Static and Dynamic Controller Assignment Algorithms for Software-Defined Networks

Abstract

Software-Defined Networks (SDNs) are replacing traditional networks in several data centers by virtue of its programmability and efficient use of resources. Increasing network sizes demand multiple decision taking entities to monitor and operate the ever-changing networks. The resultant NP-Hard problem - also known as *the Controller Placement Problem (CPP)* is a center-stage for intensive research. Many approaches over the years address various parameters of the controller placement paradigm, like flow-setup latencies, load balancing, reliability and resilience. However, very few provide an efficient algorithm which addresses multiple parameters in polynomial-time complexity. W propose a novel controller assignment method which minimizes overall flow-setup latency, route-synchronization latency and controller response times simultaneously, by dividing the network into several clusters, placing controllers and assigning switches dynamically. Simulation results show that our proposed mechanism outperforms existing state-of-the-art algorithms in terms of overall network setup latencies.

Keywords: SDN, CPP, controllers, flow-setup, latency

1. Introduction

Software Defined Networks (SDNs) simplify network management by decoupling the traditional protocol stack into data and control planes, which consist of switches and controllers, respectively. The controllers replace the switches as the decision taking entities of the SDNs and the switches only retain their data-forwarding capabilities, which simplifies switch designs and configurations. Due to scalability and reliability issues [1, 2] paired with the formation of bottle-necks in moderate-sized networks, the initial design of a single controller [3] is replaced by a multiple-controller architecture. The resultant problem is well-known as the NP-Hard Controller Placement Problem (CPP) which deals with placing an optimal number of controllers to improve network throughput [4, 5, 6]. The multiple constraints that need satisfying include minimizing network latencies, deployment costs, energy consumption and maximizing reliability and resilience. Researches in recent years propose several approximations which deal with one or more of these constraints.

Initial experiments study the effects of different latencies on overall network throughput through exhaustive controller deployment [7]. Eventually, other constraints like reliability [8] and resilience [9] are addressed. Some approaches balance controller loads when placing them [10], while others propose a more dynamic approach [11]. Several models have been developed to reduce deployment costs [12] and controller response times [13]. Some researches also address multiple constraints by providing trade-off decisions to network operators, between certain performance parameters. Seeing that most of the afore-mentioned methods provide exhaustive solutions to the CPP, a few researches propose efficient polynomial-time algorithms to cluster the network [14] and balance load through both static [14] and dynamic assignments of controllers [15]. However, they perform optimization of a specific parameter, instead of providing a complete system. To the best of our knowledge, there is no such method which clusters the network, places the controllers and assigns them dynamically to minimize overall latency and load imbalance in polynomial time complexity. In this paper, we propose a novel controller assignment method which minimizes the following:

1. **Flow-setup latency and 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 the concerned switches. Similarly, when there is a change in the network, the concerned controllers and switches have to be notified. The delay produced by these processes are called the flow-setup latency and the route-synchronization 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. A great volume of processing is required to facilitate smooth operation of the network, which is called the *load of a controller*. An imbalanced distribution of load can lead to an exponential increase of controller response times.

The processing delays of SDN switches are significantly reduced compared to traditional networks and propagation latencies are also negligible. Therefore, overall network latency depends on transmission and queuing latencies of packets [16]. Our proposed method aims to divide the network into k balanced clusters and place controllers to reduce transmission latency. Furthermore, the controller response time is improved through dynamic load balancing, which reduces queuing latency. Simulation results suggest that my proposed method outperforms the state-of-the-art algorithms in terms of overall network latency and load balancing.

The remainder of our paper contains the problem formulation in Section 2, the detailed proposed mechanism in Section 3, the simulations and experimental analysis in Section 4, and the conclusion in Section 5.

2. Problem Formulation and Assumptions

Systematic stratification 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 $G = (S, L)$, where, S represents the set of switches (or nodes) and L represents the set of links (or edges) between the switches. The edge weights represent the transmission delays (bandwidth) of the links. The graph G is clustered into k disjoint set of switches such that every switch falls into a sub-network. Any switch of a sub-network forwards packets following a set of forwarding rules, in the form of flow-tables. However, encountered with a new flow, it sends a query packet to the controller of the sub-network, which notifies the concerned switches about the new rule. Provided that, any switch in the flow-path is of another sub-network, its respective controller is notified instead. The total number of such queries generated by a switch in unit time is the *switch load* and the total number of queries generated by a sub-network is the *controller load.*

In an ideal scenario, each controller has a load proportional to its processing power. The imbalance of controller loads can cause a substantial increase in average response time of a controller, which according to Little's law [13], can be calculated as-

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

Here $Power_{C_i}$ is the maximum processing power and $Load_{C_i}$ is the load of any controller C_i of the network. Following these deductions, our goal is to -

1. Divide the network into clusters while minimizing inter-cluster and inter-cluster distances.
2. Place controllers to minimize flow-setup and route-synchronization latencies.
3. Dynamically reassign switches to reduce average controller response times.

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

3. Proposed Method

Our proposed method consists of three algorithms - Latency Based Clustering (LBC), Controller Selection Algorithm (CSA) and Best-first-search Load Balancing (BLB). LBC clusters the network into a given number of sub-networks (k) and CSA places a controller in each sub-network, resulting in a static controller-switch assignment. BLB is a dynamic load balancing algorithm which periodically reassigns switches to avoid over-burdening a controller. The above mentioned algorithms are explained in detail in sections 3.1, 3.2 and 3.3, respectively.

