Versatility in Controller Placement Approaches through Graph Theory and Greedy Search Techniques

Talha Ibn Aziz[a,*] (Researcher), Muhammad Mahbub Alam[b] (Supervisor) and Md Sakhawat Hossen[b] (Co-ordinator)

[a] *Department of Computing Science, University of Alberta*
[b] *Department of Computer Science and Engineering, Islamic University of Technology*

ARTICLE INFO

Keywords:
SDN
CPP
controllers
flow-setup
latency
control traffic

ABSTRACT

Software Defined-Networks (SDNs) decouple the traditional protocol stack into the control plane and the data plane consisting of - controllers and switches, respectively. The controllers are placed in the network considering numerous fundamental parameters like flow-setup latency, switch-controller control traffic, route synchronization latency, reliability, and security. While many researchers address the resultant Controller Placement Problem (CPP), very few provide a thorough and efficient process that considers multiple core parameters. In this paper, we propose three algorithms to cluster the network, place controllers, and assign switches dynamically to controllers in polynomial time. Extensive simulations on two hundred and forty-three (243) existing network topologies suggest that our approach outperforms current state-of-the-art controller placement algorithms in terms of flow-setup latency and traffic-awareness.

1. Introduction

The booming usage growth of the Internet, and the integration of a plethora of Internet of Things (IoT) devices into the network system - causes a shortage of network management resources. The complexity of configuring the already-troublesome network, is further increased by the amalgamation of switches from various vendors. 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 traditional network-layer functions to simplify network management and configuration, making efficient use of scarce resources. The forwarding capabilities are retained by the switches and routers, while new agents are introduced as the decision making entities. These authoritative devices - namely *controllers*, have superior processing power and memory compared to the switches, which are simplified in terms of both costs and design.

The original SDN architecture [17] includes a single controller, resulting in the formation of bottlenecks, single point of failure, and scalability issues [11, 38]. Consequently, the multiple-controller architecture is proposed for larger networks; 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 [40, 30, 8]. 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 the concerned switches. The total delay incurred in 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 [15]. 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 perform optimization of a specific latency or parameter instead of providing a complete sys-

*Corresponding author
taziz@ualberta.ca (T.I. Aziz); mma@iut-dhaka.edu (M.M. Alam); sakhawat@iut-dhaka.edu (M.S. Hossen)
sites.google.com/view/talha-ibn-aziz (T.I. Aziz)
ORCID(s): 0000-0001-8747-8119 (T.I. Aziz)

tem, while a few of them address multiple parameters simultaneously [23, 28]. However, they either provide exhaustive solutions or reduce accuracy to improve efficiency in terms of time and/or space 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.

2. Background and Related Works

Several research papers have addressed multiple aspects of the Controller Placement Problem (CPP) in the last decade. Heller et al. [18] study the effects of different latencies on overall network throughput through exhaustive controller deployment and generate two essential questions - *How many controllers are needed?* and *Where in the topology should they go?* Solutions to the CPP address these questions by optimizing various parameters when placing controllers, namely - switch-controller and controller-controller delays, route synchronization latency, flow-setup latency, reliability and resilience, deployment costs, traffic-awareness, etc. Some of the notable works are mentioned in this section along with the parameters they addressed.

