Computer Communications
=0pt
SDN CPP controllers flow-setup latency control traffic

## 1. Introduction

The booming Internet usage growth and integration of a plethora of IoT devices into the network system generates a shortage of resources in terms of network management. Furthermore, the amalgamation of different types of switches from various vendors increases the complexity of network configuration, which is already troublesome. In light of the above-mentioned predicament, a novel network architecture known as Software Defined Network (SDN) is proposed, which has been a center-stage of cutting-edge research for the past few decades. SDNs distribute the functionalities of the traditional network layer switches to simplify network management and configuration, making efficient use of scarce resources. The forwarding capabilities are retained by the switches while new agents are introduced as the decision making entities. These authoritative devices - namely *controllers*, have superior processing power and memory compared to the SDN switches, which are simplified in terms of both costs and design.

The original SDN architecture [1] includes a single controller, resulting in the formation of bottle-necks, single point of failure, and scalability issues [2, 3], when the network size increases. Consequently, the multiple-controller architecture is proposed; however, the solution comes with its own NP-Hard Controller Placement Problem (CPP). The CPP deals with placing an optimal number of controllers to optimize one or more constraints - network latencies, deployment costs, energy consumption, reliability, and resilience [4, 5, 6]. In this paper, we propose a novel controller assignment method which minimizes the following:

**(1) Latency:** We minimize two latencies simultaneously, namely, *route synchronization latency* and *flow-setup latency*. When there is a change in the network, the concerned controllers and switches are notified immediately. The delay produced here is called the route-synchronization latency. When a new flow arrives at a switch, the flow-setup process is initiated through which the corresponding controller calculates a new path and notifies concerned switches. The total delay incurred through this process is called the flow-setup latency.

**(2) Controller response time and Load imbalance:** Due to the influx of numerous new flows, several query packets from multiple switches impose on the controllers. The great volume of processing required by a controller to facilitate the smooth operation of the network is called the *load of a controller*. Imbalanced distribution of load can lead to an exponential increase in controller response times.

The processing delay of SDN switches is significantly reduced compared to traditional networks and propagation latencies are negligible due to greater wave propagation velocities. Therefore, the overall flow-setup latency relies primarily on transmission latency and queuing latency of control packets [7]. Our proposed method aims to divide the network into $k$ balanced clusters and place a controller in each cluster to reduce transmission latency. Furthermore, the controller response time is improved through dynamic load balancing, which reduces queuing latency.

Most research approaches perform optimization of a specific latency or parameter instead of providing a complete system, while a few of them address multiple parameters simultaneously [8, 9]. However, they either provide exhaustive solutions or reduce accuracy greatly to improve efficiency in terms of time and memory complexity. To the best of our knowledge, there is no such method that clusters the network, place controllers considering both inter-controller and intra-controller latency, and assigns switches dynamically to minimize flow setup latency, route synchronization latency, load imbalance, and controller response times in polynomial complexity. Therefore, we develop multiple algorithms to address the problem at hand, and our main contributions are summarized as follows:

- We minimize overall flow-setup latency and route synchronization latency while placing controllers. Furthermore, we provide a way to trade-off between inter-controller and intra-controller latencies for network managers.
- We develop three polynomial-time algorithms to cluster the network, place controllers and balance controller loads. We also suggest an optimal number of controllers.
- Our proposed method is traffic-aware and reduces controller response time by balancing controller loads in real-time. It can be improved further to work with any network parameter.
- We perform simulations on 243 existing network topologies and our simulations show that our method outperforms state-of-the-art algorithms for controller placement in terms of flow-setup latency, controller response time, and load balancing.

The remainder of our paper consists of the background and related works in Section 2, the problem formulation in Section 3, the detailed proposed mechanism in Section 4, the performance analysis in Section 5, the simulation results in Section 6, and the conclusion in Section 7.

Controller Placement (RAC) and Fast-RAC (FRAC). Yang et al. [17] introduce a greedy heuristic algorithm for single-link failures and the Monte Carlo Simulation for multi-link failures

Placing minimal controllers makes the SDN vulnerable to failures and security attacks. Contrarily, deploying maximal controllers reduces delays, response times of the controllers, and throughput of the network; it is, however, not feasible in terms of cost and inefficient in terms of resources. Sallahi et al. [9] propose a mathematical model to determine the optimal number of controllers and their locations in a network while minimizing the cost of the SDN. Chaudhary et al. [18] develop the Placement Availability Resilient Controller (PARC) scheme to provide a stable partitioning of the network, a co-operative game theory-based localization of controllers, and an optimal controller number calculation including extra backup controllers while minimizing network cost. Several research proposals also address the wireless paradigm of SDN - Dvir et al. [19] minimize propagation latency, maximize throughput, and link failure probability while placing controllers. Toufga et al. [20] propose and Integer Linear Programming (ILP) based placement method which dynamically changes controller placement based on road traffic fluctuations for Software-Defined Vehicular Networks (SDVNs). Papa et al. [21] place controllers in large-scale satellite networks to minimize the average flow setup latency with respect to varying traffic demands.

The fluctuations of switch-controller control traffic impose varying loads on the controllers. Consequently, the performance of a controller is significantly reduced when its load exceeds its processing capacity. Wang et al [22] distributes the traffic using wildcard rules on supported switches to transfer traffic to OpenFlow [23] server replicas. Yao et al. [24] define the Capacitated Controller Placement Problem (CCPP) which takes loads of controllers into consideration and also introduces an efficient algorithm to solve it. Yao et al. [25] propose a multi-controller load balancing approach called HybridFlow for software-defined wireless networks, which is a flow-based dynamic solution. Chen et al. [26] propose a load balancing scheme that provides quality of service (QoS) for machine-to-machine (M2M) networks through traffic identification and rerouting. Choumas et al. [27] assess and explore control-traffic minimization and devise multiple heuristic algorithms to propose an efficient solution. Coronado et al. [28] propose - *Wi-Balance*, an algorithm that balances traffic load and maintains an equal division of network resources. Qilin et al. [29] improve traffic scheduling by reducing the number of flow tables. Schutz et al. [30] formalizes the CPP mathematically to place controllers while keeping a balanced load distribution.

The explosive amount of data traffic generated due to the advent of the Internet of Things (IoT) and improvement of Mobile networks demands better and more efficient load balancing algorithms. Under these circumstances, multiple variations of stable matching algorithms are proposed to ensure maximum utilization of available resources. Wang et al. in [31] propose a combination of matching theory and conditional games to perform load balancing of controllers. Filali et al. [32] formulate the CCPP as a one-to-many matching game and use a well-known stable matching algorithm - Multistage Deferred Acceptance Algorithm (MSDA) [33] to solve it.

## 3. Problem Formulation and Assumptions

Systematic segregation of the network layer into control and data planes require the division of larger networks into multiple sub-networks, where each sub-network is governed by a single controller.

We represent the network as a weighted bi-directional graph $G = (S, E)$, where $S$ represents the set of switches (or nodes) and $E$ represents the set of links (or edges) between the switches. The weights of the edges represent the transmission delays, which is the reciprocal of the bandwidth of the links. The graph $G$ is clustered into $k$ mutually exclusive and collectively exhaustive sets of switches (i.e., sub-networks) denoted by $S_i$, where $i = 1, 2, ...., k$. Switches in a sub-network forward packets following a set of rules, using flow-tables. However, when a *new flow* arrives, the source switch sends a query packet to the controller of the sub-network. The controller notifies the concerned switches in the path about the new rule. It also notifies the controller of a switch in the path if the switch is not in its own sub-network. The flow setup procedure for a switch can be divided into three phases, 1) the switch sends a query packet to the its controller, 2) after a period of queuing delay (if any), the controller processes the query and takes a routing decision, and finally, 3) the controller informs the new rule to all the switches in the path. The total time required to complete all the three phases is called the *flow-setup latency*.

The delay incurred by the first phase of the flow-setup procedure depends on the link between the concerned switch and controller. Contrarily, the delay in the second phase depends on the time that a controller takes to take a routing decision+, and is called the *controller response time*. The response time can be calculated if the *loads* of the switches and the controllers, and the *processing power* of the controllers are known. The total number of query packets generated by a switch in a unit time is the load of the switch (hereafter referred to as *switch-load*) and the total number of query packets generated by all the switches in a sub-network is the load of the controller (hereafter referred to as *controller-load*).

We assume that both the arrival times and the service times of the queries follow a Poisson process like [31]. Therefore, the delays associated with the queries at the controller can be modeled as an M/M/1 queue and the

average system time (i.e., the summation of waiting time and service time) can be determined using Little's Law as follows [31]:

$$RT_{avg} = \frac{1}{(Power_{C_i} - Load_{C_i})} \tag{1}$$

where $Power_{C_i}$ and $Load_{C_i}$ of any controller $C_i$ in the network are the maximum processing power and the load of the controller, respectively .

When all the controllers are working at their maximum capacity, the imbalance of the network load does not play a role here. When one switch is overloaded and other switches

For a heavily loaded network the controllers are processing at or close to their maximum capacity. Consequently, a balance or imbalance of loads do not play a vital role to reduce controller response times. However, when some controllers are overloaded and others are underloaded, balancing their loads can improve both the waiting time and processing time. In this case, assigning loads to controllers according to their processing capacity can significantly improve their response times.