3.1. Latency Based Clustering (LBC)

Our previous work Degree-based Balanced Clustering (DBC) [17] selects the nodes with the highest degrees in an unweighted network as cluster heads to reduce intra-cluster distances. Ensuring that the cluster heads are a minimum distance (T_d) apart from each other, DBC places controllers based on inter-controller and intra-controller distances. In this paper, we propose a novel clustering algorithm LBC, which selects cluster-centers instead of cluster-heads as the foundations of the clusters. They ensure that the clusters are evenly distributed throughout the network and not centralized at a certain region.

Assuming the network is small enough to be managed by a single controller, it should be placed at the center of the network to minimize controller-to-switch delay. The center of a graph [18] can be determined by minimizing the average or maximum distance from every other node:

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

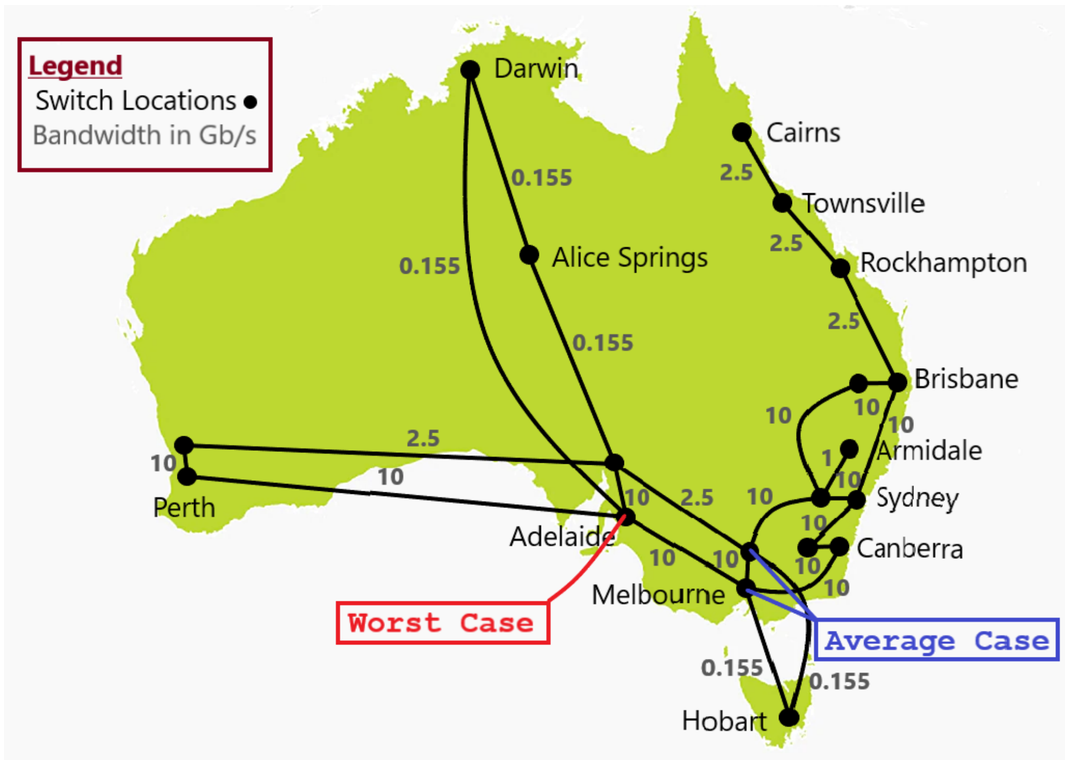


Figure 1: Existing network of Australia (AARNET [19]) showing the centers of the network.

$$center_{max} = \min_{s \in S} \Big(max_{d \in S} \big(dis(s, d) \big) \Big) \tag{3}$$

here, $center_{avg}$ and $center_{max}$ are the centers of the network S in terms of average and maximum distances, respectively, and $dis(s, d)$ is the shortest path distance from nodes s to d. The maximum distance or worst case latency (Equation 3), when determines the center of the network, can vary greatly in the presence of distant and isolated nodes (Figure 1). Conversely, the average case latency (Equation 2) is unaffected by outliers, therefore, we select the node with the minimum average distance ($center_{avg}$) as the center of the network.

For larger networks, when placing multiple controllers, several sub-networks must be formed which makes the center selection process troublesome. To ensure equal division of the network, the cluster-centers must 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 of computer science - *The vertex k-center problem* [20], which has many sub-optimal approximations. Our algorithm LBC (Algorithm 1) utilizes the latencies between switches to divide the network optimally while ensuring that each cluster contains potential controller positions.

Ideally, a network S containing $|S|$ switches and k controllers, has $\frac{|S|}{k}$ switches in each cluster. Therefore, every two cluster-center must have enough distance in between them to accommodate at least $\frac{|S|}{k}$ switches, which is the *ideal cluster-size*. The average distance of a node from all other nodes in the network increases gradually from the center towards the periphery. Accordingly, the node s_{cc} with the minimum average distance $\overline{Dis_{s_{cc}}}$, is selected as the first cluster-center and the cluster is expanded hop by hop until the cluster-size reaches

Algorithm 1: Latency Based Clustering (LBC)

```
**Result:** Set of Cluster-Centers, $CC$
$dis :=$ all possible node pair shortest distances;
for all $s \in S$, $\overline{Dis}_s := \frac{\sum_{s,d \in S}^{d \neq s} dis(s,d)}{|S|-1}$;
$CC := \emptyset$, $k :=$ required number of controllers;
while $|CC| < k$ do
    $CC := CC \cup \{s_{cc}\}$ for all $s_{cc} \in S$, where $\overline{Dis}_{s_{cc}} \leq \overline{Dis}_{t \in S}$;
    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 $\frac{dis(s_i,s)}{|S|-1}$ from $\overline{Dis}_s$ for all $s \in S - S_i$
    end
    $S = S - S_i$
end
```

$\frac{|S|}{k}$ nodes. The nodes of the cluster are then removed from the network S and their distances are subtracted from the average distances of the remaining nodes. This process is continued until k clusters have been formed.

