Controller placement with critical switch aware in software-defined network (cpcsa)

Nura Muhammed Yusuf [Corresp., 1, 2], **Kamalrulnizam Abu Bakar** [1], **Babangida Isyaku** [Corresp., 1, 3], **Abdelzahir Abdelmaboud** [4], **Wamda Nagmeldin** [5]

[1] Department of Computer Science, Faculty of Computing, Universiti Teknologi Malaysia, Johor, Johor Bahru, Malaysia

[2] Department of Mathematical Science, Faculty of Sciences, Abubakar Tafawa Balewa University, Bauchi, Bauchi, Nigeria

[3] Department of Computer Science, Faculty of Computing and Information Technology, Sule Lamido University, Kafin Hausa, Jigawa State, Nigeria

[4] Department of Information Systems, King Khalid University, Abha, Al-Namas, Saudi Arabia

[5] Department of information systems, College of Computer Engineering and Sciences, Prince Sattam bin Abdulaziz University, Al-Kharj, Al-Kharj, Saudi Arabia

Corresponding Authors: Nura Muhammed Yusuf, Babangida Isyaku
Email address: ymnura@atbu.edu.ng, bangis4u@gmail.com

Software-Defined Networking (SDN) is a networking architecture with improved efficiency achieved by moving networking decisions from the Data Plane to provide them critically at the Control Plane. In a traditional SDN, typically, a single controller is used. However, the complexity of modern networks due to their size and high traffic volume with varied quality of service requirements have introduced high control message communications overhead on the controller. Similarly, the solution found using multiple distributed controllers brings forth the “Controller Placement Problem” (CPP). Incorporating switch roles in the CPP modelling during network partitioning for controller placement has not been adequately considered by any existing CPP techniques. This paper proposes Controller Placement Algorithm with Network Partition Based on Critical Switch Awareness (CPCSA). CPCSA identifies critical switch in the Software Defined Wide Area Network (SDWAN) and then partition the network based on the criticality. Subsequently, a controller is assigned to each partition to improve control messages communication Overhead, Loss, Throughput, and Flow setup Delay. The CPSCSA experimented with real network topologies obtained from the Internet Topology Zoo. Results show that CPCSA has achieved an aggregate reduction in the controller’s overhead by 73%, Loss by 51%, and Latency by 16% while improving throughput by 16% compared to the benchmark algorithms.

CONTROLLER PLACEMENT WITH CRITICAL SWITCH AWARE IN SOFTWARE-DEFINED NETWORK (CPCSA)

Muhammad Nura Yusuf 1&2, Kamalrulnizam bin Abu Bakar1, Babangida Isyaku1&3, Abdelzahir Abdelmaboud4 and 5Wamda Nagmeldin

[1]Faculty of Computing, Universiti Teknologi Malaysia, Johor 81310, Malaysia
[2]Department of Mathematical Science, Abubakar Tafawa Balewa University, PMB 0284, Bauchi, Nigeria
[3]Department of Computer Science, Faculty of Computing and Information Technology Sule Lamido University, P.M.B 047, Kafin Hausa, Jigawa State, Nigeria
[4]Department of Information Systems, King Khalid University, Muhayel Aseer 61913, Saudi Arabia
[5]Department of information systems, College of Computer Engineering and Sciences Prince Sattam bin Abdulaziz University Al-Kharj 11942, Saudi Arabia

Corresponding Author:
Muhammad Nura Yusuf [1&2]and Babangida Isyaku[3]
Johor 81310, Malaysia and Bauchi, Nigeria.
Johor 81310, Malaysia and Kafin Hausa, Jigawa State, Nigeria
Email address: (ymnura@atbu.edu.ng), (bangis4u@gmail.com).

Abstract

Software-Defined Networking (SDN) is a networking architecture with improved efficiency achieved by moving networking decisions from the Data Plane to provide them critically at the Control Plane. In a traditional SDN, typically, a single controller is used. However, the complexity of modern networks due to their size and high traffic volume with varied quality of service requirements have introduced high control message communications overhead on the controller. Similarly, the solution found using multiple distributed controllers brings forth the "Controller Placement Problem" (CPP). Incorporating switch roles in the CPP modelling during network partitioning for controller placement has not been adequately considered by any existing CPP techniques. This paper proposes Controller Placement Algorithm with Network Partition Based on Critical Switch Awareness (CPCSA). CPCSA identifies critical switch in the Software Defined Wide Area Network (SDWAN) and then partition the network based on the criticality. Subsequently, a controller is assigned to each partition to improve control messages communication Overhead, Loss, Throughput, and Flow setup Delay. The CPSCSA experimented with real network topologies obtained from the Internet Topology Zoo. Results show that CPCSA has achieved an aggregate reduction in the controller's overhead by 73%, Loss by 51%, and Latency by 16% while improving throughput by 16% compared to the benchmark algorithms.

Keywords: SDN; Controller Placement; Controller Overhead; Switch role; Network Partition

Introduction

Software-Defined Networking (SDN) is an emerging network paradigm offering simple network management by separating network control logic and data forwarding elements. This way, the Control Plane (CP) is responsible for providing and enforcing network policies on the switches at the Data Plane (DP). To achieve this, the controller uses a link layer discovery protocol (LLDP) to identify the OpenFlow switches connected at the DP[1]. It then continuously monitors them for changes due to events like failures or the arrival of new flows. It collects network statistics concerning traffic arrival patterns, traffic types, and other changes for various applications like routing, congestion control, and security to run their algorithm instances[2]. For any state change at DP, the controller must immediately recalculate updated instructions for the DP switches, sending them as a packet-out message to all edge switches (for ARP) and a flow-mod message to all switches along the same path for installation on their flow tables[3]. Recently, the controller has been experiencing a substantial increase in communication overhead due to an exponential growth in new flow arrival rates caused by the proliferation of IoT devices and the expansion of network size [4]. Consequently, the DP may frequently encounter state change events like link failure [5], requiring the controller to reconfigure new rules [6].

This process has implications for the workload of the controller. For instance, if a flow traverses an average path length of 6 switches and the network has 100 edge switches, the controller is estimated to spend around 6ms to handle each flow [7]. A prior study reports that processing these messages adds an overhead and delay of approximately 0.5ms and 0.2ms, respectively. As a result, the cumulative burden on the controller amounts to (0.5 * 6 + 0.2 * 100) [7]. Moreover, another study highlights a direct correlation between the number of switches in a network and the volume of flow setup requests. According to [8], configuring a flow route for a network with N switches incurs an overall cost of approximately 94 + 144N, with an additional 88N byte attributed to flow-removed messages. Thus, CP design is critical to the performance of SDN.

A single controller (csCP) design is widely used for small network sizes. However, it may fail to give the desired performance due to high control message processing overhead. It also exhibits reliability concerns due to a single failure point (SPOF), as the failure tendencies are higher when the Network is large. As such researchers leverage multiple controllers (dmCP), which better performance compared to csCP. Figure 1 illustrates the differences between the former and the latter. For example, an extensive network may have switches that can generate up to 750 to 20,000 flow per second[6]; others say it might reach up to 10 million flow requests per second[9], [10]. Unfortunately, this is beyond the capacity of a single controller, as some controllers can only accommodate 6000 flow requests per second [11]. On the other hand, designing the CP with multiple controllers opens up a Controller Placement Problem (CPP) challenge. For any given network, the CPP deals with finding and optimising (i) the number of controllers in the Network. (ii) The controllers should be placed strategically on the network to minimise congestion, overhead, and Latency between controllers and switches. Heller et al. [12], who initiated the concept of (CPP), built their solution while considering the impact of Latency. The solution performs well for small-scale networks; however, it ignores the effects of Scalability, Reliability, and Congestion in large networks such as WAN. Assigning controllers to switches in an extensive network can exhibit an imbalance distribution of load among the controllers. Therefore, for Wide Area Network (SDWAN), a partitioning algorithm is employed to cluster the Network into smaller subnets for controller placement[13].

Several CPP solutions employ network partitioning techniques in their approaches. For example, methods such as [14]–[22] are designed based on k-means. A K-median is used by [23], [24], while [17], [19], [25]–[27] used Spectral Clustering. Density-based Clustering, Affinity Propagation, and Partitioning Around Medoids (PAM) are also used in [28]–[31]. Others hybridised two techniques in their solution [20], [21] [4][32]. All these techniques share the common idea of partitioning the SDWAN into smaller sub-domains, allowing for assigning one or more exclusive controllers to cover each subdomain. The k-means algorithm is one of the common methodologies used to partition a network topology. It uses Euclidean distance as its similarity metric during the partition process. However, computing Euclidean distance in real networks is not always possible due to the lack of physically connected pathways in some instances. Similarly, the strategy has no generally agreed-upon way to determine the first k partitions. The method varies in how it initialises the first set of cluster heads. Hence, the initial cluster head selection significantly affects the solution quality; thus, it is a significant limitation.

On the other hand, PAM is quite similar to k-means, except that it minimises the impact of outliers by selecting a node at the cluster's centre as the head. Although PAM does not require prior knowledge of k, it has a considerably high complexity to the tune of about cubic time. Additionally, while these approaches may be suitable for initial controller placement, repeatedly segmenting the entire network to adapt to its dynamic nature is unrealistic. At the same time, Spectral Clustering tends to produce small, isolated components and clusters of skewed sizes. In addition, all the solutions did not quantify the controller's overhead and Response Time(RT) in their performance validation.

In the rapidly evolving landscape of SDN, the efficient placement of controllers plays a pivotal role in network performance and reliability. The 'Controller Placement with Critical Switch Aware (CPCSA)' paper addresses this critical challenge by introducing an innovative approach that optimises controller placement and considers the impact on critical switches within the network. The existing solution did not adequately consider the roles of switches in the network. It is important to note that switches have different roles; some switchers are very critical, and others are non-critical. The former can have a significant impact on the efficient controller placement solution. Identifying critical switches is crucial for optimal controller placement during network partitioning decisions. Critical switches possess a high degree and betweenness criticality measures that tend to send higher flow rule requests to the controller. As a result, they often augment the flow setup delay and cause high update operations. This problem results in additional overhead on the controller if multiple critical switches reside in the same partition. Therefore, this paper proposes Controller Placement Algorithm with Network Partition Based On Critical Switch Awareness (CPCSA) to mitigate these issues. CPCSA identifies critical switch in the SDWAN and then partition the network based on the criticality. Subsequently, a controller is assigned to each partition to improve control messages communication Overhead and other dependent QoS metrics like Loss, Throughput, and Flow setup Delay.We itemized the contributions of this paper as follows.

- We devised a network partitioning model based on the switch role in the network to determine the number of controllers.
- A switch to controller placement strategy was introduced based on switch criticality factor to improve the control plane's performance.
- The performance evaluation result of CPCSA using real networks from Internet Topology Zoo in comparison to other relevant CPP algorithms.

The remainder of the paper is structured as follows: Section 2 discusses related works in SDN. Section 3 Analyse the problem. Next, section 4 presents the proposed solution. Then, Section 5 describes the experimental setup and performance evaluation. Lastly, Section 6 concludes the study and makes recommendations for future research.

Related Works

Selecting a suitable position in SDWAN for Controller Placement is crucial to its performance[12]. Inappropriate Controller Placement can increase communication overhead and Flow Setup Delay. Therefore, several CPP solutions have been proposed[33]. The CPP solutions presented in [17], [19], [25]–[27] utilised spectral Clustering to partition the wide-area Network into many subnetworks. Some authors infer the count of subnets by exploiting the concept of eigenvectors, using the Haversine equation to calculate the similarity graph. Each resulting subnetwork is assigned a dedicated controller at a location that minimises the control message Latency. Researchers in [27] formulate the CPP as an Integer Linear Programming (ILP) with the optimisation objective of reducing the network cost. They design a heuristic method to solve the ILP. However, Spectral Clustering tends to produce small, isolated components and clusters of similar sizes. In addition, all the solutions did not quantify the controller overhead and Response Time(RT) in the performance validation.