Li et al. [24] minimize the average switch-controller transmission delay to achieve effective deployment of multiple controllers in a continuous two-dimensional space by adopting the PSO (particle swarm optimization). Das et al. [9] presents a Steiner tree-based model that analyses the network state synchronization delay between the controllers by minimizing inter-controller latency. The model addresses both failure-free and post-failure scenarios to make the placement resilient. Multiple researchers explicitly address the reliability and resilience of multiple controller placement. Hu et al. [20] introduces a metric - expected percentage of control path loss and formulates the reliability-aware control placement problem. Zhang et al. [39] propose a min-cut based graph partitioning algorithm to place controllers, which aims at maximizing the resilience of the network and shows its advantage over greedy algorithms. Ashrafi et al. [1] address this issue by developing a mathematical model to minimize latency while taking all failure scenarios and scalability into consideration. Fan et al. [13] minimize the average drop-rate of flow set-up requests due to single-link failure using two algorithms - Reliability Aware Controller Placement (RAC) and Fast-RAC (FRAC). Yang et al. [35] 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. [28] 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. [4] 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. [12] minimize propagation latency, maximize throughput, and link failure probability while placing controllers. Toufga et al. [31] 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. [26] 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 [32] distributes the traffic using wildcard rules on supported switches to transfer traffic to OpenFlow [19] server replicas. Yao et al. [36] 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. [37] propose a multi-controller load balancing approach called HybridFlow for software-defined wireless networks, which is a flow-based dynamic solution. Chen et al. [5] 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. [6] assess and explore control-traffic minimization and devise multiple heuristic algorithms to propose an efficient solution. Coronado et al. [7] propose - *Wi-Balance*, an algorithm that balances traffic load and maintains an equal division of network resources. Qilin et al. [27] improve traffic scheduling by reducing the number of flow tables. Schutz et al. [29] 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 [33] propose a combination of matching theory and conditional games to perform load balancing of controllers. Filali et al. [14] formulate the CCPP as a one-to-many matching game and use a well-known stable matching algorithm - Multistage Deferred Acceptance Algorithm (MSDA) [16] 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 concerned switch sends a query packet to the controller of the sub-network. The controller notifies the switches in the path of the flow about the new rule. It also notifies the respective controllers of the switches in the path (unless the switch is in the same sub-network).

The flow setup procedure for a switch can be divided into three phases, 1) the switch sends a query packet to 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 all the switches of the flow-path about the new rule. The 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. The delay in the second phase is the time a controller takes to decide a flow-path, 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 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 [33]. 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 waiting time and service time) can be determined using Little's Law as follows [33]:

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

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 overloaded and working at full capacity, the system is unstable as the queues will grow unboundedly. Therefore, balancing the network load does not contribute to the performance of the system. However, when overall network load does not exceed the system capacity, balancing controller-loads greatly improve throughput and performance, especially when one or more controllers are overloaded. An efficient load-balancing approach in this case, is to assign loads to controllers according to their processing capacity. In order to maintain the balance and avoid controller overload, loads can be distributed constantly as the network operates (*dynamic traffic-awareness*).

The third phase of the flow-setup procedure depends on the controller-switch latencies between the concerned controllers and their respective switches in the flow-path, and the inter-controller latencies among the involved controllers. Furthermore, the time required for a controller to announce any change within its sub-network, namely the route-synchronization latency, is also determined by the controller-switch la-

tency. 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 (LBCA), Latency-based Controller Selection Algorithm (LCSA), and Greedy Load Balancing Algorithm (GLBA). LBCA clusters the network into a given number of sub-networks (k) and LCSA places a controller in each of the sub-networks, resulting in a static controller-switch assignment. GLBA is a dynamic load balancing algorithm that periodically reassigns switches to avoid over-burdening a controller.

4.1. Latency-based Clustering Algorithm (LBCA)

The Latency-based Clustering Algorithm (LBCA) 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

Before forming a cluster, LBCA determines the footholds from which a cluster will be created, to ensure that the clusters are not concentrated in a certain region. Consequently, LBCA re-clusters the network to avoid forming geographically irregular clusters (varying shapes and sizes). A recent controller placement algorithm DBC [2], places controllers based on inter-controller and intra-controller distances. DBC further ensures 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. Contrarily, LBCA determines 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 LBCA.

In a single-controller network, placing the controller at the center - minimizes the controller-switch delay. Considering the network as a graph G, the center is the node with the minimum - average or maximum delay - from every other node in G [34]. The average delay

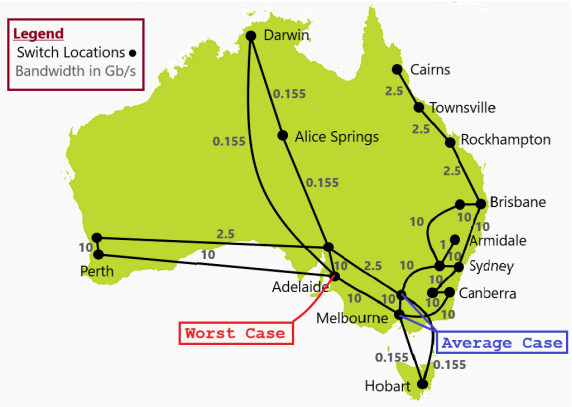

Figure 1: Existing network of Australia (AARNET [22]) showing the centers of the network.

can be calculated as:

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

where, $center_{avg}$ is the node with the minimum average delay and $T(s,d)$ is the shortest path distance from node s to d.