3.2. Controller Selection Algorithm (CSA)

The Controller Selection Algorithm (CSA) creates clusters once more from the cluster-centers generated by LBC to avoid formation of overlapping or isolated clusters and selects a controller position for each cluster. Each node is included in the cluster of the nearest clusters-center in terms of shortest path distance $dis(i,j)_{i,j \in S}$, which ends the cluster formation and initiates the controller selection process (Algorithm 2). It is to be noted that the shortest path distance is in terms of transmission delay (not hop-count).

Ideal controller positions have maximum connectivity with each other and the switches of the network. However, both controller-to-controller and controller-to-switch distances 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 which replaces inter-controller and intra-controller distances with inter-cluster (σ) and intra-cluster (ϕ) distances, respectively [17]. A normalized constant α, 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:

$$\phi(s)_{s \in S_i} = \frac{1}{|S_i|} \sum_{u \in S_i} dis(s,u) \tag{4}$$

$$\sigma(s)_{s \in S_i} = \frac{1}{|S - S_i|} \sum_{v \in (S - S_i)} dis(s,v) \tag{5}$$

$$C_i = \min\Big(\big(\phi(s) \times \overline{\sigma} \times \alpha\big) + \big(\sigma(s) \times \overline{\phi} \times (1-\alpha)\big)\Big) \tag{6}$$

Algorithm 2: Controller Selection Algorithm (CSA)

```
**Result:** Set of Controllers, $C$
$dis$ := all possible node pair shortest distances;
$CC_{i=1}^{k}$ := LBC 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
    $s_c := \min\Big(\big(\phi(s) \times \overline{\sigma} \times \alpha\big) + \big(\sigma(s) \times \overline{\phi} \times (1-\alpha)\big)\Big)$;
    $C := C \cup \{s_c\}$;
end
```

here, $\overline{\phi}$ and $\overline{\sigma}$ are the mean intra-cluster and inter-cluster distances, respectively. The highest possible value of α is 1, which nullifies the effect of σ 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 and inter-cluster distances are prioritized equally. The value of α can be changed to better suit the requirement of the network administrator.

3.3. Best-first-search Load Balancing (BLB)

For a fixed controller-switch assignment scheme ($S \to C$), 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 (BLB, Algorithm 3) which uses the well-known Uninformed Search technique – Best-First Search (BFS). BLB considers each 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 varying load, I denote the loads of the switches as $l_1, l_2, ..$ 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, ideally, the load of a controller 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 (Figure 2). The extra traffic contributes to an overall increase in latency and decrease in network throughput. To

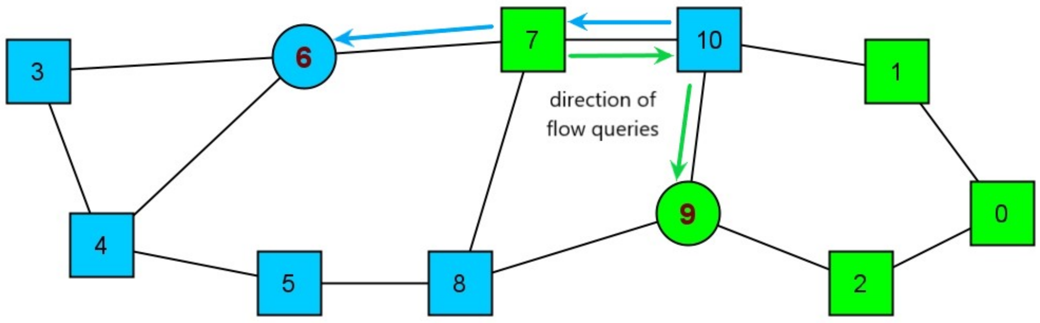


Figure 2: Redundant traffic is generated for new flow queries of switches 7 and 10 to controllers 9 and 6 respectively, which can be avoided by swapping the clusters of the switches

avoid such overlapping of clusters, I prioritize distance over load balancing, and express the problem as a Heuristic (Informed) Search with the following foundations:

3.3.1. 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 LBC and CSA are applied on a network.

3.3.2. 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.

3.3.3. Error Function

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

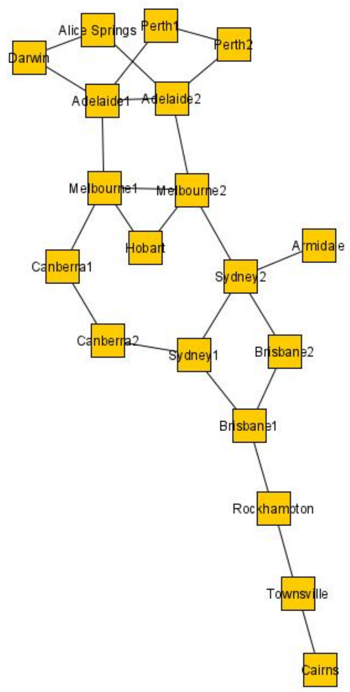
$$\varepsilon(state) = \sum_{i=1}^{k} \left[\left(L_i - \frac{L_{net}}{k} \right)^2 \right] \tag{8}$$

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

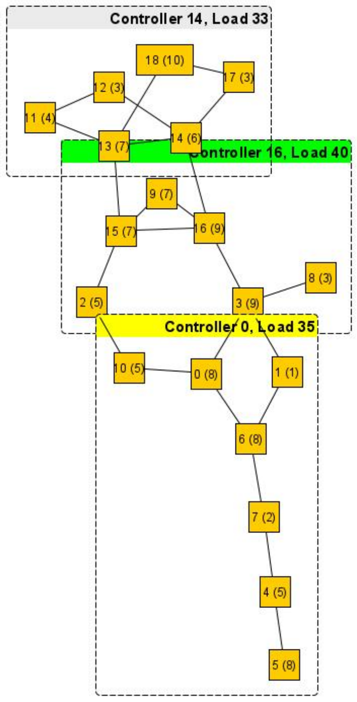
where, L_i is the load of the i^{th} controller, which is the cumulative load of the switches assigned to it. Squaring the differences give more priority to controllers whose loads are more imbalanced compared to others.

