On Placement of Controllers and Hypervisors in Software Defined Networks

Bala Prakasa Rao Killi

On Placement of Controllers and Hypervisors in Software Defined Networks

Thesis submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

by

Bala Prakasa Rao Killi

Under the Supervision of

Prof. S. V. Rao

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
Guwahati 781039, India
November 2018

Dedicated to

My parents

For their blessings, constant inspiration, and love

Declaration

I certify that

a. The work contained in this thesis is original, and has been done by myself under the general supervision of my supervisor.

b. The work has not been submitted to any other institute for any degree or diploma.

c. Whenever I have used materials (data, theoretical analysis, results) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.

d. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving required details in the references.

Place: IIT Guwahati

Date:

Bala Prakasa Rao Killi
Research Scholar
Dept of Computer Science and Engineering,
Indian Institute of Technology Guwahati,
Guwahati 781039, India.

CERTIFICATE

This is to certify that this thesis entitled " ***On Placement of Controllers and Hypervisors in Software Defined Networks****" being submitted by Mr. Bala Prakasa Rao Killi to the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, is a record of bona fide research work under my supervision and is worthy of consideration for the award of the degree of Doctor of Philosophy of the Institute.*

The results contained in this thesis have not been submitted in part or full to any other university or institute for the award of any degree or diploma.

Place: IIT Guwahati

Date:

Prof. S. V. Rao
Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati,
Guwahati 781039, India.

Acknowledgements

I would like to take this opportunity to thank many individuals who directly or indirectly supported me during my PhD process.

First, I want to thank my advisor Prof. S. V. Rao for his guidance and constant encouragement throughout my PhD. He has helped me understand what research truly is and constantly motivated me to pursue research. Despite his preoccupation with several assignments, he has been kind enough to spare his valuable time and gave me necessary council and guidance. I am thankful for all that I learned from him. I also want to thank my doctoral committee members, Dr. T. Venkatesh, Prof. S. K. Bose, and Dr. Arnab Sarkar, for their valuable comments and suggestions during my PhD. I am also thankful to the faculty, Dr. R. Inkulu, Prof. S. K. Bose, Dr. T. Venkatesh, Dr. Samit Bhattacharya, Dr. Aryabhatta Sahu, Dr. Santosh Biswas, Dr. Susantha Karmakar, Prof. G. Sajith, Prof. Sukanth Pati and Prof. Sriparna Bandopadhyay, for imparting knowledge through various courses. I also want to thank the department technical officers, technical superintendents and administrative staff for their wholehearted and unconditional support.

Next, I would also like to express my heartfelt gratitude to the Director, the Deans, and other management of IIT Guwahati, whose collective effort has made this institute a place for world-class studies and research. I am thankful to all the faculty and the staff of the Department of CSE for the support received. I thank all my friends, Chiranjeevi, Rakesh, Ghalib, Manoj, Hema, Sukarn, Saptharsi, Deepak, Surajit, Rajesh, Pawan, Debanjan, Vignesh, Akash, Subhrendu, Sandeep, Shounak, Panthadeep, Abhishek, Awnish, Sunil, Moustafa, Sanjeev, Amit, Bhidu, Jogen, Ashish and Pappu only to name a few, with whom I have spent most of the time.

Place: IIT Guwahati

Date: **Bala Prakasa Rao Killi**

Abstract

Software defined networking shifts the control plane of forwarding devices to one or more external entities known as controllers. The placement of controllers in the network influences every aspect of a decoupled control plane, from state distribution options to fault tolerance to performance metrics. Determining the number and placement of controllers is an important problem in software defined networking. Failure of a controller results in disconnections between the controller and the switches that are assigned to it. The administrator can reassign each switch of the failed controller to a working controller with enough capacity that is nearest to the switch. However, the reassignment of switches result in a significant upsurge in the worst case latency.

In this thesis, we propose optimization models for the failure foresight capacitated controller placement that avoids disconnections, repeated administrative intervention, and drastic increase in the worst case latency in case of controller failures by maintaining a list of $\mu(> 1)$ reference controllers for every switch. The objective is to minimize the worst-case latency between switches and their μ^{th} reference controllers while satisfying the capacity and closest assignment constraints. First, we design an optimization model for a single controller failure and extend it to multiple controller failures. We also design a variant of failure foresight capacitated controller placement that minimizes the sum of worst-case latencies from switches to their 1^{st}, 2^{nd} ,..., μ^{th} reference controllers. Next, we relax the failure foresight assumption of switches and investigate a capacitated next controller placement strategy that not only considers capacity and reliability of controllers but also plans ahead for controller failures. We design an optimization model for a single controller failure and extend it to multiple controller failures. We also present a simulated annealing heuristic to produce fast and viable solution on large networks.

