Degree-based Balanced Clustering for Large-Scale Software Defined Networks

Talha Ibn Aziz*, Shadman Protik*, Md Sakhawat Hossen*, Salimur Choudhury† and Muhammad Mahbub Alam*
*Department of Computer Science and Engineering, Islamic University of Technology, Dhaka, Bangladesh
†Department of Computer Science, Lakehead University, Thunder Bay, Ontario, Canada
Email: *talhaibnaziz@iut-dhaka.edu, *shadmanprotik@iut-dhaka.edu, *sakhawat@iut-dhaka.edu,
†salimur.choudhury@lakeheadu.ca, *mma@iut-dhaka.edu

***Abstract*—The Controller Placement Problem (CPP), in Software Defined Networks (SDNs), deals with placing an optimal number of controllers. The CPP aims to maximize the network throughput and minimize different latencies like flow-setup latency and route-synchronization latency. In recent years, many solutions to the CPP have been proposed– some approaches work with a single parameter like the average propagation delay of a network, reliability, load balancing, etc., while some other approaches provide exhaustive solutions which optimize multiple parameters. However, very few researches propose non-exhaustive solutions which simultaneously optimize more than one parameter. We propose a novel controller placement algorithm which clusters the SDNs in polynomial time complexity. Our proposed algorithm Degree-based Balanced Clustering (DBC) minimizes overall flow-setup latency as well as route-synchronization latency and balances the loads of the controllers at the same time. DBC divides a network into several clusters, places a controller in each cluster, and also selects an optimal number of controllers. Simulation results suggest that DBC outperforms existing state-of-the-art algorithms in terms of different latencies and also performs load balancing among the controllers.**

***Index Terms*—CPP, SDN, latency, controller, load-balancing**

I. Introduction

Software Defined Networks (SDNs) decouple the traditional protocol stack into control plane and data plane. The control plane consists of controllers which take the routing decisions and relay this information to the data plane. The data plane is the collection of switches which forward the data according to the routing decisions provided by the control plane. The initial design of SDN included a single controller in the control plane [1]. However, even for moderate-sized networks, a bottleneck is created due to heavy traffic concentrated at the controller. Furthermore, problems like scalability and reliability surface as network sizes increase [2], [3]. In order to deal with these problems, many researches propose multiple controllers [4], [5], [6], [7], which results in the emergence of the Controller Placement Problem (CPP). The CPP deals with placing a minimal number of controllers in the SDNs, with an aim to address the above mentioned problems.

A solution to the CPP is to select an optimal number of controllers, to place them in the best possible locations, and to assign them switches in an optimal way [8], [9]. In addition, there are multiple constraints that need to be satisfied– minimizing the latency among switches and controllers, maximizing reliability and resilience, and minimizing deployment cost as well as energy consumption, which results in an NP-Hard problem. Despite the complications, in recent years, several solutions have been proposed to address the CPP [10], [11], [12], [13], [14], where some of them deal with optimizing a particular constraint like latency or reliability, while some others introduce a compound metric to address two or more constraints.

Heller et. al. [15] calculate the average and worst case latencies to place controllers and mention that multiple controllers are more significant in ensuring fault-tolerance than to minimize latency. Yao et. al. [16] propose dynamic scheduling strategies based on changing flows to manage controllers aiming to balance their loads. Hu et. al. [17] propose multiple algorithms for reliability-aware controller placement. Zhang et. al. [18] use a min-cut based graph partitioning algorithm to maximize resilience of SDNs and compare the algorithm with greedy approaches. In [19], Lange et. al. propose POCO (Pareto-based Optimal COntroller placement), which represents the CPP as a combinatorial optimization problem. POCO exhaustively iterates all possible combinations as viable solutions to optimize multiple metrics and uses heuristics-based PSA (Pareto Simulated Annealing) to improve the computational complexity at the cost of accuracy. Sallahi et. al. [20] propose a mathematical model for optimal controller placement considering the cost of installing controllers, loads of the controllers and path setup latency. Exhaustive solutions like [19], [20] can give optimal solutions, however, their computational complexities are very high.