The third phase of the flow-setup procedure depends on both the controller-switch and controller-controller latencies of the involved controllers and their respective switches that are in the flow-path. Furthermore, the time required for a controller to announce any changes among the switches in its sub-network, is called the route-synchronization latency, which is also determined by the controller-switch latency. Therefore, our goal is to divide the network into clusters while,

1. placing controllers to minimize the overall flow-setup and route synchronization latencies, and
2. dynamically balancing the loads among the controllers to reduce average controller response times.

For simplicity, we assume that the processing delay of all switch-to-controller control packets is negligible. Furthermore, a controller can only be placed on a switch location and all the controllers have identical processing capacity.

## 4. Proposed Methodology

In our proposed method, we develop three algorithms - Latency-based Clustering Algorithm (LCA), Latency-based Controller Selection Algorithm (LCSA), and Best-first-search Load Balancing Algorithm (BLBA). LBCA clusters the network into a given number of sub-networks ($k$) and CSA places a controller in each of the sub-networks, resulting in a static controller-switch assignment. BLBA is a dynamic load balancing algorithm that periodically reassigns switches to avoid over-burdening a controller.

### 4.1. Latency-based Clustering Algorithm (LCA)

The Latency-based Clustering Algorithm (LCA) clusters the network in two phases: the first phase selects the cluster-centers and the second phase forms the clusters surrounding the cluster-centers.

#### 4.1.1. Cluster-center selection

We determine footholds from which a cluster will be created, before forming a cluster, to ensure that the clusters are not concentrated in a certain region. Consequently, we re-cluster the network to avoid forming irregular clusters in terms of geographical shape and size. In [34], DBC places controllers based on inter-controller and intra-controller distances, while ensuring that these footholds, namely cluster-heads, are a minimum distance ($T_d$) apart from each other. However, the cluster-heads are randomly selected and the $T_d$ distance is calculated virtually. In this paper, we determine the foundations of the clusters based on their average distance from all other nodes. Accordingly, we name the foundations as *cluster-centers* and propose a novel clustering algorithm LCA.

When a network is managed by a single controller, it should be placed at the center of the network to minimize the controller-to-switch delay. If $G$ represents such a network then the center of $G$ can be determined by minimizing the average or maximum delay from every other node [35]:

$$center_{avg} = \min_{s \in S} \left( \frac{\sum_{d \in S, d \neq s} T(s,d)}{|S| - 1} \right) \tag{2}$$

$$center_{max} = \min_{s \in S} \left( max_{d \in S} \big( T(s,d) \big) \right) \tag{3}$$

where, $center_{avg}$ and $center_{dia}$ are the centers $G$ in terms of average and maximum delays, respectively, and $T(s,d)$ is the shortest path distance from nodes $s$ to $d$. The maximum distance or worst-case latency (Equation

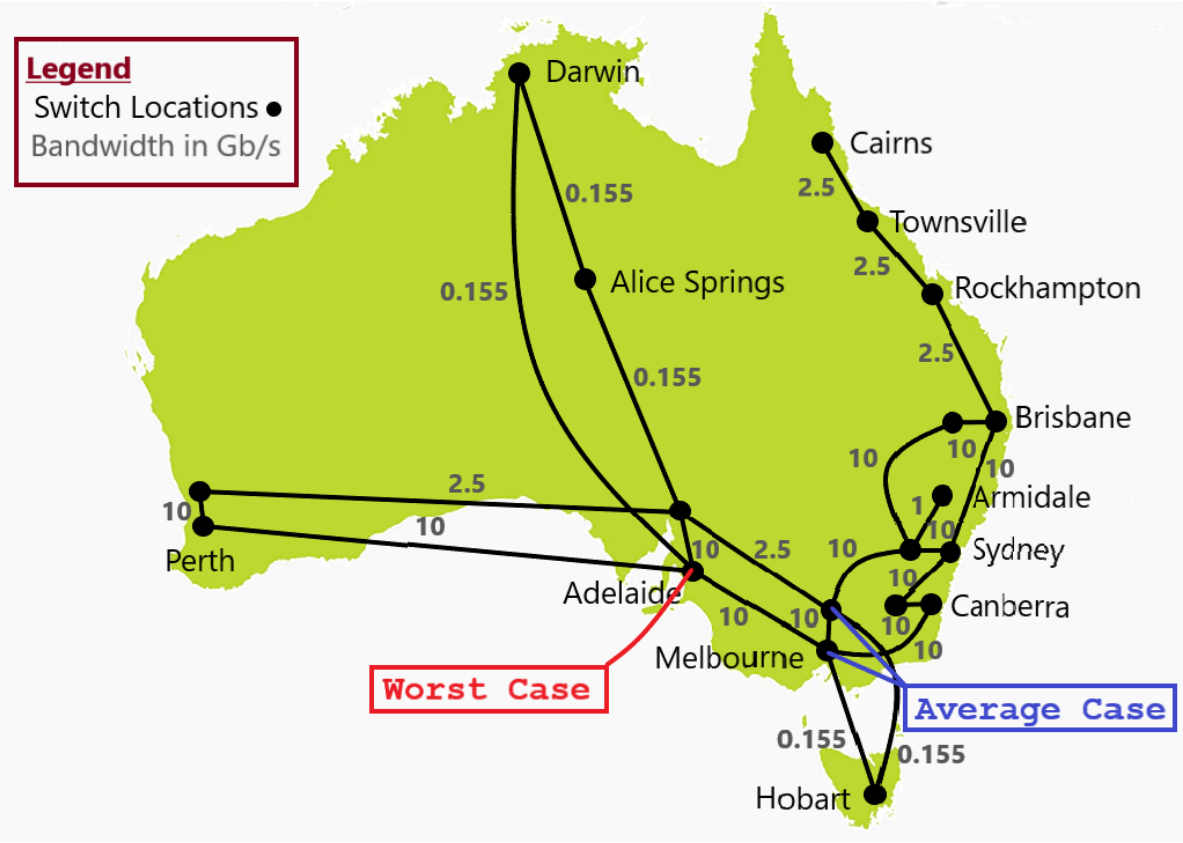



**Figure 1**: Existing network of Australia (AARNET [36]) showing the centers of the network.

3), when determines the center of the network [35], can vary greatly in the presence of distant and isolated nodes (Figure 1). Conversely, the average case latency (Equation 2) is less affected by outliers, therefore, we select the node with the minimum average distance ($center_{avg}$) as the center of the network [35].

For larger networks, when placing multiple controllers, each controller is placed in a sub-network which makes the center selection process complex. To ensure equal division of the network, the cluster-centers are expected to be equidistant from each other and the clusters must expand from the centers. This selection of centers in a network is a classical NP-hard problem - *The vertex k-center problem* [37], which has many sub-optimal approximations. Our algorithm LCA (Algorithm 1) utilizes the latencies between switches to divide the network optimally while ensuring minimum overlapping among neighboring clusters.

Initially, we calculate the delays between every possible pairs of nodes in terms of shortest source to destination path delays (Algorithm 1 Line 1). Using the attained delays, we determine the average delay $\overline{T}_{s \in S}$, of every node $s$ from every other node in the network $S$ (Algorithm 1 Line 2). Consequently, we select the first cluster-center with the minimum average delay and form a cluster including the nearest $\frac{|S|-1}{k}$ nodes in terms of hop-distance. The delays of the nodes of this cluster are not considered in the calculation of the average delays of the remaining nodes (Algorithm 1 Line 8). We select the node with the lowest average delay among the remaining nodes and include $\frac{|S|-1}{k}$ nodes again to this newly formed cluster. We continue the process until $k$ cluster-centers are selected.

#### 4.1.2. Clustering-member selection

The cluster-centers of the previous phase are used to form the clusters in this phase. Each node of the network $G$ is included in the cluster of the nearest cluster-center in terms of the shortest path latency $T(i,j)_{i,j \in S}$. However, the shortest path is in terms of transmission delay instead of hop-count. The cluster-member selection is node after the cluster-center selection to avoid the formation of overlapping or isolated clusters.

### 4.2. Controller Selection Algorithm (CSA)

The Controller Selection Algorithm (CSA) selects a controller for each cluster and allows the network administrator to place controllers while prioritizing certain parameters.

