On SDN Controllers Placement Problem in Wide Area Networks

Firas Fawzy Zobary[1,2(✉)] and ChunLin Li[1]

[1] School of Computer Science and Technology, Wuhan University of Technology, Wuhan, People's Republic of China
firas_zobary@hotmail.com
[2] Faculty of Information Engineering, Damascus University, Damascus, Syrian Arab Republic

Abstract. Software Defined Networking (SDN) is a new paradigm where the forward plan is decoupled from the control plan. The controller is a central program that tells the switches and routers how to react to the incoming flows and different network changes. The placement of the controllers considering different metrics becomes a challenge in SDN WAN. In this paper, we study the controller placement problem in terms of propagation delay and load balancing. An extended K-means algorithm is introduced to partition the network into several subnetworks and place the controllers in nodes that minimize the network delay. Then a load balance index is calculated to check the effectiveness of the load balancing considering a metric β as the load difference between controllers. The result analysis shows that a trade off should be done between the delay and load balancing depending on the priority of the network and no optimal case can be found that minimize both of the metrics at the same time.

Keywords: Software Defined Network · Controller placement · Propagation delay · Load balancing

1 Introduction

Even with their widespread adoption, traditional IP networks are still complex, very hard to manage, and start to show their limitations. Software Defined networking (SDN) is the new paradigm for managing the programmability of the networks. The idea behind SDN is to decouple the control plane from the data/forwarding plane (network devices). The control plane is presented as a central program called "Controller" which will tell the forwarding plane how to respond to different incoming network flows and changing network conditions. For Local Area Networks (LANs) like datacenters, a single controller will be enough and the performance will be well. The challenge will arise when we apply SDN to Wide Area Networks (WANs) where the delay between nodes can be high and the coverage area is large. Multiple controllers should be applied to cover the whole WAN. Network devices and the communications between controllers themselves should be studied carefully. Where to put the controllers is a very important question in this case as different controller placements will affect the delay and the behavior of

© ICST Institute for Computer Sciences, Social Informatics and Telecommunications Engineering 2020
Published by Springer Nature Switzerland AG 2020. All Rights Reserved
H. Gao et al. (Eds.): ChinaCom 2019, LNICST 312, pp. 471–479, 2020.
https://doi.org/10.1007/978-3-030-41114-5_35

the whole network. In this paper, we will focus on SDN WAN controller placement in terms of delay and load balancing. The main goal is to partition the network into several subnetworks each is managed by one controller, and to find the best location to place that controller taking into consideration to minimize the delay and maximize the load balance for the controllers. The contribution of this paper is as follows:

1. An Extended K-means algorithm is proposed to partition the network.
2. In addition to the delay, we tried to balance the load of controllers by reassigning the switches between the controllers depending on the difference of loads between each pair of controllers.

The rest of paper is organized as Sect. 2 will brief the related work. In Sect. 3 we formulate the problem. Section 4 explains the Extended K-means algorithm and the load balancing algorithm. The performance evaluation and results analysing are described in Sect. 5. Finally, Sect. 6 summarizes the conclusion and future work.

2 Related Work

Controller placement problem (CPP) is always been as a critical issue when it comes to large sized SDN as many controllers are introduced and their development is increasing such as Kandoo [1] and Hyperflow [2]. Heller et al. [3] firstly proposed a solution in terms of propagation latency, but focusing only on delay may lead to imbalanced controller distribution. This work was the most cited work for controller placement problem. Their work considered the average latency and worst case latency. A step forward was done by Yao et al. [4] as they take into consideration not only the delay but also the load balancing. Their proposed algorithm named Capacitated Controller Placement Problem (CCPP) reduced the number of controllers required to remove the overload as well as reducing the load on the heaviest controller.

Mamushiane et al. [5] asked a question about how many controllers are needed and where should they go given an SDN topology. They proposed three algorithms to address the controller placement problem, but first they mentioned only the metric of latency. However, their algorithms are accurate but are exhaustive and don't work well in the presence of time constraints.

In [6], the authors studied the controller placement problem in WAN network.

Their goal was to partition the network into several small subnetworks by using spectral clustering algorithm. They used the propagation delay between nodes as weights for edges, and according to that delay the nodes that have low weighted edges creates the small subnetworks while the links with high weights will be used to connect the subnetworks together.