Liao et. al. [21] propose two faster algorithms Density Based Controller Placement (DBCP) and Capacitated-DBCP (CDBCP) where DBCP minimizes overall latency based on the local density of the nodes and CDBCP balances the load of the controllers. However, they work separately on the given network to perform controller placement and to balance the load of the controllers. To the best of our knowledge, no other methods minimize overall latency and perform load balancing simultaneously, while clustering SDNs in polynomial time complexity. In this paper, we place minimum (optimal) number of controllers in a SDN to minimize the followings:

1) **Flow setup latency:** When a switch receives a packet for which no path exists, the flow-setup procedure is initiated. The latency introduced by this procedure is the maximum delay required to set the path (and also to inform the intermediate switches) for the flow.

2) **Route synchronization latency:** When there is a change in the network, routes are changed, and all the controllers are updated (in sync) about the changes. This latency deals with controller-to-controller latency.
3) **Load of a controller:** The volume of control traffic and processing that the switches impose on the controllers.

This requires placing the controllers in a way that minimizes both the average controller-to-switch and average controller-to-controller latencies and limits the load of the controllers. In this paper, we propose and develop an algorithm named Degree-based Balanced Clustering (DBC) to achieve the above with a careful selection of controllers. Simulation results suggest that our proposed algorithm DBC outperforms the state-of-the-art algorithms in terms of overall latency and load balancing. We also observe that the above criteria can be achieved if we can divide the SDN into k balanced clusters.

The rest of the paper is organized as follows– we provide our problem formulation along with some assumptions in Section II, a detailed description of our proposed mechanism in Section III, we present simulation results and performance evaluation in Section IV, and we conclude in Section V.

II. System Model and Assumptions

We consider that the network is a bi-directional graph $G = (S, L)$, where, the set of nodes S represent the switches and the set of edges L represent the links between the switches. The edges can be either weighted or unweighted based on the requirements. We cluster the graph G into multiple sub-networks such that, each sub-network is a disjoint set of switches. There cannot be any common switch between two sub-networks, and all of the switches of the network must fall into a sub-network, where each sub-network will have one and only one controller. Accordingly, if we assume that the network is partitioned into k sub-networks, then there will be k controllers and each sub-network will be a disjoint set of switches containing a single controller. For simplicity we make the following four assumptions:

1) The transmission delay of each control packet is identical and negligible.
2) The propagation delay, queuing delay and processing delay of each switch is identical.
3) The load imposed by each switch is fixed.
4) The controller can only replace a switch and cannot be placed in any other locations.

Under these assumptions, our objective is to minimize the flow-setup latency, the inter-controller distances and the controller-to-switch distances in each sub-network, all of which are measured in hop counts. We also aim to balance the load of each controller, which is the number of switches assigned to each controller.

III. Proposed Mechanism

We propose a solution to the CPP referred to as Degree-based Balanced Clustering (DBC) for SDN. The proposed mechanism divides the networks into k-clusters, assigns a controller to each cluster, and finally, finds the minimum value of k for the network which obtains the minimum latency. In the sequel, we explain the mechanism in detail.

A. Cluster Formation

The cluster formation mechanism divides the network into k clusters (sub-networks), when k is given. It finds the k cluster heads first and then, assigns each of the nodes to exactly one sub-network. The clustering mechanism deals with two goals–

- Equal load on the controllers, which requires equal number of nodes in each cluster. We assume that the switches have equal load.
- Minimum intra-cluster distances between the nodes and cluster heads, which requires direct (or shortest path) connectivity of the nodes and the cluster heads.

