, Sep. 2013.

[15] Z. Cai, A. L. Cox, and T. S. E. Ng, "Maestro: A system for scalable openflow control," CS Dept., Rice Univ., Houston, TX, USA, Tech. Rep. TR10-11, Dec. 2010.
[16] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. 2nd USENIX Conf. Hot Topics Manage. Internet, Cloud, Enterprise Netw. Services (Hot-ICE)*, Berkeley, CA, USA, Apr. 2012, pp. 1–6.
[17] D. Erickson, "The beacon openflow controller," in *Proc. 2nd ACM SIGCOMM Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, New York, NY, USA, Aug. 2013, pp. 13–18.
[18] *Floodlight OpenFlow Controller*. Accessed: Apr. 24, 2014. [Online]. Available: http://www.projectfloodlight.org/floodlight/
[19] S. Mallon, V. Gramoli, and G. Jourjon, "Are today's SDN controllers ready for primetime?" in *Proc. IEEE 41st Conf. Local Comput. Netw. (LCN)*, Nov. 2016, pp. 325–332.
[20] E. G. Renart, E. Z. Zhang, and B. Nath, "Towards a GPU SDN controller," in *Proc. Int. Conf. Workshops Netw. Syst. (NetSys)*, Mar. 2015, pp. 1–5.
[21] A. Ksentini, M. Bagaa, and T. Taleb, "On using SDN in 5G: The controller placement problem," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.
[22] D. L. C. Dutra, M. Bagaa, T. Taleb, and K. Samdanis, "Ensuring end-to-end QoS based on multi-paths routing using SDN technology," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2017, pp. 1–6.
[23] T. Taleb, M. Bagaa, and A. Ksentini, "User mobility-aware virtual network function placement for virtual 5g network infrastructure," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jun. 2015, pp. 3879–3884.
[24] M. Bagaa, T. Taleb, and A. Ksentini, "Service-aware network function placement for efficient traffic handling in carrier cloud," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2014, pp. 2402–2407.
[25] M. Bagaa, T. Taleb, A. Laghrissi, and A. Ksentini, "Efficient virtual evolved packet core deployment across multiple cloud domains," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2018, pp. 1–6.
[26] R. A. Addad, D. L. C. Dutra, M. Bagaa, T. Taleb, H. Flinck, and M. Namane, "Benchmarking the ONOS intent interfaces to ease 5G service management," in *Proc. IEEE GLOBECOM*, 2018.
[27] R. A. Addad, T. Taleb, and H. Flinck, "Towards modeling cross-domain network slices for 5G," in *Proc. IEEE GLOBECOM*, 2018.
[28] S. H. Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proc. 1st Workshop Hot Topics Softw. Defined Netw. (HotSDN)*, New York, NY, USA, Aug. 2012, pp. 19–24.
[29] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," in *Proc. ACM SIGCOMM Conf.*, New York, NY, USA, Oct. 2010, pp. 351–362.
[30] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *Proc. ACM SIGCOMM Conf.*, New York, NY, USA, 2011, pp. 254–265.
[31] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 473–478, Sep. 2012.
[32] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in SDN-based core networks," in *Proc. 25th Int. Teletraffic Congr. (ITC)*, Sep. 2013, pp. 1–9.
[33] Y. Jiménez, C. Cervelló-Pastor, and A. J. García, "Defining a network management architecture," in *Proc. 21st IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2013, pp. 1–3.
[34] M. F. Bari *et al.*, "Dynamic controller provisioning in software defined networks," in *Proc. 9th Int. Conf. Netw. Service Manage. (CNSM)*, Oct. 2013, pp. 18–25.
[35] R. Shah, M. Vutukuru, and P. Kulkarni, "Cuttlefish: Hierarchical SDN controllers with adaptive offload," in *Proc. IEEE 26th Int. Conf. Netw. Protocols (ICNP)*, Sep. 2018, pp. 198–208.
[36] H. Wang, H. Xu, L. Huang, J. Wang, and X. Yang, "Load-balancing routing in software defined networks with multiple controllers," *Comput. Netw.*, vol. 141, pp. 82–91, Aug. 2018.
[37] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic SDN controller assignment in data center networks: Stable matching with transfers," in *Proc. 35th Annu. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2016, pp. 1–9.
[38] T. Wang, F. Liu, and H. Xu, "An Efficient online algorithm for dynamic SDN controller assignment in data center networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 5, pp. 2788–2801, Oct. 2017.
[39] V. Huang, Q. Fu, G. Chen, E. Wen, and J. Hart, "BLAC: A bindingless architecture for distributed SDN controllers," in *Proc. IEEE 42nd Conf. Local Comput. Netw. (LCN)*, Oct. 2017, pp. 146–154.
[40] A. Muthanna *et al.*, "SDN multi-controller networks with load balanced," in *Proc. 2nd Int. Conf. Future Netw. Distrib. Syst.*, Jun. 2018, p. 57.