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

Here, node $center_{dia}$ has the minimum of all the maximum delays in the network and is always on the network diameter.

When the center of the network is determined by the maximum delay or worst-case latency (Equation 3), the position of the center 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 or distant nodes. Therefore, we select the node with the minimum average distance ($center_{avg}$) as the center of the network. This solution of the single-controller placement sub-problem, can be extended to place controllers in the multi-controller environment of a larger network, where each controller is placed in a sub-network. To ensure equal division of the network into sub-networks or clusters, the cluster-centers are expected to be equidistant from each other and the clusters must expand from the cluster-centers. This selection of centers in a network is a classical NP-hard problem - *The vertex k-center problem* [21], which has many sub-optimal approximations. The prosposed algorithm LBCA (Algorithm 1) utilizes the latency among switches to divide the network optimally while ensuring minimum overlapping among neighboring clusters.

Initially, LBCA calculates the delays between every possible pairs of nodes in terms of shortest source to

Algorithm 1: Latency-based Clustering Algorithm (LBCA)

Result: Set of Clusters, S_i
1 $S :=$ set of nodes in network G;
2 Calculate shortest path delay $T(a, b)$ between all possible pair of nodes a and b ;
3 for any $s \in S$, $\overline{T}_s := \frac{\sum_{\forall d \in S} T(s,d)}{|S|-1}$, where $d \neq s$;
4 $CC := \varnothing$, $k :=$ required number of controllers;
5 **while** $|CC| < k$ **do**
6 Find s_{cc} such that $\overline{T_{s_{cc}}} \leq \overline{T_{\forall t \in S}}$;
7 $CC := CC \cup \{s_{cc}\}$;
8 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;
9 **foreach** *switch* $s_i \in S_i$ **do**
10 Subtract $\frac{T(s_i,s)}{|S|-1}$ from $\overline{T_s}$ for all $s \in S \setminus S_i$
11 **end**
12 $S = S \setminus S_i$
13 **end**
14 Discard previous clustering and form new clusters S_i each containing one cluster-center CC_i and all its nearest nodes, where $i = 1, 2, .., k$;

destination path delays (Algorithm 1 Line 2) using Dijkstra's algorithm [10]. Using the attained delays, the average delay $\overline{T}_{s \in S}$, of every node s, from every other node $d \in S$, is determined (Algorithm 1 Line 3). Consequently, the nodes with minimum average delay are selected from the set of nodes S, as the cluster-centers. For every selected cluster-center, the center itself and the nearest $\left(\frac{|S|}{k} - 1\right)$ neighbors are removed from S (Algorithm 1 Line 12) to avoid overlapping cluster domains. The minimum average delays of the remaining nodes are adjusted accordingly (Algorithm 1 Line 9). This process is iterated until k cluster-centers are selected.

4.1.2. Clustering-member selection

The cluster-centers of the previous phase are used to form new clusters in this phase and the preceding clusters are discarded. Each node $i \in S$, of the network G is included in the cluster of the cluster-center $j \in S$, that is nearest 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 done after the cluster-center selection process - to avoid the formation of overlapping or isolated clusters.

4.2. Latency-based Controller Selection Algorithm (LCSA)

The Latency-based Controller Selection Algorithm (LCSA) selects a controller for each cluster while allowing the network administrator to prioritize certain parameters.