The first goal demands more clusters in the denser part of the network, whereas, the second goal demands the nodes with maximum direct connections with the switches to be the cluster heads. Accordingly, a cluster head should have the highest connectivity among all the switches of its cluster so that maximum number of nodes can be reached with minimum delay. Both the two goals can be achieved if the clustering is done based on degree of the node.

A degree based solution to the second goal suggests selecting k nodes with highest degrees as the cluster heads of the network. This solution ensures minimization of intra-cluster delay when the highest degree nodes are uniformly distributed (well separated). However, in typical networks, the higher degree nodes are situated in the same locality of the network. Consequently, the dense regions of the network are dominated by most of the cluster heads, and the clusters formed are not balanced in terms of load (first goal). Therefore, the cluster heads need to be a certain distance apart from each other, which we define as the minimum inter cluster head distance, T_d. The threshold, T_d, simultaneously maintains a fixed cluster head separation and balances the load of the network.

The cluster head separation dictates the number of switches in a cluster, which should be roughly $|S|/k$ for a balanced cluster. Initially, for $T_d = 1$ hop count, the nodes in the cluster are the cluster head itself and one-hop neighbors of the cluster head. Consequently, in an average case scenario, the total number of nodes is $1 + AvgDeg$, where $AvgDeg$ denotes the average node degree of the network. According to graph theory, each link simultaneously increments the degree of two switches by one, which implies that, the summation of the node degrees of a network is $2 \times |L|$ and the average node degree is $\frac{2\times|L|}{|S|}$.

We denote the total number of nodes in the periphery of the cluster as $boundary$. Initially, $boundary = AvgDeg$, as the cluster consists of a cluster head at the center and its one-hop neighbors on the border which is equivalent to $AvgDeg$ nodes on average. Each boundary node is again connected to approximately $AvgDeg$ number of nodes including the cluster head itself. Therefore, omitting the cluster head, for every increment of the value of T_d, the number of nodes in the cluster increases following the given formula,

$$boundary = boundary \times (AvgDeg - 1) \tag{1}$$

When the total number of nodes in the cluster exceeds $|S|/k$ nodes, we terminate the increment process. The value of T_d

thus obtained is the threshold distance or cluster separation of a balanced cluster.

In DBC, we sort the nodes according to descending value of node degree (line 1 of algorithm 1) and select the node with highest degree as first cluster head. We select a cluster head from the remaining nodes, only if it is at least T_d distance away from the selected cluster heads, otherwise we omit it, and move on to the next node in the sorted list. The region of the selected node may be sparser than other parts of the network. In order to take this into account, we multiply the threshold T_d with a degree ratio (line 13 of algorithm 1), which is the maximum degree of the network divided by the degree of the selected node. The degree ratio ranges the T_d value according to the density of the node, if the degree of the node is less, the density is lower. Accordingly, the ratio has a higher value ($\frac{Maximum\ Degree}{degree\ of\ selected\ node} > 1$) and increases the cluster separation threshold to include more nodes than in a denser region. However, if k cluster heads cannot be selected in this way, we select the remaining cluster heads only based on degree and ignore their distance from other cluster heads. Consequently, all the remaining nodes are assigned to the cluster of the nearest cluster head and k clusters are formed.

B. Controller Selection

After completion of the clustering process we initiate a controller selection process. The total delay of a network depends on both controller-to-controller latency and controller-to-switch latency. The controller-to-switch latency of a network improves when the switches are in close proximity of the controller and the controller-to-controller latency depends on the latencies of the paths between controllers. Although the cluster head has high intra cluster connectivity, inter cluster distance also needs to be taken into account. Considering both intra cluster and inter cluster distances (which refers to controller-to-controller latency), the cluster may or may not be the controller. Accordingly, we define $\tau(s)$ to be the controller selection function of switch s, which comprises its intra-cluster distance $\phi(s)$ and inter-cluster distance $\sigma(s)$ values. The intra-cluster distance can be calculated as the average distance from the switch s of cluster S_i to the other nodes of S_i [21],

$$\phi(s) = \frac{1}{|S_i| - 1} \sum_{u \in S_i} dis(s, u) \tag{2}$$

here, u represents any other switch in the same cluster S_i, as switch s and $|S_i|$ is the total number of switches in S_i. The component $\phi(s)$ corresponds to the controller-to-switch latency and $dis(s, u)$ is the shortest path distance of s from u.