3.3.4. Heuristic

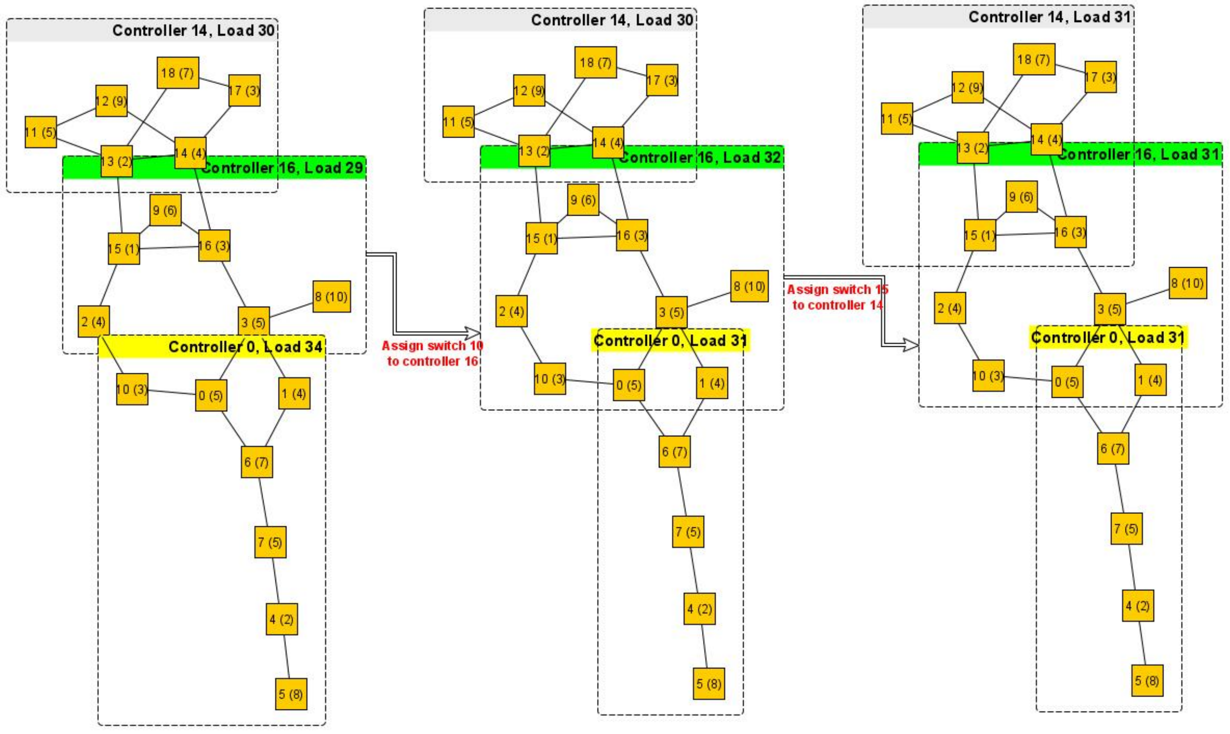
A greedy heuristic is simple and efficient, however, it provides a sub-optimal solution. Our goal is not an absolute and optimal solution, rather a local optima where no clusters



(a) Original Aarnet Network (rearranged)



(b) Root state with no child state



(c) Optimal assignment scheme achieved after three iterations

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

are overlapping. Therefore, we select the 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: BestFS Load Balancing (BLB)

```
Result: Assignment of Switches, $S \to C$
$S_{i=1}^{k}$ := CSA Clusters;
$state := S \to C$, current assignment of switches;
Set of all possible new states, $Pstates := \{state\}$;
while $Pstates \neq \emptyset$ do
    $state := \min\big(\varepsilon(Pstates)\big)$;
    $Pstates := \emptyset$;
    foreach border switch $s \in S$ do
        New assignment $Nstate := state$;
        Change assignment of switch $s$ to controller of adjacent cluster in $Nstate$;
        if $\varepsilon(Nstate) < \varepsilon(state)$ then
            $Pstates := Pstates \cup \{Nstate\}$;
        end
    end
end
```

4. Simulations

The proposed mechanism clusters the network, places controllers, and performs load balancing on the clustered network after placement. The following sections 4.1 and 4.2 give a detailed description of the simulation environment and the performance metrics. In Section 4.3 optimum values for the decision variables k and α are determined. Subsequently, in sections 4.4 and 4.5, our proposed algorithms are validated by comparing them with the state-of-the-art controller placement [14] and load balancing algorithms [15], respectively.

4.1. 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 [19]. 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.