Ideal controller positions have minimum distance from each other and the switches of the network in terms of

**Algorithm 1:** Latency-based Clustering Algorithm (LCA)

**Result:** Set of Cluster-Centers, $CC$

$T :=$ *shortest path delay between two switches*;
for all $s \in S$, $\overline{T}_s := \frac{\sum_{s,d \in S, d \neq s} T(s,d)}{|S|-1}$;
$CC := \emptyset$, $k :=$ required number of controllers;
**while** $|CC| < k$ **do**
    $CC := CC \cup \{s_{cc}\}$ for all $s_{cc} \in S$, where $\overline{T}_{s_{cc}} \leq \overline{T}_{t \in S}$;
    Create a new cluster $S_i$ which consists of $s_{cc}$ and $\left(\frac{|S|}{k} - 1\right)$ nearest neighbors of $s_{cc}$ in terms of hop distance;
    **foreach** *switch* $s_i \in S_i$ **do**
        Subtract $\frac{T(s_i,s)}{|S|-1}$ from $\overline{T}_s$ for all $s \in S - S_i$
    **end**
    $S = S - S_i$
**end**
Form clusters $S_{i=1}^{k}$, each containing a cluster-center $CC_i$ and all its nearest nodes;

**Algorithm 2:** Controller Selection Algorithm (CSA)

**Result:** Set of Controllers, $C$

$dis :=$ *all possible node pair shortest distances*;
$CC_{i=1}^{k} :=$ **LBCA** Cluster Centers;
$\phi_{s \in S}(s) :=$ intra-cluster latency of $s$;
$\sigma_{s \in S}(s) :=$ inter-cluster latency of $s$;
Form clusters $S_{i=1}^{k}$, each containing a cluster-center $CC_i$ and all its nearest nodes;
**foreach** *node* $s \in S$ **do**
    $\phi(s)_{s \in S_i} = \frac{\sum_{t \in S_i} dis(s,t)}{|S_i|}$;
    $\sigma(s)_{s \in S_i} = \frac{\sum_{t \in (S - S_i)} dis(s,t)}{|S - S_i|}$;
**end**
**foreach** *node* $s$ *in cluster* $S_i \subset S$ **do**
    $s_c := \min\left(\left(\phi(s) \times \overline{\sigma} \times \alpha\right) + \left(\sigma(s) \times \overline{\phi} \times (1 - \alpha)\right)\right)$;
    $C := C \cup \{s_c\}$;
**end**

propagation latency. However, both controller-to-controller and controller-to-switch latencies cannot be minimized simultaneously. Furthermore, the distance from a controller to any other controller or switch cannot be calculated without placing a controller beforehand, which results in a paradoxical scenario. Therefore, we utilize a controller selection method that replaces inter-controller and intra-controller delays with inter-cluster ($d$) and intra-cluster ($r$) latencies, respectively [34]. A normalized constant $\alpha$, is introduced to control their priority when selecting controller positions. Consequently, the controller position ($C_i$) for a cluster $S_i$, is calculated as follows:

$$r_{\forall s \in S_i} = \frac{1}{|S_i|} \sum_{u \in S_i} T(s,u) \tag{4}$$

$$d_{\forall s \in S_i} = \frac{1}{|S - S_i|} \sum_{v \in (S - S_i)} T(s,v) \tag{5}$$

$$D_i = \min\left(\left(r \times \beta_1 \times \alpha\right) + \left(d \times \beta_2 \times (1 - \alpha)\right)\right) \tag{6}$$

Here, $\beta_1$ is the normalization constant for intra-cluster latencies ($r$) and $\beta_2$ is the normalization constant for inter-cluster latencies ($d$). The normalization constants depend on average inter-cluster and intra-cluster latencies following the equations $\beta_1 =$.

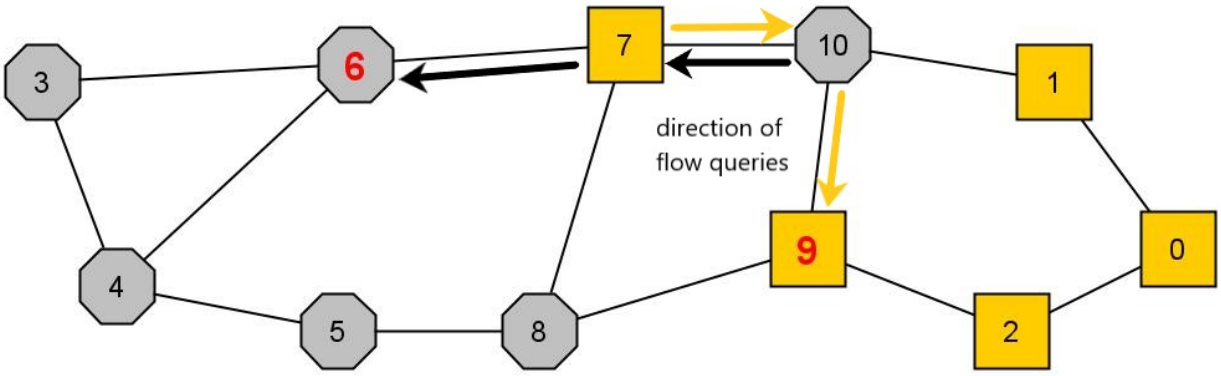



**Figure 2**: Redundant traffic is generated for new flow queries of switches 7 and 10 to controllers 9 and 6 respectively, which can be avoided by swapping the clusters of the switches

Here, $\overline{\phi}$ and $\overline{\sigma}$ are the mean intra-cluster and inter-cluster distances, respectively. The highest possible value of $\alpha$ is 1, which nullifies the effect of $\sigma$ and places controllers considering only intra-cluster distances. Meanwhile, for $\alpha = 0$, controllers are placed solely considering inter-cluster distances, and for $\alpha = 0.5$, both intra-cluster and inter-cluster distances are prioritized equally. The value of $\alpha$ can be changed to better suit the requirement of the network administrator.

### 4.3. Best-first-search Load Balancing Algorithm (BLBA)

For a fixed controller-switch assignment scheme ($S \rightarrow C$), the loads of the controllers vary due to changing loads of the switches. However, once a controller is placed, changing its position is both costly and inefficient. To balance the constantly changing loads of the controllers, we propose a dynamic load balancing algorithm Best-first-search Load Balancing (BLBA, Algorithm 3) which uses the well-known Uninformed Search technique - Best-First Search (BFS). BLBA considers each possible controller-switch assignment scheme as a separate network state and the target is to reach the optimal state, where the loads of the controllers are balanced. Assuming that each controller has identical processing capacity and each switch has a varying load, we denote the loads of the switches as $l_1, l_2, \ldots$ and so on. The total load of the network $S$, at any state, can be calculated as,

$$L_{net} = \sum_{i=1}^{|S|} l_i \tag{7}$$

Therefore, the load of a controller in an ideally load-balanced network, in the target state should be $\frac{L_{net}}{k}$, where $k$ is the number of controllers. However, when switches are assigned to distant controllers to maintain an ideal load distribution, excess traffic may be generated. For example as shown in Figure 2. The extra traffic contributes to an overall increase in latency and a decrease in network throughput. To avoid such overlapping of clusters, we prioritize distance over load-balancing and express the problem as a Heuristic (Informed) Search with the following foundations:

#### 4.3.1. Search Space and Root State

The entire search space is a graph where each node corresponds to a *state*. A state is a valid controller-switch assignment scheme, where there are multiple controllers, and every switch is assigned to a single controller. The *Root state* of the graph is the resultant assignment of switches when LBCA and CSA are applied on a network.

#### 4.3.2. Child State

All the children of any node of the search space graph must be valid *states*. Any *Child state* is similar to its parent state except one switch, which is at the border of any cluster and is reassigned to the controller of the adjacent cluster. All such combinations constitute the set of Child states of any parent state.

#### 4.3.3. Error Function

The error function $\varepsilon$ determines the acceptability of the current *state* and can vary according to requirement. The function to minimize the maximum average controller response time is calculated as follows:

$$\varepsilon(state) = \sum_{i=1}^{k} \left[ \left( L_i - \frac{L_{net}}{k} \right)^2 \right] \tag{8}$$

$$L_i = \sum_{s \in S_i} l_s \tag{9}$$

where, $L_i$ is the load of the $i^{th}$ controller, which is the cumulative load of the switches assigned to it. Squaring the differences gives more priority to controllers whose loads are more imbalanced compared to others.

#### 4.3.4. Heuristic

Our ultimate goal is not an absolute and optimal solution, rather a local optimum where no clusters are overlapping. Therefore, we use a greedy heuristics for its simplicity and efficiency as its only disadvantage is it provides a sub-optimal solution. We select the above-mentioned error function as the heuristic and we use the Best First Search (BFS) technique with pruning to solve this problem, which selects the child state with the least error as the new assignment scheme in each iteration. When there is no child state with less error compared to the Root state (Figure 3b), the child states are pruned to minimize computational complexity. Conversely, all the child states with equal or more error compared to the parent state are pruned at each iteration and the algorithm continues until the target state or a controller-switch assignment scheme with minimum error is achieved (Figure 3c). Extensive simulations suggest that the algorithm converges after a few iterations.

**Algorithm 3:** Best-first-search Load Balancing Algorithm (BLBA)

```
   Result: Assignment of Switches, S → C
1  S_{i=1}^{k} := CSA Clusters;
2  state := S → C, current assignment of switches;
3  Set of all possible new states, Pstates := {state};
4  while Pstates ≠ ∅ do
5  |  state := min(ε(Pstates));
6  |  Pstates := ∅;
7  |  foreach border switch s ∈ S do
8  |  |  New assignment Nstate := state;
9  |  |  Change assignment of switch s to controller of adjacent cluster in Nstate;
10 |  |  if ε(Nstate) < ε(state) then
11 |  |  |  Pstates := Pstates ∪ {Nstate};
12 |  |  end
13 |  end
14 end
```

## 5. Performance Analysis and Evaluation

The proposed mechanism clusters the network, places controllers, and performs load balancing on the clustered network after placement. The following sections 5.1 and 5.2 give a detailed description of the simulation environment and the performance metrics. In Section 5.3 optimum values for the decision variables, $k$ and $\alpha$ are determined. Subsequently, in sections 6.1 and 6.2, our proposed algorithms are validated by comparing them with the state-of-the-art controller placement [38] and load balancing algorithms [32], respectively.