The inter-cluster distance, which is the other component of total latency τ corresponds to the controller-to-controller latency. However, we cannot calculate inter-controller distances before the controller selection process as there are no controllers to begin with. Hence, we calculate $\sigma(s)$ for a switch s of cluster S_i, as the average distance of s from all other nodes of the network that are not in S_i [21],

$$\sigma(s) = \frac{1}{|S - S_i|} \sum_{v \in (S - S_i)} dis(s, v) \tag{3}$$

here, v denotes the switches that belong to network S but does not belong to sub-network S_i. The set of all nodes except sub-network S_i, is denoted by $S - S_i$.

The controller selection function $\tau(s)$ can be defined as the direct summation of the component latencies $\phi(s)$ and $\sigma(s)$. Although, this assigns equal weight to the inter-cluster latency and intra-cluster latency, the inter-cluster latency dominates overall latency as controller-to-controller distance is usually greater than controller-to-switch distance. We introduce a weight variable α to control this dominance and give the user the freedom to emphasize on any component as per requirement,

$$\tau(s) = \alpha \times \phi(s) + (1 - \alpha) \times \sigma(s) \tag{4}$$

In our experiments, we set the values of α as 0.5. As a result, both components contribute equally in the controller selection process. The pseudocode of our proposed method is given below (algorithm 1). The algorithm takes the network switches, S, links, L, and number of clusters, k, as input, and returns k optimal network clusters $C_1, C_2, ..., C_k$ and the set of $Controllers$ for the clusters. The worst case time complexity of the sorting (line 1 of algorithm 1) is $O(|S| \log(|S|))$, the cluster head separation determination (line 11-19) is $O(|S|)$, and the controller selection (27-31) is $O(|S|^2)$. Consequently, the over all worst case time complexity of DBC is $O(|S|^2)$.

C. Optimum k Selection

We evaluate the performance of each clustering using its average flow setup latency and denote this evaluation function by $\Omega(S)$ for a network S. When a data packet arrives at a switch s_i, for which no entry exists in its flow table, the switch forwards the packet encapsulated in a query message to its controller c_i. The controller decides the path for the packet and notifies the other switches of the path about the new flow rule, through their controllers. The switches on the path from s_i to destination s_d, may have different controllers. Consequently, the generated control packet containing the new route, needs to be forwarded to the controllers, which in turn will instruct the concerned switches to update their flow tables. Since the process is parallel, the path setup latency is dominated by the maximum of the sum of the distances from the corresponding controller of the source c_i, to controller c_j of any intermediate node s_j, and from c_j to s_j. Therefore, the evaluation function can be calculated as the average setup latency of all possible switch pairs [21],

$$\begin{aligned} \Omega(S) = & \frac{2}{|S| \times |S - 1|} \sum_{s_i, s_d \in S} \{dis(s_i, c_i) + \\ & \max_{s_j \in path_{i,d}} (dis(c_i, c_j) + dis(c_j, s_j))\} \end{aligned} \tag{5}$$

here, $path_{i,d}$ is the shortest path from source s_i to destination s_d. We use this function as the performance evaluation metric in section IV.