In a different approach to formulating a clustering-based CPP [34], researchers utilise Integer Programming (IP). They reduce the network's transmission time by employing a modified version of k-means with the shortest path as the similarity metric. In [29], the authors formulate a binary variable model of the CPP and cluster it using an Affinity Propagation Technique (APT). APT maximised similarity across short distances and moderated preference control to a mean value. In another approach, [28] propose Density-Based Controller Placement (DBCP) to partition a network into various sub-networks. The DBCP grouped tightly connected switches within the same subnet and less-connected switches in a different subnet. The value of k and members of each subnet is determined based on the distance to a higher-density node. Each sub-network is assigned a single controller. In other techniques, PAM-B Clustering and NSGA-II were utilised by [30] to solve the Network partitioned-based CPP with the multi-objective problem of optimising Latency, capacity, and availability. In another approach, using the shortest path as the similarity metric [18], [35], partitioned a Network for CPP using k-means. Starting with a random centroid, the Algorithm iterates continuously until it divides the network into k clusters. In a similar effort, researchers utilised Simulated Annealing (SA) and the k-median Algorithm [23] to determine the optimal location for a satellite gateway in a 5G network, aiming to reduce Latency. The authors implemented a clustering strategy to improve connectivity reliability between satellites and controller nodes. Also, [15] confronts the Network partitioning problem by employing the k*-means for a CPP. Initialised the partitioning with more than k clusters and later merged the nodes into the k clusters recursively based on the shortest path distance and cluster load. While in a different approach proposed by [14], for Network partition-based controller placement to reduce Latency, the authors utilise a k-means algorithm with initialisation based on cooperative game theory. Cooperative game with a set of switches as players are used to mimicking the division of the Network into subnetworks. The switches attempt to build alliances with other switches to increase their value. They also suggest two variations of the cooperative k-means technique to create size-balanced partitions. However, these approaches did not consider load balance issues. The authors in [31] formulated the CPP as an IP. The Network was divided into partitions using a k-medoid clustering technique. However, the value of k is determined via a brute-force approach. In contrast, CPP was tackled using a k-centre/k-median Clustering strategy by [24]. The authors suggested creating a local and global controller hierarchy. When a controller fails, it is replaced using the re-election procedure. To assess load balancing [20], [21] defines two distinct cost functions regarding the network topology structure and flow traffic distribution. They then hybridise the network partition scheme to tackle the problem of where to locate the load-balancing controller. Each of the numerous sub-domains that comprise the overall Network has one dedicated controller. Finally, a Simulated Annealing Partition-based K-Means (SAPKM) to address the placement is proposed. SAPKM incorporates a centroid-based clustering to achieve load-balancing among the controllers. The k-means Algorithm uses Euclidean distance as its similarity metric. However, the problem is that it is not always possible to compute the Euclidean distance in real networks due to the lack of physically connected pathways. Similarly, K-means has no agreed-upon way to determine the first k partitions. The method varies in how it initialises the first set of clusters head. Thus, the initial cluster head selection significantly affects the solution quality in k-means; this is considered a significant limitation. On the other hand, PAM is quite like k-means, except that it establishes a node in the cluster's centre as the head to minimise the effects of the outliers. Although they do not require prior knowledge of k, they have a significantly higher level of complexity to the tune of about cubic time. At the same time, Spectral Clustering tends to produce small, isolated components and clusters of similar sizes.

Network Clustering for CPP using Data Field Theory (DFT) was proposed by [36]. The DFT considers the strength of the wireless nodes' transmissions and reception signal power to determine the controller placement inside each cluster to reduce Latency and energy. While [37],

[38] presents an SDN partition strategy for controller placement in IoT environments to reduce Latency using the Analytical Network Process (ANP). The authors thoughtfully consider multiple latency-inducing parameters to guide their ranking and selection process with ANP. However, it's worth noting that one parameter that wasn't considered in their analysis is the controller's overhead. This omission is significant as it can impact performance and should ideally be factored into such an optimization strategy.

Another work [32] employed a graph theory to identify the number of controllers and their initial location. A Depth-First-Search algorithm is applied to determine Articulation Points (AP) based on two conditions. To obtain the required number of controllers and placement positions, they utilize APs. Additionally, they discretize a supervised machine learning concept using Manta-Ray Foraging Optimization (MRFO) and Salp Swarm Algorithm (SSA) to solve CPP based on network partitioning. [4]. However, the lack of a standardized and rich dataset for model training has been a serious concern in any AI-based solution for SDN problems [39], [40]. However, privacy and confidentiality issues associated with Networks have made sharing this data difficult and scarce. Additionally, the approaches may be suitable for acquiring the first controller placement. However, it is unrealistic to repeatedly segment the entire Network to meet the evolution of dynamic network changes. Thus, they lack an adaptable CPP that responds to the dynamics of each given Network. Therefore, based on the discussed literature, it can be conclude that all the solutions have not adequetely consider the switch's role in the Network to identify and separate a set of critical from non-critical switches. Recognizing the critical switches is crucial during network partition decisions for optimum controller placement. Such sets of switches possess high degree and betweenness criticality measures with many rules in their flow table entries. As a result, they often augment the flow setup delay and cause more update operations. The problem leads to additional overhead on the controller if multiple critical switches are in the same partition. See Table 1 for the summary of these approaches.

Materials & Methods

Analysis Of Controller Overhead

SDN controller overhead refers to the computational and resource requirements imposed on the SDN controller as it manages and controls the network. Although, the controller operates based on either proactive or reactive mode. The former may have lower overhead but may not cope with the real network []. The latter is widely used due to its flexibility in real-time network. However, any newly arrived Flow nF_i at switch $s_i \in S$ without corresponding forwarding rule entries in its flow table will introduce an overhead of composing and sending a Packet_IN message to its controller $SPr_{overhead}$ on the switch. Likewise, on its part, the controller C also suffers the overhead of computing the required forwarding rule and subsequent installation in the switches $s_i \in S$ flow Table via Packet_OUT message $CPr_{overhead}$. Due to these overheads, the new flow nF_i, will experience a path setup time delay $FSetUp_{SC}$, while waiting to be directed by a controller C. The flow/path setup delay emanates from five sources (i) a queue waiting time

wtS at the switch S_i before being served for duration stS, (ii) a switch s_i to controller C Packet_IN message propagation time $P_{in}(s_i,C)$ (iii) a queue waiting time wtC at controller C before being served for (iv) a duration $stC$and (v) controller C to switch S Packet_OUT message propagation time $P_{out}(C, S_i)$. Therefore, cumulatively, the flow setup time delay is determined by.

$$FSetUp = wtS + stS + P_{in}(S_i,C) + wtC + stC + P_{out}(C, S_i) \quad (1)$$

The Eqn (1) above fundamentally comprised the switch S_i processing overhead, the controller C processing overhead, and the round-trip time between switch S_i, and the controller C, given by Eqn (2), Eqn (3), and Eqn (4), respectively.

$$S_iPr_{overhead} = wtS + stS \quad (2)$$

$$CPr_{overhead} = wtC + stC \quad (3)$$

$$R_{TT} = P_{in}(S,C) + P_{out}(C, S) \quad (4)$$

Considering a network topology with an S set of switches $s_i \in S$ and E, as the communication links between the switches, can be represented as graph $G = (S, E)$. Any mapping of a set of switches $s_i \in S$ with a controller C impose an overhead $\mathrm{CPr}_{\mathrm{overhead}}$ on the controller that is directly proportional to the cost of the flow rule setup request and subsequent rule installation in the flow table.

$$\mathrm{CPr}_{\mathrm{overhead}} \propto \sum \mathrm{SPr}_{\mathrm{overhead}} \quad (5)$$

The $SPr_{overhead}$ at the switch S_i is determined by the load of the switch due to the new flow nF_i arrival rate from both the external source (Host) and internal source (s_j). As stated in Eqn (5), the overhead $SPr_{overhead}$ directly increases the $CPr_{overhead}$. Therefore, if nF_{h_0,S_i}, denote the external new flows arrival rate at the switch s_i from host h_0 . Let $X_{im} \in \{0,1\}$ variables indicate whether the switch s_i is under the control of the controller C_m or not, using $X_{ia} = \begin{cases} 1, if\ s_i \rightarrow C_m \\ 0, if s_i \nrightarrow C_m \end{cases}$. Thus, the nF_i arrival rate at S_i from host h_0 will induce rule computation overhead on the controller equivalent to:

$$\sum_{s_i \in S} (nF_{h_0,S_i}) X_{im} \quad (6)$$

Hence if nF_{h_0,S_i}, denote the internal new flows arrival rate at the OpenFlow switch S_j from host S_i. The arrival rate will induce rule computation overhead at the SDN controller C_m equals to

$$\sum_{s_i \in S} (nF_{S_i,S_j}) X_{im} \quad (7)$$

Therefore, for all the OpenFlow switches controlled by the controller C_m, The total overall overhead on the controller for rules installation in the OpenFlow switch S_i is equal to:

$$CPr_{overhead} = \sum_{s_i \in S}(nF_{h_0, s_i})X_{im} + \sum_{s_i s_j \in S}(nF_{s_i, s_j})X_{im} + \sum_{s_i s_j \in S}(nF_{s_i, s_j})X_{im} + \sum_{s_i \in S}(nF_{s_i, h_0})X_{im}$$

The objective is to minimize the $CPr_{overhead}$ to improve the overall $FSetUp$ and other QoS metrics. High controller overhead directly increases flow setup time which consequently causes performance retardation, especially for traffic with deadline violation constraints.

Design of the Proposed Solution

The proposed Controller Placement Algorithm with Critical Switch Awareness (CPCSA) for Software-Defined Wide Area Network partitioned the Network based on the switch role and assigned the required number of controllers to each partition. The operational procedure of CPCSA consists of three phases, with the output of each phase serving as input to the next phase. (i) The Critical Switch Identification Phase (CSIP) for reading the network topology to identify critical switches. (ii) Network Partition Phase (NPP) for partitioning the discovered topology based on the number of critical switches identified in (CSIP) and (iii) Controller Placement and Assignment Phase (CPAP), which uses the mathematical concept of facility location method to select a strategic position to place an SDN controller for each of the partitions formed in (NPP). This way, CPCSA placed an SDN controller in each partition formed based on the distance between the critical and non-critical switches within the partition to minimize the communication overhead and delay. Subsections (3.2 - 3.6) provide a detailed description of each phase. At the same time, the flowchart shown in Figure 2 presents the overall procedure of the proposed Algorithm (CPCSA).

Network Model and Placement Metrics

Consider an SDWAN topology modelled as a graph $\mathrm{G} = (\mathrm{V}, \mathrm{E})$, with V representing a set of nodes and E the communication links between the nodes. The network node V comprised a group of OpenFlow switches S and an SDN Controllers C, i.e., $\mathrm{S}, \mathrm{C} \in \mathrm{V}$. The collection of the OpenFlow Switches S includes critical switches (CS) and non-critical switches (nCS). For controller placement, the technique partitions G into multiple sub-nets $\mathrm{SDWAN\_Partitions}_i$ to improve latency performance and reduce a Controller's overhead. In this study, we formulate the network partition problem by considering the switch's role in the Network. This help in identifying the critical and non-critical switches in the Network. We defined the set of critical switches (SCS) as:

$$\mathrm{SCS} = \sum_{i=1}^{k} \mathrm{CS}_i \qquad (10)$$

Where k represents the Network's total number of critical switches and gives us the number of subnets to partition the Network G. At the same time, we can obtain the set of non-critical switches from

$$\mathrm{SnCS} = \mathrm{S} \backslash \mathrm{SCS} \qquad (11)$$

Therefore, by partitioning the OpenFlow switches $S \in G$ into k sub-nets, namely, $SDWAN\_Partitions_i \; \forall \; i = 1,2,..,k$ according to the number of critical switches $CS \subset V$. The resulting $SDWAN\_Partitions_i$ can be defined as:

$$SDWAN\_Partitions_i = (V_i, E_i) \quad (12)$$

Such that:

$$SDWAN\_Partitions_i \text{ is a component} \quad (13)$$

$$\sum_{i=1}^{k} CS_i = 1 \quad (14)$$

$$\forall i \neq j \in k \, ; SDWAN\_Partitions_i \cap SDWAN_{Partitionsj} = \{\emptyset\} \quad (15)$$

$$\bigcup_{i=1}^{k} V_i, \bigcup_{i=1}^{k} E_i \quad (16)$$

Eqn(13) indicates that the sub-net of any of the $SDN\_partition_i$ is made up of connected OpenFlow switches with links. Eqn(14) ensures only one critical switch CS_i is assigned to each partition. Eqn(15) implies that an OpenFlow switches s_i can only be allocated to a single domain. While Eqn(16) ensures all the Network switches are in one of the subnets. See Table 2 for the summary and description of symbols and notation used in our model.