### 5.1. Simulation Environment

The simulation environment is developed using a high-level language C++, to perform experiments on existing network topologies collected from the Internet Topology Zoo [36]. The Topology Zoo contains a total of 261 existing networks, out of which a few (18) have islands (isolated nodes). Therefore, we perform our simulations on the remaining networks. A summary of the experimental networks is given in Table 1.

The initial weights of the links are their bandwidths in *Gb/s* (Gigabits per second), which are converted into transmission latencies (milliseconds), assuming each control packet is 1500 bytes long. The maximum bandwidth

**Table 1**
A summary of our Experimental Networks

| Category | Data |
|---|---|
| Total number of networks | 243 |
| Number of unweighted networks | 134 |
| Maximum number of nodes | 754 |
| Minimum number of nodes | 4 |
| Networks with multi-edges | 82 |
| Average Edge per Node | 1.285 |

is considered for links with variable bandwidths and all fiber-optic cables without available bandwidth information are assumed to have 1 $Gb/s$ bandwidth. The networks with identical edge weights are considered as unweighted.

We perform our load balancing simulations on the network with the highest number of nodes ($|S| = 754$) to illustrate the performance of BLBA. The switches are assumed to have loads of 1000 to 5000 in terms of active flows per second, which is equivalent to the loads of data-center switches [39]. We assume the network has 10 controllers that are placed using LBCA and CSA, and each controller has a maximum processing capacity of $1000K$ $flows/s$ which is adequate to support the maximum load of the networks in our simulations.

## 5.2. Performance Metrics

SDN switches match every incoming packet with appropriate flow tables. The result of a match can either be a *hit* - which means the appropriate flow is already in a flow table, or, a *miss* - in which case the flow is new and the switch asks its assigned controller for the next course of action. The assigned controller calculates the path with the least delay for the flow and notifies the concerned switches to update their flow tables. However, for switches assigned to another controller, the corresponding controller is notified instead (Figure 4). Therefore, the total time required to notify all the concerned switches about the new flow is the *flow-setup latency*. We represent the average flow-setup latency ($\Omega_{avg}$) of the network as the average time to notify all possible pairs of switches in the network. For a network with $|S|$ switches, the average flow-setup latency can be calculated as follows:

$$\Omega_{avg} = \frac{\sum_{s_i,s_d \in S} \left(dis(s_i, c_i) + maxPath(s_i, s_d)\right)}{|S| \times |S-1|} \tag{10}$$

$$maxPath(s_i, s_d) = \max_{x \in path(i,d)} \left(dis(c_i, c_x) + dis(c_x, s_x)\right) \tag{11}$$

where, $path(i, d)$ is the path with least delay from source $s_i$ to destination $s_d$, and $s_x$ is any switch in that path. The controllers of switches $s_i$ and $s_x$ are $c_i$ and $c_x$ respectively.

The processing latency of the controllers increases significantly for an imbalanced network, which can be calculated if the loads of the switches and controllers are known. Accordingly, the overall flow setup latency for a network $S$ is the maximum time any switch needs to install a new-flow rule, including controller processing delays, and is denoted by:

$$\Omega_S = \max_{s_i \in S} \left(\Delta_{s_i} \times l_{s_i}\right) \tag{12}$$

where $\Delta_{s_i}$ is the average time required for a switch $s_i$ to add a new flow to its flow table and $l_{s_i}$ is the switch load. Using the average flow-setup latency (Equation 10) and average controller response times (Equation 1) of a network, $\Delta_{s_i}$ is derived as follows:

$$\Delta_{s_i} = \frac{\sum_{s_i,s_d \in S} \left[dis(s_i, c_i) + RT_{c_i} + maxPath(s_i, s_d)\right]}{|S| - 1} \tag{13}$$

$$maxPath(s_i, s_d) = \max_{s_j \in path_{id}} \{dis(c_i, c_j) + RT_{c_j} + dis(c_j, s_j)\} \tag{14}$$

where $RT_{c_i}$ and $RT_{c_j}$ are the average controller response times for controllers $c_i$ and $c_j$ respectively.

### 5.3. Decisive Variables

LBCA clusters the network into $k$ sub-networks and CSA places a controller in each sub-network using the constant $\alpha$, which is a real number ranging from 0 to 1. The constant dictates the placement of controllers by controlling the priority of intra-cluster and inter-cluster distances. Applying LBCA on a small network with varying values of $k$ and $\alpha$ (Figure 5) provides valuable insight on how the two variables affect the overall flow-setup latency of a network (Equation 10).

An increased number of clusters are formed when an excessively large amount of controllers are placed in a network. Consequently, the cluster-sizes are diminished, greatly reducing the number of viable controller positions in a specific cluster. Therefore, for a specific and large value of $k$, varying the value of $\alpha$ causes little or no change in placement and has no effect on overall flow-setup latency ($k = 6$). For smaller values of $k$, the flow-setup latency gradually decreases and then increases ($k < 5$). In some cases, the latency only increases as inter-controller distances increase ($k = 5$), which indicates that no better placement is found for greater controller separation. The flow-setup latency is the lowest when $\alpha \geq 0.2$ and $\alpha \leq 0.6$, which indicates that inter-controller communication contributes more to the flow-setup procedure when there are many controllers. However, in our experiments, we have used $\alpha = 0.5$ to give equal priority to controller-to-switch and controller-to-controller communication.

The flow-setup latency of a network decreases as the number of controllers increase, with a few exceptions due to variations in network topology (Figure 6). The rate of decrease is greater for smaller networks compared to larger networks as controller/switch ratio increase drastically for smaller networks. Accordingly, the setup latency is minimum when $k$ is equal to the total number of nodes in a network, which, however, invalidates one of the firsthand benefits of placing controllers (simplifying nodes and reducing costs). In order to determine the optimum number of controller $k$ for a network $S$, we define an improvement ratio:

$$Improvement\ Ratio_k = \frac{Latency_1}{Latency_k \times k} \tag{15}$$

where $Latency_1$ and $Latency_k$ are the flow-setup latencies when the number of controllers is 1 and $k$ respectively.

The improvement ratio for a network decreases gradually with respect to increasing number of controllers (Figure 7). We observe that the improvement rate *change* decreases drastically to less than 0.1 from more than 0.2 and 0.15, after adding 4 controllers to the network where $|S| = 64$ and 3 controllers to the other two networks, respectively. Therefore, we cease adding controllers when the improvement rate change drops below 0.1. Consequently, the resultant average switch/controller ratio of the 238 networks is 12.79. According to expert opinions [10], a network with 34 nodes requires approximately 3 controllers to function efficiently and one more to handle failures, which supports our optimal $k$ derivation in terms of average switch/controller ratio.

## 6. Simulation Results

### 6.1. Controller Placement

In this section, we evaluate our static controller placement method (LBCA+CSA) by comparing it to the well-known algorithm DBCP [38] and the algorithm DBC [34]. DBCP places controllers based on the density of nodes and the minimum distance to higher density nodes. In order to compare the algorithms, we simulate both DBC and LBCA with the same number of controllers as DBCP when clustering the networks from the Zoo Topology. DBCP underperforms compared to LBCA and DBC when the network has high connectivity (e.g. star topology) or when all nodes have an equal density (e.g. ring topology). Our simulations using the remaining networks suggest that LBCA+CSA outperforms both DBCP and DBC in terms of flow-setup latencies.

The flow-setup latency results of the simulations vary greatly for different networks. Therefore, we represent the latencies of DBCP and DBC as a ratio of the latencies of LBCA+CSA and plot their averages for a given range of network sizes (Figure 8). The average latency of all the networks with network sizes less than 10 are plotted for $|S| = 5$, those greater than 9 and less than 20 are plotted for $|S| = 15$, and so on. Although DBC performs better for certain unweighted networks, LBCA+CSA outperforms both DBCP and DBC in 78% and 72% of the simulated networks, respectively.

### 6.2. Load Balancing

BLBA balances the loads of the controllers by swapping clusters of border nodes to avoid overlapping cluster formations. We set the maximum iteration limit of BLBA to 100 and compare with the algorithm MSDA [32, 33] in terms of maximum load imbalance. The loads of the switches are randomly assigned within the range of 1000 to 5000 with equal probability. Therefore, the average switch load is $2500 flows/s$ and the target controller load is $188.5 K flows/s$ (Figure 9). The global precedence list of MSDA is calculated by multiplying transmission latency with average sojourn time or controller processing time. However, in our experiments, we observe that transmission

latencies are greater than processing latencies, which results in load imbalance. Furthermore, when the maximum controller capacity is decreased substantially, MSDA underperforms as the preferences of the controllers and switches cannot be satisfied. The simulation suggests that BLBA outperforms MSDA in terms of controller load, especially for controllers 8 and 9 (Figure 9). Although the algorithm converges after 10 iterations approximately, the maximum iteration limit can be increased for better accuracy and performance.

Figure 10 represents the comparison between BLBA and MSDA in terms of maximum controller response times for different average switch loads. The response times increase with increasing switch loads as the controller loads also increase significantly. The comparison shows that BLBA outperforms MSDA in terms of average controller response times. The average controller response time is reduced by an average of 13%.

We also compare both the algorithms in terms of maximum flow setup latency of a switch in Figure 11 and observe that BLBA outperforms MSDA. The flow-setup latency for higher average switch loads shows the minimum change for MSDA as the controller-switch preferences are prioritized. BLBA on the other hand reduces latency as long as a local optimum is available. Consequently, the algorithm terminates if further optimization causes overlapping clusters, which is observed for an average switch load of 3500 flows/s as it produces unnecessary traffic (Figure 2).