We increment k (starts at 1) and apply DBC repeatedly until the improvement in terms of $\Omega(S)$ is negligible. We define the improvement ratio or termination criteria ξ as follows,

$$\gamma = \frac{(\Omega_{old\ k}(S) - \Omega_{new\ k})/\Omega_{old\ k}}{new\ k - old\ k} \tag{6}$$

Algorithm 1 : Degree-based Balanced Clustering (DBC)

procedure DBC
input: k, S, L
Sort S in descending order of degree
Initialize $AvgDeg := \frac{2\times|L|}{|S|}$ and $boundary := AvgDeg$
Initialize $limit := 1 + AvgDeg$
Initialize threshold distance $T_d := 0$
Initialize $clusterHeads := \emptyset$
  while $limit < (|S|/k)$ **do**
    $boundary = boundary \times (AvgDeg - 1)$
    $limit := limit + boundary$; $T_d := T_d + 1$
  for each switch s in S **do**
    $adjT_d = T_d \times \frac{max\ degree}{degree\ of\ s}$
    if $clusterHeads = \emptyset$ **then**
      $clusterHeads.add(s)$
    else if $clusterHeads.size = k$ **then** break
    else if $dis(s, clusterHeads) < adjT_d$ **then**
      continue
    else
      $clusterHeads.add(s)$
  if $clusterHeads.size < k$ **then**
    Select remaining $clusterHeads$ with max degree
Initialize clusters $C_1, C_2, ..., C_k$ as $\emptyset$
Initialize $Controllers := \emptyset$
  for each cluster C_i **do**
    $C_i := clusterHeads[i]$
    $C_i := C_i \cup \{nearest\ nodes\ of\ C_i\}$
  for each cluster C_i **do**
    for all $s \in C_i$ **do**
      Calculate $\phi(s)$ and $\sigma(s)$
      $\tau(s) := \alpha \times \phi(s) + (1 - \alpha) \times \sigma(s)$
    $Controllers.add(node\ with\ min(\tau(s)))$
output: $Controllers$
output: $C_1, C_2, ..., C_k$

here, $old\ k$ is the previous value of k and $new\ k$ is the incremented value. In our simulations we notice a sharp drop in the value of γ after a certain number of increments.

IV. Performance Evaluation

A. Simulation Environment

We developed a simulation environment using C++ (High-level language) which takes the number of nodes and connections between each node as input and clusters the network. We use randomly generated networks with different numbers of switches starting from 40 to 100, at regular intervals of 10. There are 10 different datasets for each interval, giving a total of 70 different networks. The link to switch ratio is from 1.3 to 1.45 to keep the networks considerably sparse (similar to current worldwide networks). The simulation environment and selected 70 networks are same for both our algorithm and DBCP. The networks do not contain any self-loops or multiple links between two switches but may have cycles. The average number of edges of the scenarios are 54, 66.7, 81.4, 99.3, 111.2, 125.5 and 138.9 respectively.

B. Performance Metrics

We use various performance metrics to study and evaluate different characteristics of SDNs like inter-controller latency, controller-to-switch latency, and overall network switch to switch latency. We calculate the average inter-controller latency as follows,

$$\eta^{interController} = \frac{1}{k \times (k-1)} \sum_{1\leq i,j\leq k, i\neq j} dis(c_i, c_j) \quad (7)$$

where, c_i and c_j are controllers of the SDN. We evaluate the controller-to-switch latency using the following equation,

$$\eta^{controllerSwitch} = \frac{1}{k} \sum_{i=1}^{k} \left[\frac{1}{|S_i - 1|} \sum_{u\in S_i} dis(c_i, u) \right] \quad (8)$$

here, S_i is the i^{th} sub-network, c_i is the controller and u, the switch of S_i. Both $\eta^{controllerSwitch}$ and $\eta^{interController}$ consider the shortest distance from the controller to any other controller or switch. However, for setup of a new route, the switch sends a packet to the controller, which decides the new route and relays the information to other concerned controllers and switches. We calculate this latency using the equation 5 which has been discussed in section III-C.

