This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2999072, IEEE Transactions on Vehicular Technology

PARC: Placement Availability Resilient Controller Scheme for Software-Defined Datacenters

Rajat Chaudhary, *Student Member, IEEE* and Neeraj Kumar, *Senior Member, IEEE*

***Abstract*—Software-Defined Datacenters (SDDC) have been widely used for load-aware data management for different applications across the globe. Due to its centralized architecture, the issues of scalability along with resilience (to overcome the failure of single or multiple controllers) are still challenging because of an exponential increase in the data generated from different smart devices. Most of the solutions reported in the literature for this problem use a single controller which may not address the scalability issues. However, the issues as mentioned above of scalability and resilience in SDDC can be solved by deploying multiple distributed controllers at the control plane. However, the primary concern in a network having various controllers is the optimal Controller Placement Problem (CPP) to resolve the issues of fault-tolerance, latency among controllers, availability, and placement. Hence, to resolve the issues described above, in this paper, we propose Placement Availability Resilient Controller *(PARC)* scheme. The PARC scheme works in the following four phases: (i) stable network partitioning (ii) localization of controllers using the cooperative game theory (iii) computation of an optimal number of multiple controllers and (iv) computation of minimal extra backup controllers to improve the overall network cost. The numerical results of the PARC scheme are evaluated on Internet2 OS3E topology using POCO-toolset simulated in Matlab. The experimental results demonstrated that the cost of deploying the number of controllers using the PARC scheme has reduced to 12.98%, 8.16%, and 6.25% as compared to the POCO-SA, POCO-MOALO, and CNCP schemes respectively. Moreover, the PARC scheme outperforms the existing state-of-the-art schemes (POCO-SA, POCO-MOALO, and CNCP) for inter-controller as well as switch-to-controller latency.**

***Index Terms*—Software-Defined Networking (SDN), Network Partitioning, Controller Placement Problem (CPP), Resilience.**

TABLE I: NOMENCLATURE

Notation	*Description*
$G(V, E)$	connected graph with $V = (X \cup Y)$ as a set of nodes and E as a set of edges between nodes.
X, Y	set of switches and controllers in the network.
N, M	number of switches and controllers, where $(N > M)$.
$A_{i,j}$	matrix of shortest path for each pair of switches x_i, x_j.
$A_{\max}$	maximum allowed latency between any switches pair (i, j).
a_{ij}	propagation latency from switch i to j.
θ_i	processing capacity of each controller i.
$\lambda_j(t)$	flow arrival rate on switch j at time t.
$g : (N, U)$	cooperative or convex game.
N	set of players in the game or switches in the network.
$U : 2^N \rightarrow \mathbb{R}$	utility function.
$\gamma[a_{ij}]$	similarity function.
$P = \{P_1, ..P_L\}$	set of partition.
$\|P\|$	total number of partition of N finite players.
$F = \{f_1, ..f_n\}$	set of payoff allocation of individual switch.
$\psi_i(N, U)$	shapley value of a player i.
$U(P \cup \{i\}) - U(P)$	marginal contribution of a player i.
ζ	set of all possible permutations of N players.
$\delta \in \zeta$	position of player i in the ordering.
f^{δ}	payoff vector, where $\delta(i)$ is the position of player in the ordering.
$R_{i,j}$	reliability factor to optimize the shortest distance between nodes so that each switch is mapped to its nearest controller.
α	number of controllers associated with any switch.
β	number of backup controllers (planning ahead parameter for the master controller failure).
$(\alpha - \beta)$	number of master controllers which are actively processing the switch requests.
C	set of controllers directly linked to the core switches.
C_f	number of failed controllers.
C_k	load on each k^{th} controller.
$\overline{C} = (C_i - W_{ij})$	set of extra backup controllers deployed in the network.
Z	record of the switches deployed to a single controller.
$Z^{\star}$	record of the switches assigned to two controllers.
h	threshold on the hop count.
$a_{x_i x_r}$	shortest distance between every switch pairs controlled by the same controller.
$Y_{initial}, Y_{final}$	initial and final position of controllers.

I. Introduction

FROM the last few years, there has been an exponential increase in the Software-Defined Datacenters (SDDC) across the globe. SDDC provides flexibility for the data storage as compared to the traditional datacenters. They have three major components, i.e., storage, computation, and networking [1]. The integration of Network Functions Virtualization (NFV) along with Software-Defined Networking (SDN) leads to the popularity of SDDC. The hardware infrastructure equipment, i.e., storage, network, and the server, can be virtualized to deliver Infrastructure-as-a-Service (IaaS) which is the building block of SDDC [2]. It improves elasticity and agility within the traditional IaaS datacenter because the virtualized datacenters are automated and provisioned by the intelligent policy-based software [3]. SDN is widely implemented in the datacenters to manage the workloads in cloud automation and cloud operations with a unified data management software [4]. According to the Allied Research report [5], the market of the SDDC is expected to grow annually by 32% on an average and is expected to reach worth $139 billion by 2022.

The limitations of the traditional datacenters are that they provide limited services to the end-users in terms of bandwidth and storage facility, high latency, manual devices configuration, and traffic congestion [6]. SDDC is network programmable to provide intelligence analytics for infrastructure and workload provisioning [7]. Thus, it efficiently addresses all the issues of traditional datacenters. Due to their centralized architecture, the issue of scalability alongwith resilience (to overcome the single point of failure) are still major concerns,

R. Chaudhary is with the Department of Computer Science and Engineering, Thapar Institute of Engineering and Technology, Patiala (Punjab), India, N. Kumar is with the Department of Computer Science and Engineering, Thapar Institute of Engineering and Technology, Patiala, India, Department of Computer Science and Information Engineering, Asia University, Taiwan and King Abdul Aziz University, Jeddah, Saudi Arabia. (e-mail: rchaudhary_phd16@thapar.edu, rajatlibran@gmail.com and neeraj.kumar@thapar.edu).

which need new solutions from the research communities [8].

The issues mentioned above of scalability and resilience in SDDC can be addressed by deploying multiple controllers at the control plane such that the data plane works smoothly in case of failure of a single network controller. It can be used to achieve a high-performance gain in large-scale networks. It extends the control plane functionality by improving scalability, reliability, efficiency, latency, redundancy, and data consistency [9]. However, the roles of a controller are as follows (i) to define and install the flow rules proactively on the data plane (ii) to perform traffic engineering by preventing network failure based on periodic traffic monitoring and flow management (iii) to virtualize the network functionalities such as multiple virtual controllers, load balancers, virtual switches, and gateways, and (iv) to monitor end-to-end data traffic through network links, and feedback collection from the forwarding plane. In addition, the responsibilities of the OpenFlow switches are as follows (i) to store flow tables in ternary content addressable memory (TCAM) (ii) efficiently handle flow table entries in order to perform routing of the incoming requests to the final destination (iii) follow forwarding decision installed by the controller and send notification back to the controller, and (iv) encapsulation and tunneling of the packets [10], [11].

Though the distributed controllers deployment is the most suitable technique to enhance the efficiency of a system, it may have challenges for its design principles. For example, one of the biggest challenges is the Controller Placement Problem (CPP), which results in the generation of the issues of fault-tolerance, latency among controllers, availability and their placement [12], [13]. The CPP deals with three important issues (i) *volume* (the number of distributed controllers required in the network topology) (ii) *placement* [14], and (iii) *resilience* (backup controllers). Hence, latency and cost are the two parameters which can affect the network performance. Since there are few limited latency-sensitive applications so deploying too many controllers may cause an increase in the overall network cost [15], [16]. In this paper, we propose a near optimal solution for the CPP for SDDC.

The main contributions of this paper are as follows.

- To resolve the issues of CPP, a problem is formulated in the form of Mixed Integer Non-Linear Programming (MINLP). Then, the PARC scheme is proposed which divides the CPP into different sub-problems, i.e., network partitioning, the controller's availability, the minimal number of primary active controllers, and the minimal number of backup controllers required for resilience. The proposed scheme uses cooperative game theory to compute a stable network partitioning, and an optimal position of the distributed controllers in each sub-network.
- The overall network cost of the proposed scheme is minimized by finding the minimal number of distributed controllers along with the extra backup controllers using a heuristic approach.
- Moreover, the *PARC* scheme performs global data consistency of the updated events among multiple controllers based on timestamp at the control plane, which provides synchronization among different events.
- The Pareto Optimal COntroller (POCO) toolset is used to simulate the numerical results in Matlab. The performance analysis compares the proposed PARC scheme with the other existing state-of-the-art schemes such as Pareto Optimal COntroller based on the simulated annealing (POCO-PSA), Pareto Optimal COntroller Multi-Objective Ant Lion Optimization (POCO-MOALO), and Capacitated Next Controller Placement (CNCP) scheme.

Rest of the paper is structured as follows. Section II discusses the related work followed by the problem formulation in Section III. Section IV presents the proposed PARC scheme. The performance evaluation is discussed in Section V. Section VI concludes the paper.

II. Related Work

To solve the above-discussed issues of CPP, several proposals exist in the literature. For example in [17], a framework named Pareto Optimal COntroller (POCO) has been presented by the authors for the CPP. The POCO framework covers four aspects of CPP (i) inter-controller latency (ii) controller failure (iii) network disruption (controller-less nodes) and (iv) load imbalance. Also, Lange *et al.* [18] designed a Pareto optimal solution based on the simulated annealing (POCO-SA). Similarly, in [19], a POCO Multi-Objective Ant Lion Optimization (POCO-MOALO) algorithm was designed as a Pareto-Optimal solution to select the optimal position for the controller placement. The proposed algorithm uses the concept of load capacity factor and propagation latency simulated on POCO toolset for the Internet2 OS3E topology. Ksentini *et al.* [20] proposed a bargaining game for CPP. Similarly, in [21], the authors proposed the distributed approach based on the non-zero sum game theory for CPP. The proposed approach initially computes the payoff on each controller and then takes the decision about insertion, deletion, or load migration among the controllers by comparing the payoff with the adjacent controllers. Killi *et al.* [22] adopted a Capacitated Next Controller Placement (CNCP) scheme to minimize the average latency between the switch to it's closest reference controller and between two neighboring reference controllers with sufficient residual capacity. It ensures the sustainability of the network in case of any node or link failure in the network.

Tanha *et al.* [23] used the approximation algorithm for CPP to prolong the network reliability and to mitigate the issues of any node or link failure in the network. Moreover, in [24], the authors have extended their research work to address the issues of latency and a capacity-aware guarantee of controllers. The proposed scheme uses a clique-based technique in the network topology to ensure the recovery of both the node and the link failure. Zhang *et al.* [25] formulated the CPP by using a multi-objective function and proposed an optimization approach named as adaptive bacterial foraging to have the network reliability. In [26], authors proposed *Compare* and *Drop* algorithms to compute the average and worst-case latency. The experimental results have been tested on Internet topology Zoo to evaluate the impact on the number of dropped controllers.

In [27], the authors designed a primary-backup controller scheme to ensure fault tolerance. The results illustrated that the proposed scheme minimizes the number of controllers up

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2999072, IEEE Transactions on Vehicular Technology

to 50%. Moreover, the controller assignment problem is addressed by using a genetic algorithm and analytic hierarchy approach in [28]. Similarly, Li *et al.* [29] proposed a quality-of-service (QoS) aware resource scheduling scheme by integrating the SDN approach and instance placement optimization algorithm for virtual networks. The proposed scheme designed a Pareto-optimal solution based on non-cooperative game theory to improve the service-level agreement. In addition to the aforementioned issues in CPP, data inconsistency is still one of the crucial issues using the multiple controllers. Data consistency means synchronization of the messages based on the ordering of events among multiple controllers. In [30], the authors proposed a consensus mechanism for CPP to perform coordination of events among different controllers, each having its own database. The proposed scheme uses a leader-follower approach on the control plane to maintain global data consistency.

Motivated from the above proposals, the PARC scheme is designed in this paper to address the issues of scalability, resilience, and consistency on the control plane. The existing proposals worked on the issues of CPP in large-scale networks. However, most of the existing proposals resulted in finding the least number of controllers required and their placement with minimum focus on maximizing the fault-tolerance on the control plane. In addition, none of the authors has addressed the issue of synchronization of messages among the distributed controllers with scalability and resilience. In contrast to the existing proposals, the proposed PARC scheme mainly resolves the following issues (i) computation of the minimal number of primarily distributed controllers (ii) computation of the minimal number of extra backup controllers (iii) localization of controllers, and (iv) synchronization of events based on timestamp among the distributed controllers in order to maintain global data consistency at the control plane.

III. Problem Formulation

It comprises of SDN architecture, network model, and problem statement elaborated as follows.