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

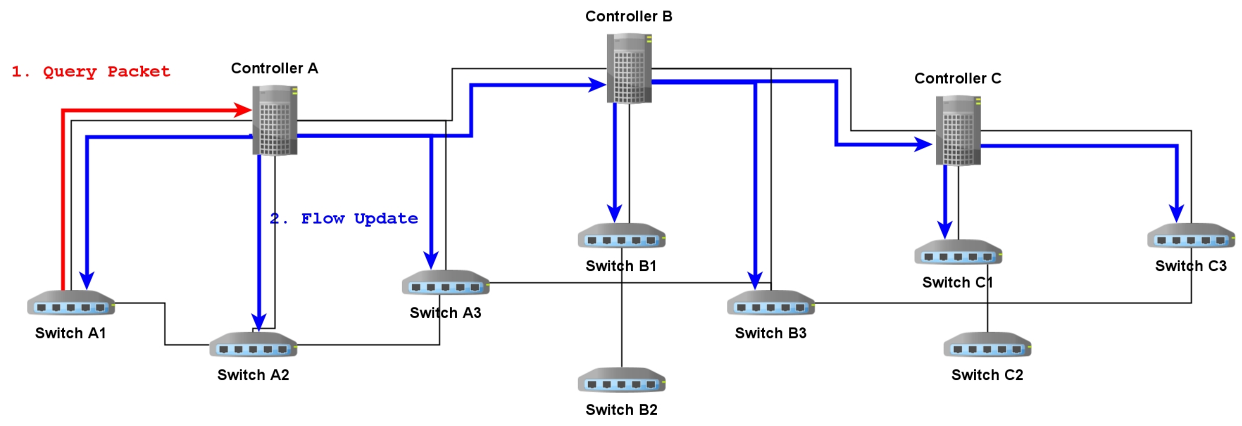


Figure 4: Setup of a new flow in the flow tables of the switches in the network

The initial weights of the links are their bandwidths in Gb/s (Giga-bits per second), which are converted into transmission latencies (milliseconds), assuming each control packet is 1500 bytes long. The maximum bandwidth 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 Best-first-search Load Balancing (BLB). 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 [21]. We assume the network has 10 controllers which are placed using LBC 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.

4.2. 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 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 (Ω_{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:

$$\Omega_{avg} = \frac{\sum_{s_i, s_d \in S} \{dis(s_i, c_i) + \max_{s_j \in path_{i,d}} (dis(c_i, c_j) + dis(c_j, s_j))\}}{|S| \times |S-1|} \tag{10}$$

here, $path_{i,d}$ is the path with least delay from source s_i to destination s_d, and s_j is any switch in that path. The controllers of switches s_i and s_j are c_i and c_j respectively.

The processing latency of the controllers increase 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:

$$\Omega_S = \max_{s_i \in S} \left(\Delta_{s_i} \times l_{s_i} \right) \tag{11}$$

here, Δ_{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, Δ_{s_i} is derived as follows:

$$\Delta_{s_i} = \frac{\sum_{s_i, s_d \in S} \left[dis(s_i, c_i) + RT_{c_i} + \max_{s_j \in path_{id}} \{dis(c_i, c_j) + RT_{c_j} + dis(c_j, s_j)\} \right]}{|S| - 1} \tag{12}$$

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

4.3. Decisive Variables

LBC clusters the network into k sub-networks and CSA places a controller in each sub-network using the constant α, 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 LBC on a small network with varying values of k and α (Figure 5) provide 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 α 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 in 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.

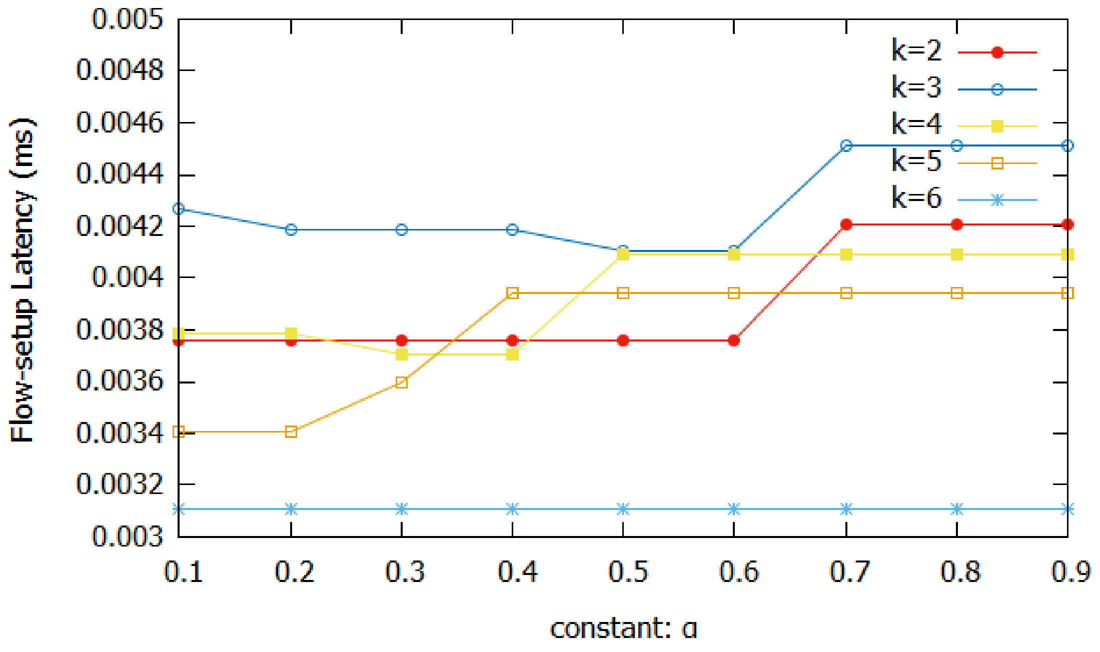


Figure 5: Average flow-setup latencies for varying values of α and k in a network containing $|S| = 11$ switches

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:

$$improvement\ ratio_k = \frac{Latency_1}{Latency_k \times k} \tag{13}$$

here, $Latency_1$ and $Latency_k$ are the flow-setup latencies when the number of controllers are 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 [7], 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.