Yang Xu (S'05–M'07) received the B.E. degree from the Beijing University of Posts and Telecommunications in 2001 and the M.Sc. and Ph.D. degrees in computer science and technology from Tsinghua University, China, in 2003 and 2007, respectively. From 2007 to 2008, he was a Visiting Assistant Professor at NYU-Poly, Brooklyn, NY, USA. He is currently a Research Associate Professor with the Department of Electrical and Computer Engineering, New York University Tandon School of Engineering, New York City, NY, USA. He has published more than 60 journal and conference papers and holds over ten U.S. and international granted patents on various aspects of networking and computing. His research interests include software-defined networks, data center networks, network function virtualization, and network security. He served as a TPC member for many international conferences, as an editor for the Elsevier Journal of Network and Computer Applications, and as a Guest Editor for the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS—Special Series on Network Softwarization and Enablers and the *Wiley Security and Communication Networks Journal*—Special Issue on Network Security and Management in Software-Defined Network.



Marco Cello received the Ph.D. degree in telecommunication engineering from the University of Genoa in 2012. In 2013, he was a Post-Doctoral Research Fellow with the Polytechnic Institute of New York University, New York City, NY, USA, and a Visiting Researcher with NYU Abu Dhabi. In 2014 and 2015, he was at the University of Genoa, focusing on software-defined network (SDN) and nanosatellite communications. In 2016 and 2017, he was a Research Fellow at Nokia Bell Labs, Dublin, Ireland, and at the Application Platforms and Software Systems Research Laboratory, focusing on elastic serverless architectures, container-based cloud infrastructures, and SDN. He got deep experience in queuing theory, Markov chain, C/C++ and Python, and an in-depth knowledge in Linux-based emulation of telecommunication networks, IP networking technologies, SDN, L2 to L4 forwarding, QoS, traffic engineering, and routing protocols. He is currently the IT Manager at Rulex, a software company specialized in artificial intelligence and autonomous decisions. He is an In Charge of the whole IT infrastructure with the objective to make the entire Rulex platform, more reliable, worldwide available, and full-cloud compliant. He has co-authored over 20 scientific works, including international journals, conferences, and patents.

I-Chih Wang was born in Changhua, Taiwan, in 1994. He received the B.E.E. and M.Sc. degrees from National Chiao Tung University, Hsinchu, Taiwan, in 2016 and 2018, respectively. He is currently in a joint dual Ph.D. Program between National Chiao Tung University and the NYU Tandon School of Engineering, New York City, NY, USA. He is doing research about software-defined network/NFV at the Computational Intelligence on Automation Laboratory, Institution of Electrical and Computer Engineering, National Chiao Tung University, and is also doing research about V2X at the High Speed Networking Lab, NYU Tandon School of Engineering.

Anwar Walid received the B.S. and M.S. degrees in electrical and computer engineering from New York University, New York City, NY, USA, and the Ph.D. degree from Columbia University, New York City, NY, USA. He was at Nokia Bell Labs, Murray Hill, NJ, USA, as the Head of the Mathematics of System Research Department and as the Director of University Research Partnerships. He is currently the Director of Network Intelligence and Distributed Systems Research and a Distinguished Member of the Research Staff at Nokia Bell Labs. He is also an Adjunct Professor at the Electrical Engineering Department, Columbia University. He has over 20 U.S. and international granted patents on various aspects of networking and computing. His research interests are in the control and optimization of distributed systems, learning models and algorithms with applications to Internet of Things (IoT), digital health, smart transportations, cloud computing, and software-defined networking. He is a fellow of the IEEE, and an Elected Member of the International Federation for Information Processing Working Group 7.3 and the Tau Beta Pi Engineering Honor Society. He received awards from the IEEE and ACM, including the 2017 IEEE Communications Society William R. Bennett Prize and the ACM SIGMETRICS/IFIP Performance Best Paper Award. He served as an Associate Editor for the IEEE/ACM TRANSACTIONS ON CLOUD COMPUTING, *IEEE Network Magazine*, and the IEEE/ACM TRANSACTIONS ON NETWORKING. He served as the Technical Program Chair for the IEEE INFOCOM, as the General Chair for the 2018 IEEE/ACM Conference on Connected Health (CHASE), and as a Guest Editor for the IEEE IoT Journal—Special Issue on AI-Enabled Cognitive Communications and Networking for IoT.