## 7. Conclusion

In this paper, we propose a novel controller assignment mechanism that clusters the network, places controllers, and balances loads to reduce the overall flow-setup latency of the network. Our proposed method addresses the Controller Placement Problem (CPP) and outperforms multiple state-of-the-art algorithms based on various performance metrics. Our proposed method has many advantages over other algorithms which include - having polynomial time complexity, providing an optimal number of clusters, decreasing controller response time, and flow-setup latency simultaneously. Our proposed algorithm BLBA can be extended to optimize any parameter by introducing and improving different error functions. Future work can include variable controller capacities when balancing controller loads. Our proposed method can also be extended to facilitate simultaneous optimization of several core network parameters, which will be significantly helpful for network operators.

## References

## References

[1] K. Greene, Tr10: Software-defined networking, Technology Review (MIT) (2009).

[2] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, R. Kompella, Towards an elastic distributed sdn controller, in: ACM SIGCOMM Computer Communication Review, Vol. 43, ACM, 2013, pp. 7–12.

[3] S. H. Yeganeh, A. Tootoonchian, Y. Ganjali, On scalability of software-defined networking, IEEE Communications Magazine 51 (2) (2013) 136–141.

[4] Y. Zhang, L. Cui, W. Wang, Y. Zhang, A survey on software defined networking with multiple controllers, Journal of Network and Computer Applications 103 (2017) 101–118.

[5] A. K. Singh, S. Srivastava, A survey and classification of controller placement problem in sdn, International Journal of Network Management 28 (2018) e2018.

[6] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, H. L. Owen, Advancing software-defined networks: A survey, IEEE Access 5 (2017) 25487–25526.

[7] A. B. Forouzan, Data communications and networking (sie), 4th Edition, Tata McGraw-Hill Education, 2017.

[8] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, M. Hoffmann, Heuristic approaches to the controller placement problem in large scale sdn networks, IEEE Transactions on Network and Service Management 12 (1) (2015) 4–17.

[9] A. Sallahi, M. St-Hilaire, Optimal model for the controller placement problem in software defined networks, IEEE communications letters 19 (1) (2015) 30–33.

[10] B. Heller, R. Sherwood, N. McKeown, The controller placement problem, in: Proceedings of the first workshop on Hot topics in software defined networks, ACM, 2012, pp. 7–12.

[11] Y. Li, W. Sun, S. Guan, A multi-controller deployment method based on pso algorithm in sdn environment, in: 2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), Vol. 1, IEEE, 2020, pp. 351–355.

[12] T. Das, M. Gurusamy, Controller placement for resilient network state synchronization in multi-controller sdn, IEEE Communications Letters (2020).

[13] Y. Hu, W. Wendong, X. Gong, X. Que, C. Shiduan, Reliability-aware controller placement for software-defined networks, in: Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on, IEEE, Ghent, Belgium, 2013, pp. 672–675.

[14] Y. Zhang, N. Beheshti, M. Tatipamula, On resilience of split-architecture networks, in: Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE, IEEE, Kathmandu, Nepal, 2011, pp. 1–6.

[15] M. Ashrafi, A.-T. Faroq, N. Correia, Placement of controllers in software defined networking under multiple controller mapping, KnE Engineering (2020) 394–404.

[16] Y. Fan, T. Ouyang, X. Yuan, Controller placements for improving flow set-up reliability of software-defined networks, in: Urban Intelligence and Applications, Springer, 2020, pp. 3–13.

[17] S. Yang, L. Cui, Z. Chen, W. Xiao, An efficient approach to robust sdn controller placement for security, IEEE Transactions on Network and Service Management (2020).

[18] R. Chaudhary, N. Kumar, Parc: Placement availability resilient controller scheme for software-defined datacenters, IEEE Transactions on Vehicular Technology (2020).

[19] A. Dvir, Y. Haddad, A. Zilberman, The controller placement problem for wireless sdn, Wireless Networks 25 (8) (2019) 4963–4978.

[20] S. Toufga, S. Abdellatif, H. T. Assouane, P. Owezarski, T. Villemur, Towards dynamic controller placement in software defined vehicular networks, Sensors 20 (6) (2020) 1701.

[21] A. Papa, T. de Cola, P. Vizarreta, M. He, C. Mas-Machuca, W. Kellerer, Design and evaluation of reconfigurable sdn leo constellations, IEEE Transactions on Network and Service Management (2020).

[22] R. Wang, D. Butnariu, J. Rexford, et al., Openflow-based server load balancing gone wild., Hot-ICE 11 (2011) 12–12.

[23] F. Hu, Q. Hao, K. Bao, A survey on software-defined network and openflow: From concept to implementation, IEEE Communications Surveys & Tutorials 16 (4) (2014) 2181–2206.

[24] G. Yao, J. Bi, Y. Li, L. Guo, On the capacitated controller placement problem in software defined networks, IEEE Communications Letters 18 (8) (2014) 1339–1342.

[25] L. Yao, P. Hong, W. Zhang, J. Li, D. Ni, Controller placement and flow based dynamic management problem towards sdn, in: Communication Workshop (ICCW), 2015 IEEE International Conference on, IEEE, London, UK, 2015, pp. 363–368.

[26] Y.-J. Chen, L.-C. Wang, M.-C. Chen, P.-M. Huang, P.-J. Chung, Sdn-enabled traffic-aware load balancing for m2m networks, IEEE Internet of Things Journal 5 (3) (2018) 1797–1806.

[27] K. Choumas, D. Giatsios, P. Flegkas, T. Korakis, Sdn controller placement and switch assignment for low power iot, Electronics 9 (2) (2020) 325.

[28] E. Coronado, J. Villalón, A. Garrido, Wi–balance: Sdn–based load–balancing in enterprise wlans, in: 2017 IEEE Conference on Network Softwarization (NetSoft), ieee, 2017, pp. 1–2.

[29] M. Qilin, S. Weikang, A load balancing method based on sdn, in: 2015 Seventh International Conference on Measuring Technology and Mechatronics Automation, Vol. 1, IEEE, 2015, pp. 18–21.

[30] G. Schütz, J. Martins, A comprehensive approach for optimizing controller placement in software-defined networks, Computer Communications (2020).

[31] T. Wang, F. Liu, J. Guo, H. Xu, Dynamic sdn controller assignment in data center networks: Stable matching with transfers, in: IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE, 2016, pp. 1–9.

[32] A. Filali, A. Kobbane, M. Elmachkour, S. Cherkaoui, Sdn controller assignment and load balancing with minimum quota of processing capacity, in: 2018 IEEE International Conference on Communications (ICC), IEEE, 2018, pp. 1–6.

[33] D. Fragiadakis, A. Iwasaki, P. Troyan, S. Ueda, M. Yokoo, Strategyproof matching with minimum quotas, ACM Transactions on Economics and Computation (TEAC) 4 (1) (2016) 1–40.

[34] T. I. Aziz, S. Protik, M. S. Hossen, S. Choudhury, M. M. Alam, Degree-based balanced clustering for large-scale software defined networks, in: 2019 IEEE Wireless Communications and Networking Conference (WCNC), IEEE, 2019, pp. 1–6.

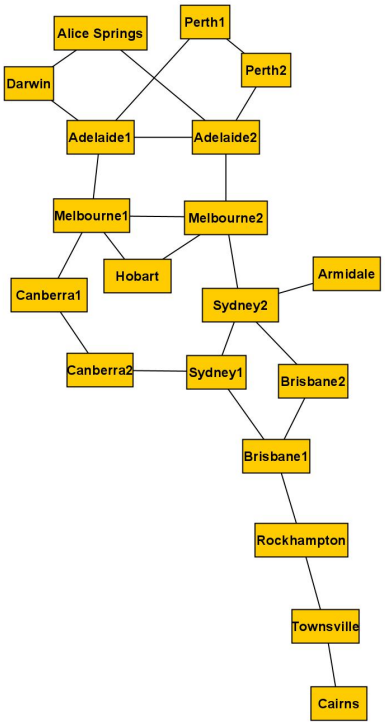
[35] R. J. Wilson, Introduction to graph theory, Pearson Education India, 1979.

[36] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, M. Roughan, The internet topology zoo, IEEE Journal on Selected Areas in Communications 29 (9) (2011) 1765–1775.

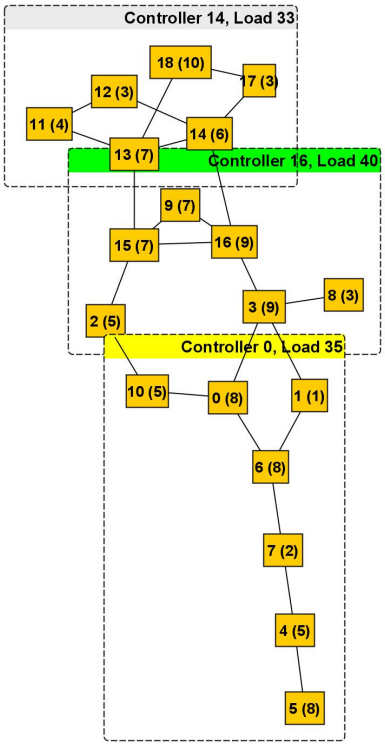
[37] O. Kariv, S. L. Hakimi, An algorithmic approach to network location problems. i: The p-centers, SIAM Journal on Applied Mathematics 37 (3) (1979) 513–538.