Network Topology Read Phase

Algorithm 1 reads a GraphML file containing a network topology of SDWAN located at graphml_path. An empty graph object stores the network topology as $G = (V, E)$ created in line 1 of the algorithm. V represents a set of switches in the Network, and E the physical communication links between the nodes. The network switch V comprised some OpenFlow switches S and SDN controllers' C, i.e., $S, C \in V$. However, the OpenFlow switches S consist of critical CS and non-critical switches nCS. The study defines a set of critical switches SCS in Eqn(10). Algorithm 1 reads the file to generate a graph object representing the network topology in line 2. Then, the algorithm returns the graph object in line 3 to identify these critical switches. The read_graphml function is a pre-existing function that reads and parses GraphML files.

Algorithm1: ReadNetworkGraphTopology GraphC_m

Input: - graphml_path: the path to the GraphML file containing the network topology
Output: - G: a graph object representing the network topology
STAT of Algorithm

1. G ← new Graph()
2. G ← read_graphml(graphml_path)
3. For each s_i to $s_j \in G$

4. Compute N_{sp} shortest path, $N_{sp}(s_i s_j)$
5. Return, G, and $N_{sp}(s_i s_j)$

END of Algorithm

Switch Role and Critical Switch Identification Phase (CSIP)

CSIP distinguishes between switches based on their roles to identify critical switches within a network. Because some switches within the network have a significantly higher frequency of communication with the SDN controller for rule installation than others. These switches are called critical switches because they impact the responsiveness of the SDN controller within the network. Therefore, a switch $s_i \in V_i$ with high communication frequency with SDN controller for rule installation is considered more critical $C_{s_i}^{I}$ compared to an ordinary switch.

To establish the criticality of a switch s_i, we used the switch criticality metrics in a network, and the switch flow rule requests overhead on the controller. We assume that information in the network G_i from different sources $s_i\ \forall\ i = 1,2.., N$ is propagated in parallel from the source s_i to the destination s_j along the shortest path (geodesic), denoted as d_{ij}. Based on these assumptions, a switch $s_i\ \forall\ i = 1,2.., N$ in a communication network $G_i = (V_i, E)$ is critical to the extent of its criticality factor $s_i Cr_f$. Therefore, we use the switch's connectivity in the network and its flow rule request overhead on the controller to model the switch criticality factor $s_i Cr_f$.

To determine the switch connectivity in the network, CSIP uses Algorithm 1 to return the number of shortest paths N_{sp} passing through the switch starting at $s_i \in V$ and ending at $s_j \in V$. Thus, we calculate the metric using the formula Eqn (17). On the other hand, to compute the switch traffic overhead on a controller, we consider the weighted new flow rule request sent from the source switch to the controller due to a new flow arrival based on Eqn (6) using Eqn (18). Following that, we compute the switch criticality factor $s_i Cr_f$ using the formula presented in Eqn (19) using these parameters. Finally, we demonstrate the procedure for critical switch identification in Algorithm 2.

$$s_i BC = \sum_{s_i s_j \in V, s_i \neq s_j} \frac{N_{sp}(s_i s_j | V)}{N_{sp}(s_i s_j)} \quad (17)$$

$$s_i nF_i = \sum_{s_i \in S} (nF_{S_i, c_m}) \quad (18)$$

$$s_i Cr_f = s_i BC + s_i nF_i \quad (19)$$

In (lines 1-2), Algorithm 2 initializes two empty dictionaries, SCS and SnCS.The dictionaries are used to store critical-switch and non-critical-switch information, respectively. For each switch $s_i \in V$ in the SDWAN G, Algorithm 2 determines whether the switch s_i is critical or non-critical using Equation (10) and by calculating its criticality factor ($s_i Cr_f$) using Equation (19). The total ($total\_s_i Cr_f$) and average ($ave\_s_i Cr_f$) criticality factors for all switches in the network are also computed (lines 3-8). Algorithm 2 then checks the criticality factor ($s_i Cr_f$) of each switch s_i in the network topology G against the average criticality factor value ($ave\_s_i Cr_f$) (lines 10-11). If ($s_i Cr_f$) is greater than ($ave\_s_i Cr_f$), the switch is classified as critical and added to the set of critical_switch SCS containers along with its criticality factor. Otherwise, it is classified as non-critical and added to the collection of non_critical_switch nSCS containers (lines 12-13). Next, for each critical switch (CS) in the SCS container, Algorithm 2 retrieves the list of its neighbours and calculates its shortest path distance to all other switches in the network topology. The resulting information is added to the CS_neighbors and distances containers (lines 14-20). Finally, Algorithm 2 returns the sets of critical_switch, non_critical_switch, critical_switch_neighbors, and distances in (line 21).

Algorithm2: Critical Switch Identification

Input: - G, and $N_{sp}(s_i s_j)$:

Output- {SCS, SnCS,CS_neighbours, distance}

STAT of Algorithm

1. SCS ← {}
2. SnCS ← {}
3. **FOR** $s_i \in V$:
4. $s_i BC$ ← calculate switch connectivity in G using Eqn (17)
5. $s_i nF_i$ ← calculate switch flow rule request using Eqn (18)
6. $s_i Cr_f$ ← calculate the switch criticality factor using Eqn (19)
7. $total\_s_i Cr_f$ ← sum_of_values ($s_i Cr_f$)
8. $ave\_s_i Cr_f$ ← $total\_s_i BC$ / length_of_values ($s_i Cr_f$)
9. **FOR** each s_i, in ($s_i Cr_f$):
10. **IF** ($s_i Cr_f$)> $ave\_s_i Cr_f$:
11. add s_i and $s_i Cr_f$ to SCS.
12. **ELSE**:
13. add s_i and $s_i Cr_f$ to SnCS.
14. CS_neighbors ← {}
15. **FOR** each s_i, in $s_i Cr_f$:
16. add a list of CS 's neighbours to CS _neighbours.
17. distance ← {}

---

18. **FOR** each CS in SCS:
19. **For** s_i, distance in shortest_path_length from CS in G:
20. add (s_i, CS) and distance to distance.
21. return SCS, SnCS, CS_neighbours, distance

END of Algorithm

Network Partition Based on Switch Criticality

The study designed a CSANP to partition the SDWAN (G) into smaller networks based on the number of critical switches (num_CS). The CSANP collects inputs from Algorithm 2, where the critical switches of G are identified. The input parameters include the set of critical switches (SCS), non-critical switches (SnCS). The procedure is as shown in (Algorithm 3). CSANP starts by initializing the number of Critical Switches (num_CS) and non-Critical Switches (num_nCS) on lines 1 and 2. It then calculates the average number of non-Critical Switches to be associated to each Critcal Switch and the remaining non-Critical Switches (num_CS_plus) on lines 3 and 4. The SDWAN_Partitions list is initialized with empty lists, where each list represents a partition associated with a Critical Switch (CS), on line 5. The algorithm then iterates through each non-Critical Switch (s_j) in SnCS (line 6) and determines its closest Critical Switch (CS) based on the minimum distance (lines 7 to 14). The non-Critical Switch is then assigned to the corresponding partition in SDWAN_Partitions (line 14). Next, the algorithm iterates through each non-Critical Switch again (s_j) (line 15) and assigns it to the appropriate partition in SDWAN_Partitions based on balancing criteria (lines 17 to 29). If a partition has fewer than avr_num_nCS, the current non-Critical Switch is added to it (line 24). If the partition has avr_num_nCS and there are remaining non-Critical Switches (num_CS_plus), one of them is added to the partition (lines 26 to 28). If the partition has avr_num_nCS, and there are no remaining non-Critical Switches, a new partition is created for the current non-Critical Switch (line 30). The process continues until all non-Critical Switches are assigned to partitions, and the resulting SDWAN_Partitions list contains the partitions, each associated with its respective Critical Switch. Finally, the algorithm returns the list of SDN [{SDWAN_Partitions},{SDWAN_Partitions}.........|num_CS|] in line 31. Refer to the Network Partition Formation Phase of Figure (2) for the flowchart for the algorithm.

Algorithm 3: Critical Switch Aware Network Partition (CSANP)

Input: (G, SCS, SnCS)

Output: SDWAN_Partitions

STAT of Algorithm

1. num_CS = len(SCS)
2. num_nCS = len(SnCS)
3. avr_num_nCS = num_nCS // num_CS
4. num_CS_plus = num_nCS % num_CS

Add all Critical Switches to SD-WAN partitions

```
5.  SDWAN_Partitions = [[] for _ in range(num_CS)]

# Assign non-Critical Switch to Critical Switch based on minimum distance
6.  For s_j in SnCS:
7.    closest_CS = None
8.    min_distance = float('inf')
9.    For i, si in enumerate(SCS):
10.       dist = distance[s_i][s_j]
11.       If dist < min_distance:
12.           min_distance = dist
13.           closest_CS = i
14.   SDWAN_Partitions[closest_CS] = SDWAN_Partitions[closest_CS] + [s_j]
      # Balance partitions and assign non-Critical Switches to Critical Switch
15. For i, s_j in enumerate(SnCS):
16.   closest_CS = None
17.   min_distance = float('inf')
18.   For j, s_i in enumerate(SCS):
19.       dist = distance[s_i][s_j]
20.       If dist < min_distance:
21.           min_distance = dist
22.           closest_CS = j
23.   cluster_index = closest_CS
24.   If len(SDWAN_Partitions[cluster_index]) < avr_num_nCS:
25.       SDWAN_Partitions[cluster_index] = SDWAN_Partitions[cluster_index] + [sj]
26.   Elif len(SDWAN_Partitions[cluster_index]) < avr_num_nCS + 1 and num_CS_plus >
0:
27.       SDWAN_Partitions[cluster_index] = SDWAN_Partitions[cluster_index] + [s_j]
28.       num_CS_plus -= 1
29.   Else:
     # If no condition is met, create a new partition for the non-Critical switches
30.       SDWAN_Partitions = SDWAN_Partitions + [[s_j]]

31. return SDWAN_Partitions
END of Algorithm
```

Critical Switch Aware Controller Placement (CSACP)

The proposed Critical Switch Aware Controller Placement (CSACP) algorithm is responsible for placing an SDN controller in each of the resulting network partitions (subnets) produced by CSANP. This placement problem is a variant of a facility location problem. Therefore, for each of the resulting subnets[{$\text{SDWAN\_Partitions}_1$},...{$\text{SDWAN\_Partitions}_{|num\_CS|}$}] obtained from the CSANP, we designed a CSACP algorithm to place the SDN controller on each $SDWAN\_Partitions_i = (V_i,E_i)$ within the shortest distance of each demand point in the subnets. We assigned C to represent the set of controllers $c_j \in C\ \forall\ j = 1,2...,m$ for the k sub-nets. Next, for each, $\forall$ $SDWAN\_Partitions_i$, our placement model maps the controller $c_j \in C\ \forall\ j = 1,2...,m$ to the demand points $s_i \in V$, which are the OpenFlow switches, in a way that the $dist(s_i c_j)$ is the shortest distance between the candidate controller locations $j \in SDWAN\_Partitions_i$ and the mapped controller $c_j \in C$. Thus, the proposed CSACP algorithm finds a suitable position in each resulting partition to place the controller. Algorithm 4 provides a detailed description of the proposed controller placement method.

$$Min \frac{1}{|SDWAN\_Partitions_i|} \sum_{s_i \in SDWAN\_Partitions_i} dist(s_i c_j) \qquad (20)$$

Such that

$$s_i, c_j \in SDWAN_{Partitionsi} \qquad (21)$$

The proposed CSACP algorithm takes inputs from CSANP (Algorithm 2), which includes the SDWAN partitions, critical and non-critical switches, and their criticality factors. Each partition is a set of switches within the SDWAN network. The algorithm initializes an empty dictionary called controller_positions to store the controller positions for each SDWAN partition in line 1. Then, for each partition in the input set of partitions, the algorithm identifies the critical switch with the highest criticality factor max $\_s_i Cr_f$. In (lines 2-11), Algorithm 4 calculates the distance to the identified critical switch using a pre-computed distance metric stored in a distance dictionary for each non-critical switch in the partition. Next, the algorithm finds the non-critical switch within the partition that has the minimum distance to the identified critical switch and assigns it as the controller position for that partition. The algorithm then stores the controller position for that partition in the controller_positions dictionary in (lines 12-26). Finally, the algorithm returns the controller_positions dictionary as the Algorithm output in line 27.