When deploying controllers in real networks, large networks such as wide area networks are always partitioned into several smaller ones. To this end, we propose a controller placement strategy that partitions the network using k-means algorithm with cooperative game theory based initialization and deploys a controller in each

of the partitions. We model the partitioning of the network into subnetworks as a cooperative game with the set of all switches as the players of the game. The switches try to form coalitions with other switches so as to maximize their value. We also propose two variants of the cooperative k-means strategy that tries to produce partitions that are balanced in terms of size.

The locations of the hypervisors and controllers together determines the latency of network elements in a virtualized software defined network. In this thesis, we propose two strategies for determining the placement of hypervisors and controller in a virtualized software defined network. The first strategy fixes the hypervisor(s) in the physical network and then determines the placement of controllers in each of the virtual network. It allows the network operator to dynamically add new virtual networks on demand basis. The second approach jointly determines the placement of hypervisors in the physical network and controllers in each virtual network.

All the proposed strategies are evaluated on various networks from the Internet Topology Zoo and Internet 2 OS3E. Results demonstrated that, it is possible to avoid disconnections, repeated administrative intervention, and drastic increase in the worst case latency in case of controller failures by planning ahead for failures. Our proposed models not only performs better in terms of the worst case latency in the event of failures but also in terms of maximum and average inter controller latencies. Results also show that the simulated annealing heuristic is able to achieve near optimal solutions in less than half of the time required by the optimized formulations. The k-means algorithm with cooperative game theory based initialization not only results in solutions that are close to optimal solution but also deterministic in nature. The load aware cooperative k-means strategies results in solutions with less partition imbalance when compared to the load unaware cooperative k-means approach. Determining the placement in each of the virtual network while fixing the hypervisor(s) in the physical network and jointly determining the placement of hypervisors in the physical network and controllers in each virtual network are efficient than determining the hypervisor(s) in the physical network while fixing the controllers in each of the virtual network.

Contents

List of Figures

List of Tables

List of Abbreviations

ACL	Access Control List
AS	Autonomous System
BGP	Border Gateway Protocol
BIP	Binary Integer Program
BIRD	Bird Internet Routing Daemon
CCO	Controller to controller Communication Overhead
CE	Control Element
CP	Controller Placement
CCP	Capacitated Controller Placement
CNCP	Capacitated Next Controller Placement
CO-FFCCP	Combined Objective Failure Foresight Capacitated Controller Placement
DC	Data Center
FIB	Forward Information Base
FFCCP	Failure Foresight Capacitated Controller Placement
ForCES	Forwarding and Control Element Separation
FTCP	Fault Tolerant Controller Placement
HPP	Hypervisor Placement Problem
IBGP	Internal Border Gateway Protocol
IETF	Internet Engineering Task Force
IGP	Interior Gateway
ILP	Integer Linear Program
IS-IS	Intermediate System to Intermediate System
JHCP	Joint Hypervisor Controller Placement
LDP	Label Distribution Protocol
LPM	Longest Prefix Match
MDO	Multiple Data Ownership
MGAP	Multi-objective Generalized Assignment Problem
MPLS	Multi Protocol Label Switching
NAT	Network Address Translator
OLSR	Optimized Link State Routing
OSPF	Open Shortest Path First

LIST OF ABBREVIATIONS

PCE	Path Computing Element
POCO	Pareto Optimal COntroller Placement
QoS	Quality of Service
RCP	Routing Control Protocol
RCS	Route Control Server
RIP	Routing Information Protocol
RR	Routing Reflector
RSVP	Resource Reservation Protocol
SCO	Switch to controller Communication Overhead
SDN	Software Defined Networking
SDNs	Software Defined Networks
SDO	Single Data Ownership
TED	Traffic Engineering Database
VCPP	Controller Placement in Virtualized Software Defined Network
VSDNs	Virtulized Software Defined Networks
WAN	Wide Area Network
Z-DNO	Zone based Distributed Network Optimization

Chapter 1

Introduction

Software Defined Networking (SDN), often referred as "radical idea in networking", is an emerging network architecture in which the control plane functionalities of network elements such as routing, signaling and label distribution are moved to one or more external entities known as controllers. However, the data plane functionalities such as forwarding, queuing and policing remain within the network elements. It also supports programmability of the control plane via open interfaces. Neither programmability of the control plane nor decoupling control from forwarding is new to the networking community. Active networking solution such as Switchware [3] and software routing suites such as Click modular router [4] and Bird Internet Routing Daemon (BIRD) [5] are released prior to SDN that supports programmability of the control plane. Routing Control platform (RCP) [6], Forwarding and Control Element Separation (ForCES) framework released by Internet Engineering Task Force (IETF) [7, 8], 4D design of network architecture [9], Path Computation Element (PCE) architecture [10,11], Ethane [12] and OpenFlow [13] are released prior to SDN that supports separation of the control and data planes. What's new in SDN is, programmability of the control plane is achieved by decoupling data and control planes.