[38] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li, T. Li, Density cluster based approach for controller placement problem in large-scale software defined networkings, Computer Networks 112 (2017) 24–35. doi:10.1016/j.comnet.2016.10.014.

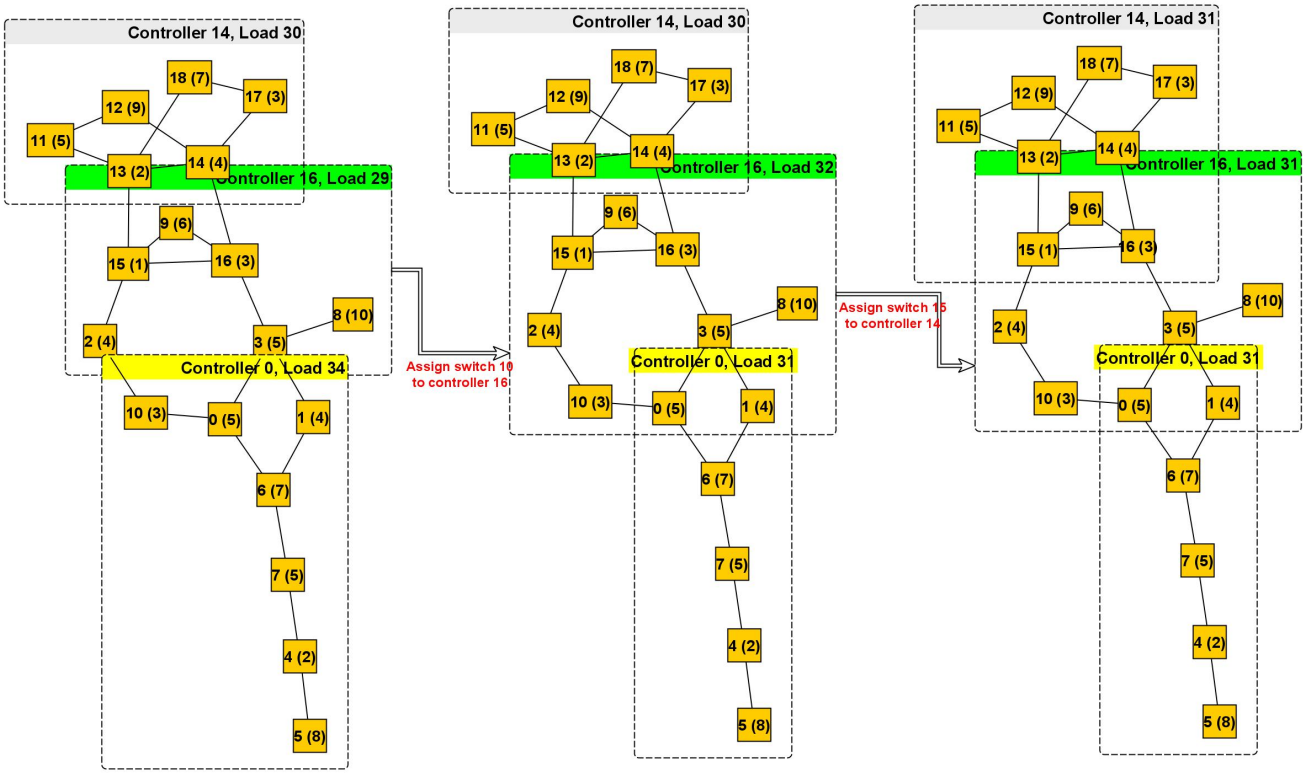
[39] T. Benson, A. Akella, D. A. Maltz, Network traffic characteristics of data centers in the wild, in: Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, 2010, pp. 267–280.




(a) Original Aarnet Network (rearranged)




(b) Root state with no child state




(c) Optimal assignment scheme achieved after three iterations

**Figure 3**: The BLBA algorithm applied on Aarnet Network variations with randomized loads, where each switch is represented by a switch id, followed by its load and each cluster has its own controller position and load marked above.

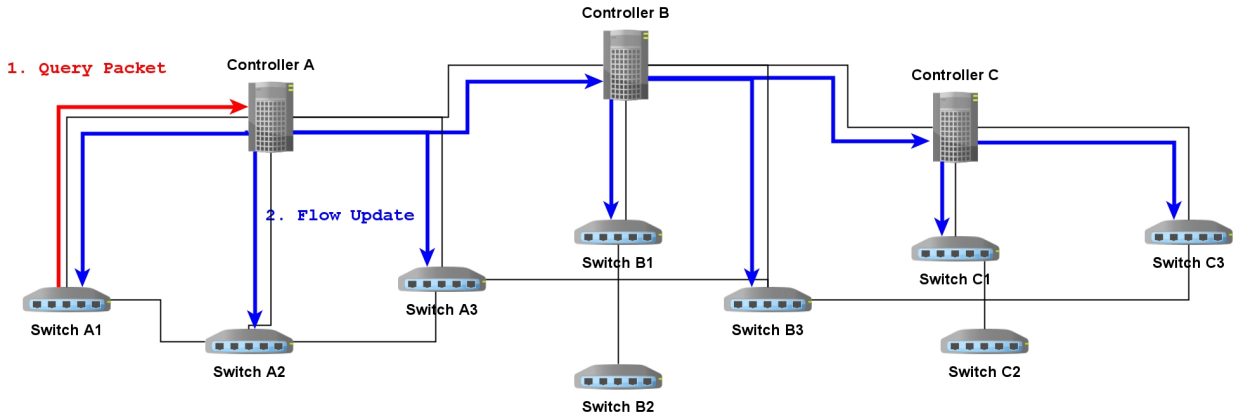



**Figure 4**: Setup of a new flow in the flow tables of the switches in the network

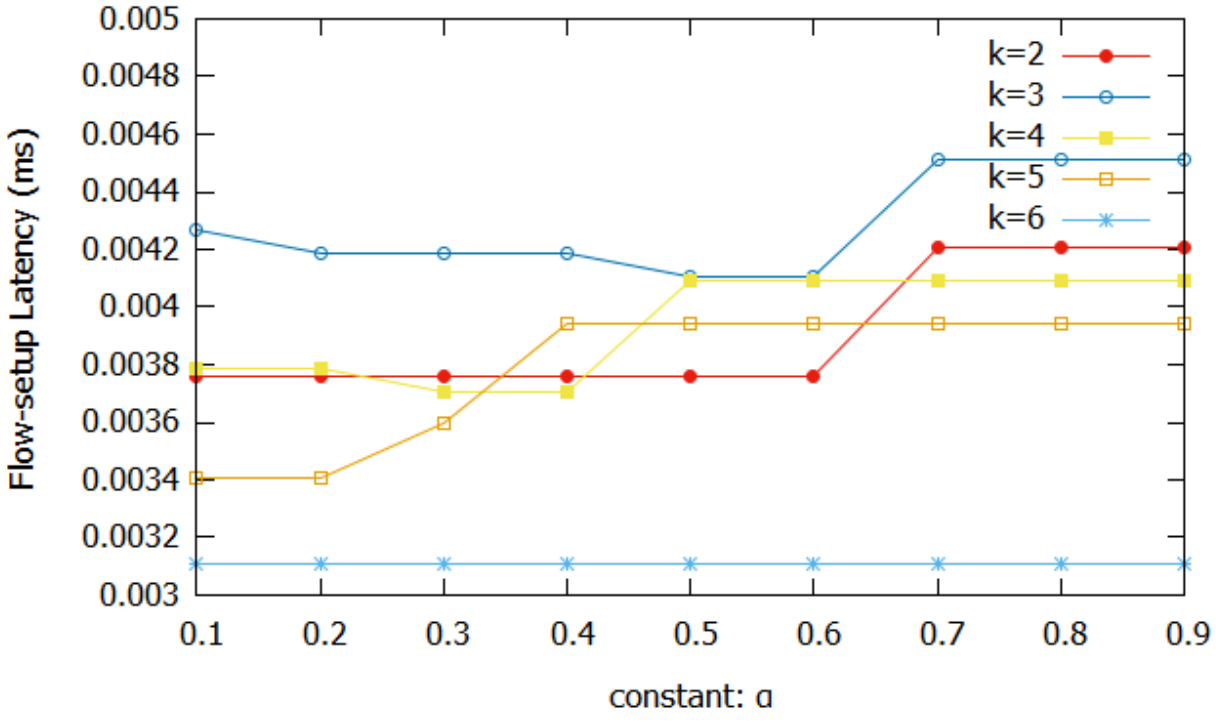



**Figure 5**: Average flow-setup latencies for varying values of $\alpha$ and $k$ in a network containing $|S| = 11$ switches