Algorithm 2: Latency-based Controller Selection Algorithm (LCSA)

Result: Set of Controllers, C
1 $S_{i=1}^{k} :=$ **LBCA** Clusters;
2 **foreach** *cluster* $S_i \subset S$ **do**
3 Find node s_c with minimum value of D_i (see explanation in section 4.2 for details);
4 $C := C \cup \{s_c\}$;
5 **end**

Controllers are in constant communication with each other and with the switches of their respective clusters. However, both controller-to-controller and controller-to-switch latencies cannot be minimized simultaneously. On the contrary, the round-trip time (RTT) from a controller to any of its switches or other controllers, cannot be calculated without placing a controller beforehand. 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 [2]. The average inter-cluster latency is strictly greater than the intra-cluster latency for a multi-controller network. We normalize the latencies by multiplying them with the constants β_2 and β_1, respectively. Another 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:

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

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

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

Here, the node with minimum D_i is the controller C_i for the cluster S_i.

For our experiments, we use the average intra-cluster and inter-cluster latencies as β_1 and β_2, respectively. The highest possible value of α is 1, which nullifies the effect of d and places controllers considering only intra-cluster delays. Conversely, for $\alpha = 0$, controllers are placed solely considering inter-cluster distances. For $\alpha = 0.5$, both intra-cluster and inter-cluster distances are prioritized equally. The value of α can be changed to better suit the requirement of the network administrator.

4.3. Greedy Load Balancing Algorithm (GLBA)

For a fixed controller-switch assignment scheme ($S \rightarrow C$), the loads of the controllers vary with time - 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, a dynamic load balancing algorithm is proposed - which uses a simple yet efficient Greedy Search technique, namely Greedy Load-Balancing Algorithm (GLBA). GLBA 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 (i.e., load is proportional to processing power). Assuming that each controller has identical processing capacity and each switch has a varying load, the loads of the switches are denoted as $l_1, l_2, \ldots, l_n$, where $n = |S|$. The total load of the set of switches 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 a perfectly balanced network or the *goal* state should be $\frac{L_{net}}{k}$, where k is the number of controllers. However, to avoid excess traffic generation, the switches must not be assigned such that one or more clusters are overlapping. To this end, GLBA prioritizes distance over load-balancing and represents the controller-switch assignment as a Greedy Search problem.

4.3.1. Problem Definition

The problem search space is a tree where the *root* is the initial controller-switch assignment (before applying GLBA). Every state of the search tree is a valid controller-switch assignment and must be generated from its parent by swapping one and only one switch. Swapping a switch means changing the cluster of any switch at the border of a cluster, to an adjacent cluster. The search starts with the root state and for every parent state, the generated state with the best heuristic value is selected as the new current state. This process is repeated until a leaf state is reached, i.e., no further states can be generated with better heuristic values. The method of searching a solution by locally selecting the state with the best heuristic is called Greedy Search.

4.3.2. Heuristic

Greedy search is simple, and therefore, its performance greatly depends on the heuristic used. The heuristic value is formulated using an error function which determines the acceptability of the current *state*. A low error value must correspond to a lower imbalance of loads, while a high error value must correspond to a higher load imbalance. Therefore, the error function is denoted by,

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

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

where, L_i is the load of the i^{th} controller and is the cumulative load of the switches assigned to it. The error ε of a state is the squared sum of the differences between the loads of the controllers and the average controller-load. The closer the load of a controller is to the average load, the better the heuristic. The squared nature of the difference - facilitates the prioritization of highly imbalanced controllers over slightly imbalanced ones. Our ultimate goal is not perfectly balanced network (as explained in section 4.3), rather it is a local optima where no clusters are overlapping. In this regard, the greedy heuristic ε is advantageous for its simplicity and efficiency.

4.3.3. Greedy Search

GLBA uses the controller-switch assignment provided by LCSA and generates all possible new states (Algorithm 3, Line 3) from the root state by swapping border nodes as explained in section 4.3.1. The state with the best heuristic from among the generated states is selected as the new current state (Algorithm 3, Line 5). When generating states from the current state, only states with better heuristic values are considered (Algorithm 3, Line 10). The remaining state-generations are discarded to avoid generating unnecessary states. The state with minimum heuristic is again selected as the new current state - this process is continued until no more states can be generated with better heuristic values. Extensive simulations suggest that most networks converge after a finite number of such iterations.