The switches/routers are of no intelligence, simply accept rules from the controller and forward packets. Each forwarding element maintains a flow table whose entries are supplied by the controller. Upon the arrival of a packet, the switch matches the header of the packet with the entries of the flow table. If there is a match, then the action corresponding to the matched entry is performed on the packet, otherwise the switch sends a PACKET_IN message to the controller. The controller then install rules to forward the packet.

A single controller within a network is beneficial as it provides centralized management. That is, routing decisions and policies are based on the global network view. However, this significantly increases the latency of switches that are far away from the controller. Hence, a single controller is clearly a bottleneck in terms of processing power and is the single point of failure for the entire network. Therefore, the control plane is logically centralized, but physically distributed across multiple controllers to satisfy both the response time and fault tolerant requirements. HyperFlow [1] and Onix [14] are logically centralized but physically distributed implementations of the control plane. Fig. 1.1 describes the distributed architecture of HyperFlow. Each controller runs NOX [15] with the HyperFlow application atop, subscribes to control, data, and its own channel in the publish/subscribe system (depicted with a cloud). Events are published to the data channel and periodic controller advertisements are sent to the control channel. Controllers directly publish the commands targeted to a controller to its channel. Replies to the commands are published in the source controller. The publish and subscribe mechanism is implemented using WheelFS [16]. All the controllers maintain a consistent global view to ensure proper network operation. A switch can be assigned to more than one controller (one as primary and others as backup) to ensure reliability.

The placement of controllers in the network influences every aspect of a decoupled control plane, from state distribution options to fault tolerance to

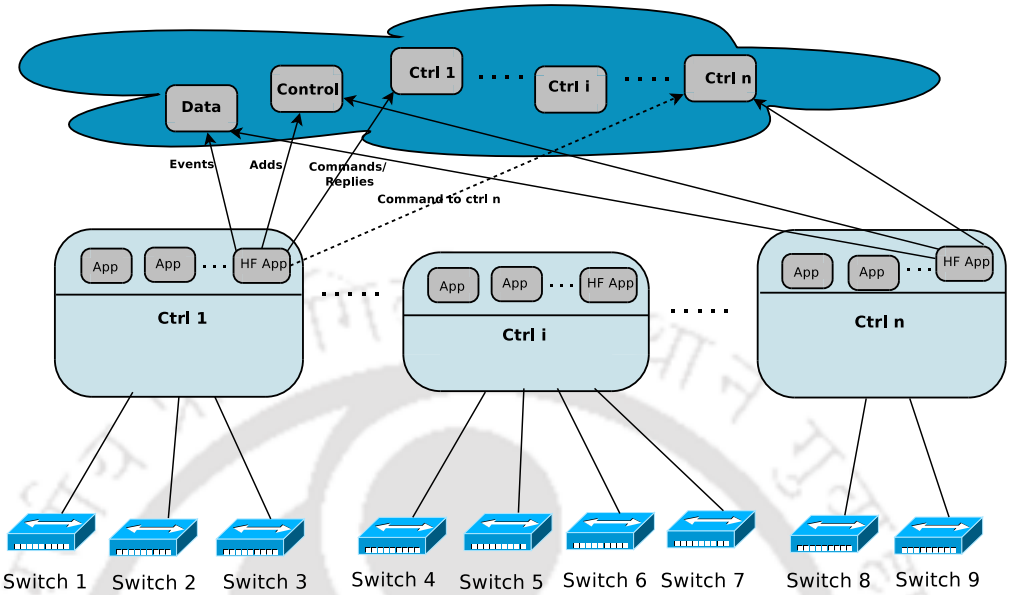


Figure 1.1: High-level overview of HyperFlow [1].

performance metrics. In long-propagation-delay Wide Area Networks (WANs), it places fundamental limits on availability and convergence time. Deploying the controllers at random locations in the network may not result in optimal performance. Therefore, determining the number and placement of controllers is an important problem in Software Defined Networks (SDNs). Heller *et al.* initiated the study of the Controller Placement (CP) problem in SDNs [17]. They analyzed the impact of number of controllers and their placement on latency, and the choice of metric on placement. However, they did not considered the capacity of controllers. Tootoonchian *et al.* varied the controller load from 2^{11} to 2^{15} requests and observed that there is a substantial increase in the processing delay of the controller when the load reaches a threshold [18]. Therefore, the capacity of controllers is an important factor to be considered. Guang Yao *et al.* proposed a controller placement strategy by incorporating a constraint on controller capacity [19]. The controller placement problem mainly deals with the following:

- What is the optimal number and location of controllers for a given network?