We also evaluate the load of each controller, considering that all switches have the same load. For an ideal clustering of an SDN, the number of nodes per cluster should be $|S|/k$. However, the actual clustering may vary from the ideal case, which can be considered as the deviation. Accordingly, the least deviation from the ideal case is the most balanced in terms of load. The deviation can be denoted by,

$$\eta^{loadDeviation} = \frac{1}{k} \max_{i=1}^{k}(||S_i| - |S|/k|) \quad (9)$$

where, $|S_i|$ is the number of nodes in the i^{th} sub-network and $|S|$ is the total number of nodes in the network.

C. Simulation Results

We validate our proposed method through extensive simulations. We compare our proposed algorithm with DBCP [21] in terms of flow-setup latency, average inter-controller latency, average controller-to-switch latency and load balancing. To compare flow-setup latency, we use the value of k provided by DBCP to cluster the network and place the controllers. We plot the flow setup latencies as the average of the 10 instances in Fig. 1. We calculate the setup latency of each instance using equation 5 of section III-C.

Fig. 1 shows that DBC outperforms DBCP, however, for smaller networks the difference in performance is less (when number of switches is 40). In all other cases, the difference in performance is noticeable. We can observe that, for both the algorithms, the latency values (Ω) increase with the number of switches in the network. However, when the number of nodes is 90, the latency for both of the algorithms decrease slightly due to inconsistency of underlying topology.

In Fig. 2, we plot the flow-setup latency with respect to controller number k for networks containing 60, 70, and 80 switches. For greater values of k, the rate of improvement

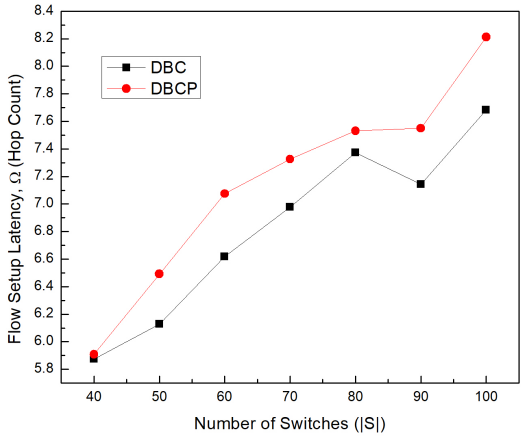

Fig. 1: Comparison of flow-setup latency between DBC and DBCP using the value of k provided by DBCP

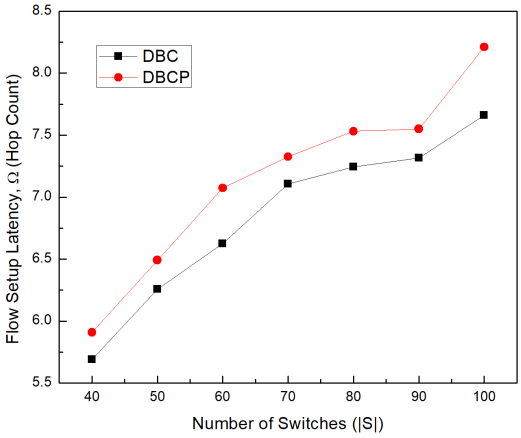

Fig. 3: Comparison between DBC and DBCP in terms of flow-latency, Ω

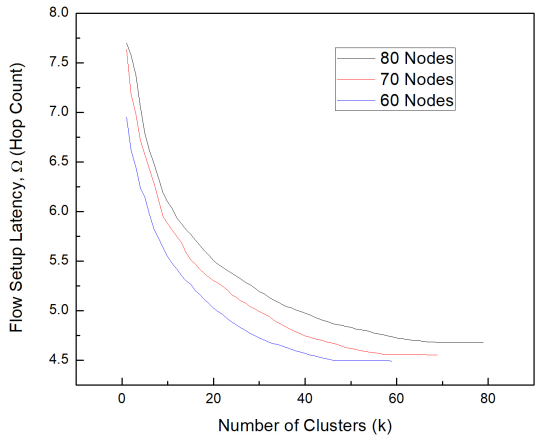

Fig. 2: Decreasing flow-setup latency with respect to increasing cluster numbers (k) for networks containing 60, 70, and 80 nodes

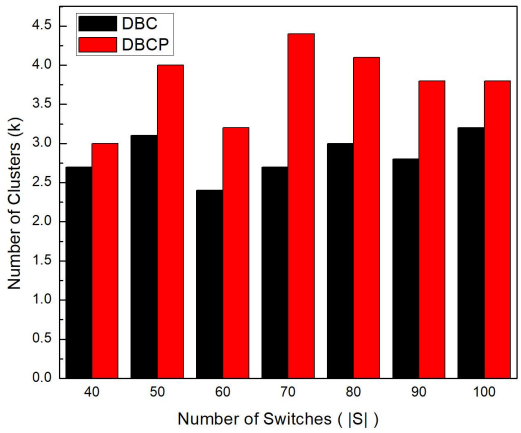

Fig. 4: Values of k given by DBCP and DBC for different scenarios

in terms of flow-setup latency is less as the placement of more controllers does not decrease the controller-to-switch distances significantly. We use this property to define an optimum k selection function (equation 6), using which we terminate the process when the improvement ratio, γ, is negligible (meaning negligible improvement). We notice a sharp drop in improvement rate at higher values of k where $\gamma > 0.05$, so we set the terminating threshold at 0.05. Accordingly, we plot the flow-setup latencies for the new values of k in Fig. 3, where, we observe that our algorithm is more consistent than previously selected k values which were selected by DBCP. The flow-setup latencies have also improved, and the number of controllers have decreased (Fig. 4). The number of controllers is comparatively less as the clusters are more balanced due to the consideration of a standard cluster-separation T_d. As a result of the formation of such clusters, the controller-switch and controller-controller distances are minimized. Consequently, in all the scenarios, DBC outperforms DBCP in terms of both flow-setup latency and minimum number of controllers.

We control the contribution of inter-controller latency and intra-cluster latency using a weight variable α (equation 4) which ranges from 0.0 to 1.0. When the value of α is increased, the intra-cluster latencies (equation 8) increase and the inter-controller latencies (equation 7) decrease as presented in Fig. 5. When the value of α is set to 0.5, inter-controller latencies are given priority by default, as the inter-controller distances are greater than intra-cluster distances. Our proposed algorithm outperforms DBCP in terms of inter-controller latency, however, DBCP gives better controller-to-switch latencies. We prioritize controller-to-controller latencies as control packets are essential in setting up new paths and conveying broken link information.

In Fig. 6, we compare the average, maximum, and minimum loads, of the controllers by adopting the number of switches per cluster which corresponds to the load of each controller. We observe that the deviation from the ideal load per controller in a balanced SDN ($|S|/k$), which can be calculated using equation 9, is higher for DBCP. Our proposed algorithm outperforms by forming clusters which are more balanced in terms of load per controller.

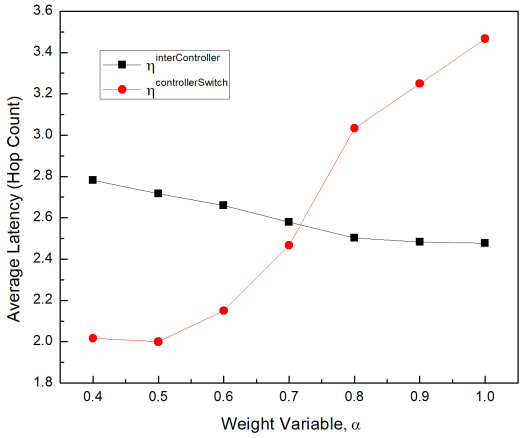

Fig. 5: Average controller-switch and controller-controller latency with respect to increasing weight variable, α

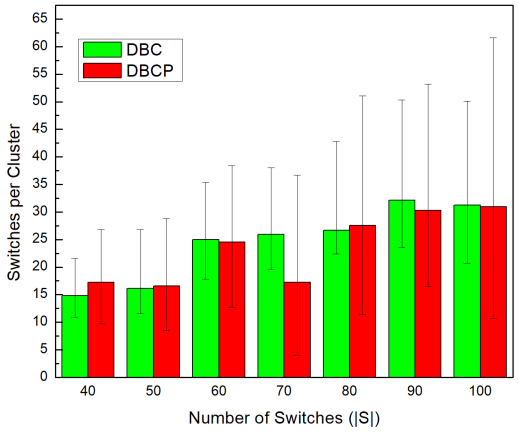

Fig. 6: Comparison between DBCP and DBC in terms of average load per controller.

V. Conclusion

In this paper, we address the Controller Placement Problem (CPP) of SDN and propose a novel clustering algorithm named Degree-based Balanced Clustering (DBC). DBCP is a recent clustering algorithm which addresses the same research problem and minimizes overall setup latency of the SDN. Our proposed algorithm, DBC outperforms DBCP in terms of different latencies and balances the load of the controllers. We have shown that our algorithm has many advantages over other algorithms. DBC minimizes flow-setup latency and route synchronization latency through minimization of controller-to-switch and controller-to-controller distances. DBC creates balanced clusters with similar number of nodes and has polynomial time complexity. Our proposed algorithm can be extended to work on weighted networks which consider ongoing traffic of links and different delays like queuing delay in a congested network to intelligently select flow routes. The hop count can be replaced with average edge weights and the inter cluster separation updated accordingly. Future work can also include variable loads of switches and balance the cluster so that minimum load is imposed on each controller.

References

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

[2] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 7–12.

[3] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine*, vol. 51, no. 2, pp. 136–141, 2013.

[4] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks." in *Nsdi*, vol. 10, 2010, pp. 19–19.

[5] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu *et al.*, "B4: Experience with a globally-deployed software defined wan," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 3–14.

[6] M. Berman, J. S. Chase, L. Landweber, A. Nakao, M. Ott, D. Raychaudhuri, R. Ricci, and I. Seskar, "Geni: A federated testbed for innovative network experiments," *Computer Networks*, vol. 61, pp. 5–23, 2014.

[7] I. T. Haque and N. Abu-Ghazaleh, "Wireless software defined networking: A survey and taxonomy," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2713–2737, 2016.

[8] R. Ahmed and R. Boutaba, "Design considerations for managing wide area software defined networks," *IEEE Communications Magazine*, vol. 52, no. 7, pp. 116–123, 2014.

[9] S. Lange, S. Gebert, J. Spoerhase, P. Rygielski, T. Zinner, S. Kounev, and P. Tran-Gia, "Specialized heuristics for the controller placement problem in large scale sdn networks," in *Teletraffic Congress (ITC 27), 2015 27th International*. Ghent, Belgium: IEEE, 2015, pp. 210–218.

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

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

[12] K. Sudheera, M. Ma, and P. Chong, "Controller placement optimization in hierarchical distributed software defined vehicular networks," vol. 135, pp. 225–239, Apr 2018.

[13] J. H. Cox, J. Chung, S. Donovan, J. Ivey, R. J. Clark, G. Riley, and H. L. Owen, "Advancing software-defined networks: A survey," *IEEE Access*, vol. 5, pp. 25 487–25 526, 2017.

[14] K. Sood, S. Yu, and Y. Xiang, "Software-defined wireless networking opportunities and challenges for internet-of-things: A review," *IEEE Internet of Things Journal*, vol. 3, no. 4, pp. 453–463, 2016.

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

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

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

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

[19] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale sdn networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 4–17, 2015.

[20] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE communications letters*, vol. 19, no. 1, pp. 30–33, 2015.

[21] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li, and T. Li, "Density cluster based approach for controller placement problem in large-scale software defined networkings," *Computer Networks*, vol. 112, pp. 24–35, 2017.