An algorithm named K^*-means was used by Qi et al. [7] to merge clusters that are connected with the shortest edge. Kuang et al. [8] used the same previous algorithm to merge the clusters until the number reaches k, but they imbedded it in a hierarchical K-means algorithm to solve controller placement problem. In [9] Sahoo et al. have considered CPP as a multi-objective combinatorial optimization problem and solved it using two population-based meta-heuristic techniques. They investigated the problem in

terms of three metrics: controller to switch latency, inter-controller latency and multi-path between the switch and controller. Their proposed scheme improves the survivability of the control path in case of a link failure, but they didn't mention the load balancing of the network. In [10] Hu et al. proposed a greedy algorithm to study the reliability in placement problem. Lu et al. [11] used a hierarchical approach to find better initial centers by treating the clustering problem as a weighted clustering problem. K-centers is the most well-known algorithm that have been used in controller placement problem [3, 12] and this problem is introduced as an NP-Hard [4].

3 Problem Formulation

In this section, we describe and then formulate the controller placement and network partition problems in terms of network delay and load balancing. The network topology is denoted as an undirected graph $G = (V, E)$, where V is the set of nodes, and E is the set of edges between nodes. Each edge between two nodes has a weight $w \in W$, where W is the set of weights which refers to the delay w_{ij} as the time delay between node i and node j. The time delay is calculated by the distance between the two nodes divided by speed of light $2 * 10^8$ m/s.

$$w_{ij}(millisecond) = \frac{distance_{i,j}(meter)}{2 * 10^8 \left(\frac{meter}{second}\right)} * 1000 \tag{1}$$

The shortest path distance between two nodes $(u, v \in V)$ is denoted by $d(u, v)$. The network is divided into k subnetworks and we define a set C contains the controller places c_i, where $i \in k$.

Each one of the subnetworks can be referred to as an SDN network and will contain nodes and edges between them as $SDN_i(V_i, E_i)$ with only one controller manages it. We define a binary variable x_{vc} equals to 1 if the node v is connected to the controller c, otherwise it equals to 0. These SDNs should follow the following rules and constraints:

$$\cup_{i=1}^{k} SDN_i = SDN \quad \cup_{i=1}^{k} V_i = V \quad \cup_{i=1}^{k} E_i = E \tag{2}$$

$$SDN_i \cap SDN_j = \emptyset \, where \, i, \, j \in k \tag{3}$$

$$\sum_{c=1}^{V} x_{vc} = 1 \, \forall \, v \in V \tag{4}$$

Equation (2) indicates that the total number of subnetworks will shape the whole big network. Equation (3) ensures that each node, edge belongs to only one subnetwork. Equation (4) refers to a constraint guaranteed that every node is connected to one and only one controller.

For load balancing, we assume the number of switches the controller manages as the load of that controller $L(c_i)$. We will define a metric β as the maximum difference between the loads of each two clusters. The constraint is as follows:

$$\left|L(c_i) - L\left(c_j\right)\right| \leq \beta \qquad \forall i, \, j \tag{5}$$

4 An Extended K-means Algorithm

For partition problems, K-means was widely used and it's an effective and fast method to do that [13, 14]. We proposed an extended K-means to partition the network in terms of time delay and load balancing. The initial nodes where the controllers are placed will be called 'centres' and the updated centres after applying the algorithm are called 'centroids'.

First, we will explain the standard K-means which is used for clustering the network. This algorithm should follow main steps as follows: (1) randomly select k points from the data set and assign each point to be as a center for a cluster, where one cluster has one and only one center, (2) each data point is assigned to its nearest cluster based on the Euclidean distance to the center of that cluster; (3) update the centers of each cluster; (4) repeat step 2 and 3 until there are no changes in clusters' centers.

Although K-means is widely used for most clustering problems, it has many drawbacks when it is applied for network topology partitioning. First, if we choose random centers we will not guarantee the minimum time delay between the node and its centroid. Second, the algorithm doesn't ensure that the updated centroids will be chosen from the nodes in the topology as the centroids must be chosen from $(v \in V)$ to be sure there is a real connection between the centroid and the nodes in that cluster because K-means choose the mean between the two nodes. Third, by applying the Euclidean distance we will not ensure that the link between the centroid and the node is physically exists.

For these reasons we proposed an extended K-means as explained in Algorithm 1 in order to adjust the previous drawbacks in standard K-means. The major steps in this algorithm are related to the initializing the centers, distributing the nodes to clusters and updating the centroids. The randomly initializing the centers as presented in standard K-means is not used in our algorithm and be replaced by another process for initializing the clusters' centers. In fact, initializing the centers in K-means was widely studied and many solutions were proposed to make this process more effective as in [15, 16]. We will use the initializing method proposed in [16] as it is an efficient technique and it reduces the complexity of computational and achieves a better local minimum. In Step 2, the nodes will be distributed to the clusters depends on their shortest path distance to the centers. In Step 3, the updating process of the centers will depend on the nodes inside the cluster.