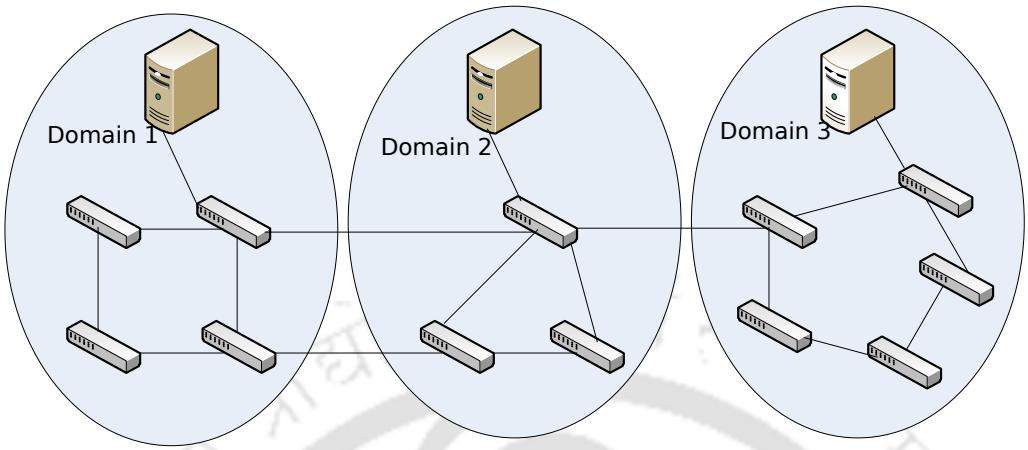


Figure 1.2: Partitioning WAN into multiple domains.

- What is the optimal assignment of switches to the controllers?

When deploying SDN in real networks, large networks such as WAN are always partitioned into several smaller ones due to numerous reasons: privacy, scalability, incremental deployment, security and so on [20, 21]. The central idea is to partition the WAN into multiple domains and assign a controller to each domain as depicted in Fig 1.2. A domain can be a sub-network in a Data Center (DC), an enterprise network or an Autonomous System (AS). Hence, the controller placement problem in a WAN mainly deals with the following:

- How to partition the WAN into multiple domains?
- What is the location of controller(s) in each domain?

Failure of a controller results in disconnections between the controller and the switches that are assigned to it. Since the controller software runs on a server, characterization of server failures in a production environment or cluster give us characterization of controller failures. The server failures can be due to hardware or software failures [22]. In addition, physical network elements may also suffer from hardware problems and stop working. The failure of the link connecting the

controller and the switch where the controller is deployed is also logically same as the controller failure. Hence, the switches are disconnected from controllers whenever there is a controller failure (hardware or software) or a link failure (connecting the controller and the switch where the controller is deployed). Therefore, the reliability of controllers is an important factor to be considered while deploying controllers.

Network virtualization allows multiple tenants to coexist on the shared physical substrate in isolation from one another. The virtualization of software defined networks facilitates to leverage the combined benefits of SDN and network virtualization. That is, each tenant can bring their own controller to control their slice of the network. This can be realized by using the network hypervisor, which we refer as hypervisor for brevity. The hypervisor abstracts the physical substrate into multiple virtual networks and provides isolation among multiple tenants [23]. FlowVisor is the first and seminal hypervisor for sharing and virtualizing SDNs [24]. It is a switch level virtualization platform that enables sharing the same forwarding plane among multiple virtual networks, each with different forwarding logic. Many network hypervisors such as AdVisor [25], VeRTIGO [26], Enhanced FlowVisor [27] etc. are built using the FlowVisor. A complete survey on network virtualization hypervisors for SDN is presented in [28]. In order to meet the scalability and fault tolerant requirements, the hypervisors are generally distributed across the network. In Virtualized SDNs (VSDNs), the PACKET_IN messages of switches must pass through the hypervisor in order to reach the corresponding controller. The latency experienced by a network element is the sum of latency from the network element to the hypervisor and the latency from the hypervisor to the controller corresponding to the network element. Further, the locations of the hypervisors and controllers together determine the latency of network elements in VSDNs. Therefore, determining the placement of controllers and hypevisors are two important problems in VSDNs.

In this thesis, we focus on leveraging planning ahead for controller failures in minimizing worst case latency in the event of controller failures. We address the problem of controller placement with planning ahead for failures with and without failure foresight assumption of switches, so that the worst case latency in the event of controller failures is minimized while satisfying constraints such as placement, demand, closest assignment and multi-level closest assignment. For a wide area network, the controller placement strategy is addressed using k-means network partition approach with cooperative game theory based initialization. The placement of both the controllers and hypervisors in VSDNs is important to reduce the latency experienced by the switches. For this, we address various approaches for minimizing the latency of a network element in VSDNs.