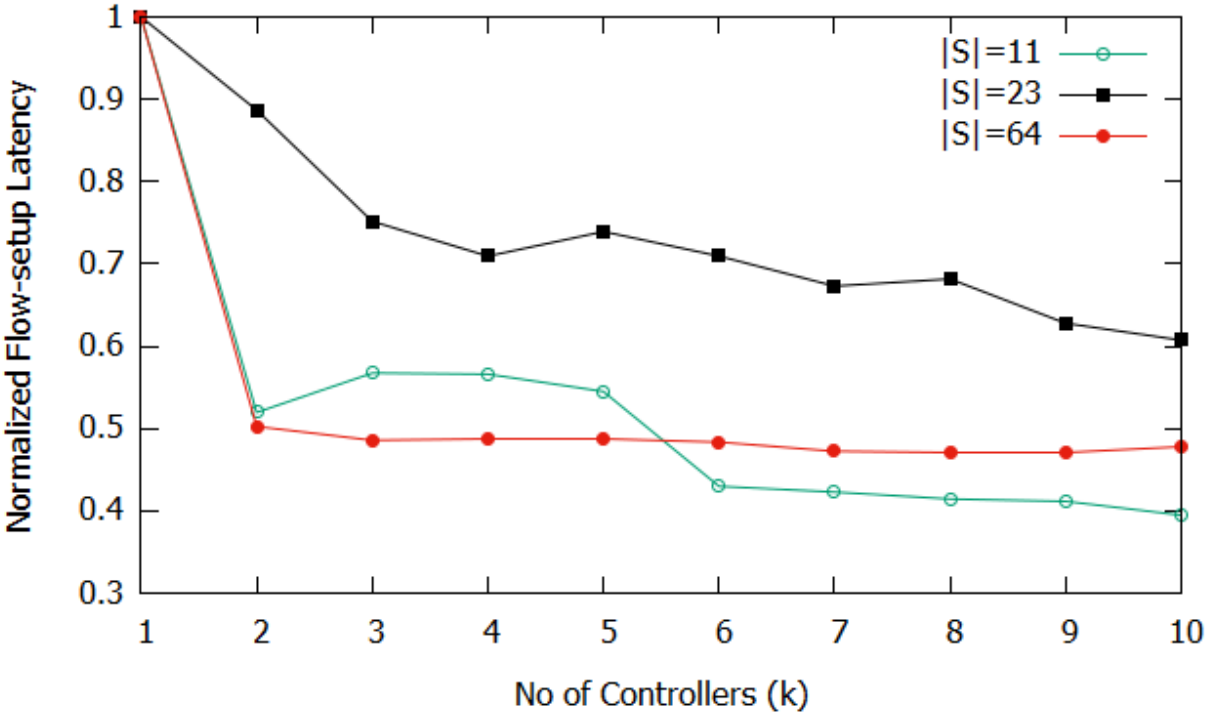



**Figure 6**: Decreasing average flow-setup latencies with respect to number of controllers ($k$) for different networks. As the networks have varying latencies, they are normalized for comparison.

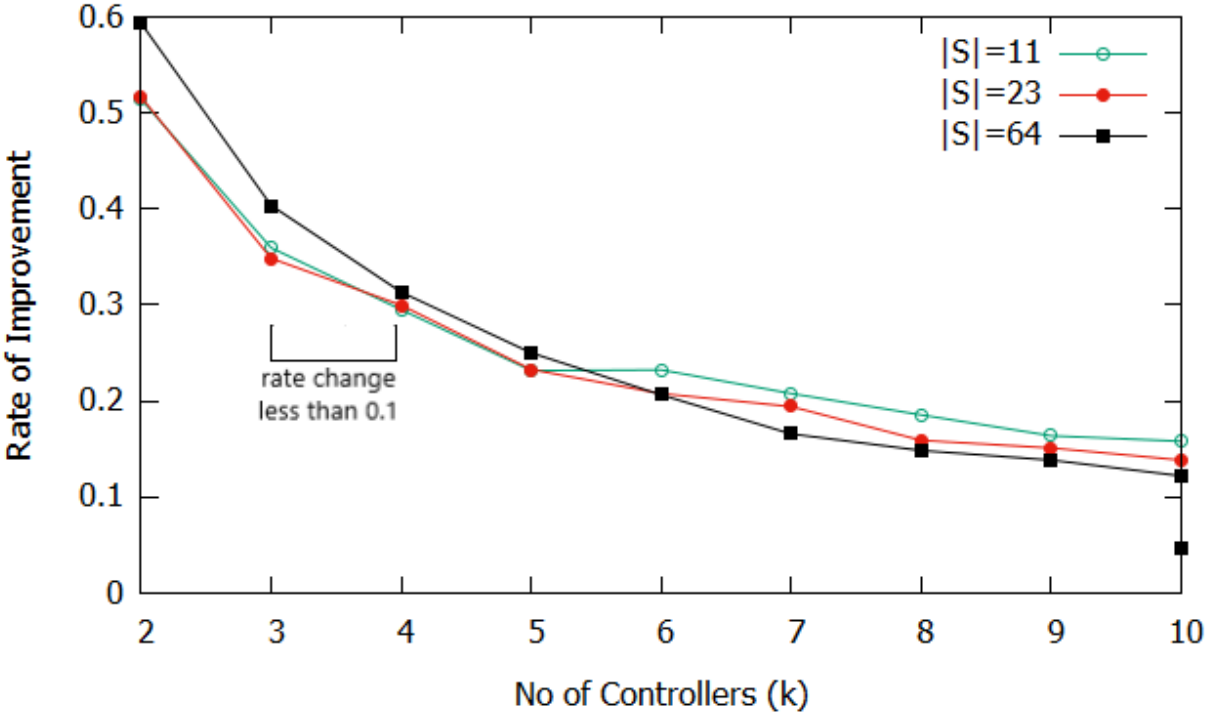



**Figure 7**: Gradually decreasing improvement ratio with respect to number of controllers ($k$) for networks of different sizes