Algorithm 1: Extended K-means

Algorithm 1: Extended K-means

Step 1: choose the first node randomly as a center.

Step 2: distribute nodes to clusters, each node ($v \in V$) to one cluster as following:

$$if\ d(v, c_i) < d(v, c_j) then\ v \in cluster_i \qquad \forall\ j \in \{1,2, \dots, k\}$$

Step 3: for each $cluster_i$, calculate the sum of shortest path distances to every node in the cluster. The node that has the minimum sum will be chosen as the new center for $cluster_i$.

$$if \sum_n d(v, v_n) = minimum\ then\ c_i^* = v \qquad \forall n \in size(cluster_i),$$
$$(v, v_n \in cluster_i), \qquad i = 1..k$$

Step 4: repeat steps 2, 3 until the centers are not updated anymore.

Step 5: choose the node which has the largest shortest path distance from the previous centroid from step 4 to be the next initial center.

Step 6: repeat step 2, 3, 4 until the network is partitioned into k clusters.

Algorithm 2: Load Balancing

Step 1: Check $|L(c_i) - L(c_j)| \qquad \forall i, j$

Step 2: If $|L(c_i) - L(c_j)| \geq \beta$ then reassign the nodes between $cluster_i$ and $cluster_j$.

Step 3: repeat step 1 and step 2 until the load is balanced.

The main idea about Algorithm 2 is to balance the load depending on the difference between the clusters loads. As we mention before, we assume the number of nodes that controller c_i manages as the cluster load. The algorithm compares the load of each two clusters and reassigns the nodes between clusters depending on the constraint β. The reassigning procedure is done by removing nodes from the cluster that has more nodes and assign it to the cluster that has fewer nodes. This process will be repeated until the load of all clusters is balanced.

5 Performance Evaluation

5.1 Experimental Setup

In this section we evaluate Algorithms 1, 2 in terms of network delay and load balancing with different numbers of clusters. The network topology on which we deploy the algorithms is Internet2 OS3E topology [17] and the simulation is done by using Matlab R2018a, 2.50 GHz Intel Core i5 CPU, 4.00 GB of RAM and Windows 7 operating system. The topology has 34 nodes and 42 edges between them. The weight of the edge is the delay between the two nodes.

For network delay we calculate the time delay of each cluster depending on the shortest path distance between each node and the controller it mapped to as in Eq. 5, and then we take the average of them to find the delay of the whole network as in Eq. 6.

$$delay(cluster_i) = \frac{1}{N_i} \sum d(v, c_i) \tag{6}$$

$$delay(Netwrok) = \frac{1}{k} \sum_{i=1}^{k} delay(cluster_i) \tag{7}$$

Where N_i is the number of nodes in $cluster_i$ and k is the number of clusters. For load balancing effectiveness we use the standard deviation as a balance index, the smaller the value the better the balance of the load is. The balance index is as follows:

$$Balance\ Index = \sqrt{\frac{1}{k} \sum_{i=1}^{k} (N_i - \frac{N}{K})^2} \tag{8}$$

where N is the number of switches in the whole topology.

5.2 Results Analysis

The simulation is applied on the topology shown in Fig. 1 as the number of controllers is different for every simulation, and we also change the value of β for load balancing results.

The number of controllers k will vary from 1 to 5. When $k = 1$, it doesn't matter which node will be selected as initial center as the algorithm will choose node 16 as the center every time because it always has the minimum sum of shortest path distances to every other node in the cluster, and the next initial center is node 32 because this node has the largest shortest path distance to the node 16, but as the network has only one controller so the load on that controller will be the maximum and the nodes can't be reassigned so the results can be omitted. For $k = 2$, even the initial centers are nodes 16, 32 but the final centroids will be chosen as nodes 16 and 31 because they are the nodes which have the minimum sum of the shortest path to every node in the clusters they belong to. The node that has the largest shortest path distance to node 31 is node 9, so the initial centers will be nodes 16, 31 and 9 but the centroids will be 16, 31 and 10. When $k = 4$ the centroids are nodes 16, 28, 10 and 33; and for $k = 5$ the centroids are nodes 8, 28, 10, 33 and 22.