1.1 Motivation of the Research Work

Given a set of switches, an integer indicating the number of controllers to be deployed, and a set of locations for installing controllers as input, the Capacitated Controller Placement (CCP) select a subset of locations to deploy controllers and assign switches to them so as to minimize the worst-case latency from any switch to its controller while satisfying the capacity constraints. The CCP corresponds to the capacitated p-center problem [29]. It is formulated as follows:

$$\min z$$

Subject to:

$$\sum_{j \in P} y_j = p \tag{1.1}$$

$$\sum_{j \in P} r_{ij} = 1 \quad \forall i \in S \tag{1.2}$$

$$r_{ij} \leq y_j \quad \forall i \in S \quad \forall j \in P \tag{1.3}$$

$$\sum_{i \in S} L_i r_{ij} \leq U_j y_j \quad \forall j \in P \tag{1.4}$$

$$z \geq \sum_{j \in P} d_{ij} r_{ij} \quad \forall i \in S \tag{1.5}$$

$$y_j, r_{ij} \in \{0, 1\} \quad \forall i \in S, \forall j \in P \tag{1.6}$$

where y_j and r_{ij} are decision variables, S is the set of switches, P is the set of locations for installing controllers, and p is the number of controllers to be deployed in the network. If a controller is deployed at j then $y_j = 1$, otherwise $y_j = 0$. Similarly, if a switch i is assigned to the controller j then $r_{ij} = 1$, otherwise $r_{ij} = 0$. Constraint (1.1) ensures that exactly p controllers are deployed in the network. Constraint (1.2) guarantees that every switch is assigned to a unique controller. Constraint (1.3) prevents a switch from an assignment to a location where a controller is not deployed. Constraint (1.4) is known as demand constraint, which ensures that the total demand of switches assigned to a controller does not exceed its capacity. Constraint (1.5) ensures that, for every switch, the objective value is greater than the latency between the switch and its controller.

Planning ahead for failures: Deploying controller at a location is equivalent to deploying a server at that location because the controller software runs on a server. Hence, the controller placement problem is off-line in nature which needs to be solved once in the beginning to determine suitable locations for deploying controllers and assignment of switches to these controllers. Failure of a controller results in disconnections between the controller and the switches that are assigned to it. Let C be the set of controllers deployed in the network and $FC \subseteq C$ be the set of controllers that are failed due to hardware or software issues. Note that the hardware failures can have long recovery times compared to the software failures. The disconnected switches ($S^{'} \subset S$) need to be reassigned to one of the working controllers ($WC = C - FC$) with enough capacity in case of failures. The administrator can reassign each switch of the failed controller to a working controller with

enough capacity that is nearest to the switch by solving the following assignment problem:

$$\min \quad z$$

Subject to the following constraints:

$$\sum_{k \in WC} r_{ik} = 1 \quad \forall i \in S^{'} \tag{1.7}$$

$$\sum_{i \in S'} L_i r_{ik} \leq U_k^{'} \quad \forall k \in WC \tag{1.8}$$

$$z \geq \sum_{k \in WC} d_{ik} r_{ik} \quad \forall i \in S^{'} \tag{1.9}$$

$$r_{ik} \in \{0, 1\} \quad \forall i \in S^{'}, \forall k \in WC \tag{1.10}$$

where $U_k^{'}$ is the residual capacity of controller k. Constraint (1.7) guarantees that every switch is reassigned to a unique working controller. Constraint (1.8) ensures that the total demand of switches reassigned to a controller k does not exceed its residual capacity $U_k^{'}$. Therefore, the goal is to reassign the switches of failed controllers to other working controllers with enough spare capacity so as to minimize the worst case latency. This reassignment generally requires administrative intervention[1]. However, the reassignment of switches result in a significant upsurge in the worst case latency. Therefore, we think there is a need for planning ahead to avoid disconnections, repeated administrative intervention, and drastic increase in the worst case latency in case of controller failures.

[1]Self-Organizing Networks (SONs) replace human intervention through automation. SON is organized without any external or central dedicated control entity [30]. Self organization also appears in the context of failure resilience and network restoration [31]. Here, the main goal is to design self-healing or self stabilizing networks that react to link and node failures. Protocols are needed to detect such events quickly and reroute the affected traffic in a self-organized manner.