4.3.4. Example

This section illustrates the application of GLBA on multiple load-balancing scenarios. The example topology used is an existing 19-node network of Australia, called *Aarnet*. Figure 2 shows the graph representation of the network and Figure 3 is a state with no possible state generation (i.e., it is a leaf state). When GLBA is applied using a leaf state as root, the root state is returned - without any change. Contrarily, in Figure 4, the goal state is reached after two iterations. The initial load distribution is 30 units for controller 14, 29 units for controller 16, and 34 units for controller 0. The total load is 93 units, and the average load is 31 unit. Table 1 represents the details of each iteration. The first column is the current iteration, the second column is the error of the current state, and the third

Algorithm 3: Greedy Load Balancing Algorithm (GLBA)

Result: Assignment of Switches, $S \rightarrow C$

$S_{i=1}^{k}$:= **LCSA** Clusters;
$state := S \rightarrow C$, current assignment of switches;
Set of all possible new states, $Pstates := \{state\}$;
while $Pstates \neq \varnothing$ **do**
  $state := \min\left(\varepsilon(Pstates)\right)$;
  $Pstates := \varnothing$;
  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

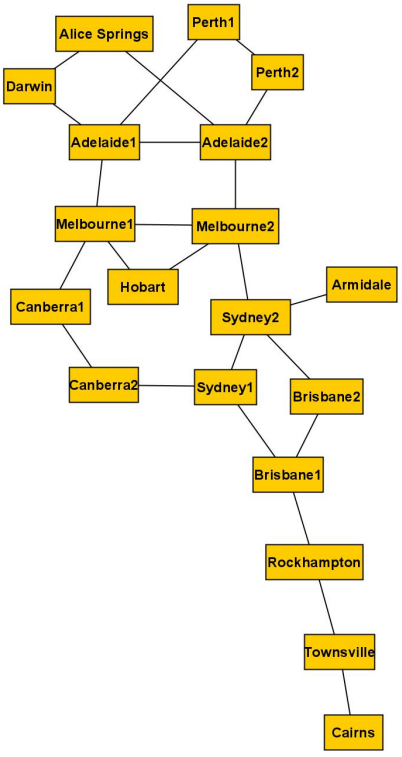

Figure 2: Original Aarnet Network (rearranged)

column is the best heuristic value of all the generated states. In iteration 3, the algorithm will terminate and return the current assignment as the current heuristic is 0.

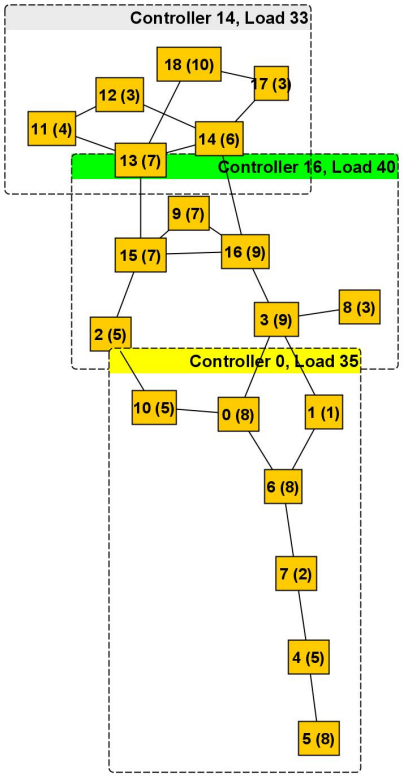

Figure 3: Root state with no child state

Table 1
BLBA applied on example network (Fig. 4)

Iteration	Error	Best Heuristic
1	14	2
2	2	0
3	0	N/A

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 α are determined.

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 [22]. 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 2.