Algorithm 4: Critical Switch Aware Controller Placement (CSACP)

Input:$\{SCS, SnCS\}$ $[\{SDWAN\_Partitions_1\},...\{SDWAN\_Partitions_{|num\_CS|}\}]$

Output- controller_positions

STAT of Algorithm

1. controller_positions = {}
2. **For** SDWAN_Partitions_num, partition in enumerate(SDWAN_Partitions) **Do**
3.   max_critical_switch = null
4.   max $\_s_i Cr_f$ = -1
5.   **For** switch in partition, **Do**
6.   **If** switch in critical_switch and critical_switch[switch] > max $\_s_i Cr_f$ **Then**

7. max_critical_switch = switch
8. max _$s_i Cr_f$ = critical_switch[switch]
9. End If
10. End For
11. distances_within_partition = {}
12. **For** a node in partition, **Do**
13. **If** the node in non_critical_switch, **Then**
14. distances_within_partition[node] = distances[(node, max_critical_switch)]
15. End If
16. End For
17. min_distance_node = null
18. min_distance = infinity
19. **For** a node in distances_within_partition, **Do**
20. **If** distances_within_partition[node] < min_distance, **Then**
21. min_distance_node = node
22. min_distance = distances_within_partition[node]
23. End If
24. End For
25. controller_positions[SDWAN_Partitions_num] = (max_critical_switch, min_distance_node)
26. End For
27. return controller_positions.

END of Algorithm

Experimentation Setup and Performance Evaluation of CPCSA

In this section, the performance of CPCSA is evaluated and compared with other representative solutions in the literature. The study utilizes three (3) real network topologies obtained from the Internet Topology Zoo (ITZ) [41] and randomly generates topologies for conducting the experiments. The database provides researchers access to hundreds of real network topologies from various service providers. Thus, the study selects AsnetAm, Arpanet19728, and ARNES networks for the experiments. Table 3 gives additional information on other aspects of the chosen network topologies, which vary in size and structure. The partitioning phase is performed offline with a script written in Python 3.8.0 and NetworkX components. The experiment uses Mininet version 2.3.0 to build the topologies of these partitions with an OpenvSwitch for interaction with a Ryu SDN controller in each partition based on OpenFlow v1.5.1 specifications. The paper borrows traffic matrix scenarios in the GÉANT network [42] for understanding traffic patterns. The traffic matrix of [42] describes the traffic between nodes and its transfer speed, highlighting what constitutes a new flow. A D-ITG utility injects a TCP/UDP flow on 1024 Mbps transmission lines of the Mininet architecture to generate the traffic. Hence, the study model, one new flow for every 100 000 KB, exchanged, according to Poisson traffic distribution in terms of Packet Inter Departure Time (PIDT). The reliance of the packet_IN message on whether the switch piggybacked the first packet of a flow to a controller. Dixit et al. [3]. The paper considers its size and Packet count as in [43] to account for it. Additionally, as proved in [43], there must be a packet OUT message (flow_mod Packet) for every packetIN message; thus, the study considers their sizes and packet count equal.

We start off the evaluation of CPCSA by providing a visual representation of its controller placement result in Fig 1(a) through 1(i) in section 5.1. We then presented the overhead incurred by the controller placed in a network using the proposed CPCSA compared to other related CPP solutions in section 5.2. While in section 5.3, the study investigates the impact of CPCSA on fault tolerance by evaluating the rate of control packet loss. Lastly, the evaluation of Throughput and average switch-to-controller Latency is done in sections 5.3 and 5.4, respectively. We conduct all the experiments on a machine with Intel(R) Core (TM) i7-10750H CPU @ 2.60 GHz, 2.59 GHz, and 16.0 GB memory.

Results

Network Partitions and Controller Placement Positions

The diagrams presented in Figure 3, from (3a through 3l) illustrate the network partitions and selected positions for controller placement as determined by the proposed CPCSA algorithm. These Figures depict the outcomes of the controller placement output when applied to the Arpanet19728, ARNES, and AsnetAm topologies. As demonstrated in Figures (3a), (3e), and (3i), before network partitioning, node 4, node 7, and node 22 are designated as the controller positions. This selection occurs based on the switch criticality factors s_iCr_f ranging from 0.25, 0.50-0.61, to 0.59-0.66 in the respective topologies. Conversely, as shown in Figures (3b), (3f) and (3j), when the switch criticality factors are 0.25, 0.18-0.49, and 0.27-0.55 in the corresponding networks, the networks are partitioned into two subnets. Consequently, in Arpanet19728, nodes 4 and 13 are chosen as the controller positions, while in ARNES, nodes 7 and 30 are selected. In the AsnetAM topology, the controller positions are nodes 22 and 7. Furthermore, by reducing the switch criticality factors s_iCr_f to 0.22, 0.14-0.15, and 0.15-0.25, the respective networks experienced partitioning into four subnets. This resulted in the inclusion of nodes 23 and 28 as additional controller positions in the Arpanet19728 topology. Similarly, in the case of ARNES, nodes 23 and 29 were selected as new placements. While for AsnetAM topology, CPCSA chooses nodes 8 and 26 to place the new controllers. Please refer to Figures (3d), (3h), and (3l) for visualization

Controller Overhead

Fig. 4 shows the accumulated controller's rule installation overhead in the Arpanet19728, ARNES, and AsnetAm network topologies with SPDA[44], gravCPA[45], and the proposed

CPCSA, respectively. The experiment results show that CPCSA incurred lower rule installation overhead than SPDA[44] and gravCPA[45] in all the topologies. As shown in Fig 4(a), the proposed CPCSA had reduced the SDN controller's overhead compared to SPDA and gravCPA in the AsnetAM topology by 63% and 49%, respectively. Meanwhile, in Fig 4 (b), with the Arnes topology, the proposed technique is shown to cut the overhead by 54% and 36%. Lastly, CPCSA minimizes the overhead of SPDA[44] and gravCPA[45] by 63% and 51% in the Arpanet19728 topology, as revealed in Fig 4(c). The achievement of the overhead reduction is attributable to the control of the number of critical switches CPCSA assigns to a single SDN controller. A switch is critical if it continually appears along the shortest path of many dissimilar host-to-destination communicating pairs. This type of switch receives an augmented number of rule installation instructions from the controller on what to do with the flow. Because, by default, Flows are usually routed along the shortest path from the source to the destination host in most networks. Thus, the controller with a higher number of critical switches in a partitioned SDWAN incurs higher overhead. The additional controller overhead will amount to the number of switches assigned to the controllers by a factor of their generated control traffic.

Control Packet Loss

In this section, this study measures the impact of control packet loss during switch-to-controller communication to verify CPCSA's fault-tolerance benefits. High control plane overhead can induce a network problem, which can cause some switches to lose connections with their controllers, resulting in dropped packets. The study expects CPCSA to reduce the possibility of Network failures owing to excessive controller overhead, which can lead to substantial packet loss. Because, by design, the CPCSA differentiates among network switches and restricts the number of critical switches for each partition. We use Python 3.8.0 with NetworkX and Matplotlib library components for simulation. However, unlike the previous experiments with real network topologies, fully connected networks are randomly generated using Barabási–Albert (BA) model. After 50 repeated experiments, the average results findings in comparison to alternative approaches are shown in Figure 5. The y and x-axis in the Figure display the average control packet loss as a function of the x-axis representation of the total network nodes, n. As expected, CPCSA has the lowest average packet loss rate of the four routing algorithms due to minimising the controller's overhead. On DBCB, the proposed CPCSA reduced packet loss by 31%, while on SPDA and gravCPA, it reduced it by 61%. The Minimum Controller's overhead correlates better with preventing network failure and lower control packet loss. Therefore, a low average control packet loss indicates the technique's ability to avoid network faults due to high overhead.

Throughput

Figure 6 displays the network throughput evaluation result between the proposed CPCSA and the benchmark algorithms. The Throughput metric gives information about the performance of the techniques regarding the number of control data packets sent from a source host and successfully delivered at the destination host during a transmission period[44]. The throughput metric is relevant in assessing Controller Placement Algorithm performance about how it reacts to network-changing events that can trigger flow setup requests or failure. Figure 6(a) shows the result of CPCSA's Throughput with different numbers of controllers. Figure 6(b) shows the CPCSA's Throughput versus that of gravCPA[38] and SPDA[43]. As can be seen from Figure 6(b), CPCSA outperformed the benchmarked reference algorithms. Comparatively, the algorithm improved the throughput achieved by gravCPA and SPDA by 16% and 18%, respectively. This improvement indicates that the methodology adopted by CPCSA to minimise the Controller's overhead significantly influenced the control packet delivery rate. Thus, this analysis affirms the research question, "Can controlling the number of critical switches under the control of an SDN controller improve the Quality of Service in a network."

Switch to Controller Average Latency

In this subsection, the study demonstrates how the average switch-controller latencies respond when a controller is appropriately placed in the subnets of the network partitioned while considering critical switches. For validation and revelation of results, the study compares the performance of CPCSA with that of other Controller Placement solutions that incorporate a network partitioning strategy and allocation of a controller to each subnetwork. In the experiments, we ensure that all the benchmarked algorithms deploy the same number of controllers as CPCSA in the network for a fair evaluation. Therefore, given a controller $c_j \in C$ and the switches $s_i \in SDWAN\_Partitions_i$ in the sub-network, the CPCSA uses the relation in Equation (18) to measure the latency metrics. Based on the result obtained, Figure 7 displays the relationships between the average switch-controller latencies with the number of controllers and partitions varying from 1 to 4 on three (3) topologies. As shown in Figure 7, the result exhibits a monotonic decreasing trend in the switch-controller Latency with an increasing number of partitions and controllers. We observed this pattern throughout all four (4) algorithms under study. i.e., Increasing the number of controllers and partitions causes all the compared algorithms to behave identically regarding average switch-controller control packet processing delay. However, CPCSA performs significantly better when compared to SPDA, DBCP, and gravCPA algorithms. As shown in Figure 7(a), the proposed CPCSA reduces the average switch-to-controller Latency by 27%,12%, and 3%, respectively, compared to SPDA[44], DBCP[28], and gravCPA[45] algorithms when the Algorithms partitioned the network into 4.

Conclusions

Controller Placement Algorithm with Network Partition Based on Critical Switch Awareness (CPCSA) is a novel approach to address the challenge of transient congestion due to controllers' overhead in the existing Controller Placement Problems (CPP) solutions in SDN. CPCSA identifies the set of critical switches in a network to guide the network partition procedure for finding the optimal number of controllers and placement in the network. The algorithm has been implemented and evaluated in a laboratory testbed in a series of comparative experiments with similar solutions using multiple Real life network topologies from ITZ. The comparative experiments demonstrate CPCSA's effectiveness in reducing control message Overhead, control packet loss, switch-to-controller latency, and improved throughput. The results show that the proposed solution has achieved an aggregate reduction in the controller's overhead by 73%, Loss by 51%, and Latency by 16% while improving throughput by 16% compared to the benchmark algorithms. However, the proposed scheme does not support heterogeneous controllers and has no defense mechanism against vulnerabilities such as DDOS, common-mode fault, etc.

For future research, we plan to update the CPCSA controller placement model with traffic flow behavioural quality of service requirements for consideration. It would be intriguing to employ machine learning techniques such as deep learning to study flow behaviour based on flow history for the classification. Considering this would support designing a controller placement with traffic dynamics awareness. The aim is to partition the Network and place a controller while considering the traffic pattern in the Network. Another exploration avenue could be integrating the algorithm with heterogeneous controllers' support. We can see the motivation for these from many perspectives. First, a homogeneous CP provides a potential security risk due to the controllers' common-mode fault, often known as a common vulnerability point. Assume enemies are aware of the vulnerability of one Controller; in this instance, they can easily knock down the entire Network by exploiting the controller's shared vulnerability. Second, interoperability between various controller platforms and traditional IP networks can encourage and facilitate the commercial adoption of SDN globally. Very little research has examined this direction thus far. Therefore, undertaking further research in this direction will be a valuable contribution.

Acknowledgements

We thank the Deanship of Scientific Research at King Khalid University, Universiti Teknologi Malaysia, and Tertiary Educational Trust Fund.

References

[1] M. N. Yusuf, K. bin A. Bakar, B. Isyaku, A. H. Osman, M. Nasser, and F. A. Elhaj, "Adaptive Path Selection Algorithm with Flow Classification for Software-Defined Networks," *Mathematics*, vol. 11, no. 6, p. 1404, Mar. 2023, doi: 10.3390/math11061404.

[2] B. Isyaku and K. B. A. Bakar, "Managing Smart Technologies with Software-Defined Networks for Routing and Security Challenges: A Survey," *Comput. Syst. Sci. Eng.*, vol. 47, no. 2, pp. 1839–1879, 2023, doi: 10.32604/csse.2023.040456.

[3] M. N. Yusuf, "Review of Path Selection Algorithms with Link Quality and Critical Switch Aware for Heterogeneous Traffic in SDN Review Paper," pp. 345–370.

[4] N. Firouz, M. Masdari, A. B. Sangar, and K. Majidzadeh, "A novel controller placement algorithm based on network portioning concept and a hybrid discrete optimization algorithm for multi-controller software-defined networks," *Cluster Comput.*, vol. 0123456789, 2021, doi: 10.1007/s10586-021-03264-w.

[5] B. Isyaku, K. Bin Abu Bakar, W. Nagmeldin, A. Abdelmaboud, F. Saeed, and F. A. Ghaleb, "Reliable Failure Restoration with Bayesian Congestion Aware for Software Defined Networks," *Comput. Syst. Sci. Eng.*, vol. 46, no. 3, pp. 3729–3748, 2023, doi: 10.32604/csse.2023.034509.

[6] B. Isyaku, K. Bin Abu Bakar, M. N. Yusuf, and M. S. Mohd Zahid, "Software Defined Networking Failure Recovery with Flow Table Aware and flows classification," in *2021 IEEE 11th IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE)*, Apr. 2021, pp. 337–342, doi: 10.1109/ISCAIE51753.2021.9431786.
[7] G. Zhao, L. Huang, Z. Yu, H. Xu, and P. Wang, "On the Effect of Flow Table Size and Controller Capacity on SDN Network Throughput," pp. 1–6, 2017.
[8] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," *Comput. Commun. Rev.*, vol. 41, no. 4, pp. 254–265, 2011, doi: 10.1145/2043164.2018466.
[9] S. Ahmad and A. H. Mir, "Scalability, Consistency, Reliability and Security in SDN Controllers: A Survey of Diverse SDN Controllers," *J. Netw. Syst. Manag.*, vol. 29, no. 1, pp. 1–59, 2021, doi: 10.1007/s10922-020-09575-4.
[10] B. Yan, Q. Liu, J. Shen, D. Liang, B. Zhao, and L. Ouyang, "A survey of low-latency transmission strategies in software defined," *Comput. Sci. Rev.*, vol. 40, p. 100386, 2021, doi: 10.1016/j.cosrev.2021.100386.
[11] T. Hu, Z. Guo, T. Baker, and J. Lan, "Multi-controller Based Software-Defined Networking : A Survey," *IEEE Access*, 2017.
[12] B. Heller, R. Sherwood, and N. Mckeown, "The controller placement problem," *Comput. Commun. Rev.*, vol. 42, no. 4, pp. 473–478, 2012, doi: 10.1145/2377677.2377767.
[13] B. P. R. Killi and S. V. Rao, "Controller placement in software defined networks: A Comprehensive survey," *Comput. Networks*, vol. 163, 2019, doi: 10.1016/j.comnet.2019.106883.
[14] B. R. Killi, E. A. Reddy, and S. V. Rao, "Game theory based network partitioning approaches for controller placement in SDN," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 11227 LNCS, pp. 245–267, 2019, doi: 10.1007/978-3-030-10659-1_11.
[15] H. Kuang, Y. Qiu, R. Li, and X. Liu, "A hierarchical K-means algorithm for controller placement in SDN-Based WAN architecture," *Proc. - 10th Int. Conf. Meas. Technol. Mechatronics Autom. ICMTMA 2018*, vol. 2018-Janua, pp. 263–267, 2018, doi: 10.1109/ICMTMA.2018.00070.
[16] J. Liu, J. Liu, and R. Xie, "Reliability-based controller placement algorithm in software defined networking," *Comput. Sci. Inf. Syst.*, vol. 13, no. 2, pp. 547–560, 2016, doi: 10.2298/CSIS160225014L.
[17] P. Xiao, W. Qu, H. Qi, Z. Li, and Y. Xu, "The SDN controller placement problem for WAN," *2014 IEEE/CIC Int. Conf. Commun. China, ICCC 2014*, pp. 220–224, 2015, doi: 10.1109/ICCChina.2014.7008275.
[18] G. Wang, Y. Zhao, J. Huang, and Y. Wu, "An Effective Approach to Controller Placement in Software Defined Wide Area Networks," *IEEE Trans. Netw. Serv. Manag.*, vol. 15, no. 1, pp. 344–355, 2018, doi: 10.1109/TNSM.2017.2785660.
[19] P. Xiao, Z. Li, S. Guo, H. Qi, W. Qu, and H. Yu, "A K self-adaptive SDN controller placement for wide area networks," *Front. Inf. Technol. Electron. Eng.*, vol. 17, no. 7, pp. 620–633, Jul. 2016, doi: 10.1631/FITEE.1500350.
[20] K. Yang, D. Guo, B. Zhang, and B. Zhao, "Multi-Controller Placement for Load Balancing in SDWAN," *IEEE Access*, vol. 7, pp. 167278–167289, 2019, doi: 10.1109/ACCESS.2019.2953723.
[21] K. Yang, B. Zhang, D. Guo, M. Lin, and T. De Cola, "Partitioned controller placement in

SDWANs for reliability maximization with latency constraints," *2019 IEEE Globecom Work. GC Wkshps 2019 - Proc.*, 2019, doi: 10.1109/GCWkshps45667.2019.9024372.

[22] Zhu T, Feng D, Wang F, Hua Y, Shi Q, Xie Y, Wan Y. "A congestion-aware and robust multicast protocol in SDN-based data center networks". *Journal of Network and Computer Applications. 2017 Oct 1;95:105-17*.

[23] J. Liu, Y. Shi, L. Zhao, Y. Cao, W. Sun, and N. Kato, "Joint Placement of Controllers and Gateways in SDN-Enabled 5G-Satellite Integrated Network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 2, pp. 221–232, 2018, doi: 10.1109/JSAC.2018.2804019.

[24] H. I. Kobo, A. M. Abu-Mahfouz, and G. P. Hancke, "Efficient controller placement and reelection mechanism in distributed control system for software defined wireless sensor networks," *Trans. Emerg. Telecommun. Technol.*, vol. 30, no. 6, pp. 1–19, 2019, doi: 10.1002/ett.3588.

[25] H. Aoki and N. Shinomiya, "Network Partitioning Problem for Effective Management of Multi-domain SDN Networks," *Int. J. Adv. Networks Serv.*, vol. 8, no. 3–4, pp. 62–77, 2015, [Online]. Available: https://thinkmind.org/index.php?view=article&articleid=netser_v8_n34_2015_5.

[26] H. Aoki and N. Shinomiya, "Controller Placement Problem to Enhance Performance in Multi-domain SDN Networks," *CICN 2016 Fifteenth Int. Conf. Networks (includes SOFTNETWORKING 2016)*, no. c, pp. 95–101, 2016.

[27] Z. Zhao and B. Wu, "Scalable SDN architecture with distributed placement of controllers for WAN," *Concurr. Comput.* , vol. 29, no. 16, pp. 1–9, 2017, doi: 10.1002/cpe.4030.

[28] 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," *Comput. Networks*, vol. 112, pp. 24–35, Jan. 2017, doi: 10.1016/j.comnet.2016.10.014.

[29] J. Zhao, H. Qu, J. Zhao, Z. Luan, and Y. Guo, "Towards controller placement problem for software-defined network using affinity propagation," *Electron. Lett.*, vol. 53, no. 14, pp. 928–929, 2017, doi: 10.1049/el.2017.0093.

[30] F. Bannour, S. Souihi, and A. Mellouk, "Scalability and reliability aware SDN controller placement strategies," in *2017 13th International Conference on Network and Service Management, CNSM 2017*, 2017, vol. 2018-Janua, pp. 1–4, doi: 10.23919/CNSM.2017.8255989.

[31] A. Dvir, Y. Haddad, and A. Zilberman, "Wireless controller placement problem," *CCNC 2018 - 2018 15th IEEE Annu. Consum. Commun. Netw. Conf.*, vol. 2018-Janua, pp. 1–4, 2018, doi: 10.1109/CCNC.2018.8319228.

[32] G. R. R. Manoharan, "Enhanced optimal placements of multi - controllers in SDN," *J. Ambient Intell. Humaniz. Comput.*, vol. 12, no. 7, pp. 8187–8204, 2021, doi: 10.1007/s12652-020-02554-2.

[33] M. N. Yusuf, K. Bin Abu Bakar, B. Isyaku, and F. Mukhlif, "Distributed Controller Placement in Software-Defined Networks with Consistency and Interoperability Problems," *J. Electr. Comput. Eng.*, vol. 2023, pp. 1–33, Jan. 2023, doi: 10.1155/2023/6466996.

[34] L. Zhu, R. Chai, and Q. Chen, "Control plane delay minimization based SDN controller placement scheme," in *2017 9th International Conference on Wireless Communications and Signal Processing, WCSP 2017 - Proceedings*, 2017, vol. 2017-Janua, pp. 1–6, doi: 10.1109/WCSP.2017.8171153.

[35] G. Wang, Y. Zhao, J. Huang, Q. Duan, and J. Li, "A K-means-based network partition

algorithm for controller placement in software defined network," *2016 IEEE Int. Conf. Commun. ICC 2016*, 2016, doi: 10.1109/ICC.2016.7511441.

[36] F. Li, X. Xu, X. Han, S. Gao, and Y. Wang, "Adaptive controller placement in software defined wireless networks," *China Commun.*, vol. 16, no. 11, pp. 81–92, 2019, doi: 10.23919/JCC.2019.11.007.

[37] J. Ali and B. H. Roh, "An Effective Approach for Controller Placement in Software-Defined Internet-of-Things (SD-IoT)," *Sensors*, vol. 22, no. 8, 2022, doi: 10.3390/s22082992.