Judicious initialization of k-means for network partitioning: A single centralized controller may satisfy response time requirements of a small and medium scale network. However, a single controller does not satisfy the fault tolerant requirements of any network as it is the single point of failure. A potential solution is to divide the network into domains and assign a controller to each of these domains. The standard k-means algorithm with random initialization can be used for partitioning the network into different domains. However, the solution produced by standard k-means algorithm varies with different execution instances. Further, it results in solutions that are far from optimal. Hence, we argue that judicious initialization k-means algorithm with results in near optimal solutions.

Jointly determining placement of hypervisors and controllers: Blenk *et al.* proposed a Hypervisor Placement Problem (HPP) that determines the optimal location for deploying the hypervisor in VSDNs [32]. It assumes that the controllers are deployed in each virtual network before determining the location of hypervisor. HPP necessitates the number of virtual networks to be known before hand because it deploys the hypervisors after fixing the controllers in each virtual network. However, the virtual networks are being added dynamically (on demand) in a SDN-based cloud environment. Since the PACKET_IN messages of switches must pass through the hypervisor in order to reach the corresponding controller, the order in which we determine the locations of hypervisors and controllers effects the control plane latency. That is, determining the location of controllers in each virtual network while fixing the hypervisors is preferable than determining the location of the hypervisors while fixing controllers in each virtual network. Moreover, it allows the network operator to dynamically add new virtual networks on demand. However, determining the location of hypervisors and controllers in a sequential way results in a sub optimal performance. Therefore, the location of hypervisors and controllers must be determined jointly to obtain optimal performance. It is beneficial in a

SDN-based private cloud environment where the administrator has control over deployment of hypervisors and controllers.

1.2 Contributions of the Thesis

Based on the several motivation factors mentioned so far, we formulated a set of problems for controller placement and hypervisor placement in SDNs. We briefly describe the problems addressed in this thesis. For each problem, we discuss the optimization formulation and mention the key observations from our evaluation. The details of these are presented in subsequent chapters of the thesis.

1.2.1 Failure Foresight Capacitated Controller Placement in SDNs

Problem statement: What is the optimal placement of controllers and assignment of switches to them such that the worst case switch to controller latency is minimized in the event of controller failures while satisfying the capacity and closest assignment constraints?

We investigate the Failure Foresight Capacitated Controller Placement (FFCCP) problem in SDNs that avoid disconnections, repeated administrative intervention, and drastic increase in the worst case latency in case of controller failures. We formulate it as an optimization model, for a single controller failure using two-indexed decision variables. The objective is to minimize the worst-case latency between switches and their second reference controllers. We extend our optimization model for multiple controller failures. We also design a variant of our model, known as Combined Objective FFCCP (CO-FFCCP) that minimizes the sum of worst-case latencies in failure free and failure scenarios. We solve our models on real-world networks from Internet Topology Zoo and compare the performance obtained with the

standard CCP; one that minimized the worst case latency in failure free scenario. In the following, we summarize the key observations from the numerical evaluation.

- The FFCCP performs better than CCP and CO-FFCCP in the event of failures, because it is optimized for failures.
- The CO-FFCCP neither performs best in failure free case nor in case of failures because it is optimized for neither of these two cases. But, CO-FFCCP performs better than FFCCP in failure free case and better than CCP in case of failures because it minimizes the worst-case latencies with and without failures together.
- The FFCCP results in lesser maximum and average inter controller latencies when compared to CCP and CO-FFCCP. Since the latency between the farthest controllers increases with the number of controllers in the network, the maximum inter controller latency of FFCCP is lower than the CCP and CO-FFCCP when fewer number of controllers are deployed in the network and it is close to CCP and CO-FFCCP when more than the required number of controllers are deployed in the network.

1.2.2 Capacitated Next Controller Placement in SDNs

Problem statement: What is the optimal placement of controllers and assignment of switches to them such that the worst case switch to controller latency is minimized in the event of controller failures while relaxing the failure foresight assumption of switches and satisfying the capacity and closest assignment constraints?

We address a Capacitaed Next Controller Placement (CNCP) problem in SDNs that not only considers capacity and reliability of controllers but also plans ahead for controller failures without assuming that the switches have failure foresight.

In addition to the first reference controller, we also maintain a second reference controller for every switch. It is formulated as an Integer Linear Program (ILP). The objective is to minimize the maximum, for all switches, of the sum of the latency from the switch to the nearest controller with enough capacity (first reference controller) and the latency from the first reference controller to its closest controller with enough capacity (second reference controller). We also proposed a generalized model which can be used to minimize the average latency and extended it for multiple controller failures. Further, we present a simulated annealing heuristic to produce fast and viable solutions. We solve our models and heuristic on real-world networks from Internet Topology Zoo and compare the performance obtained with the standard CCP. In the following, we summarize the key observations from the numerical evaluation.