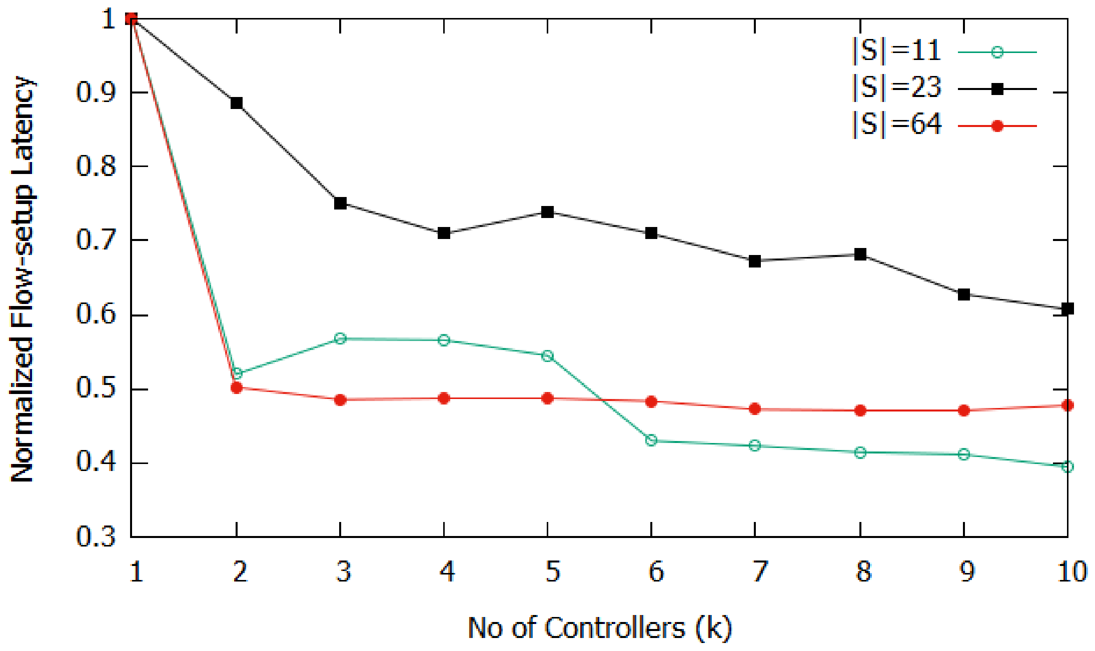


Figure 6: Decreasing average flow-setup latencies with respect to number of controllers (k) for different networks. As the networks have varying latencies, they are normalized for comparison.

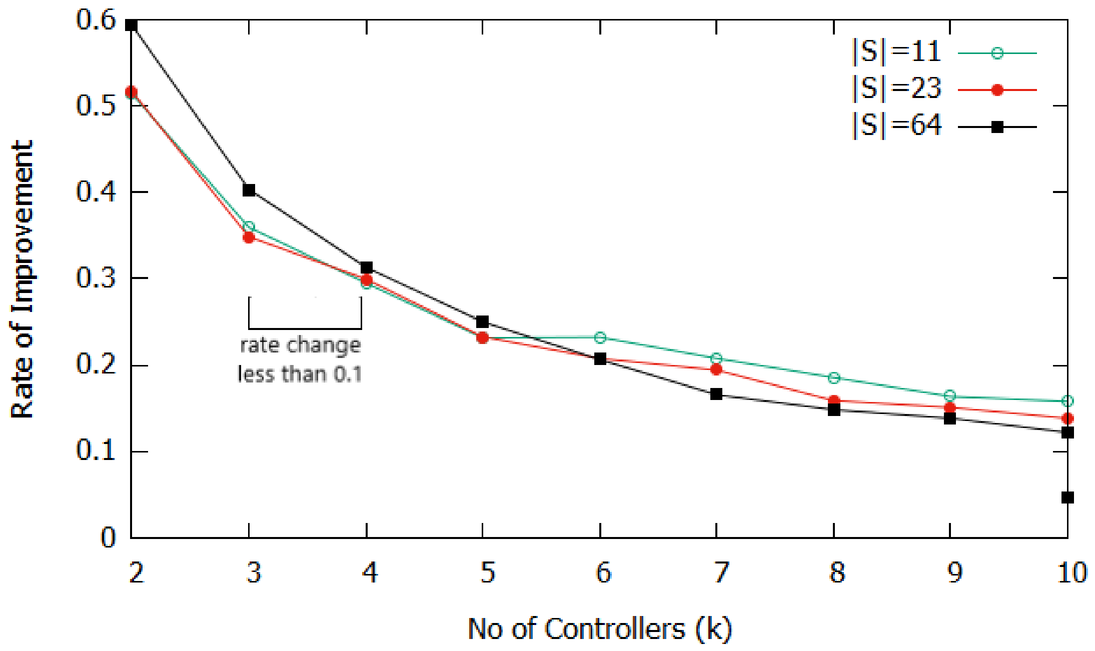


Figure 7: Gradually decreasing improvement ratio with respect to number of controllers (k) for networks of different sizes