After the network is partitioned for each k, the network delay and balance index are calculated with different β, and the results are as shown in the Fig. 2.

Figure 2 shows that as β increases, the network delay decreases and the network become less balanced as balance index increases. For k = 2, the 32 switches are divided into {24, 8} in the beginning with network delay 7.06 ms and balance index 8. As β increases, the delay decreases slowly and balance index increases. The switches are divide into {16, 16} when β = 1 with optimal load balancing but maximum delay, {17, 15} when β = 2, 3, {18, 14} when β = 4, 5 and {19, 13} when β = 6. When k = 3, the 31 switches are divided into {13, 8, 10} when network is unbalanced, {11, 10, 10} when β = 1, 2, {12, 9, 10} when β = 3, 4, and the network is back to the original state when β = 5, 6. When k = 4, the 30 switches are divided into {7, 8, 8, 7} when β = 1,

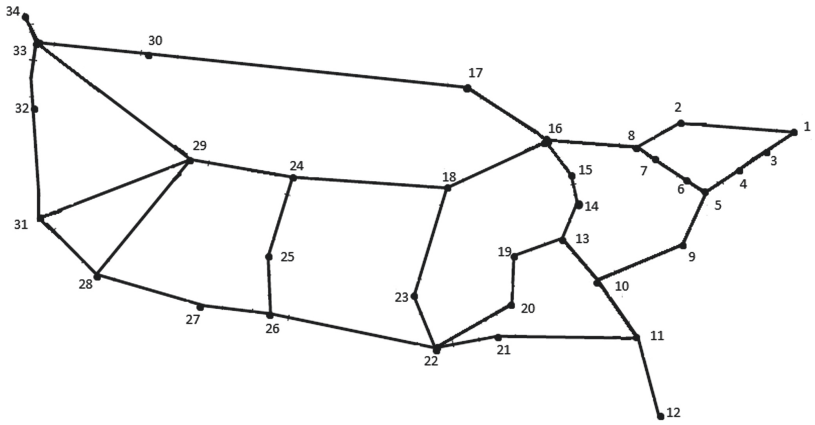

Fig. 1. Topology of Internet2 OS3E

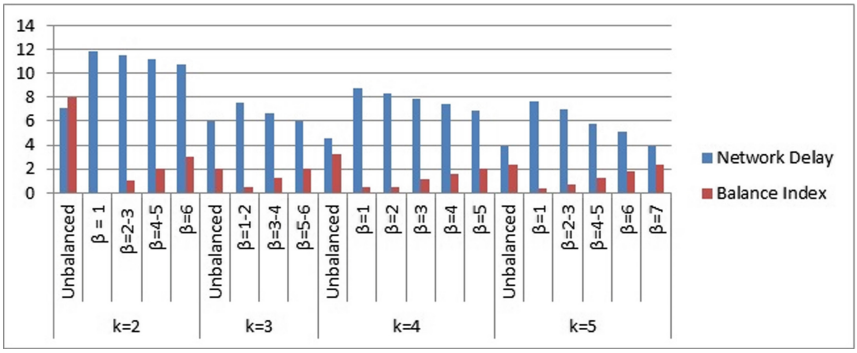

Fig. 2. Network delay and balance index for different k and β

{8, 7, 8, 7} when β = 2, {8, 7, 9, 6} when β = 3, {9, 7, 9, 5} when β = 4, {10, 6, 9, 5} when β = 5 and {10, 6, 10, 4} when β = 6, 7. When k = 5, the 29 switches are divided into {6, 6, 6, 5, 6} when β = 1, {7, 5, 6, 5, 6} when β = 2, 3, {8, 5, 6, 4, 6} when β = 4, 5, {9, 4, 6, 4, 6} when β = 6 and {10, 4, 6, 3, 6} when β = 7.

In most cases, the network delay is decreasing slowly except when k = 5. For k = 2, the delay is decreasing from 11.85 when β = 1 to 10.74 when β = 6, and when k = 3 it is decreasing from 7.5 when β = 1, 2 to 6.04 when β = 5, 6 and when k = 4 the delay is 8.7 when β = 1 and 6.04 when β = 6, 7. But when k = 5 we can see that the delay is 7.6 when β = 1 and 3.9 when β = 7. The balance index is increasing almost in the same manner. From Fig. 2, the minimum network delay will happen when k = 5 in case of unbalanced case or β = 7 as it is 3.9 ms, and the balance index is 2.4 with switches divided into {10, 4, 6, 3, 6}.

From the above discussion, we can say that there is no optimal case for the network to be in minimum delay and maximum load balancing. A trade-off should be made between the two metrics and the decision should be made according to our priority. However,

other metrics can be studied and affect the decision. For example, the best network delay we have is when k = 5, but that makes the cost higher as we should apply 5 controllers in the topology.

6 Conclusion and Future Work

In this paper we introduced the SDN controller placement problem in terms of delay and load balancing. We first apply an Extended K-means algorithm to partition the WAN into several subnetworks and the goal is to place the controllers in positions that minimize the propagation delay between nodes and controllers, and then balance the loads of the controllers. We test our algorithm using Internet2 OS3E topology. From the results we can say that there is no optimal case for the network to be in minimum delay and maximum load balancing. A trade-off should be done between the two metrics and the decision should be made according to our priority. However, other metrics such as resilience, reliability and energy saving can be addressed in the future studies so that can affect the decision more accurately.

References

1. Hassas Yeganeh, S., Ganjali, Y.: Kandoo: a framework for efficient and scalable offloading of control applications. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, pp. 19–24. ACM (2012)
2. Tootoonchian, A., Ganjali, Y.: HyperFlow: a distributed control plane for OpenFlow. In: Internet Network Management Conference on Research on Enterprise Networking. USENIX Association (2010)
3. Heller, B., Sherwood, R., McKeown, N.: The controller placement problem. In: Proceedings of the First Workshop on Hot Topics in Software Defined Networks, pp. 7–12. ACM (2012)
4. Yao, G., Bi, J., Li, Y., Guo, L.: On the capacitated controller placement problem in software defined networks. IEEE Commun. Lett. **18**(8), 1339–1342 (2014)
5. Mamushiane, L., Mwangama, J., Lysko, A.A.: Given a SDN topology, how many controllers are needed and where should they go? In: 2018 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN), pp. 1–6. IEEE (2018)
6. Xiao, P., Qu, W., Qi, H., Li, Z., Xu, Y.: The SDN controller placement problem for WAN. In: 2014 IEEE/CIC International Conference on Communications in China (ICCC), pp. 220–224. IEEE (2014)
7. Qi, J., Yu, Y., Wang, L., Liu, J.: K*-means: an effective and efficient K-means clustering algorithm. In: IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom), pp. 242–249. IEEE (2016)
8. Kuang, H., Qiu, Y., Li, R., Liu, X.: A hierarchical K-means algorithm for controller placement in SDN-based WAN architecture. In: 2018 10th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA), pp. 263–267. IEEE (2018)
9. Sahoo, K.S., Puthal, D., Obaidat, M.S., Sarkar, A., Mishra, S.K., Sahoo, B.: On the placement of controllers in software-defined-WAN using meta-heuristic approach. J. Syst. Softw. **145**, 180–194 (2018)

10. Hu, Y.N., Wang, W.D., Gong, X.Y., Que, X.R., Cheng, S.D.: On the placement of controllers in software-defined networks. J. China Univ. Posts Telecommun. **19**, 92–171 (2012)
11. Lu, J.F., Tang, J.B., Tang, Z.M., Yang, J.Y.: Hierarchical initialization approach for K-means clustering. Pattern Recognit. Lett. **29**(6), 787–795 (2008)
12. Jimenez, Y., Cervelló-Pastor, C., García, A.J.: On the controller placement for designing a distributed SDN control layer. In: IFIP Networking Conference, pp. 1–9. IEEE (2014)
13. Al-Mohair, H.K., Saleh, J.M., Suandi, S.A.: Hybrid human skin detection using neural network and K-means clustering technique. Appl. Soft Comput. **33**, 337–347 (2015)
14. López Del Alamo, C., Calla, R., Arnaldo, L., Fuentes Pérez, L.J.: Parallelization of the algorithm K-means applied in image segmentation. Int. J. Comput. Appl. **88**(17) (2014)
15. Celebi, M.E., Kingravi, H.A., Vela, P.A.: A comparative study of efficient initialization methods for the K-means clustering algorithm. Expert Syst. Appl. **40**(1), 200–210 (2013)
16. Katsavounidis, I., Kuo, C.C.J., Zhang, Z.: A new initialization technique for generalized Lloyd iteration. IEEE Signal Process. Lett. **1**(10), 144–146 (1994)
17. Internet2 Open science, scholarship and services exchange. http://www.internet2.edu/network/ose/. Accessed 22 Mar 2019