International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-8 Issue-6, August 2019

A New Algorithm for Controller Placement in SDN

G Ramya, R Manoharan

Abstract**:** ***SDN network supports centralized network management by splitting control plane and data plane of forwarding devices and places the network intelligence in a software entity called controller. The controller can be placed in selective places of network to effectively monitor and control network activities. Large scale network needs multiple controller to manage control activities of network. In order to identify the optimum number of controllers and its effective locations in the network, a new algorithm is proposed using cut-vertex concept from graph theory. The proposed algorithm is simulated using Mininet SDN emulator. To study the performance of the proposed algorithm, multiple scenarios were used in the simulation and performance was analysed using parameters viz., flow installation time, average latency of network, throughput.***

Index Terms**: Controller placements, *Cut-vertex, Mininet,SDN***

I. INTRODUCTION

With the development of new technologies like fog and Internet of Things (IoT), a large number of devices are connected to the Internet and management of these devices would be a challenging task in the future Internet. In the existing network architecture, each and every forwarding device in network performs both computations and packet forwarding and both data and control planes are tightly packed together. A huge volume of data is being generated and managing high volume of traffic is difficult. New technologies like Fog computing needs minimal latency to forward data from one end to another end and the complexity increases with increase in the number of devices connected to the Internet. CISCO predicts 50 million devices will be connected by 2020. The future network management lies in the technologies like SDN, NFV.

SDN manages network by separating the data and control planes from the hardware devices and thereby it removes dependency of the proprietary devices whereas NFV aims at virtualization of network services. The centralized approach of SDN and the programmability of data plane makes easier to distribute traffic and balance load in network. The architecture of SDN contains control, data and application planes. In SDN, network intelligence is moved to a software entity called controller which lies in control plane. Forwarding devices (switches, routers) are located in data plane whilst applications like NAT, IDS lies in the application plane of SDN. The forwarding elements of network simply forwards packet according to the rule installed by the controller. Controller performs control operations and thereby managing the network. Placing controller in appropriate locations in a network will certainly increase the performance of network.

The data flow in the data plane of SDN is verified by the controller. Every PACKET_IN messages should get permission from controller, which verify packets according to the network policy. Controller computes a route for all PACKET_IN and adds the routing information in a table called flow table located in all forwarding devices of data plane.The Controller Placement Problem (CPP) is a well-known research problem of SDN. Identifying the number of controllers required for network and placing them in its optimal locations is defined as CPP in SDN. To elaborate the problem of CPP, let us assume a large area network controlled by a single controller. The controller connected with the farthest node yields more latency thereby increasing the response time. This will certainly affect the performance of network. Besides that, single controller is always affected by "single point failure". Failure in a network may lead to link disconnection and communication will also fail. Sometimes, path may get disconnected and it is very difficult to manage that situation. In order to prevent aforementioned scenario, multiple controller which are physically distributed in the network concept was introduced. The multiple controllers enhance the performance in the aspects of scalability, availability and reliability of network. This will decrease the response time, increases the throughput and yield better results than the single controller scenario. In the case of multiple controllers, even if one controller fails; there would be several other controllers available, which can take care of control operations. But the actual challenge lies in deciding of "estimating the optimum number of controllers and the location in the network to place it". The controller to node latency plays a vital role in network performance. Because, whenever a packet arrives, flow table values can be updated in three methods:" (i) reactive (ii) proactive and (iii) hybrid". In reactive mode, after the arrival of a packet, the node sends request to controller to update flow table entry. Whereas in the case of proactive method, the controller installs and updates table entries even before the packet arrives.

Revised Manuscript Received on August 30, 2019.
* Correspondence Author
G Ramya*, Computer Science and Engineering, Pondicherry Engineering College, Pondicherry, India.
R Manoharan, Computer Science and Engineering, Pondicherry Engineering College, Pondicherry, India.

In hybrid methodology, controller can proactively update and reactively respond to dynamic scenarios. Calculating round-trip time taken to update flow table entry is termed as "flow installation time". Suppose, the amount of time taken to update data is high leading to high latency, resulting in packet delay and overall performance gets affected.

This paper adapts a concept from graph theory to identify the required number of controllers and its placement issue. The performance was analysed using metrics like average latency, throughput and flow installation time from controller to node. The proposed work is compared with the same performance metrics when the controllers were placed randomly and when the controllers were placed in the place of vertex whose degree is maximum and with the K-Medoids algorithm.