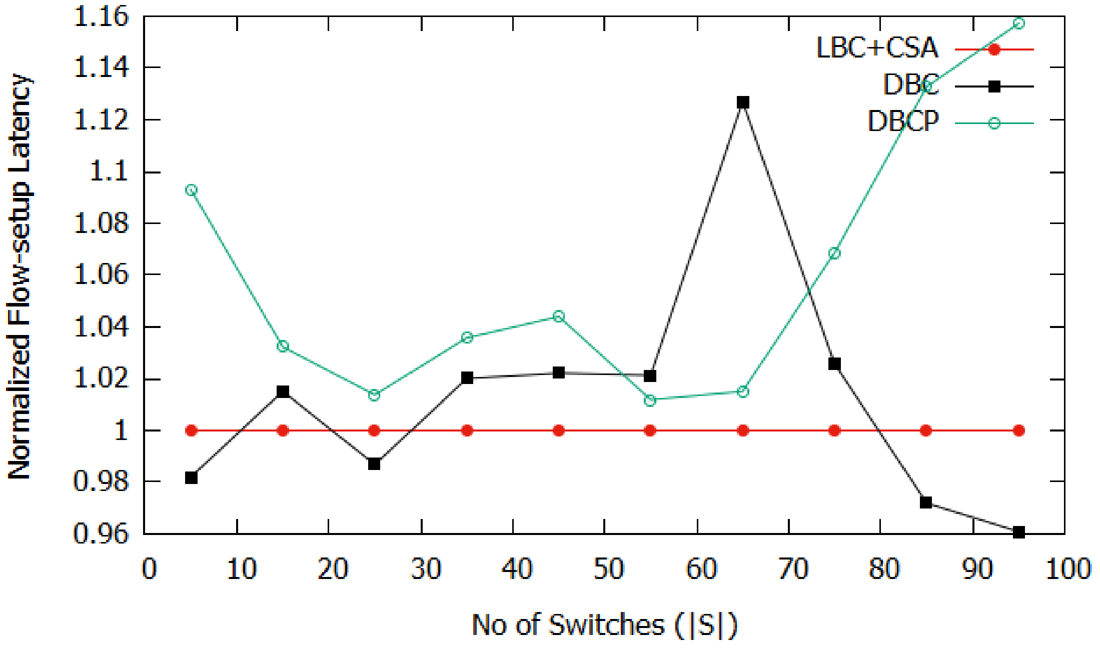


Figure 8: Comparison among LBC+CSA, DBCP and DBC, in terms of normalized flow-setup latency

4.4. Controller Placement

In this section we evaluate our static controller placement method (LBC+CSA) by comparing it to the well-known algorithm DBCP [14] and the algorithm DBC [17]. 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 LBC with the same number of controllers as DBCP when clustering the networks from the Zoo Topology. DBCP underperforms compared to LBC and DBC when the network has high connectivity (e.g. star topology) or when all nodes have equal density (e.g. ring topology). Our simulations using the remaining networks suggest that LBC+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 LBC+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, LBC+CSA outperforms both DBCP and DBC in 78% and 72% of the simulated networks, respectively.

4.5. Load Balancing

BLB balances the loads of the controllers by swapping clusters of border nodes to avoid overlapping cluster formations. We set the maximum iteration limit of BLB to 100 and compare with the algorithm MSDA [15, 22] in terms of maximum load imbalance. The loads of the switches are randomly assigned within the range 1000 to 5000 with equal probability. Therefore, the average switch load is $2500 flows/s$ and the target controller load is

$188.5K flows/s$ (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 BLB outperforms MSDA in terms of controller load, specially 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.

The Figure 10 represents the comparison between BLB 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 BLB 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 BLB outperforms MSDA. The flow-setup latency for higher average switch loads shows minimum change for MSDA as the controller-switch preferences are prioritized. BLB on the other hand reduces latency as long as a local optima is available. Consequently, the algorithm terminates if further optimization causes overlapping clusters, which is observed for average switch load of 3500 flows/s as it produces unnecessary traffic (Figure 2).

5. Conclusion

In this paper, we propose a novel controller assignment method which clusters the network, places controllers and balanced loads to reduce overall flow-setup latency of the network. Our proposed method addresses the Controller Placement Problem (CPP) and outperforms several 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 BLB 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. My proposed method can also be extended to facilitate simultaneous optimization of multiple network parameters, which will be very helpful for network operators.

[1] 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.