[38] J. Ali, S. Lee, and B. H. Roh, "Poster: Using the analytical network process for controller placement in software defined networks," *MobiSys 2019 - Proc. 17th Annu. Int. Conf. Mob. Syst. Appl. Serv.*, pp. 545–546, 2019, doi: 10.1145/3307334.3328617.

[39] B. Isyaku, M. S. Mohd Zahid, M. Bte Kamat, K. Abu Bakar, and F. A. Ghaleb, "Software Defined Networking Flow Table Management of OpenFlow Switches Performance and Security Challenges: A Survey," *Futur. Internet*, vol. 12, no. 9, p. 147, 2020, doi: 10.3390/fi12090147.

[40] M. S. Elsayed, N. A. Le-Khac, S. Dev, and A. D. Jurcut, "Machine-Learning Techniques for Detecting Attacks in SDN," *Proc. IEEE 7th Int. Conf. Comput. Sci. Netw. Technol. ICCSNT 2019*, pp. 277–281, 2019, doi: 10.1109/ICCSNT47585.2019.8962519.

[41] A. G. University of Adelaide, "The Internet Topology Zoo - Dataset," 2023. https://ieeexplore.ieee.org/abstract/document/6027859/%0Ahttp://topology-zoo.org/dataset.html.

[42] S. Uhlig, B. Quoitin, J. Lepropre, and S. Balon, "Providing public intradomain traffic matrices to the research community," in *Computer Communication Review*, 2006, vol. 36, no. 1, pp. 83–86, doi: 10.1145/1111322.1111341.

[43] M. Obadia, M. Bouet, J. L. Rougier, and L. Iannone, "A greedy approach for minimizing SDN control overhead," *1st IEEE Conf. Netw. Softwarization Software-Defined Infrastructures Networks, Clouds, IoT Serv. NETSOFT 2015*, 2015, doi: 10.1109/NETSOFT.2015.7116135.

[44] J. Guo, L. Yang, D. Rincon, S. Sallent, Q. Chen, and X. Liu, "Static Placement and Dynamic Assignment of SDN Controllers in LEO Satellite Networks," *IEEE Trans. Netw. Serv. Manag.*, vol. 19, no. 4, pp. 4975–4988, 2022, doi: 10.1109/TNSM.2022.3184989.

[45] C. Wang, H. Ni, and L. Liu, "GravCPA: Controller Placement Algorithm Based on Traffic Gravitation in SDN," *J. Control Sci. Eng.*, vol. 2022, pp. 1–12, Mar. 2022, doi: 10.1155/2022/1047898.

Table 1 (on next page)

Table 1

Network Partitioned-Based CPP *Pareto Integrated Tabu Search: (PITS), SA: Simulated Annealing*

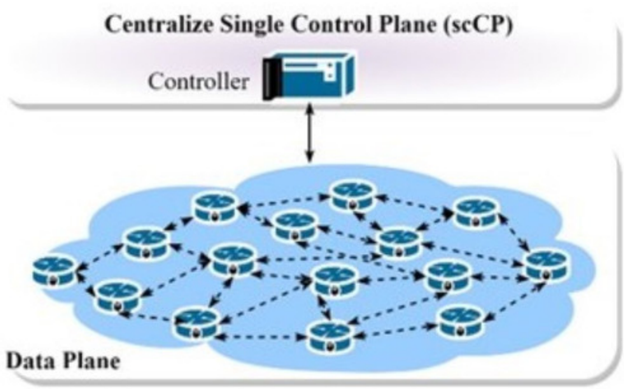
Table 1: Network Partitioned-Based CPP: *Pareto Integrated Tabu Search: (PITS), SA: Simulated Annealing*

Paper	Problem Formulation	Partition/solution Approach	Network Topology Properties			Performance Metrics Considered			Weakness	
			Path	Switch Role	Metrics	Latency	Overhead	Loss	Partition Approach	Performance Metrics
[35]	MILP	Heuristics	✓	X	X	✓	✓	X	Not Partitioned	Throughput and Loss unaccounted
[13], [15]	Network Partitioning	Spectral Clustering	✓	X	Eigen Vectors	✓	X	X	Tend to produce small, isolated components and clusters with similar sizes	High CP Overhead, Poor Load Balancing & CP Overhead and Throughput
[21], [22]			✓	X		✓	X	X		
[36]		Node Burden	✓	✓	Traversal Set	✓	X	X		
[23]	ILP	Spectral Clustering	✓	X	Eigen Vectors	✓	X	X		
[30]		K-Means	✓	X	Euclidean Distance	✓	X	X	Random centre Initialisation stage, the number of cluster determinations	
Modified-AP [25]	BIP	Affinity propagation	✓	X	Shortest Distance	✓	X	X	Not Partitioned	
[24]	Network Partitioning	Density-Based Clustering	✓	✓	Density	✓	X	X	NA	
[26]	MOCO	PAM-B	✓	X	Dijkstra	✓	X	X	Quadratic running time complexity	
SACA[19]	Mathematical	K-Median, SA	✓	X		✓	X	X	Random centre Initialisation, number of cluster determinations, the use of "means" limit its expression level, Euclidean distance" might not get a path physically connected path, one size fits it-all effect, outliers, and noise	
[11]	Network Partitioning	K-Means	✓	X	Euclidean Distance	✓	X	X		
[27]	IP	K-Mediod		X		✓	X	X		
[14], [31]		K-Means	✓	X		✓	X	X		
[37]	Mathematical Model	Clique-Based	✓	X	Shortest distance	✓	X	X	Too rigid to use in practice. It tends to produce maximally cohesive subgraph	The clique property cant guarantee optimum RT
SACKM [16], [17]		Hybridised SA with K-Means	✓	X	Euclidean Distance	✓	X	X	K-means limitation, SA limited memory to track tested solutions, low improvement rate,	
[32]		Data Field Theory	X	X	Signal Strength	✓	X	X	Interference	Ignore the CP Overhead, LB, and Throughput
[20]	IP	K-Median	✓	X	Haversine	✓	X	X	Random centre Initialisation stage, the number of cluster determinations,	
[10]	Mathematical Model	K-means with Game Theory	✓	X	Euclidean Distance	✓	X	X		
PHCPA [1]	AI	MRFO with Salp Swarm	✓	X	Cosine Haversine	✓	X	X	Lack of Sufficient Training Dataset	Increased PPT, control message Overhead
PITS [28]	Graph Theory,	DFS	--	---	----	✓	X	X		
GravCPA[38]	LP	Louvain algorithms	X	Node Traffic	Euclidean	✓	X	X	LPA and gravitation are vulnerable to oscillations and non-unique results	
ECP [39]	MILP	Linearization & Supermodular	X	X	----	✓	✓	X	The CP overhead will likely resurface due to not partitioning the network into smaller clusters.	
[40]	Greedy	None	X	X	X	✓	✓	X	Network properties not considered	No controller placement module

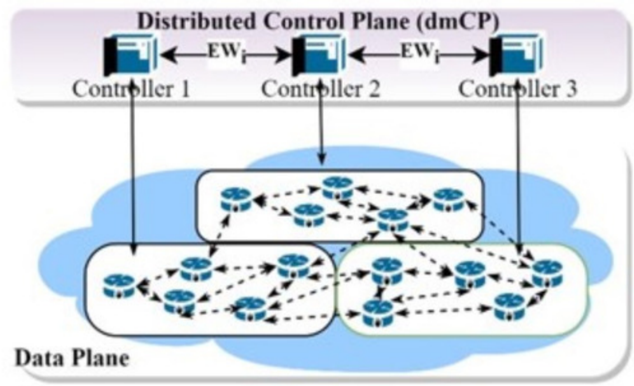
Figure 1

Control Plane Architecture

Single Control Plane Architecture(a) and Multiple Controllers (b)



Single controller (a)



Multiple Controllers (b)

Figure 2

CPCSA Flow Chart

CPCSA Flow Chart

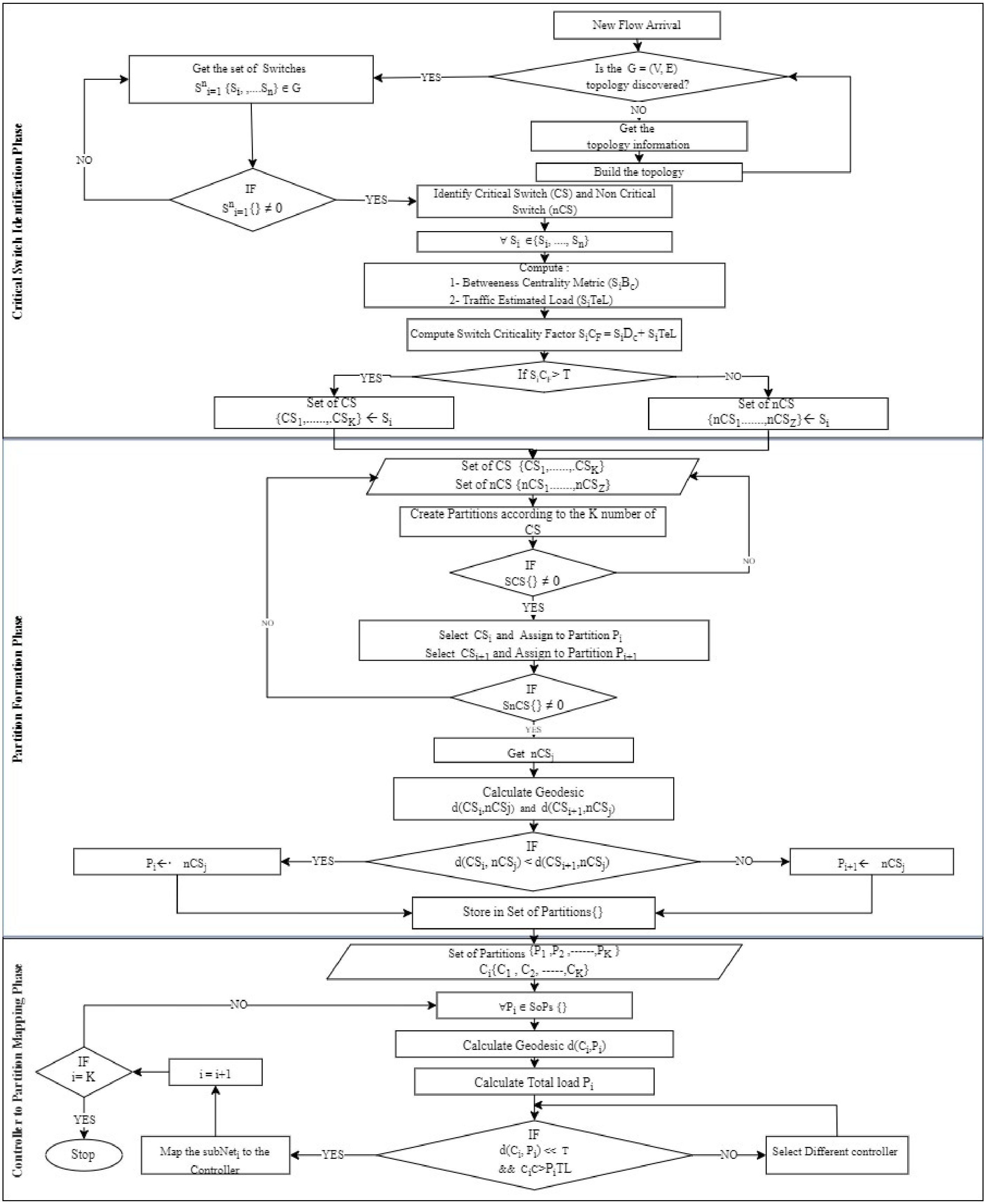

Critical Switch Identification Phase
New Flow Arrival
Is the G = (V, E) topology discovered?
YES
NO
Get the set of Switches
Get the topology information
Build the topology
IF
YES
Identify Critical Switch (CS) and Non Critical Switch (nCS)
Compute :
1- Betweeness Centrality Metric
2- Traffic Estimated Load
Compute Switch Criticality Factor
Set of CS
Set of nCS
Partition Formation Phase
Create Partitions according to the K number of CS
Get nCS
Calculate Geodesic
Store in Set of Partitions{}
Controller to Partition Mapping Phase
Set of Partitions
Calculate Total load
Select Different controller
Map the subNet to the Controller
i = i+1
Stop