- We demonstrate that the worst case switch to controller latency of the proposed model is better than CCP in case of controller failure.
- We demonstrate the impact of restricting the maximum inter controller latency on worst case switch to controller latency in failure free and failure scenarios of CNCP. The worst case latency with and without failures of CCP and CNCP with inter controller latency constraints are higher when compared to the CNCP and CCP without these constraints.
- The CNCP results in lesser maximum and average inter controller latencies when compared to CCP. Since the latency between the farthest controllers increases with the number of controllers in the network, the maximum inter controller latency of CNCP is lower than the CCP when fewer number of controllers are deployed in the network and it is close to CCP when more than the required number of controllers are deployed in the network
- The heuristic achieves near optimal solutions in less than half of the time

required by the optimized formulations.

1.2.3 Cooperative Game Theory based Network Partition for Controller Placement in SDNs

Problem statement: How to partition the wide area network using k-means algorithm and deploy controllers while producing close to optimal solutions in deterministic way?

In this problem, we address a controller placement strategy that partition the network using k-means algorithm with cooperative game theory based initialization. It is referred as cooperative k-means for brevity. We model the partitioning of the network into subnetworks as a cooperative game with the set of all switches as the players of the game. The switches try to form coalitions with other switches so as to maximize their value. We also propose two variants of cooperative k-means strategy that tries to produce partitions that are balanced in terms of size. The performance of our proposed strategy is evaluated on networks from Internet 2 OS3E topology and Internet Topology Zoo and compared it with the k-means algorithm with random initialization. In the following, we summarize the key observations from the numerical evaluation.

- The worst case latency of the solutions induced by cooperative k-means outperforms standard k-means and is very close to the optimal solution.

- The standard k-means algorithm randomly selects initial controller locations. Therefore, the solution produced by standard k-means algorithm varies with different execution instances. Since cooperative k-means algorithm does not involve any randomization or probability distribution, it is deterministic in nature.

- The partition imbalance of load aware cooperative k-means strategies is less when compared to the load unaware cooperative k-means approach.

1.2.4 Controller and Hypervisor Placement in Virtualized SDNs

Problem statement: What is the optimal placement of hypervisors and controllers in a virtualized software defined network such that the worst case latency of network element, i.e, the sum of latency from the network element to the hypervisor and the latency from the hypervisor to the controller corresponding to the network element is minimized?

We formulate the problem of determining the placement of controllers in VSDNs while fixing the hypervisor(s) in the physical network as ILP. It is referred as controller placement problem in VSDNs (VCPP) for brevity. We also present an approach for jointly optimizing the placement of hypervisors and controllers in a VSDNs. The objective is to minimize the worst case latency between the network element and its corresponding controller. It is referred as Joint Hypervisor and Controller Placement (JHCP) problem for brevity. We also present a generalized ILP model for JHCP which can be used to optimize other objectives such as the average latency, the maximum average latency, and the average maximum latency. We evaluate the performance of our proposed strategies on the ATT network of Internet Topology Zoo [33] and the results are compared with the existing hypevisor placement [32]. In the following, we summarize the key observations from the numerical evaluation.

- The worst case latency, average latency, maximum average latency and average maximum latency of VCCP is less than HPP when they are optimized for the worst case latency. Similarly, VCCP performs better than HPP when they are optimized for average latency objective.

- The worst case latency of both the methods VCPP and HPP increases with the number of virtual networks. However, the gap between the worst case latencies of VCPP and HPP increases with the number of virtual networks.

- The worst case latency, average latency, maximum average latency and average maximum latency of JHCP is less than HPP when they are optimized for the worst case latency. Similarly, JHCP performs better than HPP when they are optimized for other objectives.

- The complexity of JHCP and HPP in terms of the number of equality constraints and the number of inequality constraints are asymptotically equal. However, the complexity of VCPP is asymptotically smaller than JHCP and HPP.

1.2.5 Organisation of the Thesis

The rest of the thesis is organized as follows: In the next chapter, we present the background material required to understand the setting in which we addressed the problems discussed. We also present the state-of art literature on controller placement and hypervisor placement in SDNs. In Chapter 3, we address the problem of failure foresight capacitaed controller placement in SDNs that plans ahead for controller failures. We propose an optimization model to reduce the worst case switch to controller latency in case of controller failures. In Chapter 4, we relax the failure foresight assumption of switches (switches do not know the status of controllers) and present various mathematical models while considering single and multiple failures. We also present a simulated annealing heuristic to provide fast and viable solutions on large networks. In Chapter 5, we present a controller placement strategy that partition the network using k-means algorithm with cooperative game theory based initialization. The partitioning of the network into subnetworks is

modeled as a cooperative game with the set of all switches as the players of the game. We also propose two variants of cooperative k-means strategies that tries to produce partitions that are balanced in terms of size. In Chapter 6, we investigate a strategy for determining the placement of controllers in VSDNs while fixing the hypervisor(s) in the physical network. We also present an approach for jointly optimizing the placement of hypervisors and controllers in VSDNs. Finally, the thesis ends with summary and future work in Chapter 7.