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

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

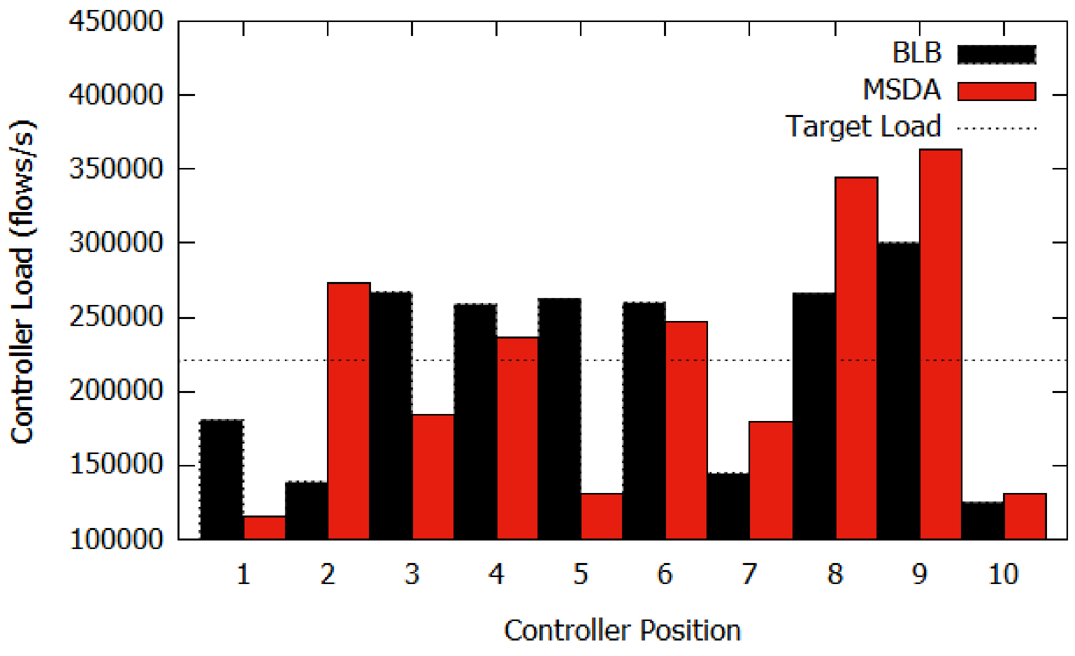


Figure 9: Comparison between BLB and MSDA in terms of load per controller

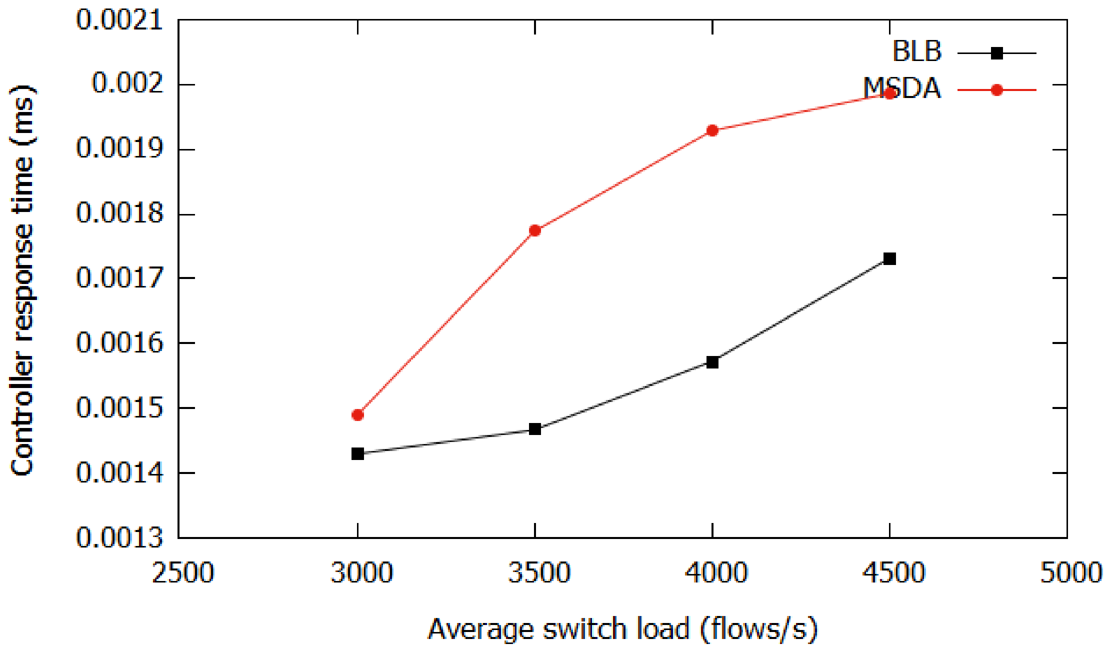


Figure 10: Gradually decreasing average response times for varying switch loads

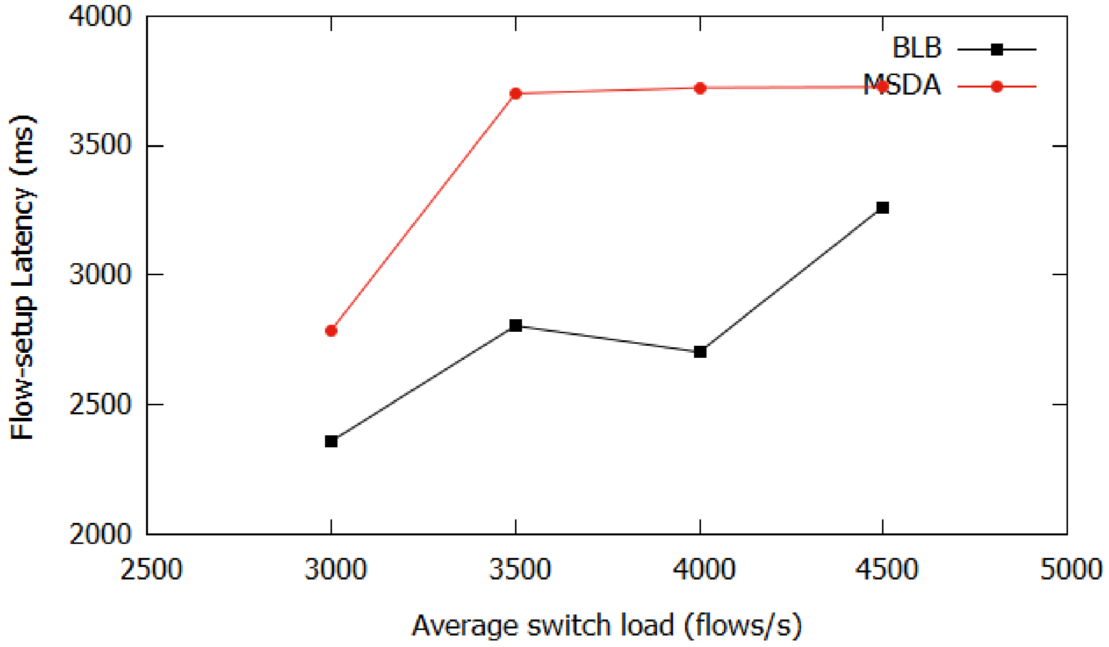


Figure 11: Comparison between BLB and MSDA in terms of maximum flow-setup latency

[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] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

[16] A. B. Forouzan, Data communications and networking (sie), Tata McGraw-Hill Education, 2006.

[17] 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.

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.