Gordon Wilfong received the B.Sc. degree (Hons.) in mathematics from Carleton University in 1980 and the M.S. and Ph.D. degrees in computer science from Cornell University, Ithaca, NY, USA, in 1983 and 1984, respectively. He is currently a Distinguished Member of Technical Staff with the Mathematics Research Group, Nokia Bell Labs, Murray Hill, NJ, USA. His major research interests are in the design and analysis of algorithms.

Charles H.-P. Wen (M'07) received the Ph.D. degree in very-large-scale integration verification and test from the University of California, Santa Barbara, Santa Barbara, CA, USA, in 2007. He is currently an Associate Professor with National Chiao Tung University, Hsinchu, Taiwan. He is a Specialist in computer engineering. His research is focused on applying data mining and machine learning techniques to SoC designs (including radiation hardening, functional verification, and timing analysis) and cloud networking (especially on performance analysis and architecture design of large-scale datacenters). He was a recipient of the Best Paper Award from the 2012 ASP-DAC, the 2014 SASIMI, the 2016 ICOIN, and the 2017 ICOIN, and the Distinguished Young Scholar Award from the Taiwan IC Design Society.

Mario Marchese (S'94–M'97–SM'04) was born in Genoa, Italy, in 1967. He received the Laurea degree *(cum laude)* and the Ph.D. (Italian "Dottorato di Ricerca") degree in telecommunications from the University of Genoa, Italy, in 1992 and 1997, respectively. From 1999 to 2005, he was with the Italian Consortium of Telecommunications, by the University of Genoa Research Unit, where he was the Head of Research. From 2005 to 2016, he was an Associate Professor with the University of Genoa. Since 2016, he has been a Full Professor with the University of Genoa. He has authored the book *Quality of Service Over Heterogeneous Networks* (John Wiley & Sons, Chichester, 2007), and has authored or co-authored more than 300 scientific works, including international magazines, international conferences, and book chapters. His main research activity concerns: networking, quality of service over heterogeneous networks, software-defined networking, satellite DTN and nanosatellite networks, and networking security. He is the Winner of the IEEE ComSoc Award "2008 Satellite Communications Distinguished Service Award" in recognition of significant professional standing and contributions in the field of satellite communications technology. He was the Chair of the IEEE Satellite and Space Communications Technical Committee from 2006 to 2008.

H. Jonathan Chao (M'83–F'01) received the B.S. and M.S. degrees in electrical engineering from National Chiao Tung University, Taiwan, in 1977 and 1980, respectively, and the Ph.D. degree in electrical engineering from The Ohio State University, Columbus, OH, USA, in 1985. He was the Head of the Electrical and Computer Engineering (ECE) Department, New York University (NYU), New York, NY, USA, from 2004 to 2014. He has been involved in research for software-defined networking, network function virtualization, datacenter networks, high-speed packet processing/switching/routing, network security, quality-of-service control, network on chip, and machine learning for networking. From 2000 to 2001, he was the Co-Founder and the CTO of Coree Networks, Tinton Falls, NJ, USA. From 1985 to 1992, he was a Member of the technical staff at Bellcore, Piscataway, NJ, USA, where he was involved in transport and switching system architecture designs and application-specified integrated circuit implementations, such as the world's first SONET-like framer chip, ATM layer chip, sequencer chip (the first chip handling packet scheduling), and ATM switch chip. He is currently a Professor of ECE, NYU. He is also the Director of the High-Speed Networking Lab. He has co-authored three networking books, *Broadband Packet Switching Technologies—A Practical Guide to ATM Switches and IP Routers* (New York: Wiley, 2001), *Quality of Service Control in High-Speed Networks* (New York: Wiley, 2001), and *High-Performance Switches and Routers* (New York: Wiley, 2007). He holds 61 patents and has published more than 260 journal and conference papers. He is a Fellow of the National Academy of Inventors. He was a recipient of the Bellcore Excellence Award in 1987, and was a co-recipient of the 2001 Best Paper Award from the IEEE TRANSACTION ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY.