A Comparative Study on Load Balancing Techniques in Software Defined Networks

Moinul Islam Sayed
Department of Computer Science and Information Technology, Patuakhali Science and Technology University, Bangladesh
Email: sayed.pstu11@gmail.com
Sajal Saha
Department of Computer and Communication Engineering, Patuakhali Science and Technology University, Bangladesh
Email: sajal.saha@pstu.ac.bd
Ibrahim Mohammed Sayem
Department of Computer and Communication Engineering, University of Western Ontario, Canada
Email: isayem@uwo.ca
Sarna Majumder
Department of Computer and Communication Engineering, Patuakhali Science and Technology University, Bangladesh
Email: sarna.majumder90@gmail.com

ABSTRACT

Software-defined networking (SDN), which decouples the control plane from data plane and provides programmability to design the network, has been considered as a viable paradigm shift to ease the management of conventional networks. Studies have identified that the placement of controllers heavily impacts network performance in SDN. Many studies proposed methods regarding controller's placement in the network to improve the performance metrics such as propagation latency, distribution of load, failure resilience, and reliability of network. However, network operators' main concern is always Quality of Service (QoS) when placing SDN controllers. Because SDN controllers are responsible for providing services to the switches, controller response time is a critical QoS criterion for network operators. In this study two different approaches of controller placement were thoroughly examined and combined to offer a solution that minimizes the propagation delay among nodes and maximizes the QoS of the network by maintaining better load balancing.

Keywords - **Software Defined Networks, Controller Placement, QoS, Load-balance, Propagation delay.**

Date of Submission: Jan 22, 2022 Date of Acceptance: Feb 05, 2022

1. INTRODUCTION

SDN has been considered as a promising new paradigm that decouples the control plane from data plane, allowing for programmability in network configuration, to make traditional network management more flexible. Controller(s) in the control plane are responsible for making routing decisions, as well as generating rules and policies for the routers and switches in the network. The data plane is a set of switches that forward data in accordance with the control plane's routing decisions. The control plane of SDN was originally designed with a single controller. Having a single controller within a network can be advantageous since it provides a single view of the entire network [13]. However, because of the massive traffic condensed at the controller, even a medium-sized network with just a single controller suffers from many efficiency and scalability issues [30], [29].

As a solution to these problems, having multiple controllers is a feasible alternative. However, the number of the controllers and their placement has immense effect on the network's efficiency and cost [13], [36]. As a result, Controller Placement Problem (CPP) emerges as a hotspot in recent SDN research. The CPP focuses on three main points: 1) determining minimum number of controllers; 2) finding optimum placement of controllers in the network; and 3) distributing controllers among switches with the aim of reducing latency cost [11], [28], enhancing reliability [32], and optimizing energy efficiency [13].

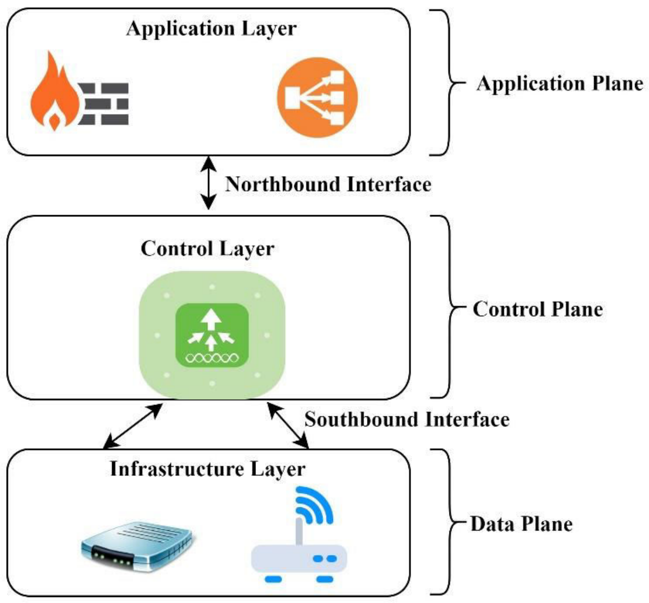

Fig 1: SDN Architecture

The main contributions of this work are as follow:

- Several research papers have been studied for analysing the solution to CPP. Some of the research papers were selected to study intensely and discussed in the following sections.
- Two algorithms that propose solution to CPP focusing on two different perspectives were selected and implemented.
- Combining two novel algorithms, a new algorithm is proposed and implemented for comparative performance analysis.

2. RELATED LITERATURE

SDN proffers more flexibility to network vendors when designing the network by isolating the network's control plane and packet forwarding plane. The controllers in the SDN control plane are responsible for prescribing network behaviors and making routing decisions on how data packets should be forwarded by building a routing table with network paths. After receiving routing decisions from the control plane, the data plane, which is made up of switches, transmits packets to the desired destination [1], [2]. The centralized control plane performs well in the smaller network. One of the first SDN controllers, NOX, can only serve 30K flow requests per second with a response time of fewer than 10 milliseconds [3]. However, the SDN controller confronts a fundamental difficulty in terms of scalability and reliability when dealing with networks with multiple flows, which we refer to as large-scale networks [4], [5], [6]. Many researchers believe that the solution to this problem is to develop a decentralized control plane that can distribute the load among multiple controllers where controllers can cooperate each other [4], [7], [8], [9]. Conversely, using multiple controllers causes the Controller Placement Problem (CPP), which refers to where numerous controllers should be placed in an SDN-enabled network [10].

The Controller Placement Problem (CPP) has an impact on network performance parameters like latency (flow setup latency/time and route synchronization latency), throughput, network availability, controller load balances, and energy consumption. It is suggested that in order to address the CPP problem, it is necessary to define not only the minimum number of controllers, but also their placements [11], [12]. Furthermore, multiple constraints must be met, including minimizing packet propagation and controller processing latency between switches and controllers, intensifying resilience and reliability, and reducing the cost of placing controllers, linking controllers to switches, and connecting controllers together, resulting in an NP-Hard problem [10], [13], [14]. However, in recent years, many researchers have proposed several solutions addressing the CPP where some of them are based on specific constraints like the total delay of the network or load balancing or reliability as well as scalability [15], [16] and while some other provides solutions which addressed multiple constraints [11], [17], [18].

Yao et al. [19] describe CPP with load constraints and propose an efficient scheduling algorithm that reduces the number of controllers required to balance loads. Bari et al. [20] propose an Integer Linear Programming (ILP) heuristic approach to address the dynamic controller provisioning problem, which decreases flow setup periods and controller workload by dynamically altering the amount and position of controllers. Zhao and Wu's [21] simulation findings suggest that a heuristic-based technique produces superior results than integer linear program outcomes. Zhang et al. [22] presented and compared a min-cut based controller placement method that increases reliability with a greedy based approach. Hu et al. [23] try to concentrate on the reliability-aware controller placement problem and propose the expected percentage of control path loss, where control path loss refers to broken control paths caused by network failures. Sallahi et al. [13] consider the cost of installing controllers, the loads on the controllers, and the path setup latency when developing a mathematical model for optimal controller location. Hock et al. [27] use the Pareto-based Optimal Controller-placement (POCO) framework to compute resilient placements by considering several variables such as network reliability, controller load, and propagation latency. According to Lange et al. [24] Some parameters, such as controller-switches latency, controller-controller latency, controller load, and network failures, are significant in identifying the number and placement of required controllers in a large-scale network. Optimal solution can be achieved found from [23], [24] but their computational cost is much higher than other proposed method.

In [26] Liao et al. proposed Density-Based Controller Placement (DBCP), in which a huge network is reduced to a small network and the overall latency is reduced based on the nodes' local density. Moreover, it provides better result for CPP compared to Lange [24]. Wang et al. [25] introduced an improved K-means network partitioning algorithm, in which a network is partitioned using the K-means algorithm and controllers are placed in the partitioned networks. Heller et al. [11] was the first to bring up the controller placement problem. The aim of this research is to reduce the average latency between switches and controllers. Since the Figure 2.1: Basic architecture of SDN. Page 4 of 20 average and maximum latency cannot be optimized simultaneously, a greedy K-median method was adopted for the optimal placement that minimizes average latency, and a greedy k-center method was used to reduce the maximum latency. However, this approach is restricted to particular topologies and ignores the controller's constraints in capacity, making it unsuitable for real-world networks. Singh et al. [35] proposed an algorithm based on optimization that minimizes the total average latency. This study also concluded that optimization-based solution shows better performance when compared to solutions based on clustering. When it comes to capacitated CPP, the load of controller is taken into account. Yao et al. [36] suggest dynamic scheduling techniques for managing controllers that seek to balance the loads based on varying flows. [33] also considers the load distribution and communication time for the controllers when mapping the control plane., To balance the load of the controllers, Liao et al. [34] proposed to partition the whole SDN into smaller clusters based on the density of nodes and with each cluster having one controller.

In [27], failure tolerance is taken into account when positioning controllers in the network. This study found that while one controller is sufficient in terms of latency, further controllers are needed to satisfy reliability requirements. In addition, inter-controller latency, balancing load of the controllers, and the trade-off between failure tolerance and latency were also considered in the study. Multiple algorithms for reliability-aware controller positioning were proposed by Hu et al. [32]. Cheng et al. [31] considered QoS parameters for the network when devising solution for CPP. In this study the loads of the switches have been an influential parameter in computing positions for the controllers in the network. Sallahi et al. [13] suggest a mathematical model for optimum controller positioning that takes into account installation costs, controller loads, and route setup latency.

From the studies mentioned above, the following determinants found for consideration when designing solution for CPP.

• Latency: The amount of time required for packet transfer between nodes in the network. This includes switch to switch, controller to switch, and controller to controller latency. the lion's share of the previous research has attempted to solve the CPP issue by minimizing switch-to-controller latency while ignoring inter-controller latency and network complete latency. However, [29] proposed a degree-based clustering method that considered the inter-controller latency.
• Load balance: Switches need to manage a limited number of outgoing and incoming requests (receiving packets or sending packets). The packets can be dropped if the switch is overloaded. As a result, packet loss can reduce the SDN performance.
• Inter-controller communication: If a large number of controllers are needed to handle a network, the inter-controller communication complexity would rise, but the network's overall efficiency would improve.
• Capacity of the controller: Because of scarce resources such as memory and cpu, an SDN controller can only manage a certain number of switches.

3. METHODOLOGY

Several methods to solve the CPP have been introduced in recent years, some of which focus on optimizing a single constraint such as latency or efficiency, while others use a compound metric to solve two or more constraints. Two innovative solutions to CPP have been chosen for analysis and implementation. Then a novel algorithm is proposed based on these two-reference algorithms, combining some of its methods to maximize the performance metrics.

3.1 Degree-based Balanced Clustering (DBC)

[29] proposed this algorithm that minimizes the flow-setup latency, route synchronization latency and also optimize the loads of the controllers. Here flow setup latency is the new path setup delay that occurred when a packet is received by a switch for which no corresponding path exists. Route synchronization involves delay in updating the routes in network. This algorithm divides the whole SDN into several smaller clusters and selects one controller for each cluster. Switches in a cluster are selected based on their connection configurations. Mostly connected and nearer (minimum intra-cluster distance) switches remain in the same cluster. However, for the load balance, this algorithm tends to assign equal number of switches in each cluster. Then a controller is selected in the cluster based on inter-controller and intra-cluster connection and the distances of these connections. As one goal of this algorithm is to minimize the intra-cluster distances in clusters, it would select controllers with higher degrees of connections. However, in practical, nodes with higher degrees of connections incline to remain in the same locality of the network. As a result, most controllers may remain in the dense part of the network. It can result in misdistribution of the clusters. To solve this problem, it uses a threshold value Td, which maintains an optimum distance from each cluster heads. This algorithm firstly selects K number of cluster heads with higher degree and maintaining the threshold distance. Then to form cluster, each cluster head expands its boundary, initially from one hop count then increasing the number of hope count until the total nodes in a cluster exceeds the number |S|/K. Then in each cluster, it selects a controller from all nodes in that cluster such that it minimizes the intra-cluster and inter-controller latency. This algorithm returns k number of clusters and controllers from a given network (switches and connections). First it selects k number of cluster heads based on the degree and edge distances. If it cannot select k cluster heads in this way, the remaining cluster heads are filled by the nodes with higher degrees. As there are k number of cluster heads, all the switches are assigned to one of these clusters. Then it selects one controller based on the edge distances. Therefore, this algorithm correctly gives output of k number of clusters. Here all the for loops iterate through each switch. The only while loop has a condition “limit < (|S|=k)”. The value of limit is guaranteed to be increased. Therefore, this algorithm has nothing to be stuck into, this has to terminate. This algorithm has a for loop with another nested for loop. All the loop statement in this algorithm can iterate maximum N times (number of nodes). Therefore, a for loop with another nested for loop will iterate at most NxN number of times. Therefore, the time complexity is: $O(N^2)$. The maximum length of any data structure used here is to store the corresponding value for all the switches. As there are N number of switches, the space complexity is: $O(N^2)$.

3.2 QoS-Guaranteed Incremental Greedy Algorithm (QGIG)

[31] Proposed this algorithm that takes into account the QoS and optimize the loads of the controllers. Here the algorithm finds the required minimum number of controllers and their placement based on the response time of the controllers so that the SDN is optimized for the QoS. The algorithm uses a heuristic algorithm named as incremental greedy algorithm that iteratively select controller candidates that can serve maximum number of switches until all switches are assigned to a controller. One unique idea made this solution novel that, this algorithm considers the requests and loads from the switches to determine the clusters. For every node in the network, this algorithm computes the clusters considering the node as a controller. It keeps adding the closest nodes in the cluster until it reaches the maximum load. In every iteration, the cluster with maximum number of switches is selected. Then it does the same in the further iterations until every node is assigned to a controller. Here, S denotes number of switches, L denotes the links, δ denotes response time bound, Xi,j denotes ith switch that is served by the jth controller, R represents the remaining switches, C represents set of controller candidate sites and Sj represents the set of switch that are served by j^{th} controller. This algorithm returns optimum number of clusters and controllers from a given network (switches and connections). First it finds closest set of switches within maximum load limit for every node. Then it keeps only one cluster with maximum nodes. It continues until all the nodes are assign to a cluster. It will terminate when there is no switch that is not assigned to any of the clusters. If there are N numbers of switches, there are three nested loops, and each have potential to iterate N times. Therefore, the time complexity is $O(N^3)$. The maximum length of any data structure used here is to store the corresponding value for all the switches. As there are N number of switches, the space complexity is: O(N).

3.3 Proposed QoS-Guaranteed Degree-based Balanced Clustering

Degree based Balanced Clustering is a novel solution that focus on reducing the intra-cluster and inter-cluster latency. The controllers are selected based on the degree of connections. Switches with higher degree have more potential to be selected as controller. However, this algorithm allows more switches in the cluster in the dense area. As a result, controllers in the dense area may encounter more loads than capacity and controllers in sparse area may encounter less loads than usual. On the other hand, Incremental Greedy Algorithm focuses on the load distribution of the controllers. This algorithm computes cluster combination for every unserved switch in every iteration. This raises the complexity for the algorithm to install and maintain large scale SDN. Also, for selecting controllers, this algorithm considers the maximum number of switches that can be served by any controller with cumulative load remain in the limit of maximum load. Therefore, intra-cluster and inter-cluster latencies for the controllers are not taken into account. Addressing the gap between two novel algorithms, a new algorithm is proposed that ensures the QoS of the SDN by maintaining load balance among controllers and at the same time the algorithm considers clusters and controllers that minimize the intra-cluster and inter-cluster latencies. We assume the network to be a bi-directional graph G = (S, L), with the nodes S representing the switches and the edges L representing the connections between them. According to the demands, the edges might be weighted or unweighted. We divide the graph G into sub-networks, each of which has a disjoint collection of switches. There can't be a common switch between two sub-networks, thus all of the network's switches must be divided into sub-networks, each with its own controller. Our proposed algorithm splits the network into n-clusters and allocates a switch to each cluster as a controller. The clustering method aims for equal load on controllers and the shortest possible intra-cluster distances between nodes. This technique uses more clusters in the denser portion to distribute loads evenly across the controllers. To reduce the latency in a cluster, the node with the highest connectivity to the other switches in the cluster is chosen as the controller. This method initially chooses n-nodes as cluster heads. Cluster heads are chosen from nodes with a greater degree of connection. However, in a real-world scenario, nodes with greater degrees are discovered in the same network location. Therefore, a certain spacing between the cluster heads is required, we defined as ThrDis. The average degree AvgDegree can be calculated as (2x Links)/ Switches as each link increases the degree of two switches by one. Limit holds the maximum allowed number of switches in clusters. initially it is made up of the cluster head and its immediate neighbours which is 1 + AvgDegree. We termed the switches on the cluster's outskirts as Boundary. Limit is increased by Boundary for each ThrDis increment. The limit is raised until the number of switches per cluster is fewer than the average. The switches are ordered by degree from greatest to smallest so that the cluster head may be chosen from the nodes with the highest degree. Following the selection of the initial cluster head with the highest degree, further cluster heads with higher degrees and a distance of at least ThrDis from the other cluster heads are chosen. However, higher-degree nodes prefer to stay in the same or close vicinity of the network. As a result, ThrDis should be modified for the network's dense and sparse parts. We used ThrDis to multiply the ratio of the maximum degree to the degree of current switch, such that ThrDis is lower in dense areas and higher in sparse areas. If n clusters cannot be chosen in this fashion, this approach simply adds the nodes with higher degrees while ignoring ThrDis to fill the n-clusters.

Algorithm 1: QoS Guaranteed Degree-based Balanced Clustering

```
1: procedure QGBC
2: input: $n, Switches, Links$
3: Initialize $AvgDegree \leftarrow (2x\ Links) / Switches$
4: Initialize $Boundary \leftarrow AvgDegree$
5: Initialize $Limit \leftarrow 1 + AvgDegree$
```

```
6:  Initialize $ThrDis \leftarrow 0$
7:  Initialize $SortedSwitches \leftarrow Switches$ sorted by degree (largest to smallest)
8:  Initialize $ClstrHeads \leftarrow \emptyset$
9:  Initialize $MaxLoad \leftarrow \Sigma\ Load_i / Switches$
10:     while $Limit < (Switches / n)$ do
11:         $Boundary \leftarrow Boundary$ x $(AvgDegree-1)$
12:         $Limit \leftarrow Limit + Boundary$
13:         $ThrDis \leftarrow ThrDis + 1$
14:     for each switch $s$ in $Switches$ do
15:         $AdjustedThrDis \leftarrow ThrDis$ x (maximum degree / degree of s)
16:         if $ClstrHeads = \emptyset$ then
17:             $ClstrHeads$.add(s)
18:         else if $ClstrHeads$.size = n then break
19:         else if $MinDistance(s, ClstrHeads) < AdjustedThrDis$ then continue
20:         else $ClstrHeads$.add(s)
21:     if $ClstrHeads$.size < n then
22:         while $ClstrHeads$.size < n do
23:             for each switch Si in $SortedSwitches$ do
24:                 if Si not in $ClstrHeads$ then
25:                     $ClstrHeads$.add($S_i$)
26:                     break
27:     Initialize Clusters $C_1, C_2, \ldots, C_n$ as $\emptyset$
28:     Initialize Cluster Loads $L_1, L_2, \ldots, L_n$ as 0
29:     for each $i$ in (1 to $n$) do
30:         $Ci$.add($ClstrHeads_i$)
31:         $Li \leftarrow Li$ + LoadOf($ClstrHeads_i$)
32:     Initialize $Controllers \leftarrow \emptyset$
33:     for each $s$ in $Switches$ do
34:         $SortedClusters \leftarrow ClstrHeads$ sorted by distance from s (smallest to largest)
35:         for each $ClstrHead$ in $SortedClusters$ do
36:             $C \leftarrow$ corresponding Cluster for $ClstrHead$
37:             if $(L_C + Load\ of\ s) < MaxLoad$ then
38:                 $C$.add(s)
39:                 $L_C = L_C + L_S$
40:                 break
41:     for each clusters $C_i$ do
42:         for each $s$ in $C_i$ do
43:             Compute $D_s = \Sigma$ Distances from nodes in $C_i$
44:         $Controllers$.add(switch with min($D_s$))
45: Output: $Controllers$
```

Initially cluster heads are the first element in every cluster. To populate these clusters, this method starts with each switch and sort the cluster heads by the distance in ascending order. Each switch is added to the cluster with minimum distance and the clusters' load is increased by the switch load. If the cluster is full, or if adding the load of the current switch to the load of the cluster exceeds the maximum permitted load, the switch moves to the next clusters in the sorted list and initiate likewise procedure until it is added to any cluster. After finishing populating these clusters, in each cluster for each switch the total distance from all the other nodes in the same cluster is computed and switch with the minimum distance is selected as the controller in a cluster.

4. RESULT AND DISCUSSION

The proposed algorithm is implemented along with the other two reference algorithm: DBC in [29] and QGIG in [31]. All these three algorithms were provided with 200 switches with exact same connections and the distances of these connections were identical. For simulation, each of the 200 switches were associated with random loads. In order to compare these algorithms in terms of inter-controller distance, intra-cluster distance, node distribution among clusters and comparative load distribution, we specified 8 clusters for 200 switches.

The proposed algorithm, DBC and QGIG were very similar in the range of numbers of switches for individual clusters. 30 was the highest number of switches in any cluster for QGIG. For DBC and the proposed algorithm, the highest number of switches in any cluster is 25. QGIG also has the lowest number of switches in any cluster, which is 16. The lowest number of switches in any cluster for DBC and proposed algorithm are 19 and 22 respectively. The range of numbers of switches for individual clusters is minimum for the proposed algorithm. Therefore, the proposed algorithm provides slightly uniform distribution of switches among clusters. Fig 2 shows the detail distribution of switches among clusters.

Intra-cluster distance is the sum of distances from each switch to every other switch in a cluster. If the intra-cluster distance is less, then switches can communicate among themselves with minimum time and cluster become cost-efficient. Among all these three algorithms, though proposed algorithm shows the lowest 36.25 total intra-cluster distance, this is very similar to the DBC algorithm which shows 36.5 total intra-cluster distance. However, QGIG shows highest 41 total intra-cluster distance. Therefore, the proposed algorithm and DBC exhibit better results in terms of intra-cluster distance. Fig 3 shows the distribution of intra-cluster distance for the proposed algorithm as well as other two reference algorithm.

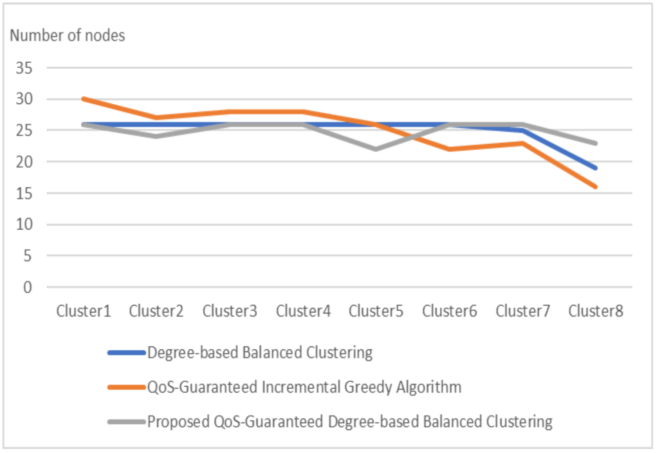

Fig 2: Distribution of nodes among clusters

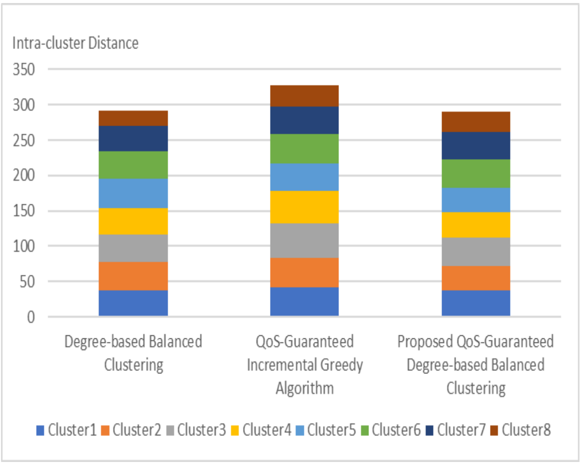

Fig 3: Distribution of intra-cluster distance

Inter-controller distance is the sum of distances from each controller to every other controller. The lesser the value for inter-controller distance the easier for the clusters to communicate. Thus, lesser value for inter-controller distance is appreciated. The proposed algorithm and DBC show similar inter-controller distances. However, QGIG algorithm shows bigger values for inter-cluster distance. Fig 4 shows the cluster-wise inter-controller distances for all the three algorithms.

For QoS of the SDN, load balance is one of the crucial performance metrics. Uniform distribution of loads among clusters makes the SDN more balanced and QoS efficient. In this simulation every switch was associated with a load. For comparison, we calculate the sum of loads in every cluster. Then we calculate the average load for the clusters in each of the three algorithms. Lasty, we calculate the difference between average load and cluster's total load. Fig 5 shows the difference between average loads and total load for every cluster in each algorithm. The results indicate impressive performance from the proposed algorithm, having most uniform hence most balanced distribution of loads among clusters. For the difference between average load and cluster's total load, the proposed algorithm has range of values from 1.375 to 24.375. In this case the range for DBC is from 40.25 to 147.75 and for QGIG is from 28.125 to 340.875. It is evident that the proposed algorithm outperformed other two reference algorithms in terms of load balancing.

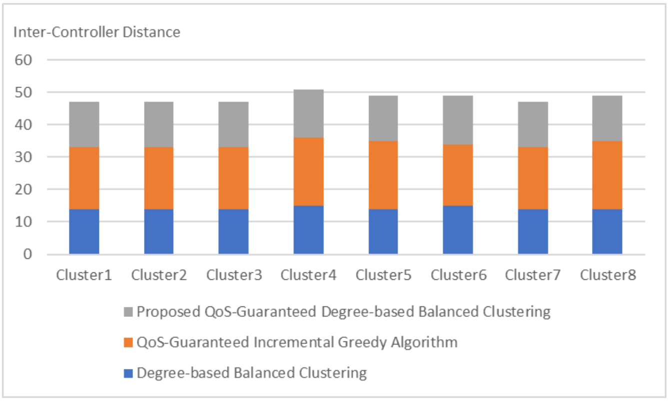

Fig 4: Cluster-wise inter-controller distances.

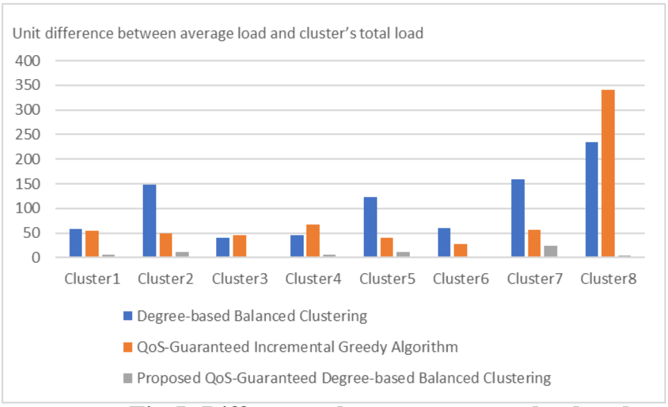

Fig 5: Differences between average load and cluster's total load.

The proposed algorithm returns k number of clusters and controllers from a given network (switches and connections). First it selects k number of cluster heads based on the degree and edge distances. If it cannot select k cluster heads in this way, the remaining cluster heads are filled by the nodes with higher degrees. As there are k number of cluster heads, all the switches are assigned to one of these clusters based on edge distances and cluster loads. Then the controller is selected from each cluster based on intra-cluster and interclassed distances. Therefore, this algorithm correctly gives output of k number of clusters. Here all the for loops iterate through each switch. The only while loop has a condition "limit < (|S|=k)". The value of limit is guaranteed to be increased. Therefore, this algorithm has nothing to be stuck into, this has to terminate. This algorithm has two for loops with another nested for loop inside. All the loop statement in this algorithm can iterate maximum N times (number of nodes). A for loop with another nested for loop will iterate at most NxN number of times. Therefore, the time complexity is O(N2). The maximum length of any data structure used here is tos store the corresponding value for all the switches. As there are N number of switches, the space complexity is O(N2). DBC has similar time and space complexity. However, QGIG exhibits O(N3) time complexity. So, in terms of complexity analysis the proposed algorithm and DBC show better results than QGIG.

5. Conclusion

Decision makers face various obstacles while designing the control plane of an SDN-based network. Even though the needed number of components in the abstracted control plane is determined, the positions of those components have a significant impact on the system's performance. This paper looks at the controller placement problem in terms of a variety of significant metrics, including intra-cluster and inter-cluster latencies, load distribution, and quality of service (QoS). Previously the methods that addressed controller placement problem, either focused on latencies and load distribution, or quality of service (QoS). Our proposed algorithm: QoS

Guaranteed Balanced Clustering focuses on all the metrics, and the proposed approach outperforms the other two reference algorithm discussed in this study, by a little margin, according to simulation results.

REFERENCES

[1]. Diego Kreutz, Fernando M.V. Ramos, and Paulo Verissimo. 2013. Towards secure and dependable software-defined networks. In Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking (HotSDN '13). Association for Computing Machinery, New York, NY, USA, 55–60.

[2]. Cabaj, Krzysztof, et al. "SDN Architecture Impact on Network Security." FedCSIS (Position Papers). 2014.

[3]. Tavakoli, Arsalan. Exploring a centralized/distributed hybrid routing protocol for low power wireless networks and large-scale datacenters. University of California, Berkeley, 2009.

[4]. Hu, J., Lin, C., Li, X., & Huang, J. (2014). Scalability of control planes for Software defined networks: Modeling and evaluation. 2014 IEEE 22nd International Symposium of Quality of Service (IWQoS). doi:10.1109/iwqos.2014.6914314

[5]. Advait Dixit, Fang Hao, Sarit Mukherjee, T.V. Lakshman, and Ramana Kompella. 2013. Towards an elastic distributed SDN controller. SIGCOMM Comput. Commun. Rev. 43, 4 (October 2013), 7–12. DOI:https://doi.org/10.1145/2534169.2491193

[6]. Yeganeh, Soheil Hassas, Amin Tootoonchian, and Yashar Ganjali. "On scalability of software-defined networking." IEEE Communications Magazine 51.2 (2013): 136-141.

[7]. A. S. -. Tam, Kang Xi and H. J. Chao, "Use of devolved controllers in data center networks," 2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), 2011, pp. 596-601, doi: 10.1109/INFCOMW.2011.5928883.

[8]. 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.

[9]. Aglan, M. A., Sobh, M. A., & Bahaa-Eldin, A. M. (2018). Reliability and Scalability in SDN Networks. 2018 13th International Conference on Computer Engineering and Systems (ICCES). doi:10.1109/icces.2018.8639201

[10]. Wang, Guodong, et al. "The controller placement problem in software defined networking: A survey." IEEE Network 31.5 (2017): 21-27.

[11]. Heller, B., Sherwood, R., & McKeown, N. (2012). The controller placement problem. Proceedings of the First Workshop on Hot Topics in Software Defined Networks - HotSDN '12. doi:10.1145/2342441.2342444

[12]. Singh, A. K., & Srivastava, S. (2018). A survey and classification of controller placement problem in SDN. International Journal of Network Management, 28(3), e2018. doi:10.1002/nem.2018

[13]. Sallahi, Afrim and M. St-Hilaire. "Optimal Model for the Controller Placement Problem in Software Defined Networks." IEEE Communications Letters 19 (2015): 30-33.

[14]. Li, T., Gu, Z., Lin, X., Li, S., & Tan, Q. (2018). Approximation Algorithms for Controller Placement Problems in Software Defined Networks. 2018 IEEE Third International Conference on Data Science in Cyberspace (DSC). doi:10.1109/dsc.2018.00043

[15]. Sood, K., Yu, S., & Xiang, Y. (2016). Software-Defined Wireless Networking Opportunities and Challenges for Internet-of-Things: A Review. IEEE Internet of Things Journal, 3(4), 453–463. doi:10.1109/jiot.2015.2480421

[16]. Zhang, Y., Cui, L., Wang, W., & Zhang, Y. (2018). A survey on software defined networking with multiple controllers. Journal of Network and Computer Applications, 103, 101–118. doi:10.1016/j.jnca.2017.11.015

[17]. 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.

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

[19]. 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.

[20]. Bari, M. F., Roy, A. R., Chowdhury, S. R., Zhang, Q., Zhani, M. F., Ahmed, R., & Boutaba, R. (2013). Dynamic Controller Provisioning in Software Defined Networks. Proceedings of the 9th International Conference on Network and Service Management (CNSM 2013). doi:10.1109/cnsm.2013.6727805

[21]. Zhao, Z., & Wu, B. (2017). Scalable SDN architecture with distributed placement of controllers for WAN. Concurrency and Computation: Practice and Experience, 29(16), e4030. doi:10.1002/cpe.4030

[22]. Ying Zhang, Beheshti, N., & Tatipamula, M. (2011). On Resilience of Split-Architecture Networks. 2011 IEEE Global Telecommunications Conference - GLOBECOM 2011. doi:10.1109/glocom.2011.6134496

[23]. Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliabilityaware 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.

[24]. 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(1), 4–17. doi:10.1109/tnsm.2015.2402432

[25]. Wang, G., Zhao, Y., Huang, J., Duan, Q., & Li, J. (2016). A K-means-based network partition algorithm for controller placement in software defined network. 2016 IEEE International Conference on Communications (ICC). doi:10.1109/icc.2016.7511441

[26]. 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

[27]. Hock, D., Gebert, S., Hartmann, M., Zinner, T., & Tran-Gia, P. (2014). POCO-framework for Pareto-optimal resilient controller placement in SDN-based core networks. 2014 IEEE Network Operations and Management Symposium (NOMS).

[28]. Aoki, H., & Shinomiya, N. (2016, February). Controller placement problem to enhance performance in multi-domain SDN networks. In Proc. ICN (p. 120).

[29]. Aziz, T. I., Protik, S., Hossen, M. S., Choudhury, S., & Alam, M. M. (2019, April). Degree-based Balanced Clustering for Large-Scale Software Defined Networks. In 2019 IEEE Wireless Communications and Networking Conference (WCNC) (pp. 1-6). IEEE.

[30]. Bo, H., Youke, W., Chuan'an, W., & Ying, W. (2016, October). The controller placement problem for software-defined networks. In 2016 2nd IEEE International Conference on Computer and Communications (ICCC) (pp. 2435-2439). IEEE.

[31]. Cheng, T. Y., Wang, M., & Jia, X. (2015, December). QoS-guaranteed controller placement in SDN. In 2015 IEEE Global Communications Conference (GLOBECOM) (pp. 1-6). IEEE.

[32]. Hu, Y. N., Wang, W. D., Gong, X. Y., Que, X. R., & Cheng, S. D. (2012). On the placement of controllers in software-defined networks. The Journal of China Universities of Posts and Telecommunications, 19, 92-171.

[33]. Jimenez, Y., Cervello-Pastor, C., & Garcia, A. J. (2014, June). On the controller placement for designing a distributed SDN control layer. In 2014 IFIP Networking Conference (pp. 1-9). IEEE.

[34]. 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.

[35]. Singh, A. K., Maurya, S., & Srivastava, S. (2020). Varna-based optimization: a novel method for capacitated controller placement problem in SDN. Frontiers of Computer Science, 14(3), 1-26.

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

Authors Biography

Moinul Islam Sayed received his Bachelor of Science in Computer Science and Engineering from Patuakhali Science and Technology University, Patuakhali, Bangladesh. Currently, he is doing as a faculty member in the Department of Computer Science and Information Technology, Patuakhali Science and Technology University, Patuakhali, Bangladesh. His research experience includes E-health, Security and Privacy, and Geographic Information Systems.

Sajal Saha received bachelor's degree in computer science & Engineering from Patuakhali Science and Technology University (PSTU) and Master of Sciene in Information Technology from Jahangirnagar University. Currently he is working as a faculty member of Computer Science & Engineering faculty in PSTU. His research interest includes computer network, machine learning, and deep learning.

Ibrahim Mohammed Sayem received his Bachelor of Science in Computer Science & Engineering from the University of Chittagong in 2018. He is currently a Masters's student at the Department of Computer Science of the University of Western Ontario. His research interest lies in the areas of network security, Machine Learning, and Intelligent networks.

Sarna Majumder received bachelor's degree in computer science & Engineering from Patuakhali Science and Technology University (PSTU) and now doing her Master of Science in Electrical and Electronics Engineering. Currently she is working as a faculty member of Computer Science & Engineering faculty in PSTU. Her research interest includes data mining, sentiment analysis, machine learning, and deep learning.