Figure 3

Overhead

Effect of flows installation cost on the Overhead on the number of controllers

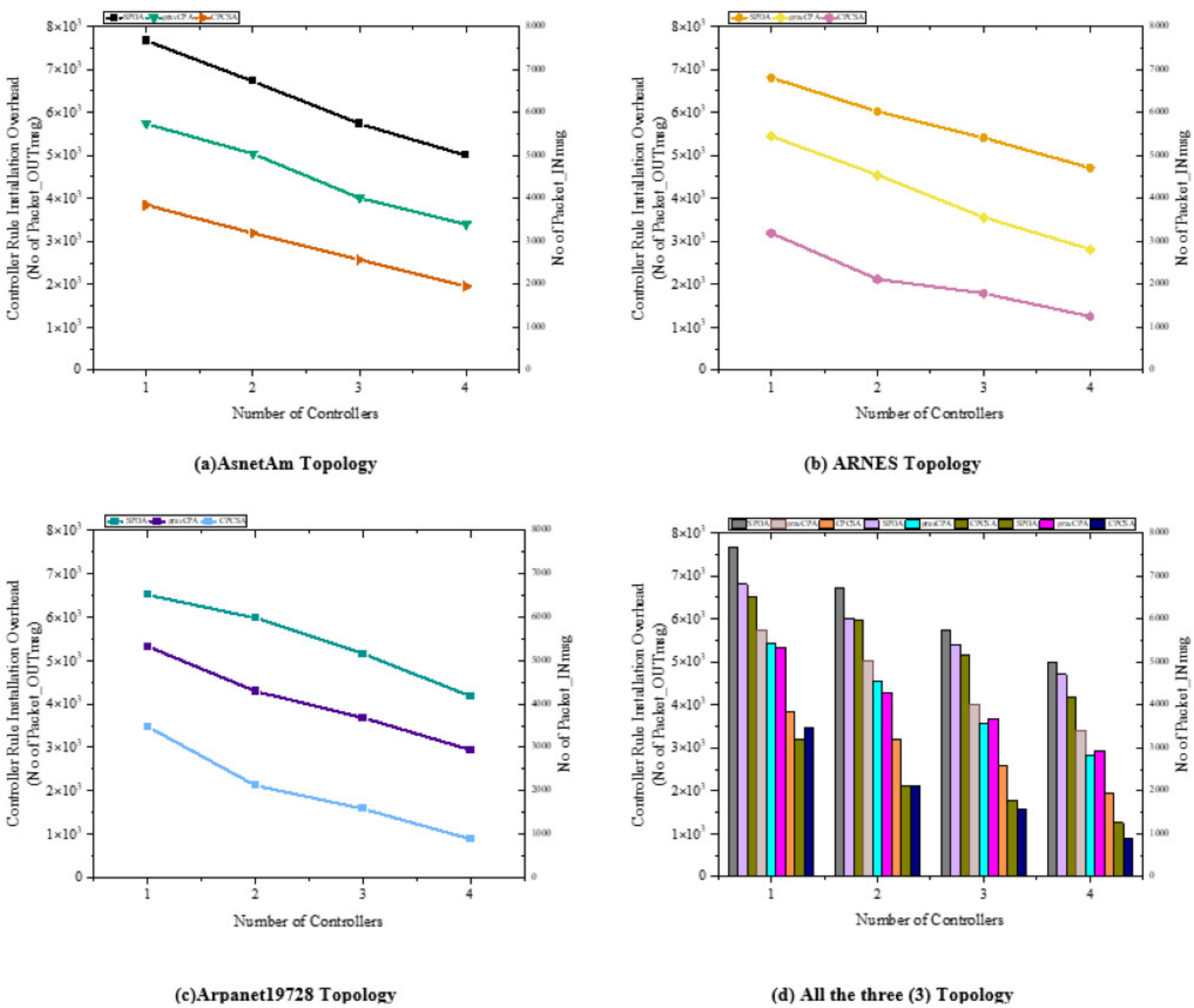


Figure 4

Figure 3 a-d

Arpanet topology

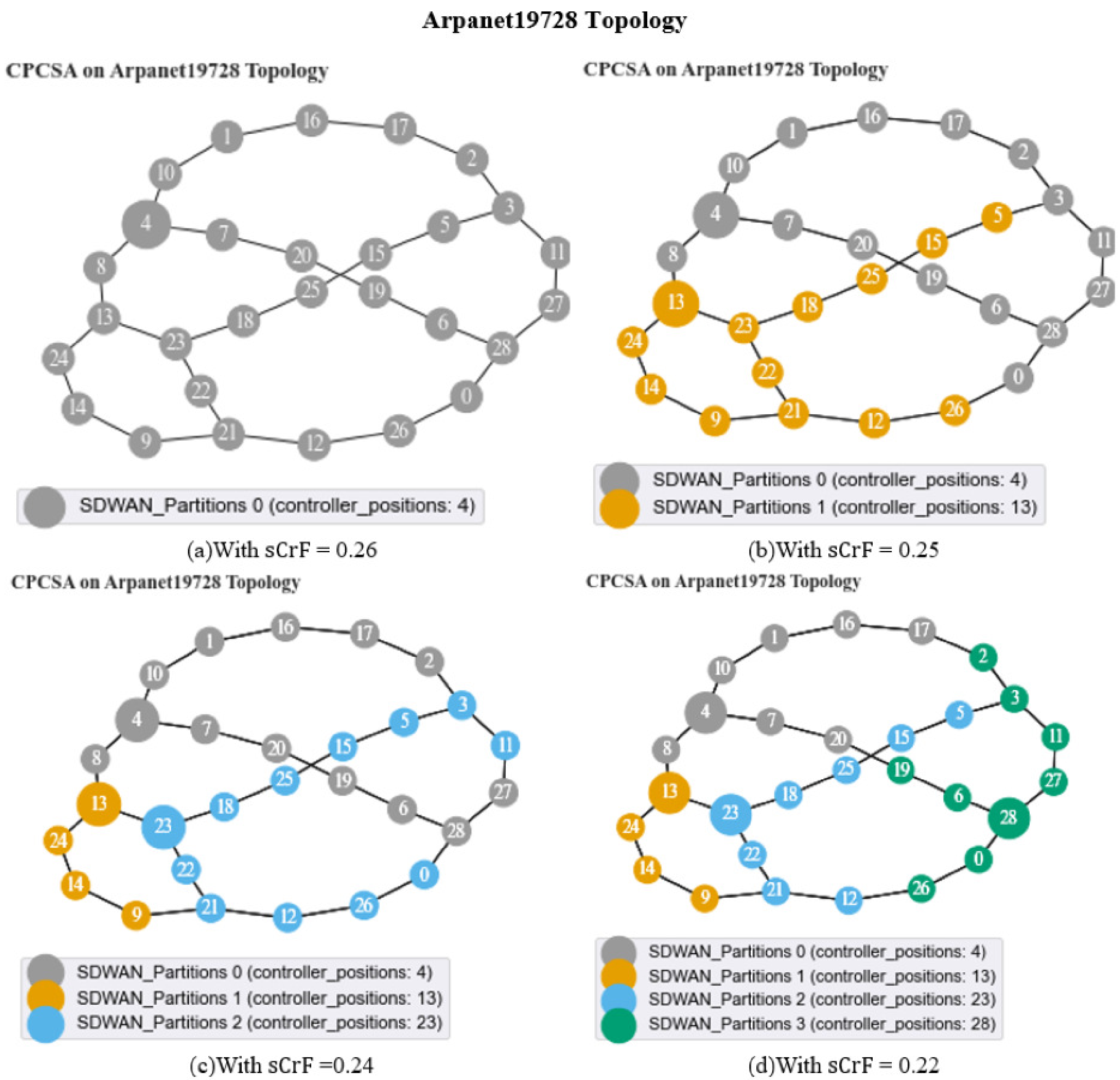


Figure 5

Packet loss result

Comparison of Packet loss

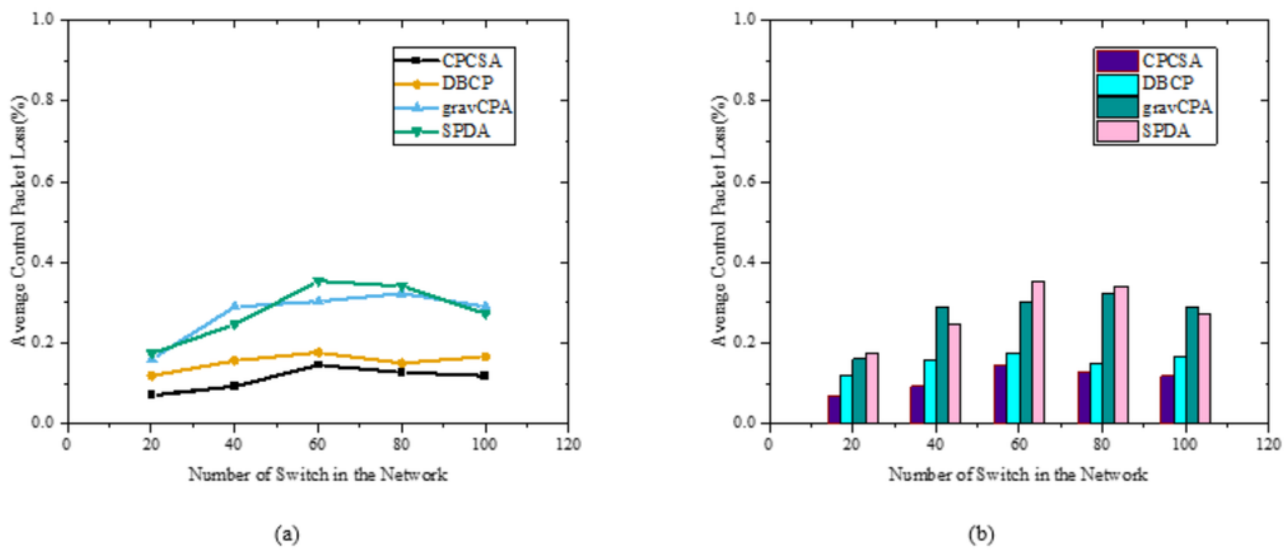
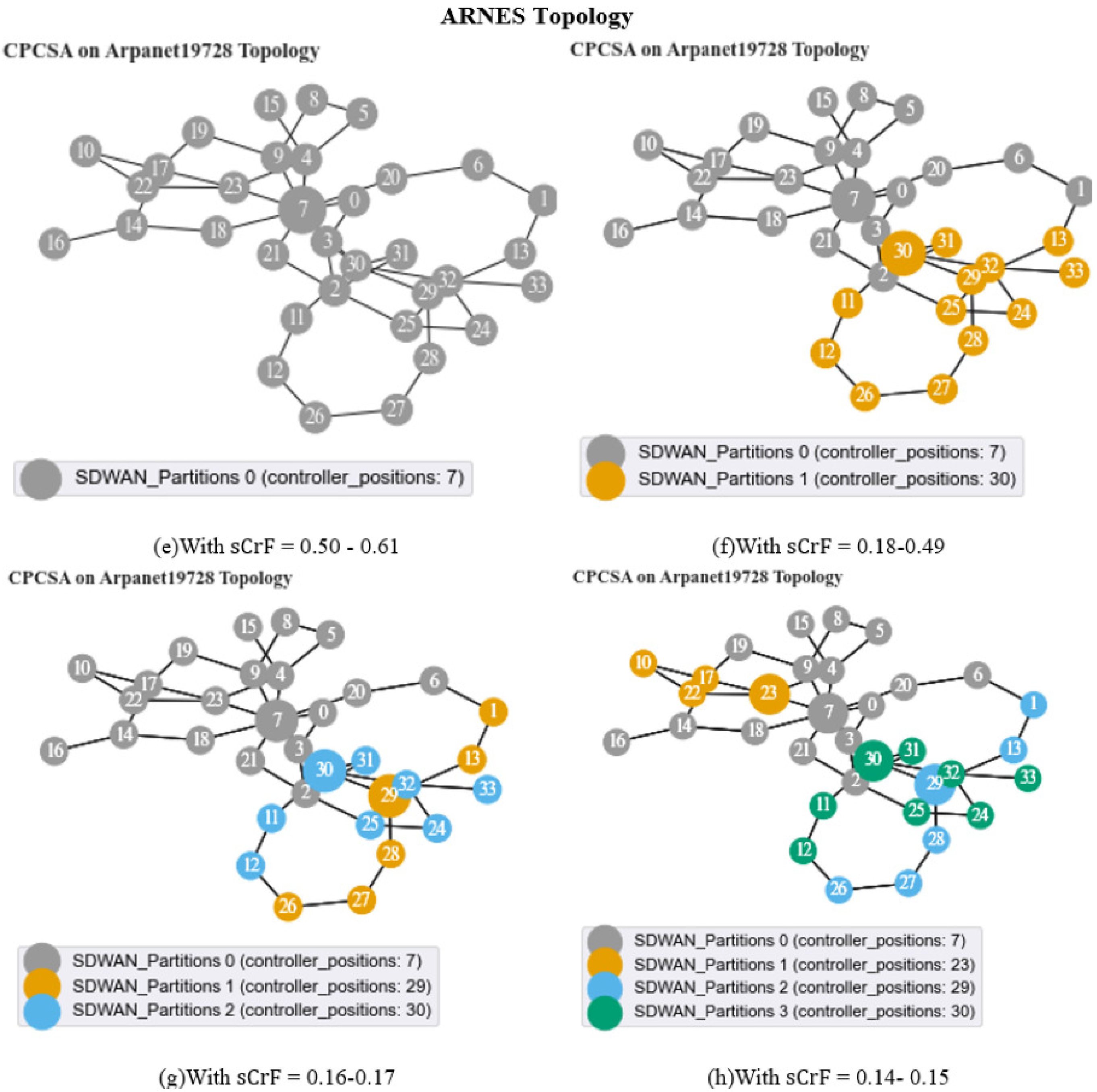