This paper is organized as follows. Section II deals with the brief description of the existing research works done in CPP. Section III describes the problem statement. Formal definitions are presented in section IV. The proposed methodology is described in section V. Results and analysis are explained in detail in section VI. Finally concludes with conclusion and future enhancements.

II. RELATED WORK

The Controller Placement Problem was studied with various aspects such as placing multiple controllers to improve scalability, reliability and distribute load among controllers[17]. In [5], the authors translated the controller placement problem into a facility location problem. The latency between node and controller was the primary parameter for controller placement. The authors also claimed that the exhaustive evaluation of search space may result in finding optimal controller placements with reference to latency. The inference from their work is the exhaustive search may find controller locations with latency as major parameter. It is also inferred that average and worst-case scenarios in terms of latencies cannot be optimized. Furthermore, in some cases, a single controller can meet network demands of communication latency. A single controller approach is always prone to single point failure. Bari.et al. [6], proposed a methodology to reduce the flow set up time.In [7][8], the authors identified controller locations using a Pareto Optimal Controller Placement (POCO) technique by considering the latency parameter. The search space for placing the k-controllers were analysed in terms of controller - node latency, controller to controller latency, load balancing amongst controllers and link/node failure situations. Moreover, the number of controller's 'k' to be placed was given as an input parameter. This work considered that nodes are connected to the nearest one, which may not be applicable to dynamic scenarios. In [9], the authors tried to balance the load by placing single controller in network and proposed a method to perform switch migration. Rath.et al. [10] applied a game theory concept for maximum utilization of controllers. The number of controllers was taken as input parameter and is limited to small-scale SDN. Yao.et al. [11] and Jimenez et al. [12] taken load as primary parameter for assigning nodes to controller. They estimated number of controllers needed only when the traffic is static. Sahoo et al. [13] proposed simulated annealing methodology by considering latency as a parameter to place controllers. They analysed the performance of network only when the traffic is static. Sanner et al. [14] proposed a hierarchical clustering of controllers for finding optimal controller placements. Clusters were formed using adapted k-mean algorithm and hierarchical clustering methodology was adapted to merge clusters for controller placements. Many heuristics and machine learning [16] methodologies have been proposed to place controllers in optimal locations. A few approaches followed cluster theory where node clusters were formed and controller was placed as a cluster head. In other approaches, translated CPP into mathematical optimization problems like facility location problem, linear integer problem and many other optimization methodologies. Random placements were adopted in few approaches and performance were studied. A few authors used heuristics techniques and performance were analysed using parameters like load, latency, throughput, etc..,. In most of the above-mentioned methodologies, the number of controller's "k" was given as input. Then the controllers were placed in optimal locations.

III. PROBLEM STATEMENT

Given a network (undirected graph), G= (V, E) where V refers to forwarding elements of data plane and E refers to the links that are connecting the elements. The problem statement is to find the optimal number of controllers i.e., a subset of V and its placement in appropriate locations in the given topology to improve the network performances in terms of flow installation time, average latency, packet delay variation and throughput.

IV. DEFINITIONS

In this section, the notations are formally presented. Given a graph G, 'S' be the set of switches and 'C' represents the controllers set. Let 'k' be the number of controllers. The switch and controller set can be defined as follows:

$$S = \{s_1, s_2, \dots, s_m\}$$

where 'm' is the number of vertices

$$C = \{c_1, c_2, \dots, c_k\}$$

Given a graph G, finding the number and its locations Pk in a search space L is a combinatorial optimization problem. The number of possible placements can be taken from $\binom{n}{k}$. The connection between the switch and controller is defined as follows (1):

$$f_{ij} = \begin{cases} 1, when\ the\ connection\ is\ established \\ 0, no\ connection \end{cases} \quad (1)$$

f_{ij}, represents i th switch is connected to jth controller.

The load of controller (Δ) is directly proportional to the number of PACKET_IN (flow request) and is given as below:

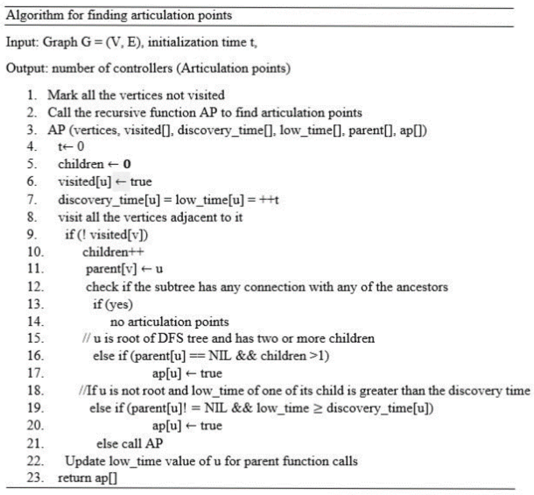
$$\Delta(C) \propto Numberofflowrequestsswitch$$

V. PROPOSED SYSTEM

A node is said to be a cut-node or articulation point only when the removal of a node disconnects the graph. Finding articulation points generates the number of controllers required and locates its placements [18].

$$k = nubmer\ of\ articulation\ points$$

In neighbourhood search algorithm or in any meta-heuristic optimization, the number of controllers required was given as input parameter. But in the case of proposed methodology, a topology is sufficient to estimate the required numbers of controller. The algorithm starts with scanning the topology to find out the articulation points in network.



```
Algorithm for finding articulation points

Input: Graph G = (V, E), initialization time t,
Output: number of controllers (Articulation points)

1.  Mark all the vertices not visited
2.  Call the recursive function AP to find articulation points
3.  AP (vertices, visited[], discovery_time[], low_time[], parent[], ap[])
4.     t← 0
5.     children ← 0
6.     visited[u] ← true
7.     discovery_time[u] = low_time[u] = ++t
8.     visit all the vertices adjacent to it
9.       if (! visited[v])
10.          children++
11.          parent[v] ← u
12.          check if the subtree has any connection with any of the ancestors
13.            if (yes)
14.                no articulation points
15.         // u is root of DFS tree and has two or more children
16.            else if (parent[u] == NIL && children >1)
17.                      ap[u] ← true
18.         //If u is not root and low_time of one of its child is greater than the discovery time
19.           else if (parent[u]! = NIL && low_time ≥ discovery_time[u])
20.                      ap[u] ← true
21.             else call AP
22.      Update low_time value of u for parent function calls
23.  return ap[]
```

Fig 1. Algorithm for finding articulation points

The standard Depth First Search algorithm (DFS) is utilized for estimating number of controllers and its locations in network. The DFS algorithm applies the concepts from 'DFS tree'. Node or vertex 'u' in topology is the parent of node 'v' in DFS tree if and only if 'v' is discoverable by u. DFS algorithm is executed by maintaining the following data:

i. depth of each and every visited node
ii. low-point

In DFS tree, a node 'u' is said to be an articulation point if it satisfies any one of the following constraints:

i. node 'u' is root node of the tree and 'u' has at least two children.
ii. Node 'u' is not a root node and it has a child called 'v' ('v' is the adjacent of 'u' in graph) such that there is no node in subtree has connectivity with any of the ancestors of 'u'.

Given a graph G, a DFS traversal is made and DFS tree is constructed. A dynamic array of adjacency list is also created. If the nodes 2 and 5 are removed from figure 2 then there is no communication link from node 3 to nodes 4, 6, and 7 and it is shown as dashed lines.

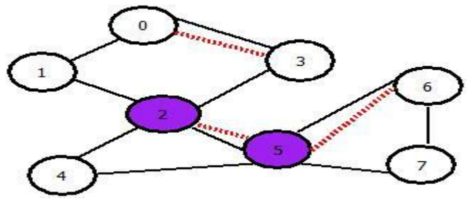


Fig 2. Example construction of DFS tree

Let 'u_i' be the node to be visited next. visited[], an array keeps track of visited nodes during the execution of algorithm. Let discovery_time[], stores discovery time of visited nodes. Let parent[], stores parent vertices in DFS tree. Finally, let ap[],stores articulation points (k). A recursive function was written to find out articulation points while performing DFS traversal. First initialize low_time to 0. Count the number of children in DFS tree. Make the current node as visited. Initialize its discovery time and update its low value by using (2).

$$D_t[visited] = l_t\,[vertex] = ++\,time \qquad (2)$$

where,
Dt - Discovery time
lt - represents Low time

Visit all the nodes adjacent to visited node. Check whether the adjacent nodes have already visited or not. In each iteration, the discovery_time and low_time values are updated. If the adjacent node is not visited, then make it as the child of already visited node v in DFS tree and call it recursively. Check if the subtree has a connectivity with any of the ancestors of v. If not return that node as articulation point.

The topologies considered for experiment was given as input to the algorithm. The algorithm generates all the AP identified in the topologies. More than one AP's were generated for all the topologies taken. For Iris topology, the nodes [5,6,39,22,10,29,2,3,0] are identified as AP. Likewise,[8,33,25,28,23,27,39,38,0], [3,33,7,55,35,41,51,43,20,42,27], [3,33,30,13] and [47,3,3,5,40,6,5,9,49,52,51] are identified as AP's for China telecom, Forthnet, LambdaNet and BTN respectively. The switches were connected with the nearest controllers.

VI. RESULTS AND ANALYSIS

This section elaborates the simulation setup, metrics used for evaluating the performance of proposed system and comparison of proposed with existing methodologies. Various topologies are taken from Internet Topology Zoo (China Telecom: 41 nodes, BTN: 52 nodes, IRIS: 50 nodes, LambdaNet: 41 nodes and Forthnet: 61 nodes). The controller nodes are the subset of vertex. Each and every switch in the network was connected with at least two hosts. Iperf, ITG were executed to generate network traffic. The proposed approach was compared with controllers placed in the random position (RP), nodes of higher degree (HP) and by using K-Medoids algorithm. The results obtained from multiple runs of all the algorithm utilized for comparison. The following are the metrics used for evaluating the performance of the proposed system.

Throughput:

The throughput may be calculated as the number of packets transmitted over the given period of time and it is calculated by using the following formula.

Flow Installation Time:

The flow installation time may be described as communication latency between the switch and controller and it can be calculated by using equation 3.

$$L_{fit} = ((L_{cs} * number\ of\ PACKET\_IN) + (L_{sc} * time\ to\ update\ flow\ table))$$
(3)

Average Latency:

The average latency can be defined as the amount of time taken to send a packet. The latency between switch and controller L_{cs} can be defined as in equation 4:

$$L_{cs} = L_{fit} + \sum_{i=0}^{|V|} \sum_{j=0}^{k} d(s_i, c_j) \qquad (4)$$

It is evident from the analysis; the proposed methodology for placing controllers outperforms RP and HD.

The experiment was conducted by varying the number of controllers (k=1,2,3) for all the topologies considered and same type of traffic was generated to all the controllers and for all the algorithms.

Flow Installation Time:

The FIT decreases with increase in the number of controllers in the network. Figure 3a,3b,3c. shows the FIT values for various topologies for the number of controllers 1,2,and 3 respectively. Though the FIT value of AP is increases with decrese in the number of controllers, the value remains low when compared with other approaches. The value ranges between 120- 790 (ms) for AP placements which is very low when compared with HD, RP, and KM. Figure 3a depicts the FIT values of AP when the number of controllers is 1,2,3. There is an average of 25% to 70% improvement in the proposed methodology when compared with RP, HD and K-Medoid.

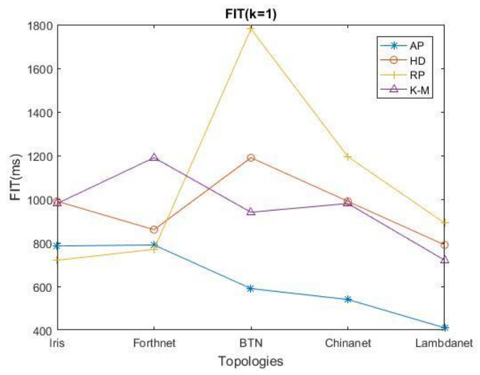


Fig. 3a. FIT (k=1)

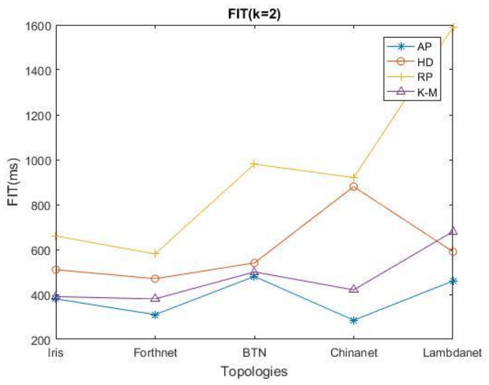


Fig. 3b. FIT (k=2)

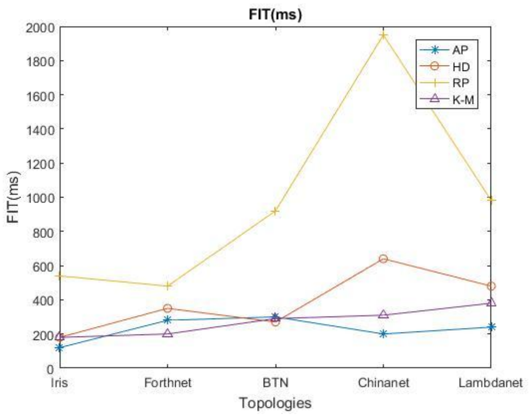


Fig. 3c. FIT (k=3)

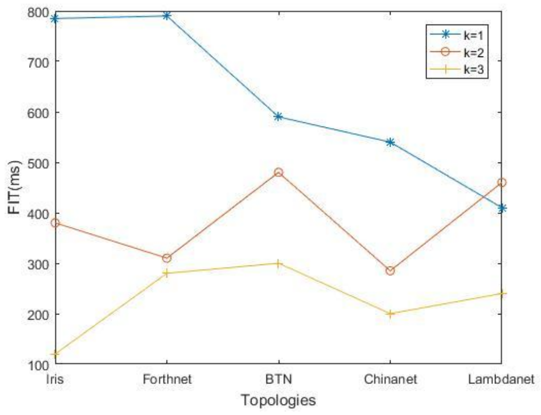


Fig. 3d. FIT values of AP

Average Delay:

The average delay decreases with increase in the number of controllers in the network. Figure 4a,4b,4c. shows the avereage delay values for various topologies for the number of controllers 1,2, and 3 respectively. The average delay of AP falls in the range of 180-800 (ms). There is 30% to 65% improvement in the average delay of the network when the controllers are placed in the articulation points (AP).

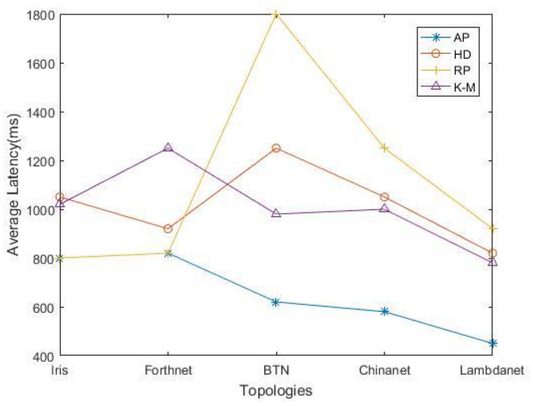


Figure 4a. Average delay (k=1)

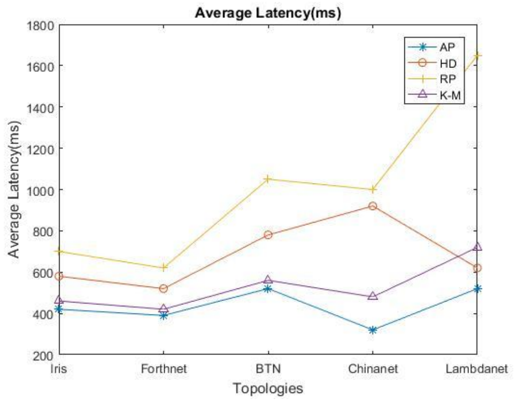


Figure 4b. Average delay (k=2)

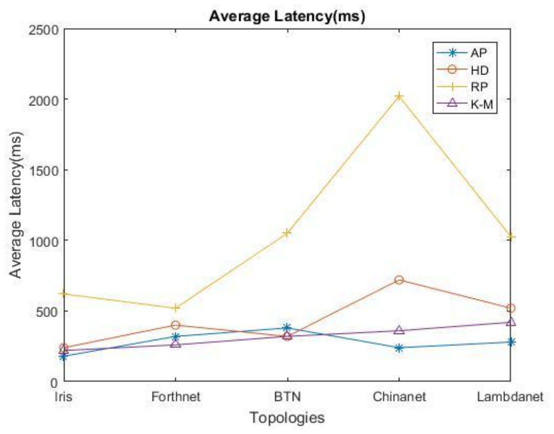


Figure 4c. Average delay(k=3)

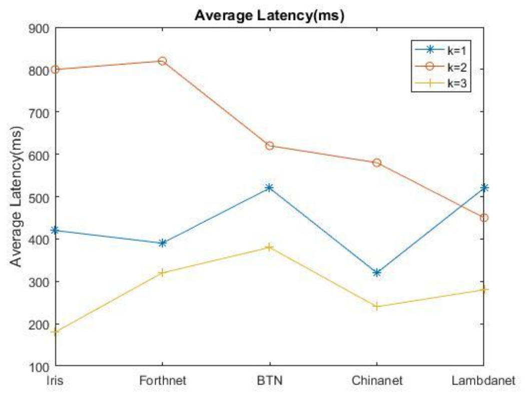


Figure 4d. Average delay of AP

Throughput:

Figure 5a,5b,5c depicts the throughput values of various topologies for the controllers 1,2, and 3. From the figure, it is evident that the throughput values increase with increase in the number of switches connected to controllers when the controller numbers are increased. Throughput of AP is increased in the range of 18% in some cases the value gets increased and falls in the average of 70%.

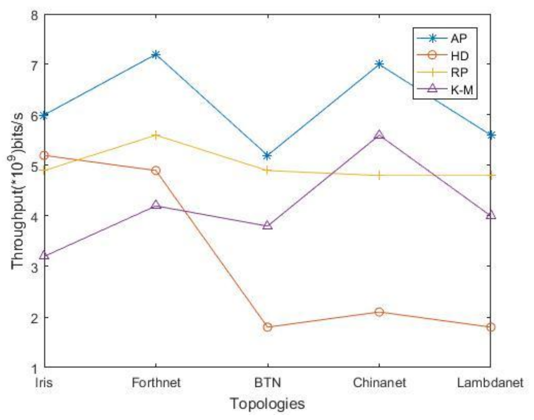


Figure 5a. Throughput (k=1)

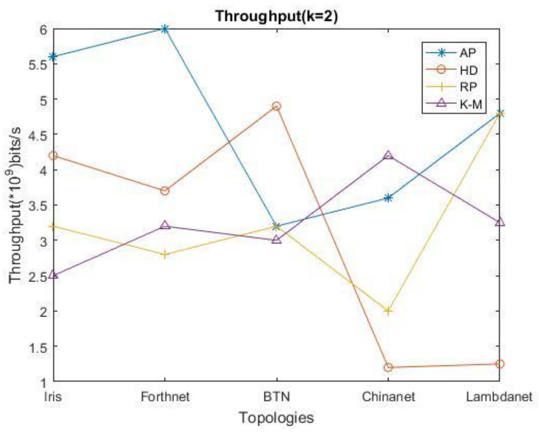


Figure 5b. Throughput (k=2)

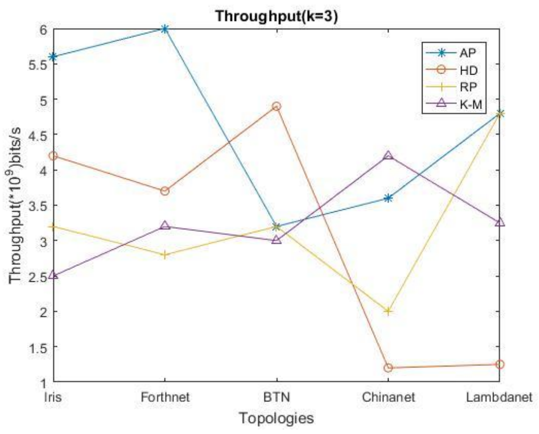


Figure 5c. Throughput (k=3)

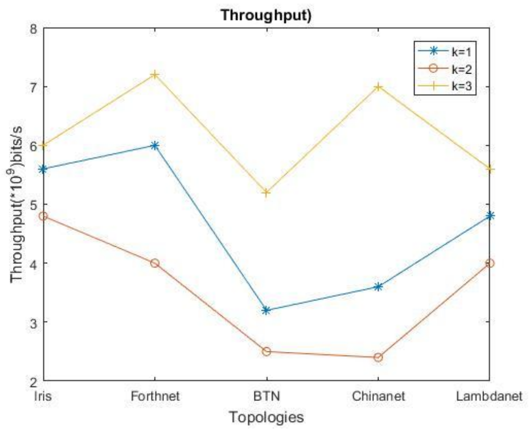


Figure 5d. Throughput values of AP

VII. Conclusion

In this paper, a concept from graph theory is adapted to estimate number of controllers and its placements. The controller placement method identifies the articulation points in network and places controller at articulation points. Various topologies from Internet Topology Zoo are simulated and the results are studied in terms of throughput, flow installation time, Average delay. From the results obtained, it is evident that the proposed approach outperforms the existing methodologies. In future, this work can be extended to study the performance of controller in terms of load balancing.

REFERENCES

1. H.Farhady et.al., “Software -Defined Networking: A survey”, Computer Networks, 2015.
2. A.Hakiri, A.Gokhale, P.Berthou, D.C. Schmidt, G.Thierry, “Software-defined Networking: Challenges and Research opportunities for future Internet,” Computer Networks, 2014.
3. J.Xie. et.al., “Control plane of software defined networks: A survey,” Computer communications, 2015.
4. Open Networking Foundations, “Software-Defined Networking: The New Form for Networks,” ONF white paper, April 2012.
5. Heller, Brandon, Rob Sherwood, and Nick McKeown. "The controller placement problem." In Proceedings of the first workshop on Hot topics in software defined networks, pp. 7-12. ACM, 2012.
6. Yao, Guang, Jun Bi, Yuliang Li, and Luyi Guo, "On the capacitated controller placement problem in software defined networks," IEEE Communications, vol. 18, no. 8, pp. 1339-1342, 2014.
7. Bo, Hu, Wu Youke, Wang Chuan'an, and Wang Ying. "The controller placement problem for software-defined networks." In 2nd IEEE International Conference on Computer and Communications (ICCC), pp. 2435-2439, 2016.
8. Lange, Stanislav, Steffen Gebert, Thomas Zinner, Phuoc Tran-Gia, David Hock, Michael Jarschel, and Marco 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.
9. Hock, David, Matthias Hartmann, Steffen Gebert, Michael Jarschel, Thomas Zinner, and Phuoc Tran-Gia. "Pareto-optimal resilient controller placement in SDN-based core networks." In 25th IEEE International Teletraffic Congress (ITC), pp. 1-9, 2013.
10. Bari, Md Faizul, Arup Raton Roy, Shihabur Rahman Chowdhury, Qi Zhang, Mohamed Faten Zhani, Reaz Ahmed, and Raouf Boutaba. "Dynamic controller provisioning in software defined networks." In 9th IEEE International Conference on Network and Service Management (CNSM), pp. 18-25, 2013.
11. Rath, Hemant Kumar, Vishvesh Revoori, S. M. Nadaf, and Anantha Simha. "Optimal controller placement in Software Defined Networks (SDN) using a non-zero-sum game." In 15th IEEE International Symposium on World of Wireless, Mobile and Multimedia Networks (WoWMoM), pp. 1-6, 2014.
12. Jimenez, Yury, Cristina Cervello-Pastor, and Aurelio J. Garcia. "On the controller placement for designing a distributed SDN control layer." In IEEE Networking Conference, pp. 1-9, 2014.
13. Sahoo, Kshira Sagar, Bibhudatta Sahoo, Ratnakar Dash, and Nachiketa Jena. "Optimal controller selection in Software Defined Network using a greedy-SA algorithm." In 3rd IEEE International Conference on Computing for Sustainable Global Development (INDIACom), pp. 2342-2346, 2016.
14. Sanner, Jean-Michel, Yassine Hadjadj-Aoufi, Meryem Ouzzif, and Gerardo Rubino. "Hierarchical clustering for an efficient controllers' placement in software defined networks." In IEEE Global Information Infrastructure and Networking Symposium (GIIS), pp. 1-7, 2016.
15. Knight, Simon, Hung X. Nguyen, Nick Falkner, Rhys Bowden, and Matthew Roughan. "The internet topology zoo." IEEE Journal on Selected Areas in Communications 29, no. 9 (2011): 1765-1775.
16. Xie, Junfeng, F. Richard Yu, Tao Huang, Renchao Xie, Jiang Liu, Chenmeng Wang, and Yunjie Liu. "A survey of machine learning techniques applied to software defined networking (SDN): Research issues and challenges." IEEE Communications Surveys & Tutorials 21, no. 1 (2018): 393-430.
17. Lu, Jie, Zhen Zhang, Tao Hu, Peng Yi, and Julong Lan. "A Survey of Controller Placement Problem in Software-defined Networking." IEEE Access 7 (2019): 24290-24307.
18. Ramya, G., and R. Manoharan. "Enhanced Multi-Controller Placements in SDN." In 2018 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), IEEE, 2018, pp. 1-5.