A. SDN Architecture

The model of the distributed SDN controllers is presented in Fig. 1 which comprises of three planes, namely, data, control, and application discussed as follows.

Data Plane: It consists of OpenFlow-enabled switches. The requests are sent as *packet-in messages* by the hosts as they are directly connected to switches such that the control plane installs the forwarding path in the form of *flow-mod messages* in the switches. The incoming *packet-in messages* arise due to new flow arrivals, link failure, or logical changes in the network topology. The edge switches follow Top-of-Rack (ToR) switch architecture, i.e., the switches are installed in racks such that every switch is directly connected to the topmost switch of each rack via a link [31]. Such ToR switches are also directly connected to the aggregate switch which in turn carries its aggregate flow. The benefits of ToR switch design are the minimum cost, reduces cabling complexity, and high data rate (the network is upgraded to 10-40 Gigabit Ethernet). Suppose, q be the total ports available at each ToR switch which helps to connect with other switches, then each ToR switch uses only p out of the available q ports to interconnect multiple switches such that $0 < p < q$. The aggregate switches are supervised by the core switches, and finally, the core switches are connected in a Peer-to-Peer (P2P) manner with the distributed controllers.

Control Plane: The multiple controllers are physically distributed but are logically centralized, wherein, each controller maintains its own database to keep the updated global knowledge. The multiple controllers at the control plane install the flow tables in proactive mode on switches and process the switch requests in case of mismatch flow entries. The controller generates the flow table consisting of flow ID, matching rule, and action performed, which is installed on the data plane. The forwarding information shared among the distributed controllers does not follow the polling (periodic updates) method, rather it uses the subscribe and publish method, which updates the network information only if there are any changes in the forwarding path in switches. The ordering of updated notification among distributed controllers is synchronized by using the timestamp. The timestamp consists of three tuples $\langle switchID, termnumber, eventnumber \rangle$. Here, *switch ID* is a unique identifier of OpenFlow switch, whereas, a *term number* is defined as an increment in the counter if the primary controller fails, and the *event number* is the counter for any new event or update. The horizontal model follows the leaderless approach, unlike the master-slave approach of the hierarchical model. The benefits of the flat model are fault-tolerance (easy to handle fail-over mechanism by avoiding a single point of failure), replication (multiple copies of the global database), and reduced inter-controller latency.

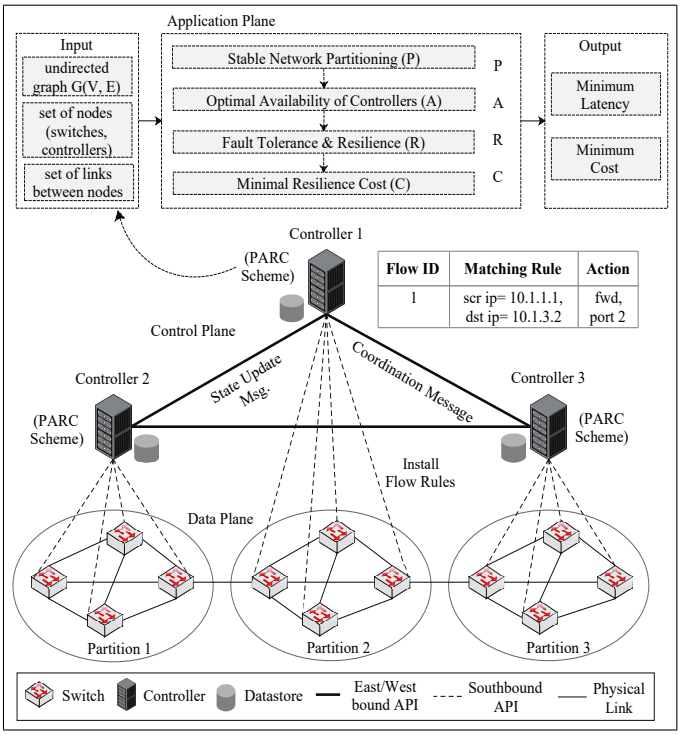

Fig. 1: Horizontal view of the distributed SDN controllers.

Application Plane: The SDN architecture is network programmable, i.e., the programmers run multiple applications and are deployed on the control plane. A software programmer resides at the remote location to provide services to the end-users at the data plane via the control plane. Moreover, the logic of the multiple SDN applications (fault tolerance, network virtualization, and load balancers) runs remotely at the application plane whose instructions are executed on the controller through a northbound interfaces. An instance migration is also an application of SDN-enabled datacenter [32]. The application of network virtualization splits a physical network into sub-networks in order to deploy atleast one distributed controller in each virtual network. Finally, the application plane ensures that the network is fully resilient with minimum network cost and delay.

B. Network Model

As discussed in the related work, the issues of multiple CPP are resolved by dividing the problem into four sub-problems. Hence, let us consider a physical network topology $G(V, E)$, where, V represents a set of nodes, and E represents a set of edges between nodes. Let $X = \{x_1, x_2, .., x_N\}$ be the set of switches, and $Y = \{y_1, y_2, .., y_M\}$ be the set of controllers in the network. Assume that, $N > M$ and $V = (X \cup Y)$ for all the nodes in G. Let A_{ij} be a matrix of shortest path distance for each pair of switches x_i and x_j. Let us assume that each distributed controller has the same processing capacity and is denoted by θ_i. Also, the flow arrival rate on switch j at time interval t is denoted by $\lambda_j(t)$.

C. Problem Statement

Fig. 1 represents an overview of the SDN architecture with data, control, and application planes. The distributed controllers are P2P linked to each other through an eastbound and westbound interface. The application plane takes the input as $G(V, E)$ to solve the four sub-problems by designing a Pareto near optimal solution for the control and data planes which returns the output with minimum latency and minimum cost. Fig. 2(a) represents an insight of the first step of the application plane i.e., network partitioning. The physical network is splitted into virtual sub-networks such that atleast one controller is mapped to supervise all the switches in each sub-network. Fig. 2(b) depicts the issue of optimal controller placement such that the minimum latency is incurred from controller-to-the-switch and inter-controller latency. The output returned is the stable network partitions. Fig. 2(c) presents the third and the fourth issue of the application plane that takes care of the fault-tolerance in case of controller or link failure. The objective is to minimize the network cost by deploying minimum number of primary and backup controllers. The detailed explanation of each sub-problem is as follows.

*1) **Network Partitioning in SDDC**:* The first issue is a stable network partitioning in SDDC so that multiple controllers can be deployed in the entire network [33]. Fig. 2(a) depicts the splitting of the large-scale network into different sub-networks in Internet2 OS3E topology. The problem of network partitioning is formulated as a cooperative game theory using a coalition formation game. The final output is to minimize switch-to-controller and inter-controller latency.

Definition 1. *Cooperative Game–* A cooperative game is denoted as a pair g:(N, U), where, N denotes the set of players in the game or switches in the network and U: $2^N \rightarrow \mathbb{R}$ is a *utility function* from the total possible number of coalitions to the overall payoff that satisfies $U(\emptyset) = 0$. A cooperative game is defined in which all the players cooperate to work together to form a coalition so that the total payoff obtained is equally distributed among all the players [34]. The coalition formation game is defined as a set of partition denoted as $P = \{P_1, P_2, ., P_L\}$ with L partitions of N finite players (switches) such that it satisfies the following conditions:
(i) $\forall l \in \{1, 2, .., L\}, P_l \neq \emptyset$, i.e., the coalitions are not empty.
(ii) $\bigcup_{l=1}^{L} P_l = N$, i.e., the union of all sub-networks should be equal to the entire network.
(iii) $P_i \cap P_j = \emptyset$, where, $i \neq j, (i, j) \in \{1, .., L\}$, i.e., every partition has distinct players. Here, $i\&j$ represents the switches.

Definition 2. *Core of the Game (Player's Payoff)–* Payoff is the profit or loss received by any player $i \in X$ in terms of network cost and latency from the outcome of the game generated by dividing the obtained $U(P)$. The core of the game (N, U) is defined as a set of stable payoff allocation to players such that each player has equal gain so as to stay in the coalition [35]. Let $F = \{f_1, f_2, .., f_n\}$ be the set of payoff allocation of a individual switch. The main criteria in a convex game are how to divide a payoff among the N players such that the stability of the game is achieved. The payoff allocation f is said to be optimal if $\sum_{i=1}^{N} f_i = U(N)$. The stable state of a player is the optimal reaction which maximizes their payoff to other player's stable strategies. The payoff condition is satisfied if any player in a group, disjoints the coalition in order to join another coalition then the payoff achieved is not greater than the sum of the payoff originally received from the coalition. The payoff allocation is said to be stable with respect to the coalition and is defined as $f : \sum_{i \in P} f_i \geq U(P)$. The core satisfies the two properties of payoff allocation:
(i) *Group Rational*: $\{f : \sum_{i=1}^{N} f_i = U(N)\}$.
(ii) *Coalitional Rational*: $\{f : \sum_{i \in P} f_i \geq U(P), \forall P \subseteq N\}$.

Definition 3. *Convex Game–* In a convex game (N, U), bigger coalitions have a large value of player's marginal contribution as compared to its value in smaller coalitions which can be mathematically defined with respect to P_1 and P_2 as follows.

$$U(P_1 \cup \{i\}) - U(P_1) \geq U(P_2 \cup \{i\}) - U(P_2), \forall P_2 \subseteq P_1 \subseteq N, i \in N. \quad (1)$$

Definition 4. *Shapley Value of a convex Game–* It is defined as a solution in which the total value of the gain is split fairly among the players as per their individual contributions in the game [36]. The Shapley value of a player i denoted as $\psi_i(N, U)$ is obtained by using the formula defined as below.

$$\psi_i(N, U) = \sum_{P \subseteq N \setminus \{i\}} [U(P \cup \{i\}) - U(P)] \frac{|P|!(n - |P| - 1)!}{n!}, \quad (2)$$

where $U(P \cup \{i\}) - U(P)$ denotes the marginal contribution of a player i, $|P|!$ is the number of possibilities of positioning

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2999072, IEEE Transactions on Vehicular Technology

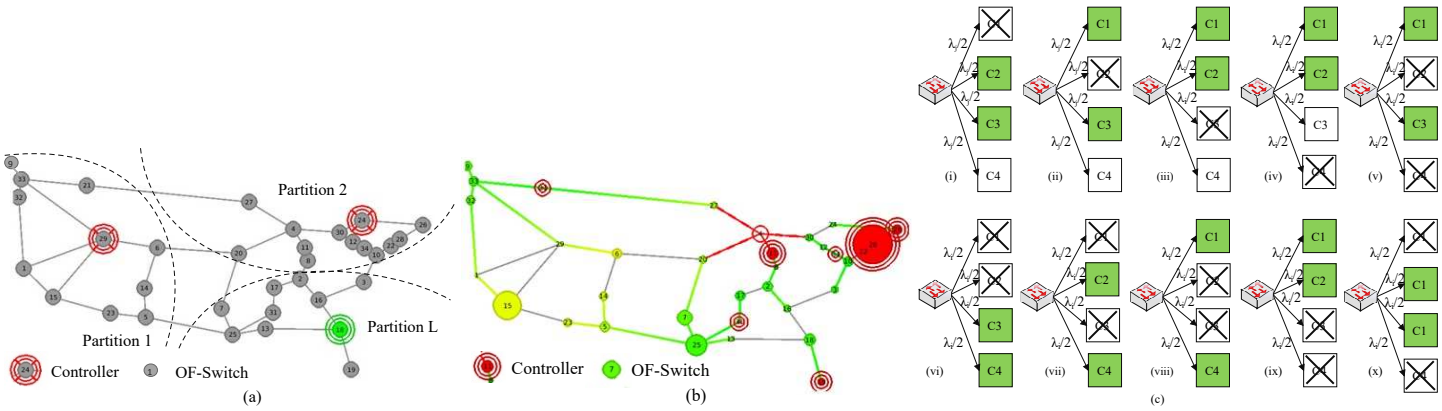

Fig. 2: (a) Network partitioning in Internet2 OS3E (b) Optimal controller placement (c) Resilience in case of controller-less nodes

the players in the beginning of position, and $(n-|P|-1)!$ is the possible positioning of players other than player i at the termination of positioning. In a convex game for a player i, the Shapley value is obtained as below.

$$\psi_i(N,U) = \frac{1}{|N|!}\sum_{\delta\in\zeta}(f_i^{\delta}) = \frac{1}{|N|!}\sum_{\delta\in\zeta}U(S_{\delta,\delta(i)}) - U(S_{\delta,\delta(i)-1}), \tag{3}$$

where ζ is the set of all possible permutations of N players and $\delta \in \zeta$ be any individual ordering, f^{δ} is a payoff vector in which $\delta(i)$ is the position of player i in the ordering, and $S_{\delta,b}$ is the starting component of ordering δ defined as $S_{\delta,b} = \{i \in N : \delta(i) \leq b\}, b \in N$.

In this problem, the objective is to split G into sub-graphs such that G is a connected sub-graphs, i.e., $G = \{G_1(V_1, E_1) \cup, .., G_n(V_n, E_n)\}$ to generate a stable network partitioning.

*2) **Optimal Availability of Distributed Controllers**:* The cooperative game theory discussed above considers the graph $G(V,E)$ that takes the set $X = \{x_1, x_2, .., x_N\}$ of switches as the input and returns the output as the initial number of controllers to be assigned in each sub-network. Fig. 2(b) illustrates the issue of the controller positioning as the switches which are comparatively far from the controller in each sub-network increase the overall latency. Initially, it is considered that each sub-network has been assigned with atleast one controller. The set of availability is used to decide the most preferred locations of each controller such that minimum latency is achieved. The optimal placement of controllers is decided on the basis of their shortest distance to the switches such that in every partition, each switch is controlled by its nearest controller. Hence, assuming the shortest distance between controller-to-controller and switch-to-controller for the set of vertices i and j be denoted by a_{ij}. The reliability factor is defined to optimize the shortest distance by using the formula defined as follows.

$$R_{i,j} = \prod_{e\in E}\omega_e.d_{e,a_{i,j}}.\prod_{v\in V}\omega_v.d_{v,a_{i,j}}, \tag{4}$$

where $d_{e,a_{i,j}} = 1$, if edge e is used for the shortest distance between given vertices i and j, otherwise 0 and $d_{v,a_{i,j}} = 1$, if vertex v follows the shortest distance between vertices i and j, otherwise 0. Here, ω_e and ω_v are the reliability of edge e and nodes v. The final output of CPP returns the final optimal location of controllers in every sub-network.

*3) **Resilience in Controllerless Nodes**:* Resilience refers to the capability of the system to maintain the quality of service in case of failure of the controllers. The network performance mainly depends upon two parameters (i) the minimal number of primary active controllers with maximum coverage so as to reduce the cost associated with each controller to supervise the network, and (ii) the minimal number of backup controllers required to improve the network resilience in case the primary controller fails. So, it is required to determine the minimum number of core switches required in the network topology. Core switches are defined as those switches in the graph $G(V,E)$ which are directly linked with the controller using a single hop count while the remaining switches are considered as normal switches. Let us assume that the number of controllers which should be associated with any switch is α and let β be the number of controller failures that the system can handle with full resilience, i.e., the allowed number of failures from the assigned controllers to a switch. Therefore, $(\alpha-\beta)$ is the number of master controllers that are actively processing the switch requests, and β is the number of backup controllers in case the master controller fails.

Fig. 2(c) represents an example of resilience in the case of controller-less nodes by considering $\alpha = 4$ as the number of controllers associated with a switch and $\beta = 2$ as the allowed number of backup controllers assigned to a switch to handle full resilience. Fig. 2(c) (i-iv) represents all the possible options of a single controller failure and Fig. 2(c)(v-x) represents the scenario of two controllers failure. Clearly, the system has the first $(\alpha-\beta)$ as the master controllers depicted by the green color while the rest β is the backup controller depicted by the white color. The inflow requests are served by the master controllers and for the flow arrival rate, λ_j of any switch j, each of associated α controllers stores a capacity of $\lambda_j/(\alpha-\beta)$ to meet the demands of the switch j. Thus, the starting $(\alpha-\beta)$ controllers are considered as the first working controllers serving the packet requests of a switch in case of no failure in the network (as mentioned in Fig. 2(c)). Initially, the average traffic flows on each primary active controller are computed then the stored capacity is analyzed for the extra backup controllers.

Thus, the load on each k^{th} controller is computed as follows.

$$C_k = \sum_{j=1}^{N} \lambda_j(t) R_{i,j,k}, \tag{5}$$

where $\lambda_j(t)$ is the average flows arrival rate on switch j. Now, the stored capacity of backup controllers for every switch is $\beta.\frac{\lambda_j}{(\alpha-\beta)}$ with β backup controllers. If the number of failed controllers is C_f then to fulfil full resilience, the model C_f should satisfy $C_f \leq \beta$ such that $(\alpha - C_f) \geq (\alpha - \beta)$ and $(\alpha - C_f)\frac{\lambda_j}{(\alpha-\beta)} \geq \lambda_j$. The second case is $C_f \geq \beta$ in which the model only fulfils partial resilience.

TABLE II: Decision variables used in problem statement.

Variables	***Description***
C_i	1, if switch i is a core switch directly connected to a controller; 0, otherwise.
$C[x]_{ij}$	1, if core switch i covers normal switch j; 0, otherwise. Here, $C[x]_{ij}$ is a decision variable for the core switches.
$x_{i,j,k}$	1, if i^{th} switch is mapped to some another k^{th} controller when j^{th} controller fails; 0, otherwise.
$R_{i,j,k}$	1, if i^{th} core switch is mapped to k^{th} controller at position j; 0, otherwise. Here, $i, j \in V$ and $1 \leq k \leq \alpha$.
$H_{i,j}$	$\frac{1}{(\alpha-\beta)}$, if switch i is assigned to one of α linked controllers at position j; 0, otherwise.
W_{ij}	1, if switch i is assigned to the controller at j^{th} position; 0, otherwise.

Problem 1: The objective function of the problem 1 is to determine the least number of core switches in the network. The number of core switches are equal to the least number of controllers to be deployed in such a manner that atleast one controller is assigned to each switch in a sub-network. In addition, the objective is to find the optimal placement of controllers on the basis of their shortest distance to the switches in every partition. The reliability factor $R_{i,j}$ is defined to optimize the shortest distance.

Mathematically, it can be represented as follows.

$$Problem\ 1 : \min \sum_{i=1}^{N} (C_i), \tag{6}$$

$$\begin{aligned}
s.t.\ \ \mu 1 \ &: \ \sum_{k=1}^{\alpha} \sum_{i=1}^{|N|} R_{i,j,k} \frac{\lambda_j}{(\alpha-\beta)} \leq \theta_i.C_i,\ \forall i \in N, \\
\mu 2 \ &: \ \sum_{r \in M, a_{ir} \leq a_{ij}} R_{i,r,k} + \sum_{h=1}^{k-1} R_{i,j,h} \geq C_i, \\
&\quad \forall i \in N, j \in M,\ k = \{1, 2, .., \alpha\}, \\
\mu 3 \ &: \ \sum_{k=1}^{\alpha} R_{i,j,k} \leq C_i, \quad \forall i \in N, j \in M, \\
\mu 4 \ &: \ \sum_{i=1} R_{i,j,1} \leq 1, \qquad \sum_{i \in N, i \neq j} R_{i,j,k} \leq 1, \\
&\quad \forall i \in N, k = \{2, .., \alpha\}, \\
\mu 5 \ &: \ C[x]_{ij}.a_{ij} \leq A_{\max}, \quad \forall i \in N, \\
\mu 6 \ &: \ C[x]_{ij} \geq 1, \quad \forall i \in N, \\
\mu 7 \ &: \ H_{ij} \in \{0, \frac{1}{(\alpha-\beta)}\}, \quad \forall i \in N, j \in M, \\
\mu 8 \ &: \ H_{ij} = \sum_{k=1}^{\alpha} \frac{R_{i,j,k}}{(\alpha-\beta)}, \quad \forall i \in N, j \in M, \\
\mu 9 \ &: \ C_i \in \{0, 1\},\ \forall i \in N, \\
\mu 10 \ &: \ R_{i,j,k} \in \{0, 1\}, \forall i \in N, j \in M, k = \{1, 2, .., \alpha\}, \\
\mu 11 \ &: \ C[x]_{ij} \in \{0, 1\}, \forall i \in N, j \in M.
\end{aligned}$$

The explanation of each constraint is provided as follows.

- $\mu 1$ ensures that the traffic flow or number of packet-in messages processed by a core switch directly linked to a controller deployed at position i does not exceed the processing capacity of that controller θ_i. Here, the processing capacity of each controller determines the total number of switches a controller can control i.e., the maximum number of packet-in messages processed per second by each controller.
- $\mu 2$ indicates that core switch i mapped to its corresponding active controllers are all α nearest controllers. The nearest distance between the core switch and controller depends on the propagation latency a_{ij}.
- $\mu 3$ ensures that switch i mapped to all the associated α controllers are at distinct position j.
- $\mu 4$ constraint denotes that each switch is mapped to a distinct k^{th} associated controller.
- $\mu 5$ ensures that the latency between any core switch i linked with normal switch j cannot exceed the maximum allowed propagation latency $A_{\max}$ between any pair of switches (i, j).
- $\mu 6$ ensures that every switch is supervised by at least one controller.
- $\mu 7$ states that the value of a decision variable H_{ij} is binary either 0 or $\frac{1}{(\alpha-\beta)}$.
- $\mu 8$ states if the controller is one of α linked controllers at position j may avail $\frac{1}{(\alpha-\beta)}$ as a primary controllers of switch i packet-in messages.
- $\mu 9 - \mu 11$ states that the decision variables are binary.

*4) **Minimal Resilience Cost**:* In the sub-problem above, it is discussed to compute the least number of distributed controllers to be deployed in the network such that at least one controller needs to be assigned in each sub-network. Hence, the binary vector is defined as $W_{ij} = \langle w_1, w_2, .., w_{|M|} \rangle$ which denotes that i^{th} switch is supervised by the controller deployed at j^{th} position. However, there are two issues which may lead to partial resilience due to which some of the nodes (switches) may become controller-less. It may be due to (i) link failure between switch and controller, and (ii) controller failure in any sub-network. The solution for the partial resilience is to deploy an extra number of backup controllers such that atleast two controllers can be mapped to each switch. The network resilience in case of β controller failures can only be achieved if each switch is controlled exactly by two or more active controllers, i.e., $[\alpha \geq 2 + \beta]$ as each controller are serving partial incoming requests. Therefore, it is assumed that $\alpha > \beta$ in the rest of the paper.

Problem 2: The objective function of the problem 2 is to minimize the cost of extra backup controllers deployed in the network in such a way that every switch can be mapped with atleast two controllers in the network topology. In addition, the

reliability factor $R_{i,j}$ ensures the reliability of nodes i and j using the shortest path in every partition such that each switch is controlled by its nearest controller.

Mathematically, it can be expressed as follows.

$$\begin{aligned} Problem\ 2 \quad &: \quad \min\ (\overline{C}) = \sum_{i \in N} (C_i - W_{ij}), \qquad (7)\\ s.t. \quad \mu 12 \quad &: \quad C[x]_{ij} \geq 2, \quad \forall i \in N,\\ \mu 13 \quad &: \quad C_{total} = \sum_{i=1}^{|N|} C_i,\\ \mu 14 \quad &: \quad \alpha \in \{\beta + 2, \beta + 3, ..., C_{total}\},\\ \mu 15 \quad &: \quad \alpha = \sum_{i=1}^{|N|} W_{i,j}, \quad \forall i \in N,\ W_{i,j} \in \{0, 1\} \end{aligned}$$

where $\mu 12 - \mu 15$ constraints are applicable in addition to $\mu 1 - \mu 11$ defined for the objective function in Eq. (6). The explanation of the additional constraint is provided as follows.

- $\mu 12$ ensures that every switch is supervised by atleast two controllers.
- $\mu 13$ ensures that the total number of controller assigned in the network is exactly equal to C_{total}.
- $\mu 14$ ensures that α is a positive integer that consider atleast $\beta + 2$ for β controller failure to provide complete resilience.
- $\mu 15$ ensures that each switch is mapped to α reference controllers in the network.

In the next section, the PARC scheme is discussed in details.

IV. PARC: Placement Availability Resilient Controller Scheme

The PARC scheme is designed to addresses the aforementioned four issues of CPP in SDN. The four sub-problems are as follows (1) network partitioning based on cooperative game theory to split a physical network into virtual sub-networks (2) optimal position for the controller placement based on reliability factor $R_{i,j}$ that ensures the reliability of nodes i and j using the shortest path in every partition such that each switch is controlled by its nearest controller (3) the minimum number of primary controllers in order to minimize the network cost, and (4) the minimum number of backup controllers deployed in the network to achieve full resilience, in case, the primary controller fails in each sub-network. All these four problems are inter-related to each other, hence cannot be solved parallely. For example, the output of the Algorithm 1 is $Y_{initial}$ which is the deciding factor for network partitioning and is considered as the input of Algorithm 2. The initial positions of controllers are computed by selecting a switch position having maximum Shapley value. The output of the Algorithm 2 is Y_{final}, Z which is considered as the input of Algorithm 3. The final stable positions of the controllers are computed based on the euclidean distance between switch pairs assigned to the same controller, and from the switch to the controller. Finally, the output of Algorithm 3 is C which is the input of Algorithm 4. The minimum number of primary controllers/core switches are computed based on the adjacent switches having only one hop distance. Finally, the output of the Algorithm 4 is $\overline{C}$ which is the set of backup controllers which are directly linked to the core switches in a single hop distance. Therefore, the sub-problems are solved one by one as the output of the first algorithm is the input of the next algorithm. The Pareto near optimal solution of the four optimization problems is described as follows.

A. Network Partitioning based on cooperative game theory

In the first phase, the cooperative game theory is used to generate the coalitions by partitioning the entire network into different sub-networks. The set of switches defined by $X = \{x_1, x_2, .., x_n\}$ is assumed as players in the cooperative game (X, U) with utility function $U : 2^{|X|} \rightarrow \mathbb{R}, U(\emptyset) = 0$. The set of switches creates coalitions in order to obtain maximum utility. A similarity function is defined as γ: $\mathbb{R}^+ \cup \{0\} = \{0, 1\}$ which is monotonically non-increasing used for partitions by finding the similar set of switches. The similarity is found by using Euclidean distance based on choosing the smallest distance between the pair of switches.

If i and j are two switches in Euclidean n-space with positions $i = \{i_1, i_2, .., i_n\}$ and $j = \{j_1, j_2, .., j_n\}$ then the Euclidean distance between switch i and j is calculated as below.

$$a_{ij} = \sqrt{(i_1 - j_1)^2 + ... + (i_n - j_n)^2} = \sqrt{\sum_{s=1}^{n} (i_s - j_s)^2}, \quad \forall i, j \in N. \qquad (8)$$

Therefore, the similarity function is defined as below.

$$\gamma[a_{ij}] = 1 - \frac{a_{ij}}{(1 + A_{\max})}, \qquad (9)$$

where $A_{\max}$ is the highest value of propagation latency between any pair of switches.

Thus, the utility function for the set of partitions $P = \{P_1, P_2, .., P_L\}$ is given as below.

$$U(P') = \frac{1}{2} \sum_{\substack{x_i, x_j \in P' \\ x_i \neq x_j}} \gamma[a_i j],\ where, P' \subseteq X, P' \neq \emptyset. \qquad (10)$$

Now, it is shown that the considered cooperative game (X, U) is a convex game. Consider, coalitions $P_2 \subseteq P_1 \subseteq X \backslash \{t\}, x_t \in X$.

$$\begin{aligned} U(P_1 \cup \{x_t\}) - U(P_2 \cup \{x_t\}) &= \frac{1}{2} \sum_{\substack{x_i, x_j \in P_1 \\ x_i \neq x_j}} \gamma[a_{ij}]\\ &+ \sum_{x_i \in P_1} \gamma[a_{it}] - \frac{1}{2} \sum_{\substack{x_i, x_j \in P_2 \\ x_i \neq x_j}} \gamma[a_{ij}] + \sum_{x_i \in P_2} \gamma[a_{it}],\\ &= U(P_1) - U(P_2) + \sum_{x_i \in P_1 \backslash P_2} \gamma[a_{ij}],\\ U(P_1 \cup \{x_t\}) - U(P_2 \cup \{x_t\}) &\geq U(P_1) - U(P_2), \qquad (11) \end{aligned}$$

which shows that the game (X, U) is a convex game. Now, it is found that the Shapley values of the switches which are similar to each other, i.e., at a closest distance to each other have approximately the same Shapley value defined as below.

$$\psi_i(X, U) = \frac{1}{|N|!} \sum_{\delta \in \zeta} (f_i^{\delta}) = \frac{1}{|N|!} \sum_{\delta \in \zeta} U(S_{\delta, \delta(i)}) - U(S_{\delta, \delta(i)-1}),$$

$$= \frac{1}{|N|!}\sum_{\delta\in\zeta}\Big[\frac{1}{2}\sum_{\substack{x_c,x_d\in S_\delta,\delta(i)\\ x_c\neq x_d}}\gamma[a_{cd}]-\frac{1}{2}\sum_{\substack{x_c,x_d\in S_\delta,\delta(i)-1\\ x_c\neq x_d}}\gamma[a_{cd}]\Big],$$

$$= \frac{1}{|N|!}\sum_{\delta\in\zeta}\Big[\sum_{\substack{x_c,x_d\in S_\delta,\delta(i)\\ x_c\neq x_d}}\gamma[a_{cd}]-\sum_{\substack{x_c,x_d\in S_\delta,\delta(i)-1\\ x_c\neq x_d}}\gamma[a_{cd}]\Big],$$

$$= \frac{1}{|N|!}\sum_{\delta\in\zeta}\Big[\sum_{\delta(j)<\delta(i)}\gamma[a_{ij}]\Big] = \frac{1}{|N|!}\sum_{\substack{\delta\in\zeta,\delta(i)=1\\ \delta(j)<\delta(i)}}\gamma[a_{ij}],$$

$$+\frac{1}{|N|!}\sum_{\substack{\delta\in\zeta,\delta(i)=2\\ \delta(j)<\delta(i)}}\gamma[a_{ij}]+..+\frac{1}{|N|!}\sum_{\substack{\delta\in\zeta,\delta(i)=n\\ \delta(j)<\delta(i)}}\gamma[a_{ij}],$$

$$= \frac{1}{|N|!}\Big[\sum_{\substack{x_j\in X\\ j\neq i}}\gamma[a_{ij}]\Big]\Big[\sum_{i=1}^{N}\frac{i-1}{N-1}(N-1)!\Big],$$

$$\psi_i(X,U) = \frac{1}{|N|!}\frac{(N-1)!}{(N-1)N!}[1+2+..+(N-1)]\sum_{x_j\in X,i\neq j}\gamma[a_{ij}]. \tag{12}$$

So, for a fixed position of switch i, there are total $(n-1)!$ ways of positioning switch i. Now, each switch except i can be placed at positions before i in $\frac{i-1}{N-1}$ ways.

Algorithm 1 computes the initial position for deploying distributed controllers which is the deciding factor for splitting the entire network into sub-networks. The input of the algorithm is the cooperative game $g:(X,U)$, the flow arrival rate denoted by $\lambda_j(t)$, the processing capacity of each controller denoted by θ_i, the shortest distance latency matrix between any two pairs of switches denoted by A_{ij}. In addition, ε is the similarity threshold parameter, where $\varepsilon\in\{0,1\}$ that decides the number of sub-networks. The output is the set of initial position of controllers denoted by $Y_{initial}$.

Algorithm 1 Computation of initial position of controllers

Input: $G(V,E), X = \{x_1,..,x_N\}, g : (X,U), \lambda_j(t), \theta_i, \varepsilon \in \{0,1\}, A_{ij}$.
Output: $Y_{initial}$ as the initial position of number of controllers deployed.

1: **procedure** POSITION
2: initialize $Y_{initial} = NULL$;
3: initial position of switches: $X^\star = \{x_1, x_2, .., x_N\}$;
4: initialize $Y_{potential} = X^\star$;
5: **for** $(i = 1; i \leq N; i{+}{+})$ **do**
6: Compute Shapley Value $\psi_i(X,U)$ = $\frac{1}{2}\sum_{j\in N, j\neq i}\gamma[a_{ij}]$;
7: **end for**
8: **while** $(Y_{potential} \neq NULL)$ **do**
9: **if** $(x_i.\lambda_j(t) \leq \theta_i)$ **then**
10: $temp \leftarrow arg\{\max_{i\in Y_{potential}}(\psi_i)\}$;
11: $Y_{initial} \leftarrow Y_{initial} \cup \{temp\}$;
12: Compute euclidean distance (a_{ij}) based on (i,j);
13: Check the similarity threshold: D_{temp}
14: = $(i \in Y_{potential} : \gamma[a_{it}] \leq \varepsilon)$;
15: **else** (controller rejects the switch demand);
16: **end if**
17: update $Y_{potential}$ $s.t.$ conditions of D_{temp} are satisfied;
18: **end while**
19: return $Y_{initial}$;
20: **end procedure**

Step 2 is initialized with a set of the initial position of controllers $Y_{initial}$ as empty and assumed that $X^\star$ as a set of the initial position of static switches in step 3. Initially, we consider that $X^\star$ be the position of controllers. Steps 5-7 execute the *for* loop to compute the Shapley value $\psi_i(X,U)$ for all the switches. Then in steps 8-17, the *while* loop works iteratively until the $Y_{potential}$ is not empty to check if the switch demand is less than the controller processing capacity. If yes, then computes the initial position of controllers $Y_{initial}$ by selecting a switch position having maximum Shapley value in each sub-network; otherwise, the controller rejects the switch demands. Then we check, the similarity between switch pairs based on the Euclidean distance (a_{ij}) and assign those switches set in the same partition, i.e., at least ε similar to $temp$. Finally, the updated set of initial controller positions $Y_{initial}$ is returned as an output.

B. Availability of Controllers based on Stable Partitioning

The second phase computes the stable network partitioning for the final placement of controllers Y_{final}, and the mapping pair between controllers and switches denoted by Z that determines which switch is assigned to a specific controller. It takes input as the initial position of controllers and returns the output as (Y_{final}, Z). Initially, we consider the set (Y_{final}, Z) as empty. Then, the *while* loop repeats the process iteratively from steps (3-21) until $(Y_{initial} \neq Y_{final})$. The first *for* loop iteratively repeats the process in steps (4-10) until the mapping of switches to a particular controller is done on the basis of shortest distance from the switch to the controller in each partition. Hence, the closest controller is assigned to each switch in every sub-network, and the updated switch-controller mapping pairs are finally stored in a set Z. The second *for* loop in steps (11-17) iteratively repeats the process in order to find the shortest distance between every switch pairs $(a_{x_i x_r})$ controlled by the same controller and is computed in $temp$. The updated partition is finally stored in Y_{final}. However, still in case $(Y_{initial} \neq Y_{final})$ then the updated Y_{final} position is the current position of controllers and returns as an output.

Time Complexity of Algorithm 2: Steps (4-11) repeats the first *for* loop with respect to the number of switches and takes $O(n)$ times in the worst case. Also, in steps (12-19) in the second *for* loop iterates with respect to the number of controllers and takes $O(n)$ times in the worst case. In addition, steps (20-22) are conditional operations which take $O(1)$ times. Finally, the time complexity is computed as below.

$\Longrightarrow T(n) = [O(n) + O(n) + O(1)] = O(n)$. The overall time computation of Algorithm 2 is 56 seconds that iteratively executes the *for* and *while* loops.

C. Minimal Resilience Computation of Primary Controllers

In this phase, the PARC scheme solves the third issue of resilience with a minimal number of primary controllers. Algorithm 3 computes the minimum number of controllers required with maximum coverage while reducing the overall cost of the network. As core switches are linked directly to the controllers, so the least number of core switches required in the network is computed. The algorithm takes the input

Algorithm 2 Computation of stable network partitioning

Input: $G(V,E), g:(X,U), \lambda_j(t), \theta_i, A_{ij}, Y_{initial}$
Output: (Y_{final}, Z), where (Y_{final}) is the updated position of controllers and Z is deployment of switches to the set of controllers.

1: **procedure** PARTITION
2: initialize $(Y_{final}, Z) = NULL$;
3: **while** $(Y_{initial} \neq Y_{final})$ **do**
4: **for** $(i = 1; i \leq N; i$++$)$ **do**
5: **if** $(x_i.\lambda_j(t) \leq \theta_i)$ **then**
6: Calculate the euclidean distance between switch and controller:
7: $temp = arg\{\min(a_{x_i.y_j})\},\ \forall x_i \in X, y_j \in Y$;
8: $Z \leftarrow Z_{temp} \cup \{i\}$;
9: **else** (controller rejects the switch demands);
10: **end if**
11: **end for**
12: **for** $(j = 1; j \leq M; j$++$)$ **do**
13: **if** $(x_i.\lambda_j(t) \leq \theta_i)$ **then**
14: Calculate the euclidean distance between switch pairs assigned to the
15: same controller: $temp = arg\{\min(a_{x_i.x_r})\}\ \forall x_i, x_r \in Z$;
16: $Y_{final} \leftarrow Y_{final} \cup \{temp\}$;
17: **else** (controller rejects the switch demands)
18: **end if**
19: **end for**
20: **if** $(Y_{initial} \neq Y_{final})$ **then**
21: $Y_{initial} \leftarrow Y_{final}$;
22: **end if**
23: **end while**
24: return (Y_{final}, Z);
25: **end procedure**

as $(G(V,E), \lambda_j(t), \theta_i, h, Z)$, where h is the threshold on the hop count, and Z is the record of the switches deployed to a single controller. The output set C is the set of switches directly connected to a controller, i.e., the set of core switches.

Initially, C and $x_i[adj]$ are considered as empty sets, where $x_i[adj]$ stores the record of the adjacent switches of switch i. Here, the adjacent switches are those switches having only one hop distance and the list of $core[x_i]$ as an empty set. Then, we compute $x_i[adj]$ and the total count is stored in $\overline{x}$. Next, the *while* loop continuously repeats the process until all the switches are not supervised by the controller. Steps (6-15) execute the *for* loop to computes the number of core switches. First, it is checked if switch x_i is not supervised by any controller, then we count the total number of adjacent switches of x_i which are one hop distance and temporarily stores the value in $total$, where it is assumed that the initial value of $sum = 0$. Now in step 9, if the value of $total$ is greater than the sum then select $total$ as the maximum value, and i is selected as the core switch that covers maximum vertices in $G(V,E)$. The *for* loop repeats the process iteratively till the core switches are computed among all the switches based on the maximum coverage of vertices. The set C stores the number of controllers directly linked to the core switches.

Next, the function coverage is executed that covers the list of uncovered switches in multiple hops by the core switches based on depth-first search in $G(V,E)$. Initially, it is checked if the hop distance is less than the allowed threshold h of hop distance and also the controller capacity is enough to serve the switch demands then cover the uncovered switches. The

Algorithm 3 Minimal resilience of primary controllers

Input: $G(V,E), \lambda_j(t), \theta_i, h, Z$ (record the switches assigned to controllers).
Output: C as a set of number of controllers directly linked to the core switches.

1: **procedure** CORE
2: initialize $(x_i[adj], C) = \emptyset$;
3: initially set $sum = 0, hop = 1, core[x_i] = 0$;
4: build set of adjacent switches to x_i in $G(V,E)$: $\overline{x} \leftarrow x_i[adj]$;
5: **while** (for switches not in Z) **do**
6: **for** $(i = 1; i \leq N; i$++$)$ **do**
7: **if** (x_i is an unsupervised switch) **then**
8: $total \leftarrow count[\overline{x}]$;
9: **if** $(sum \leq total)$ **then**
10: $sum \leftarrow total$;
11: $core[x_i] \leftarrow i$;
12: **end if**
13: $C \leftarrow C \cup \{core[x_i]\}$;
14: **end if**
15: **end for**
16: **end while**
17: $Z \leftarrow$ COVERAGE$(Z, \lambda_j(t), \theta_i, hop, h, core[x_i], total, sum)$;
18: return (C);
19: **end procedure**
20: **procedure** COVERAGE
21: **if** $((hop \leq h)$ && $(x_i.\lambda_j(t) \leq \theta_i))$ **then**
22: **for** $(j = 1; j \leq \overline{x}; j$++$)$ **do**
23: **if** (x_j is an unsupervised switch) **then**
24: switch j is covered by $core[x_i]$;
25: $\theta \leftarrow (\theta_i - 1)$;
26: hop++;
27: **end if**
28: **end for**
29: **end if**
30: COVERAGE$(Z, \lambda_j(t), \theta, hop, h, x_i)$;
31: return (Z);
32: **end procedure**

for loop in steps (22-28) continuously repeats the process that checks if any switch x_j is not supervised by any controller then switch j is supervised by the core switch and the controller residual capacity decreases by one and the value of hop count increases by one. Finally, the updated output returns are in Z to store the number of switches assigned to the controllers.

Time Complexity of Algorithm 3: The iterations in steps (6-15) in the first *while* loop computes the least number of controllers required in the network and takes $O(n)$ times in the worst case. Also, in steps (22-28) in the second *for* loop in the *Coverage* function which iterates with respect to the utmost number of core switches supervised by the distributed controllers and takes $O(n)$ times in the worst case. Finally, the time complexity is computed as below.

$\Longrightarrow T(n) = [O(n) + O(n)] = O(n)$.

The overall time computation of Algorithm 3 is 67 seconds that iteratively executes the *for* and *while* loops.

D. Minimal Resilience Cost of Backup Controllers

In the final phase, Algorithm 4 computes the minimal cost of resilience to minimize the number of backup controllers

required in case, some of the switches are left as controllerless nodes. The concept of backup controllers states that each switch is supervised by two controllers in $G(V, E)$. In addition, the algorithm performs synchronization of events among controllers at the control plane by assigning a timestamp for proper ordering of updated events. The input of the algorithm are $(G(V, E), \lambda_j(t), \theta_i, C, Z^\star)$, where, $Z^\star$ maintains the record in which switches are assigned two controllers. The output returns are the set of backup controllers denoted by $\overline{C}$ which are directly linked to the core switches. Initially, the record is built by the number of adjacent controllers of x_i and $Z^\star$. Then for all the switches check, whether the controller in set C is fully utilized. Next, in steps (11-22), the *while* loop repeats the process to find the number of extra core switches until all the switches are not assigned to two controllers. Then for all the switches check, if it is a normal switch then it again repeats a similar process to compute the extra number of core switches and returns the output in set $\overline{C}$.

Next, the function *timestamp* is executed in order to synchronize the ordering of events among all distributed controllers at the control plane. Initially, $(term_{no.}, Event_{no.})$ are considered as empty, where $term_{no.}$ increments every time in case the master controller fails and switch is assigned to another controller while $Event_{no.}$ is incremented by one if any new event arrives at the controller. Let x_{ID} is a unique switch identifier. Then, we check if the switch demand exceeds the controller processing capacity. For all the controllers, if (x_{ijk} = 1) then the *term number* is incremented by 1, otherwise, the *event number* is increased by 1. Finally, the timestamp of events stores all the attributes such as– *switch ID, term number, event number* in $Event_T$ and the controller j broadcasts the updated copy to all the M number of controllers present in the network and returns $Event_T$ as an output.

Time Complexity of Algorithm 4: The iterations in steps (6-9) in the first *for* loop iterate according to the number of switches and takes $O(n)$ times in the worst case. Next, in steps (12-21), the *while* loop returns the number of extra backup controllers required in the network and takes $O(n)$ times in the worst case. Also, the iterations in steps (30-38) in the *for* loop which iterates with respect to the number of controllers and takes $O(n)$ times in the worst case. Finally, the time complexity of Algorithm 4 is computed as below.

$\implies T(n) = [O(n) + O(n) + O(n)] = O(n)$.

The overall time computation of Algorithm 4 is 73 seconds that iteratively executes the *for* and *while* loops.

Algorithm 4 Minimal resilience cost of backup controllers

Input: $G(V, E), \lambda_j(t), \theta_i, C, Z^\star$ (record the switches assigned to two controllers).
Output: $\overline{C}$ as a set of extra backup controllers directly linked to the core switches.

1: **procedure** COST
2: initialize $(x_i[adj],\ \overline{C}) = \emptyset$;
3: initially set $sum = 0, core[x_i] = 0$;
4: build set of adjacent switches to x_i in $G(V, E)$: $\overline{x} \leftarrow x_i[adj]$;
5: build $Z^\star$ from set C;
6: **for** $(i = 1; i \leq N; i$++$)$ **do**
7: **if** $(x_i.\lambda_j(t) < \theta_i)$ **then**
8: Controller is capable of serving more switches;
9: **end if**
10: **end for**
11: **while** (for switches not in $Z^\star$) **do**
12: **for** $(i = 1; i \leq N; i$++$)$ **do**
13: **if** (x_i is a normal switch) **then**
14: $total \leftarrow count[\overline{x}]$;
15: **if** $(sum \leq total)$ **then**
16: $sum \leftarrow total$;
17: $core[x_i] \leftarrow i$;
18: **end if**
19: $\overline{C} \leftarrow \overline{C} \cup \{core[x_i]\}$;
20: **end if**
21: **end for**
22: **end while**
23: $Event_T \leftarrow$ TIMESTAMP$(Event_T, \lambda_j(t), \theta_i, Event_{no.}, term_{no.}, x_{ID})$;
24: return $(\overline{C})$;
25: **end procedure**
26: **procedure** TIMESTAMP
27: initialize $(term_{no.},\ Event_{no.}) = \emptyset$;
28: x_{ID} be a unique switch Identifier;
29: **if** $(x_i.\lambda_j(t) \leq \theta_i)$ **then**
30: **for** $(j = 1; j \leq M; j$++$)$ **do**
31: **if** $(x_{ijk} == 1)$ **then**
32: $term_{no.}$++;
33: **end if**
34: $Event_{no.}$++;
35: $Event_T = (x_{ID}||term_{no.}||Event_{no.})$;
36: $C_{|M|} \leftarrow C_j(Event_T)$;
37: update $Event_T$;
38: **end for**
39: **end if**
40: TIMESTAMP$(Event_T, \lambda_j(t), \theta_i, Event_{no.}, term_{no.}, x_{ID})$;
41: return $(Event_T)$;
42: **end procedure**

E. Analysis of the proposed scheme

The analysis of the proposed scheme is evaluated based on cost, backup capacity, and resilience. The detailed explanation is as follows.

1) Cost analysis of the proposed scheme: Fig. 2(c) illustrates the mapping of a switch to all its reference controllers. Here, $\alpha = 4$ is the total number of controllers associated to a switch with $\beta = 2$ as the allowed number of backup controllers and each reference controller holds a capacity of $\frac{\lambda_j}{(\alpha-\beta)}$ times the switch demand. In this scenario, our goal is to minimize the network cost with resilience in case of previously known β controllers. In order to reduce the backup capacity and the network cost, switches linked to two different primary controllers working without any failure can rely on the same backup capacity stored on another controller. In case, the primary controller C_j^* fails then the incoming traffic on the switch is transmitted to the backup controller C_j which is calculated by using the formula given as below.

$$Q_{j^*,j} = \sum_{i=1}^{|N|} \Bigg[\sum_{\substack{k^*=1 \\ \forall j \in N, j^* \in N, j \neq j^*}}^{\alpha-\beta} R_{i,j^*,k^*} \sum_{k^*=(\alpha-\beta+1)}^{\alpha} R_{i,j,k} \Bigg] \frac{\lambda_j}{(\alpha-\beta)}, \tag{13}$$

where $\sum_{k^*=1}^{\alpha-\beta} R_{i,j^*,k^*}$ and $\sum_{k^*=(\alpha-\beta+1)}^{\alpha} R_{i,j,k}$ test whether j^* and j are a primary active controller and backup controller serving the demand of switch i respectively. Therefore, the total switch demand served by the backup controller in case

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2999072, IEEE Transactions on Vehicular Technology

of β controller failures is computed as follows.

$$Q_j = \max_{\beta}\Big(\beta, \quad Q_{j^*,j}\Big), \ \forall j \in N, j^* \in N, j \neq j^*, \tag{14}$$

where $\max_\beta(.)$ is a function that calculates the maximum subset from $Q_{j^*,j}$ set of backup controller. The above backup capacity reserved on the backup controller can be explained using a simple example.

Let us assume a network topology with a set of three switches denoted by X_1, X_2, X_3 and four controllers denoted by Y_1, Y_2, Y_3, Y_4 respectively. Each switch is connected with exactly two primary controllers and one backup controller. The primary controller mapped with X_1, X_2, X_3 are $\{(Y_1, Y_2), (Y_1, Y_3), (Y_2, Y_3)\}$ while the backup controller is Y_4 for all the switches. Suppose, the packet-in messages or demand of switches X_1, X_2, X_3 is equal to 60, 80 and 40 respectively. For each switch, $\alpha = 3, \beta = 1$ i.e., each switch is linked to two primary and one backup controller to tackle the controller failiure. As $(\alpha - \beta) \geq 2$, the backup capacity stored to meet the switch demand is equal to $\beta\frac{\lambda_j}{\alpha-\beta}$. Using this formula, the total backup capacity stored at controller Y_4 is computed as $(30 + 40 + 20) = 90$. However, in case if a single primary controller fails then the failure of a single controller might not affect all the three switches as each switch is connected with a different pair of controllers. Hence, the total backup capacity is shared at Y_4 in case of controller failure. So, the total switch demand served by Y_4 controller is computed based on Eq.(14) defined as $\max_\beta(\{(X_1, X_2), (X_1, X_3), (X_2, X_3)\})$ = $\max_\beta(\{(30 + 40), (30 + 20), (40 + 20)\})$ = 70. This shows that the process of backup capacity sharing reduces the cost and backup capacity of the network.

2) Backup controller capacity analysis: The additional backup capacity stored at a pre-specified β number of controllers is inversely proportional to α (reference controllers) and is defined as follows.

$$\Big(\sum_{j=1}^{|N|}\beta\frac{\lambda_j}{(\alpha-\beta)}\Big). \tag{15}$$

In order to minimize the value of backup capacity at controllers, the aim is to maximize the number of reference controllers α. It can be formulated as follows.

$$\max\Big(\alpha\Big). \tag{16}$$
$$s.t.\ \mu 13, \mu 14, \mu 15$$

Algorithm 5 computes the near optimal values of α and C_{total} by taking input parameters as $G(V, E), \lambda_j(t), \theta_i, \beta, a_{ij}$. The values of α and C_{total} are dependent on each other and initially, it is considered that $\alpha = \beta + 2$. The solution of minimum cost (Problem 1) is used returns the value of C_{total}. Then, the solution of minimum backup (Problem 2) takes C_{total} as input to return the updated value of α as α^*. Similarly, α^* is used as input in Problem 1 to evaluate the updated value of C_{total} as $\overline{C}_{total}$. These two steps are iteratively performed till the value of α does not changed any further resulting in near optimal values as $\overline{\alpha}, \overline{C}_{total}$.

Algorithm 5 Computing optimal value of α, C_{total}

Input: $G(V, E), \lambda_j(t), \theta_i, \beta, a_{ij}$.
Output: $\overline{\alpha}, \overline{C}_{total}$.

procedure CAPACITY
  initialize $\alpha = \beta + 2$;
  $\alpha^* = \infty$;
  C = call_Algorithm 3$(G(V, E), \lambda_j(t), \theta_i, \beta, a_{ij})$;
  $C_{total} = C$;
  while $(\alpha! = \alpha^*)$ **do**
    $C = C_{total}$;
    $\overline{C}$ = call_Algorithm 4$(G(V, E), \lambda_j(t), \theta_i, \beta, a_{ij})$;
    $C = \overline{C}$;
    C_{total} = call_Algorithm 3$(G(V, E), \lambda_j(t), \theta_i, \beta, a_{ij})$;
  end while
  $\overline{\alpha} = \alpha^*$;
  $\overline{C}_{total} = C_{total}$;
end procedure

Therefore, the near-optimal value of the total backup capacity stored by the controllers is defined by using the formula.

$$\Big(\sum_{j=1}^{|N|}\beta\frac{\lambda_j}{(\overline{\alpha}-\beta)}\Big). \tag{17}$$

3) Resilience analysis: To achieve resilience against β controller failures, we have considered atleast $\beta + 2$ controllers should be connected to each switch. In our Problem 1 of minimum cost model for pre-specified planning ahead parameter β and fixed parameter α, the %tage of traffic flow between switch and controller unaffected by minimum cost model when there occur n_F failures is determined by using the formula.

$$\min\Big(100, \ \max\Big[\frac{(\alpha - n_F)100}{\alpha - \beta}, \quad 0\Big]\Big), \tag{18}$$

where n_F denotes the number of controller failures. Now in this case, two situations arises (i) $n_F \leq \beta < \mu$, where the proposed model achieves complete resilience with 100% traffic flow is unaffected (ii) $\beta < n_F < \mu$, where the model achieves only partial resilience against n_F controller failures.

Note: It can be noticed that as the processing capacity of a controller increases, the number of controllers needed to acheive complete resilience in case of β controller failures decreases.

V. PERFORMANCE EVALUATION

A. Numerical Settings

The list of parameters used in numerical settings to perform the experiments are listed in Table III. Simulation results are tested on Internet2 OS3E, Planet V2, and Jellyfish topology by considering the latency between node-to-controller and inter-controller as shown in Fig. 3. In this setup, the topology consists of 34 nodes with $k = 5$ as the number of controllers placed in the network which are denoted by double-circles nodes while the rest 29 nodes are OpenFlow virtual switches.

Both the Algorithm 1 and Algorithm 2 uses the concept of cooperative game theory with k-medoids, which results in a Pareto near optimal solution for the placement of controllers, as shown in Fig. 4(a). The cooperative game with k-medoids partition the network into four and five sub-networks resulting

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2999072, IEEE Transactions on Vehicular Technology

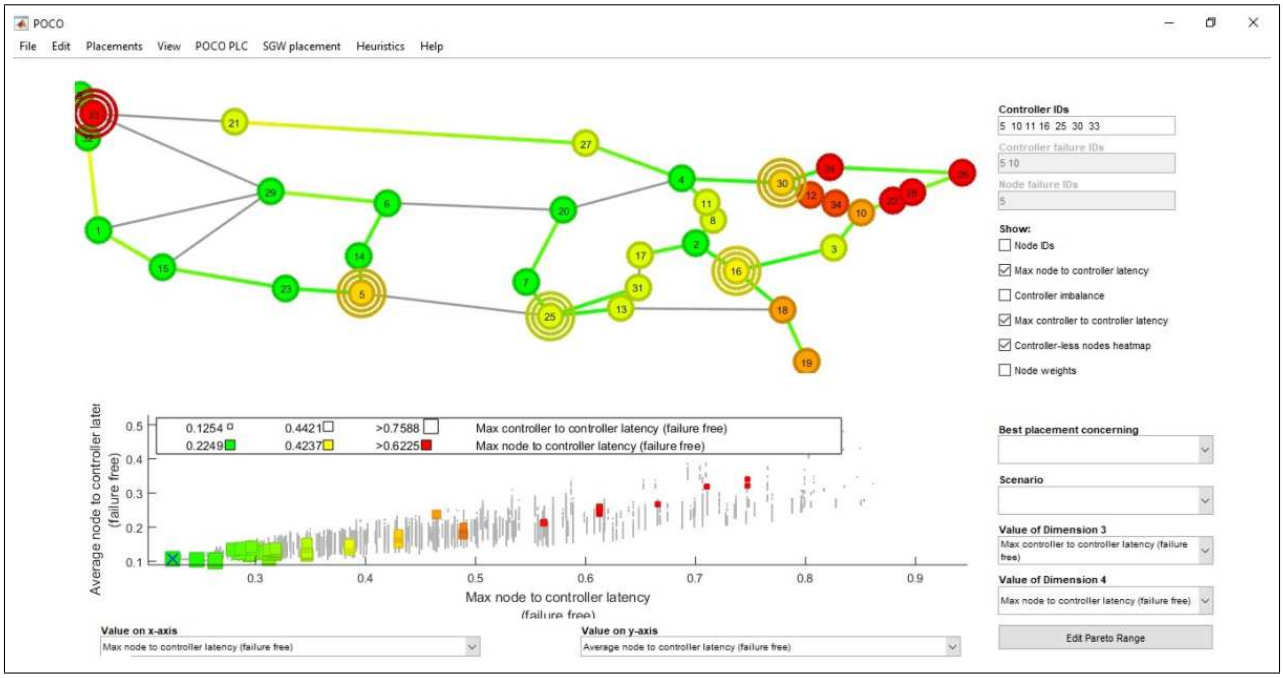

Fig. 3: Optimal Controller placement with k= 5 on Internet2 OS3E topology by considering node-to-controller and controller-to-controller latency

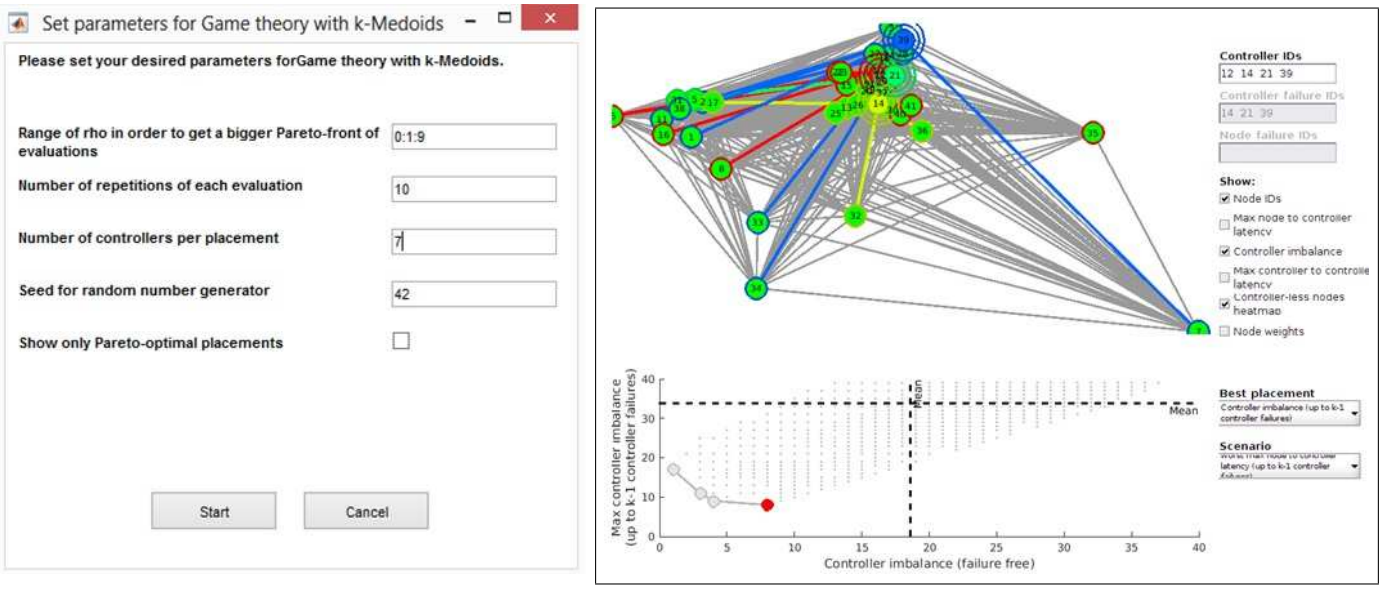

Fig. 4: (a) Pareto optimal placement using cooperative game with k-medoids (b) Controller imbalance and controller-less nodes on Planet V2 topology

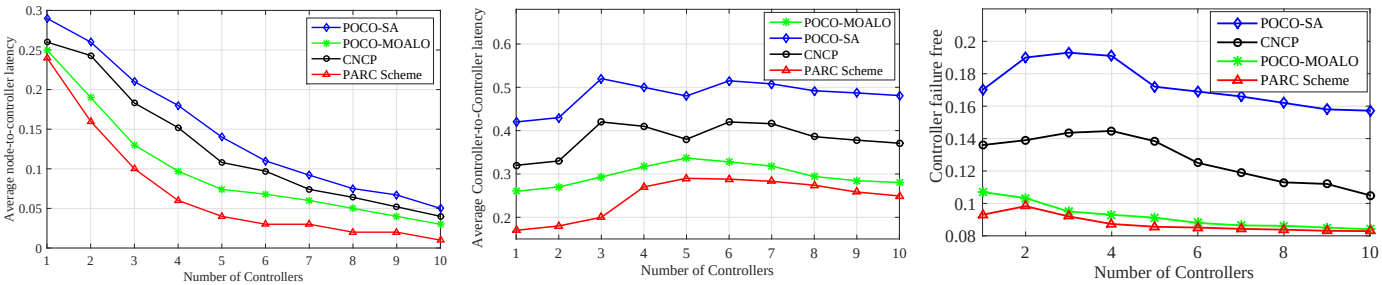

Fig. 5: (a) Average node-to-controller latency vs number of controllers (b) Average controller-to-controller latency vs number of controllers (c) Controller failure free vs number of controllers

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TVT.2020.2999072, IEEE Transactions on Vehicular Technology

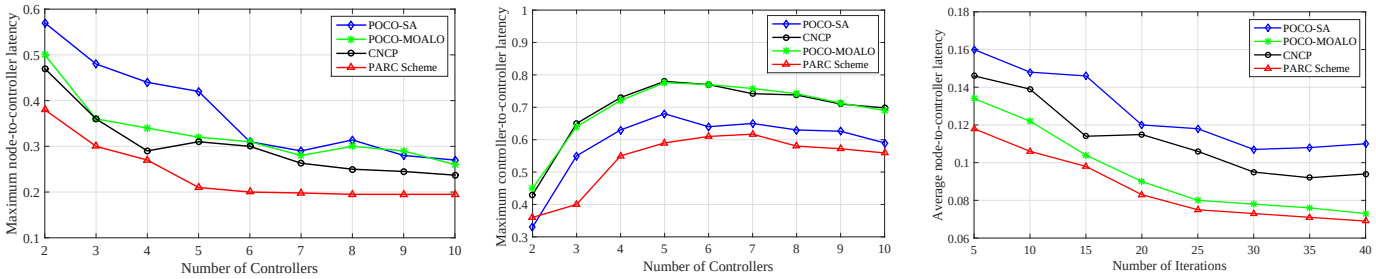

Fig. 6: (a) Maximum node-to-controller latency vs number of controllers (b) Maximum controller-to-controller latency vs number of controllers (c) Average node-to-controller latency vs number of iterations

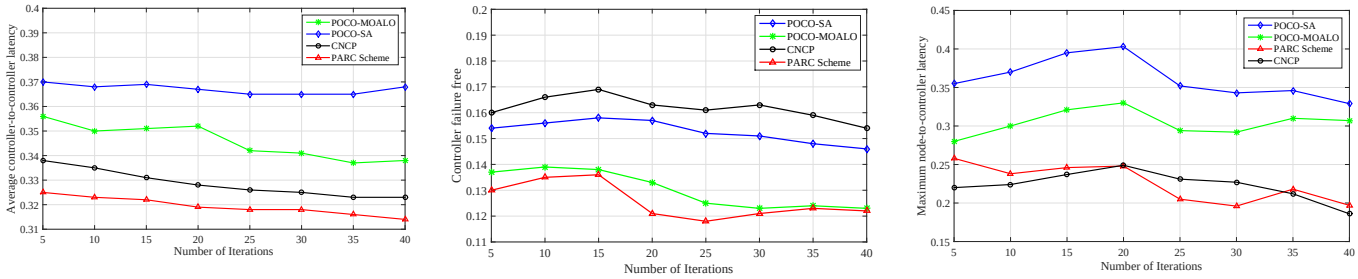

Fig. 7: (a) Average controller-to-controller latency vs number of iterations (b) Controller failure free vs number of iterations (c) Maximum node-to-controller latency vs number of iterations

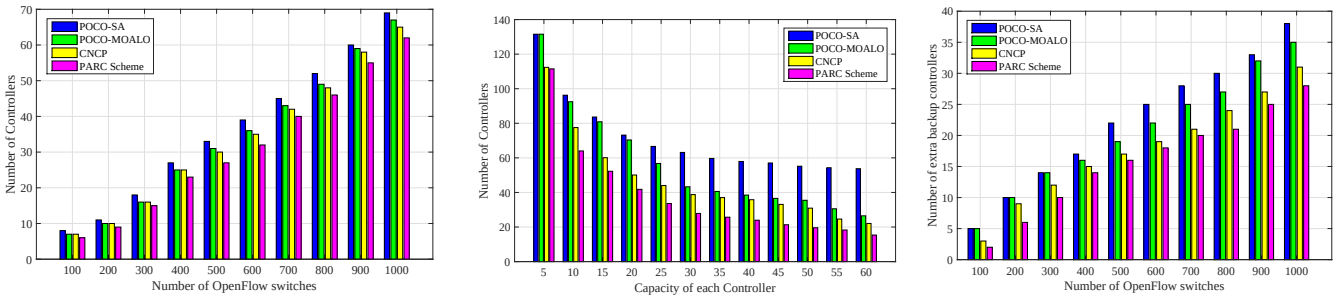

Fig. 8: (a) Number of controllers vs the number of switches (b) Number of controllers vs controllers capacity (c) Number of extra backup controllers with the number of OpenFlow switches

TABLE III: Numerical Settings.

Parameter	***Description***
System Configuration	Intel core i7 4770 with 3.20 GHz, 8 GB RAM
Simulation Platform	Matlab 2015a with POCO toolset [37]
Datasets (Topology)	Internet Topology Zoo (Internet2 OS3E [38], Planet V2 [39], Jellyfish topology [40])
Switch type	OpenFlow vSwitch version 2.5.5
Comparison Schemes	POCO-SA [18], POCO-MOALO [19], CNCP [22]
Each Switch ports	12 ports for switch interconnection
Number of nodes (V)	34 nodes, 110 nodes, 1000 nodes
Demands on switches (λ_j)	1072 flows/sec (average flow arrival rate)
Each Controller capacity (θ_i)	30 Kilo flows/sec (0.03×10^6 flows/sec)
Packet size	160 Bytes
Maximum bandwidth	2 Gbps
Number of iterations	40
Max. Resilience level	{1, 2} Controller failure

in the latency of 8.96 milliseconds and 6.51 milliseconds in the worst case. Fig. 4(b) shows the grahical user interface (GUI) representation of the best placement for Planet V2 topology in case of controller imbalance and controller-less nodes. Here, the controller-less nodes are those switches which are not linked to any controller due to controller or link failures. The graph shown in Fig. 4(b) calculates the mean value for controller failure free versus maximum controller imbalance, i.e., upto $k-1$ controller failures. The size of the datacenters varies from small-sized to large-sized networks ranging from 34-1000 switches. In the next subsection, the PARC scheme performance in terms of the inter-controller and node-to-controller latency is discussed.

B. Results and Discussion

The PARC scheme is compared with the existing state-of-the-art schemes (POCO-PSA, POCO-MOALO, and CNCP) and its performance is evaluated on the basis of latency and network cost. The computation for minimizing the average case propagation latency from switch to controller is defined by using the formula.

$$\min \frac{1}{|N|}\left(\sum_{i=1}^{|N|}\sum_{j=1}^{|N|}\sum_{k=1}^{\alpha-\beta}\frac{1}{(\overline{\alpha}-\beta)}a_{ij}R_{i,j,k}\right). \tag{19}$$

Further, the worst case propagation latency is computed as follows.

$$\min\left(\max_{i\in N}\left(a_{ij}R_{i,j,\alpha-\beta}\right)\right). \quad (20)$$
$$s.t. \quad \mu1, \mu2, \mu3, \mu4, \mu7, \mu8, \mu10, \mu15$$

The detailed explanation is as follows.

1) Impact on Node-to-Controller Latency: The node-to-controller latency is the latency incurred as the number of demands increases from the multiple switches to be served by its assigned controller. Fig. 5(a) shows the average node-to-controller latency with respect to the number of controllers. There is an increase in latency if the number of demands between the switch-to-controller increases. The average node-to-controller latency in Fig. 5(a) is monotonically decreasing as the number of controllers increases. The experimental results show that the average latency between the switch-to-controller by using the PARC scheme is reduced to 51.83%, 28.21%, and 44.22% as compared to the POCO-SA, POCO-MOALO, and CNCP schemes respectively. Likewise, Fig. 6(a) shows the maximum node-to-controller latency with respect to the number of controllers. Initially, there is a regular decrease in the latency between the node-to-controller, but with an increase in the size of controllers, there is a very slight decrease in the latency. The latency of the PARC scheme is reduced to 36.48%, 27.60%, and 21.35% in comparison to the POCO-SA, POCO-MOALO, and CNCP schemes, respectively.

Similarly, in Figs. 6(c) and 7(c), the average and maximum latencies are computed between the switch-to-controller within a sub-network with respect to the number of iterations. It has been observed from Fig. 6(c) that as the number of iterations increases, there is a gradual decrease in the latency of the *PARC* scheme. Here, the number of iterations to perform the experiments are 40. The node-to-controller latency computes the Euclidean distance from the switch-to-controller in each partition. The experimental results show that the average latency between the node-to-controller of the proposed scheme is reduced to 31.85%, 8.45%, and 23.07% in comparison to the POCO-SA, POCO-MOALO, and CNCP schemes respectively. Moreover, in Fig. 7(c), the maximum latency between the switch-to-controller of the proposed scheme is 38.26%, 26.62%, and 1.2% which is comparatively lesser in comparison to the existing POCO-SA, POCO-MOALO, and CNCP schemes respectively.

The proposed scheme outperforms the existing schemes as it considers the k-medoids method for the initialization of controllers and then optimally places the controller in each sub-network on the basis of maximum Shapley value of the switch. Thus, the solution obtained is deterministic in nature and excludes the randomness which is considered in the existing schemes.

2) Impact on Inter-Controller Latency: The inter-controller latency occurs during the communication of messages among the distributed controllers which are located in different sub-networks. Fig. 5(b) demonstrates the average controller-to-controller latency with respect to the number of controllers. Initially, the inter-controller latency increases as the size of the controller increases but as the number of controllers reaches five; the inter-controller latency shows an improvement in the proposed scheme. The average inter-controller latency using the PARC scheme reduces to 49%, 35.73%, and 17.41% in comparison to the POCO-SA, POCO-MOALO, and CNCP schemes respectively. The reason why the PARC scheme outperforms the existing schemes is due to the variation in the controller position and assignment decisions.

Fig. 6(b) demonstrates that as the controller's size increases, there is an increase in latency between the pair of controllers. The results show that the maximum inter-controller latency of the PARC scheme is reduced to 10.08%, 22.71%, and 22.56% as compared to the POCO-SA, POCO-MOALO, and CNCP schemes, respectively. Also, in Fig. 7(a), it has been observed that the average inter-controller latency does not increase gradually with an increase in the number of iterations. The average latency of the proposed scheme is compared with the existing schemes and is reduced to 7.66%, 13%, and 2% respectively. The reason for such improvements is that the PARC scheme gives more priority to place the controllers close together as it influences the inter-controller latency. The proposed scheme implements the timestamp strategy to maintain the global synchronization between two controllers while the existing schemes incur more latency as they lack in synchronization of messages.

3) Impact on the Network Disruption: The network disruption occurs due to controller or link failures, which may lead to controller-less nodes in the worst case. Figs. 5(c) and 7(b) demonstrate the scenarios of failure-free controllers in which the backup controller is assigned to the switches to avoid additional delay during failure. Fig. 5(c) shows that as the number of controllers increases, the network is resilient and incurs minimum latency. The results show that the controller failure rate is improved to 49.34%, 4.76%, and 31.37% in comparison to the POCO-SA, POCO-MOALO, and CNCP schemes, respectively. Similarly, the performance of the failure-free controllers is analyzed with the number of iterations. Fig. 7(b) shows the improvement in the existing solutions in comparison to the PARC scheme by 14.72%, 3.45%, and 22.31% respectively. The reason why the PARC scheme outperforms the existing schemes is due to the mapping of switches with exactly two β backup closest controllers. The existing schemes, i.e., POCO-SA, POCO-MOALO, and CNCP schemes consider the mapping of each switch with exactly single primary controller based on the nearest position, therefore, the issue of network disruption and delay occurs in case the single controller fails.

4) Impact on the least number of Controllers: Fig. 8(a) computes the required number of controllers to supervise all the OpenFlow switches in the network. The number of required controllers is the ratio of the processing capacity of an individual controller to the average flow arrival rate for switches. Thus, the total number of controllers required to supervise the number of OpenFlow switches are computed as follows = (30,000/1072) = 27 in the entire network. However, the required number of controllers varies as the size of the datacenter varies from 100-1000 switches and each OpenFlow switch reserves 12 out of 24 ports for switches interconnection. Initially, the lower bound is calculated by dividing the total number of switches with the processing capacity of the con-

troller. The results show that the cost of deploying the number of controllers in the proposed scheme is 12.98%, 8.16%, and 6.25% which is less in comparison to the POCO-SA, POCO-MOALO, and CNCP schemes respectively. Hence, the PARC scheme deployed less number of controllers irrespective of the switch size.

5) Impact on Controller's Capacity: Fig. 8(b) shows the controllers capacity vs the number of controllers deployed in the network. The processing capacity of each controller to control the number of switches in each sub-network differs from 10–60 switches according to the proposed Algorithm 3. Here, as the capacity of each controller increases, it leads to a decrease in the number of required controllers, but after exceeding a threshold limit of 50 switches, the number of required controllers does not decrease. The average capacity of the total number of controllers to control the switches of the PARC scheme is 51.5% while the POCO-SA, POCO-MOALO, and CNCP are 73.8%, 60.6%, 53.7% respectively. The reason for this behaviour is that the coverage range of multiple controllers overlaps and the propagation delay mainly depends upon the coverage range of a controller, i.e., the number of hops or physical distance in the topology. The PARC scheme deploys less number of controllers with a maximum coverage range of switches as compared to the existing schemes. The maximum range is defined as the distance from the controller to the farthest switch, which is two hops.

6) Impact on an extra number of backup Controllers: Algorithm 4 illustrates that the number of minimal extra backup controllers required for resilience in case any primary active controller fails. Fig. 8(c) shows that the number of extra backup controllers increases as the number of OpenFlow switches increases. The proposed scheme reduces the number of additional controllers up to 27.92%, 21.95%, and 10.11% as compared to the POCO-SA, POCO-MOALO, and CNCP schemes respectively. It has been observed that the PARC scheme redeploys less number of controllers as a backup in comparison to the existing schemes. The reason for the better performance of the proposed scheme is that the existing schemes randomly redeploy the number of extra controllers while the proposed scheme considers two-hop as the maximum coverage range which redeploys less number of controllers.

VI. CONCLUSION

In this paper, we propose the PARC scheme to address the issues of multiple CPP, with a view that the data plane continues to operate without any failure. The PARC scheme divides the CPP into four sub-problems, i.e., network partitioning, the position of controllers, the minimal number of controllers, and the minimal number of extra backup controllers for resilience in each sub-network. The simulation is performed on the Internet topology zoo on the Matlab platform. The numerical results are tested on Internet2 OS3E topology using POCO-toolset, simulated on Matlab framework. The experimental results demonstrate that the proposed scheme reduces the controller deployment cost to 12.98%, 8.16%, and 6.25% as compared to the POCO-SA, POCO-MOALO, and CNCP schemes, respectively. In addition, the PARC scheme outperforms the existing state-of-the-art schemes such as- POCO-SA, POCO-MOALO, and CNCP in terms of inter-controller and switch-to-controller latency.

In future, we will explore the resilience of POCO scheme by increasing the resilience level upto three or more controllers failure to analyze the control plane reliability.

VII. ACKNOWLEDGMENTS

The authors would like to thank the Editors and the anonymous reviewers for their invaluable feedback and comments that helped us to improve the quality and presentation of this article. This work is supported by a fellowship under the INDO-POLAND Research Project funded from Department of Science & Technology (DST), India with reference number: DST/INT/POL/P-34/2016.

REFERENCES

[1] G. S. Aujla, R. Chaudhary, N. Kumar, A. K. Das, and J. J. Rodrigues, "SecSVA: Secure Storage, Verification, and Auditing of Big Data in the Cloud Environment," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 78–85, Jan. 2018.

[2] R. Yu, J. Ding, X. Huang, M.-T. Zhou, S. Gjessing, and Y. Zhang, "Optimal Resource Sharing in 5G-Enabled Vehicular Networks: A Matrix Game Approach," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 10, pp. 7844–7856, Oct. 2016.

[3] W. Wang, M. Dong, K. Ota, J. Wu, J. Li, and G. Li, "CDLB: A Cross-Domain Load Balancing Mechanism for Software Defined Networks in Cloud Data Centre," *International Journal of Computational Science and Engineering*, vol. 18, no. 1, pp. 44–53, Dec. 2019.

[4] R. Chaudhary, N. Kumar, and S. Zeadally, "Network Service Chaining in Fog and Cloud Computing for the 5G Environment: Data Management and Security Challenges," *IEEE Communications Magazine*, vol. 55, no. 11, pp. 114–122, Nov. 2017.

[5] "Software Defined Data Center (SDDC) Market," (Available: https://www.alliedmarketresearch.com/software-defined-data-center-market) (Accessed on Jun. 2019).

[6] G. S. Aujla, R. Chaudhary, N. Kumar, J. J. Rodrigues, and A. Vinel, "Data Offloading in 5G-Enabled Software-Defined Vehicular Networks: A Stackelberg-Game-Based Approach," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 100–108, Aug. 2017.

[7] R. Chaudhary, G. S. Aujla, N. Kumar, and J. J. Rodrigues, "Optimized Big Data Management across Multi-Cloud Data Centers: Software-Defined-Network-based Analysis," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 118–126, Feb. 2018.

[8] G. S. Aujla, R. Chaudhary, K. Kaur, S. Garg, N. Kumar, and R. Ranjan, "SAFE: SDN-Assisted Framework for Edge–Cloud Interplay in Secure Healthcare Ecosystem," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 1, pp. 469–480, Jan. 2018.

[9] G. S. Aujla, R. Chaudhary, N. Kumar, R. Kumar, and J. J. Rodrigues, "An Ensembled Scheme for QoS-aware Traffic Flow Management in Software Defined Networks," in *2018 IEEE International Conference on Communications (ICC), Kansas City, MO, USA*, May 2018, pp. 1–7.

[10] H. Li, K. Ota, M. Dong, and M. Guo, "Mobile Crowdsensing in Software Defined Opportunistic Networks," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 140–145, Jun. 2017.

[11] A. Jindal, G. S. Aujla, N. Kumar, R. Chaudhary, M. S. Obaidat, and I. You, "SeDaTiVe: SDN-Enabled Deep Learning Architecture for Network Traffic Control in Vehicular Cyber-Physical Systems," *IEEE Network*, vol. 32, no. 6, pp. 66–73, Nov. 2018.

[12] R. Chaudhary and N. Kumar, "LOADS: Load Optimization and Anomaly Detection Scheme for Software-Defined Networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 12, pp. 12 329–12 344, Dec. 2019.

[13] Y. Zhang, L. Cui, W. Wang, and Y. Zhang, "A Survey on Software Defined Networking with Multiple Controllers," *Journal of Network and Computer Applications*, vol. 103, pp. 101–118, Feb. 2018.

[14] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proceedings of the first workshop on Hot topics in software defined networks, Stanford, CA*, Aug. 2012.

[15] R. Chaudhary, G. S. Aujla, S. Garg, N. Kumar, and J. J. Rodrigues, "SDN-enabled Multi-Attribute-based Secure Communication for Smart Grid in IIoT Environment," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2629–2640, Jun. 2018.

[16] Y. E. Oktian, S. Lee, H. Lee, and J. Lam, "Distributed SDN Controller System: A Survey on Design Choice," *Computer Networks*, vol. 121, pp. 100–111, Jul. 2017.

[17] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in sdn-based core networks," in *Proceedings of the 25th International Teletraffic Congress (ITC), Shanghai, China*, Nov. 2013.

[18] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic Approaches to the Controller Placement Problem in Large Scale SDN Networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 4–17, Mar. 2015.

[19] M. Ramasamy and S. Pawar, "Pareto-Optimal Multi-Controller Placement in Software Defined Network," in *2018 3rd International Conference for Convergence in Technology (I2CT), Pune, India*, Apr. 2018, pp. 1–7.

[20] A. Ksentini, M. Bagaa, T. Taleb, and I. Balasingham, "On Using Bargaining Game for Optimal Placement of SDN Controllers," in *2016 IEEE International Conference on Communications (ICC), Kuala Lumpur, Malaysia*, May 2016.

[21] H. K. Rath, V. Revoori, S. M. Nadaf, and A. Simha, "Optimal Controller Placement in Software Defined Networks (SDN) using a Non-Zero-Sum Game," in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, Sydney, NSW, Australia*, Oct. 2014.

[22] B. P. R. Killi and S. V. Rao, "Capacitated Next Controller Placement in Software Defined Networks," *IEEE Transactions on Network and Service Management*, vol. 14, no. 3, pp. 514–527, Jun. 2017.

[23] M. Tanha, D. Sajjadi, and J. Pan, "Enduring Node Failures through Resilient Controller Placement for Software Defined Networks," in *IEEE Global Communications Conference (GLOBECOM), Washington, DC*, Dec. 2016.

[24] M. Tanha, D. Sajjadi, R. Ruby, and J. Pan, "Capacity-aware and Delay-guaranteed Resilient Controller Placement for Software-Defined WANs," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 991–1005, Apr. 2018.

[25] B. Zhang, X. Wang, and M. Huang, "Multi-Objective Optimization Controller Placement Problem in Internet-Oriented Software Defined Network," *Computer Communications*, vol. 123, pp. 24–35, Jun. 2018.

[26] A. Alshamrani, S. Guha, S. Pisharody, A. Chowdhary, and D. Huang, "Fault Tolerant Controller Placement in Distributed SDN Environments," in *IEEE International Conference on Communications (ICC), Kansas City, MO*, May 2018.

[27] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "Primary-Backup Controller Mapping for Byzantine Fault Tolerance in Software Defined Networks," in *IEEE Global Communications Conference, GLOBECOM, Singapore*, Dec. 2017.

[28] A. Jalili, M. Keshtgari, R. Akbari, and R. Javidan, "Multi Criteria Analysis of Controller Placement Problem in Software Defined Networks," *Computer Communications*, vol. 133, pp. 115–128, Jan. 2019.

[29] H. Li, K. Ota, and M. Dong, "Virtual Network Recognition and Optimization in SDN-enabled Cloud Environment," *IEEE Transactions on Cloud Computing*, Sep. 2018, doi: 10.1109/TCC.2018.2871118.

[30] T. Zhang, P. Giaccone, A. Bianco, and S. De Domenico, "The role of the Inter-Controller Consensus in the Placement of Distributed SDN Controllers," *Computer Communications*, vol. 113, pp. 1–13, Nov. 2017.

[31] T. Wang, Z. Su, Y. Xia, and M. Hamdi, "Rethinking the Data Center Networking: Architecture, Network Protocols, and Resource Sharing," *IEEE Access*, vol. 2, pp. 1481–1496, Dec. 2014.

[32] H. Li, K. Ota, and M. Dong, "LS-SDV: Virtual Network Management in Large-Scale Software-Defined IoT," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 8, pp. 1783–1793, Aug. 2019.

[33] J. Wu, M. Dong, K. Ota, J. Li, and Z. Guan, "Big Data Analysis-based Secure Cluster Management for Optimized Control Plane in Software-Defined Networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 1, pp. 27–38, Mar. 2018.

[34] K. E. Avrachenkov, A. Y. Kondratev, and V. V. Mazalov, "Cooperative Game Theory Approaches for Network Partitioning," in *Computing and Combinatorics*. Cham: Springer International Publishing, Jul. 2017, pp. 591–602.

[35] C. T. Do, N. H. Tran, C. Hong, C. A. Kamhoua, K. A. Kwiat, E. Blasch, S. Ren, N. Pissinou, and S. S. Iyengar, "Game Theory for Cyber Security and Privacy," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, p. 30, Jun. 2017.

[36] X. Liang and Y. Xiao, "Game Theory for Network Security," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 472–486, Jul. 2012.

[37] "POCO: Pareto-Optimal Controller Placement." [Online]. Available: http://www3.informatik.uni-wuerzburg.de/poco

[38] "Internet2 Open Science, Scholarship and Services Exchange," (Available: http://www.internet2.edu/network/ose/) (Accessed on Jun. 2019).

[39] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE Journal on Selected Areas in Communications*, vol. 29, no. 9, pp. 1765–1775, 2011.

[40] A. Singla, C. Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking Data Centers Randomly," in *Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI}, San Jose, CA)*, Apr. 2012, pp. 225–238.

Rajat Chaudhary (S'17) received the M.Tech. degree in Information Security and Management from Uttarakhand Technical University, Dehradun, India in 2012. He is currently working toward the Ph.D. degree from Computer Science and Engineering Department, Thapar Institute of Engineering and Technology, Patiala, India. He is currently a JRF in Indo-Poland joint research project funded by Indian and Polish Governments. His research interests in the area of Internet-of-Things, software-defined networks, network functions virtualization, and post-quantum cryptography. Some of his research findings are published in top-cited journals such as IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE INTERNET OF THINGS JOURNAL, IEEE Communication Magazine, IEEE Network Magazine, Elsevier Journal of Network and Computer Applications, Elsevier Computers & Security and various International top-tiered Conferences such as IEEE ICC, IEEE GLOBECOM, IEEE ACM MobiHoc, IEEE WiMob, etc.

Neeraj Kumar (M'16, SM'17) received the Ph.D. degree in Computer Science from Shri Mata Vaishno Devi University, Katra, India. He was a Postdoctoral Research Fellow with the Coventry University, Coventry, U.K. He is currently a Full Professor with the Computer Science and Engineering Department, Thapar Institute of Engineering and Technology, Patiala, India & Department of Computer Science and Information Engineering, Asia University, Taiwan and King Abdul Aziz University, Jeddah, Saudi Arabia. He has authored or coauthored more than 300 technical research papers in leading journals and conferences from IEEE, Elsevier, Springer, John Wiley, etc and also four books from Springer and CRC Press. Some of his research findings are published in top-cited journals such as IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, IEEE TRANSACTIONS ON CONSUMER ELECTRONICS, IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY. He is an Associate Technical Editor of IEEE Communication Magazine, IEEE Network Magazine, Elsevier Journal of Network and Computer Applications, and ComCom. He is in the 2019 list of the highly cited researchers in WOS. He was the recipient of many prestigious awards from IEEE.