Figure 6

Figure 3, e-h,

ARNES topology



(e)With sCrF = 0.50 - 0.61

(f)With sCrF = 0.18-0.49

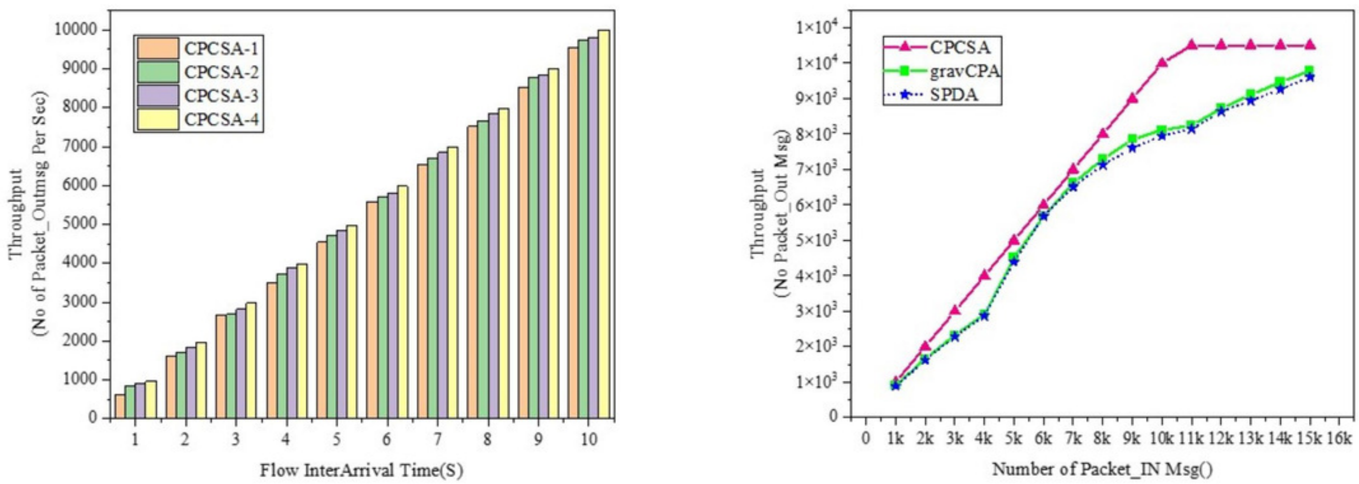
(g)With sCrF = 0.16-0.17

(h)With sCrF = 0.14- 0.15

Figure 7

Throughput

Comparison of throughput



(a) (b)

Figure 8

Figure I-L

AsnetAM topology

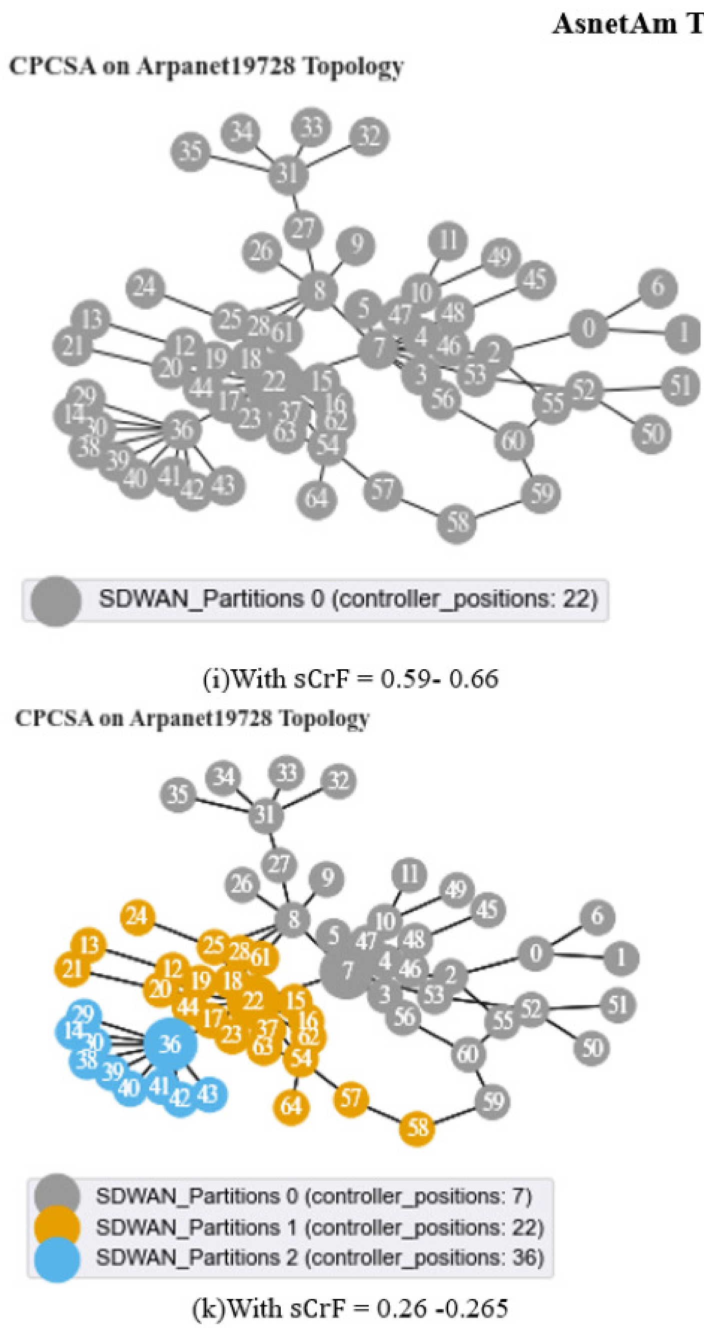


Figure 9

Latency

Relationship between switch to controller latency

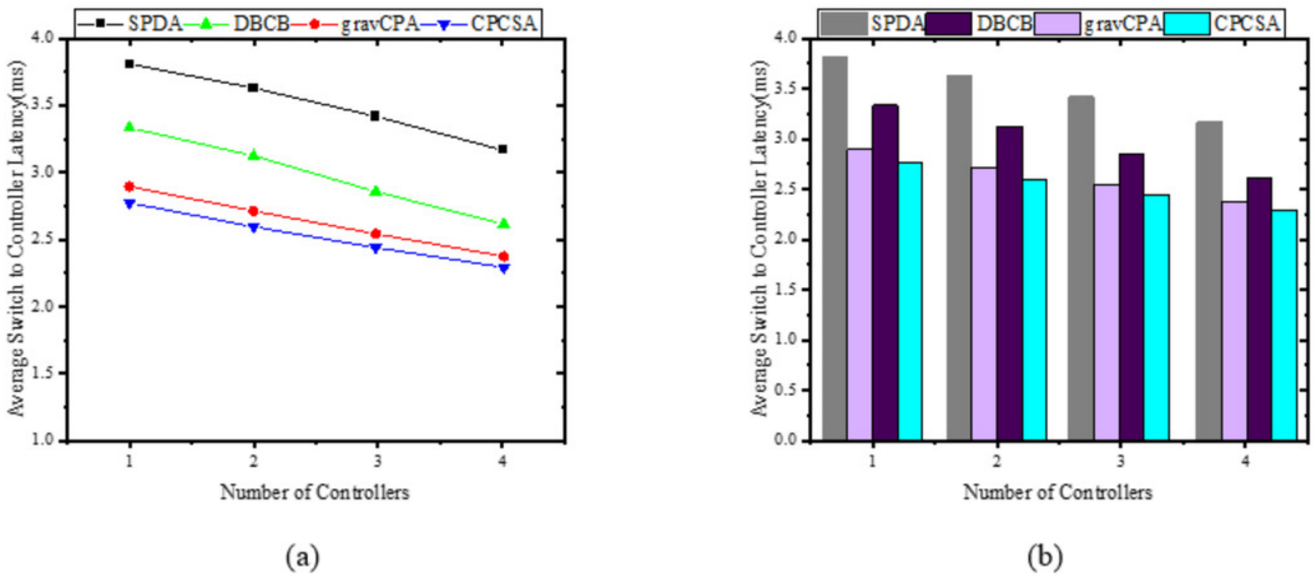


Table 2(on next page)

Table 2

Notations and Symbols

Table 1: Notations and Symbols

Notation	Description
G	SDWAN
E	Set of communication links in the Network
V	Set of network nodes (comparison of both controllers and switches)
C	Set of SDN controllers
$CPr_{overhead}$	Controller overhead
S	Set of OpenFlow switches
$SPr_{overhead}$	Switch overhead on the controller
CS	critical switches
nCS	non-critical switches
SCS	Set of critical switches
SnCS	Set of non-critical switches
$SDWAN\_Partitions_i$	Sub-net of OpenFlow Switches
$dist(s_i c_j)$	Shortest distance between the controller c_j and switch s_i in $Sdomain$
k	An integer representing the number of CS, SDWAN_Partitions, and C
nF_i	New flow
(nF_{S_i,S_j})	Number of flow between source and destination
X_{im}	$\{0,1\}$ binary variables indicating whether the switch s_i is under the control of the controller C_m

Table 3 (on next page)

Table 3

Topologies Information and Traffic Information

Table 1: Topologies Information and Traffic Information

Topologies Information					Traffic Information		
Topology	Number Switches	Number of Links	Density	Ave SBF	New Flow	Packet_IN msg size	Packet_OUT msg size
Arpanet19728	29	32	0.0788	**0.136**	For every 100 000Kb	80 bytes	80 bytes
ARNES	34	47	0.0837	**0.076**			
AsnetAm	65	79	0.0380	**0.044**			