The initial weights of the links are their bandwidths in Gb/s (Gigabits per second), which are converted into

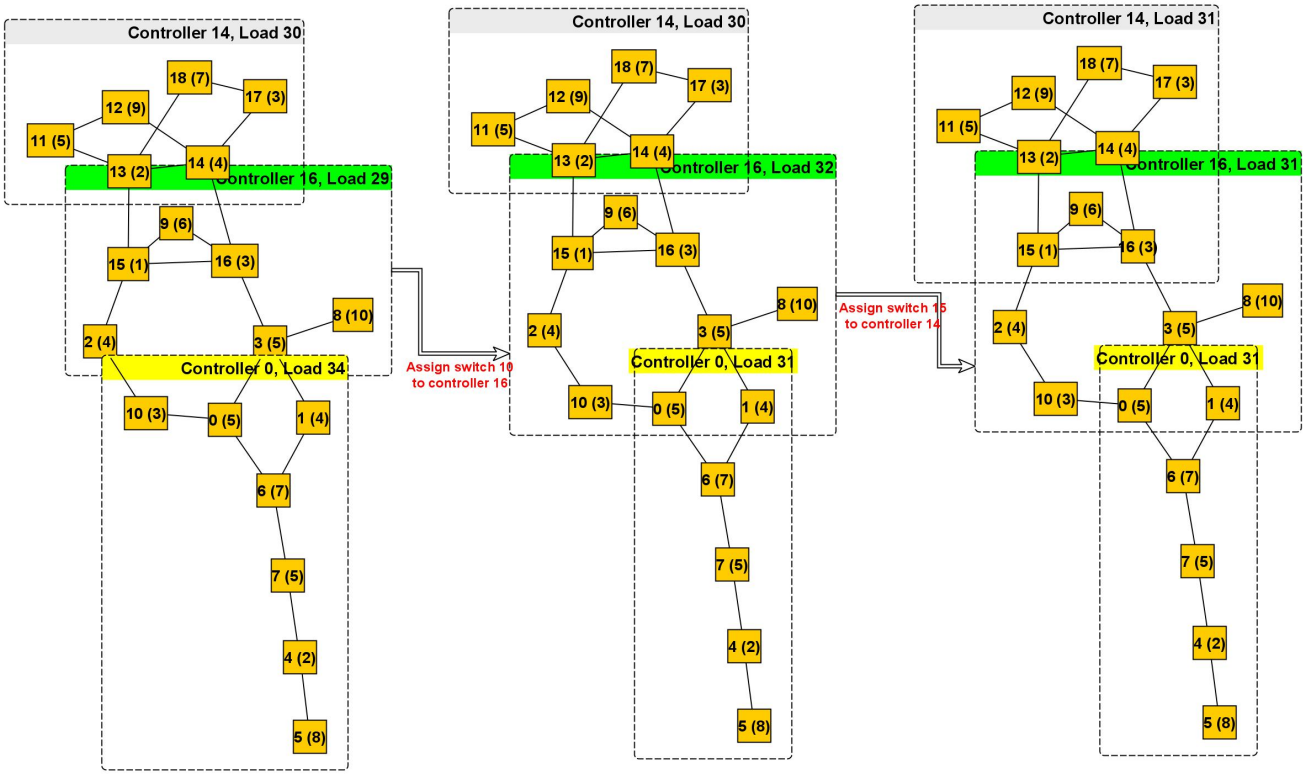

Figure 4: Optimal assignment scheme achieved after two iterations

Table 2
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

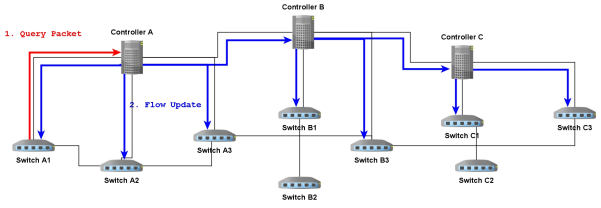

Figure 5: Setup of a new flow in the flow tables of the switches in the network

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 GLBA. 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 [3]. We assume the network has 10 controllers that are placed using LBCA and LCSA, 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 5). 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:

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

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

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 system time (service + waiting time) 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, and is denoted by:

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

where 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:

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

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

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 LCSA 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 LBCA on a small network with varying values of k and α (Figure 6) 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 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 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 7). 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,

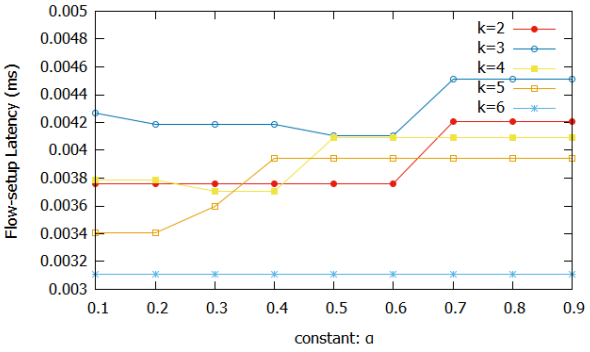

Figure 6: Average flow-setup latencies for varying values of α and k in a network containing $|S| = 11$ switches

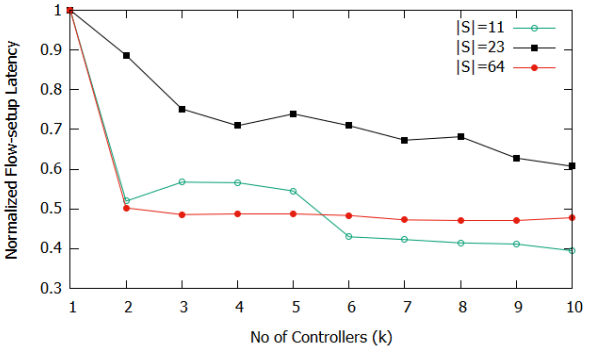

Figure 7: Decreasing average flow-setup latencies with respect to number of controllers (k) for different networks. As the networks have varying latencies, they are normalized for comparison.

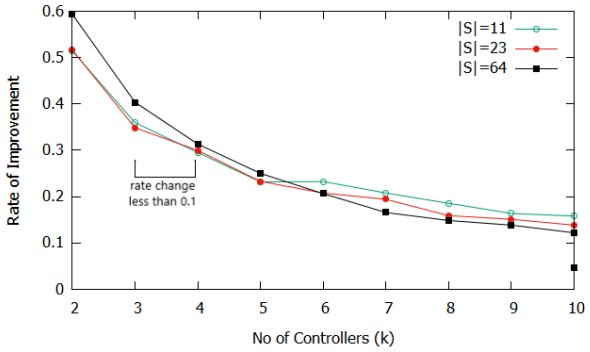

Figure 8: Gradually decreasing improvement ratio with respect to number of controllers (k) for networks of different sizes

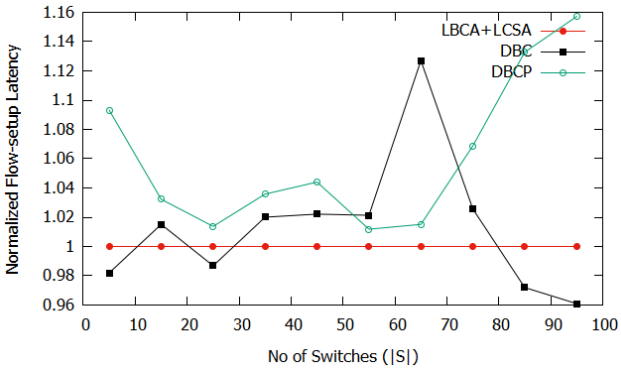

Figure 9: Comparison among LBCA+LCSA, DBCP and DBC, in terms of normalized flow-setup latency

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 8). 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 [18], 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

In the following sections 6.1 and 6.2, our proposed algorithms are validated by comparing them with the state-of-the-art controller placement [25] and load balancing algorithms [14], respectively.

6.1. Controller Placement

In this section, we evaluate our static controller placement method (LBCA+LCSA) by comparing it to the well-known algorithm DBCP [25] and the algorithm DBC [2]. 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

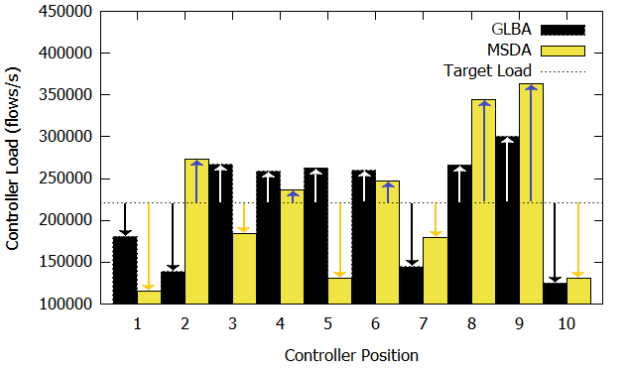

Figure 10: Comparison between GLBA and MSDA in terms of load per controller

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+LCSA 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+LCSA and plot their averages for a given range of network sizes (Figure 9). 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+LCSA outperforms both DBCP and DBC in 78% and 72% of the simulated networks, respectively.

6.2. Load Balancing

GLBA balances the loads of the controllers by swapping clusters of border nodes to avoid overlapping cluster formations. We set the maximum iteration limit of

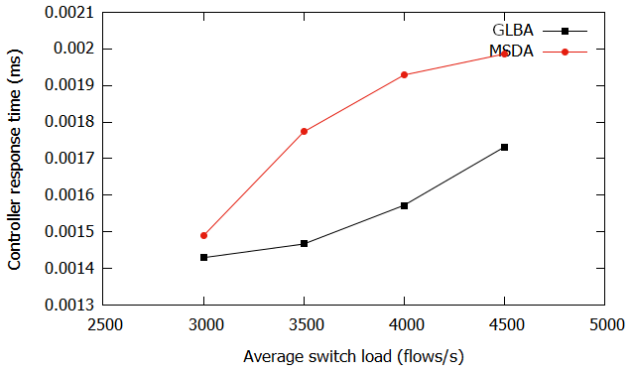

Figure 11: Gradually decreasing average response times for varying switch loads

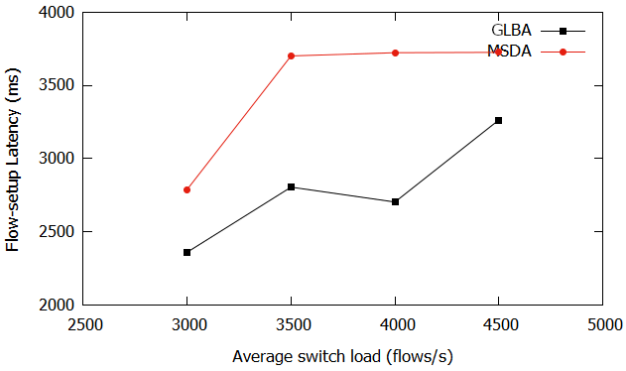

Figure 12: Comparison between GLBA and MSDA in terms of maximum flow-setup latency

GLBA to 100 and compare with the algorithm MSDA [14, 16] 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 $188.5K flows/s$ (Figure 10). 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 GLBA outperforms MSDA in terms of controller load, especially for controllers 8 and 9 (Figure 10). Although the algorithm converges after 10 iterations approximately, the maximum iteration limit can be increased for better accuracy and performance.

Figure 11 represents the comparison between GLBA 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 GLBA 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 12 and observe that GLBA outperforms MSDA. The flow-setup latency for higher average switch loads shows the minimum change for MSDA as the controller-switch preferences are prioritized. GLBA on the other hand reduces latency as long as a local optimum is available. Consequently, the algorithm terminates when further optimization causes overlapping clusters (producing unnecessary traffic), which is observed for an average switch load of 3500 flows/s.

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 GLBA 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

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

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

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

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

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

[6] Choumas, K., Giatsios, D., Flegkas, P., Korakis, T., 2020.

Sdn controller placement and switch assignment for low power iot. Electronics 9, 325.

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

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

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

[10] Dijkstra, E.W., et al., 1959. A note on two problems in connexion with graphs. Numerische mathematik 1, 269–271.

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

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

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

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

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

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

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

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

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

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

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

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

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

[24] Li, Y., Sun, W., Guan, S., 2020. 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), IEEE. pp. 351–355.

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

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

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

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

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

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

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

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

[33] Wang, T., Liu, F., Guo, J., Xu, H., 2016. 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. pp. 1–9.

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

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

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

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

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

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

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