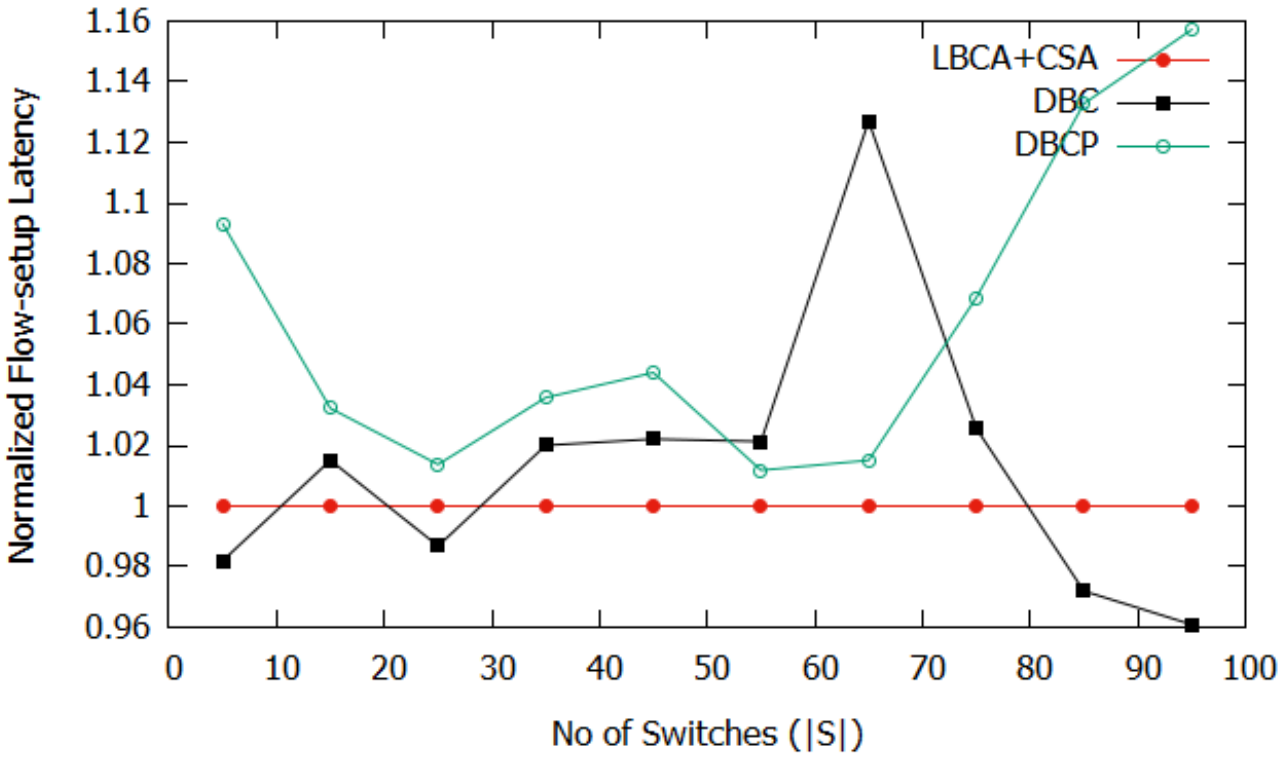



**Figure 8**: Comparison among LBCA+CSA, DBCP and DBC, in terms of normalized flow-setup latency

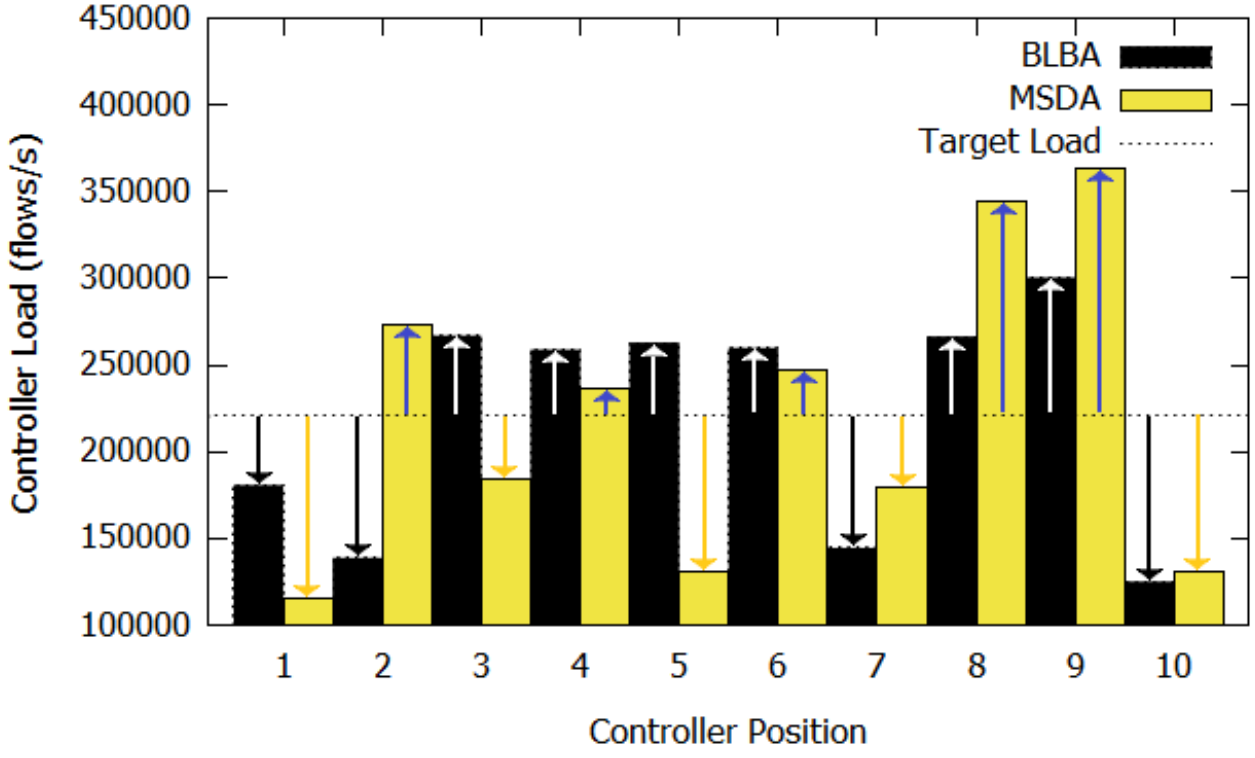



**Figure 9**: Comparison between BLBA and MSDA in terms of load per controller

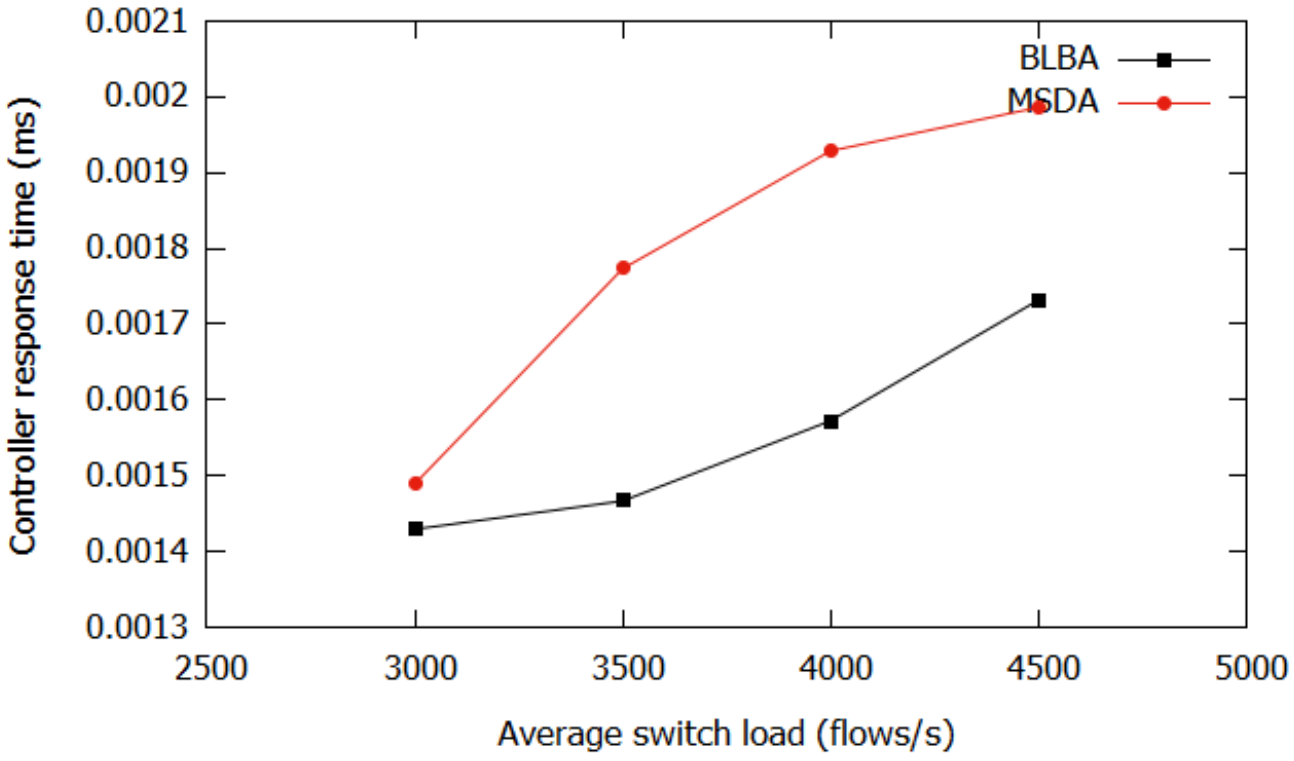



**Figure 10**: Gradually decreasing average response times for varying switch loads

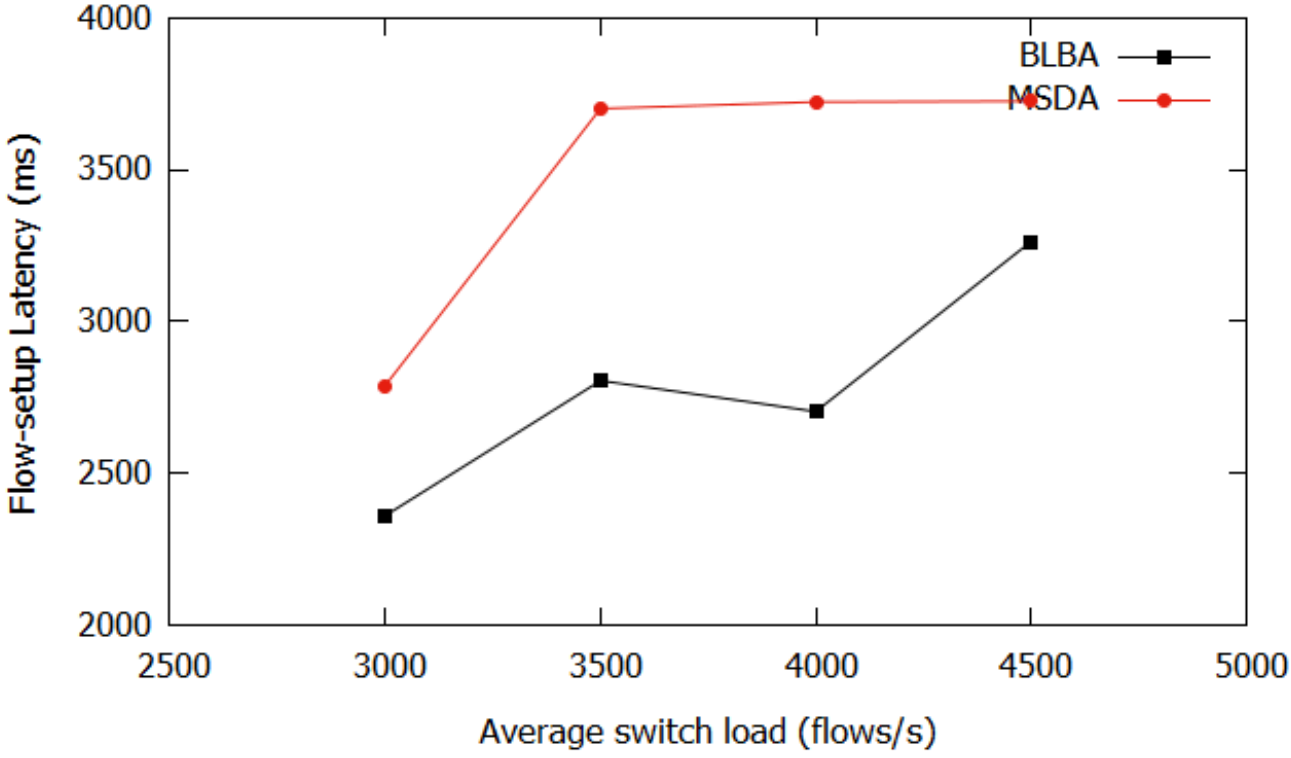



**Figure 11**: Comparison between BLBA and MSDA in terms of maximum flow-setup latency