Chapter 2

Background and Literature Survey

In this chapter we present some material describing the need for a network architecture and the limitations of the traditional network architecture. We also discuss the architecture of SDN and some of the precursors to SDN that supported programmability of control plane or decoupling control and data planes. Finally, the chapter discusses the state-of-art literature related to controller and hypervisor placement in SDNs.

2.1 Need for New Network Architecture

The emergence of cloud services and server virtualization, the explosion of mobile and social networks, multimedia content, and the growing popularity of big data analytics are among the trends driving the networking industry to reexamine traditional network architectures. Many conventional networks are hierarchical, built with tiers of Ethernet switches arranged in a tree structure. This design made sense when client-server computing is dominant, but such a static architecture is ill-suited to the dynamic computing and storage needs of todays' enterprise data centers, campuses, and carrier environments. Some of the key computing trends

driving the need for a new network paradigm [34] include:

- **Changing network traffic patterns:** The network traffic patterns of the enterprise data centers have changed significantly. Most of the applications generates huge amounts of east-west traffic before returning the data to the user. Therefore, the traditional client-server architecture where most of the communication is between user and server is no longer suitable to current changing traffic patterns.

- **Changing user traffic patterns:** The explosion of mobile and social networks resulted in drastic changes in user traffic patterns. The number of users who are using mobile personal devices such as smart phones, tablets, and notebooks to access applications and content has grown significantly. However, accessing applications and content from anywhere and at any time demands ubiquitous communication.

- **The rise of cloud services:** Emergence of the public and private cloud services resulted in on demand access to infrastructure, applications and other resources. Both the public and private clouds provide self-service provisioning which requires elastic scaling of computing, storage, and network resources. Planning for cloud services must be done in an environment of increased security, compliance, and auditing requirements, along with business reorganizations, consolidations, and mergers that can change assumptions overnight.

- **Increasing demand for Big data applications:** The growing popularity of big data applications which requires parallel processing of massive data on thousands of servers. These servers must be connected and communicated with each other to produce the final result. However, communication between thousands of servers requires a huge amount of network bandwidth.

2.2 Limitations of traditional network architecture

In traditional network architecture, both the control plane and data plane are tightly coupled in the network element. This tight coupling of control and data plane means the hardware and the software innovations of a network element are tightly coupled. Furthermore, network management becomes very tedious, error prone and time consuming process. Some of the limitations of traditional network architecture are given below.

- **Protracted network innovation:** The pace of network innovation is far too slow because hardware and software innovation are bundled together. The path from prototype demonstration to deployment, which involves standardization, incorporation into vendor hardware, user procurement, and installation is a long-delayed process. This time consuming process is an impediment to integrating new technologies, standards and services into the existing architecture. We can expedite the network innovation by decoupling hardware and software innovations.

- **Strenuous network management:** Typical computer networks constitute of variety of the devices such as routers, switches and middle boxes such as proxy servers, Network Address Translators (NATs), and firewalls etc. The network operators are responsible for setting the policies that are in line with the continuous network events. They have to manually transform these high level-policies into low-level configuration commands while adapting to changing network conditions. Often, they also need to accomplish these very complex tasks with access to very limited tools. As a result, network management and performance tuning is quite challenging and thus error-prone. The fact that network devices are usually vertically-integrated black boxes

exacerbates the challenge network operators and administrators face [35].

- **Internet Ossification:** The increasing popularity of cloud computing, mobile and social networks, multimedia content and big data analytics demands dynamic network management, ubiquitous communication, and higher bandwidth of computer networks. The capability of existing computer networks can be enhanced by investing more money on the network infrastructure. However, it will increase the complexity of the network due to the presence of heterogeneous nature and vast number of network elements. Therefore, the Internet has become extremely difficult to evolve both in terms of its physical infrastructure as well as its protocols and performance [35].

- **Inconsistent policies:** To implement a network-wide policy, IT may have to configure thousands of devices and mechanisms. For example, every time a new virtual machine is brought up, it can take hours, in some cases days, for IT to reconfigure Access Control Lists (ACL) across the entire network. The complexity of todays networks makes it very difficult for IT to apply a consistent set of access, security, Quality of Service (QoS), and other policies to increasingly mobile users, which leaves the enterprise vulnerable to security breaches, noncompliance with regulations, and other negative consequences [34].

- **Vendor dependence:** Carriers and enterprises seek to integrate new technologies, standards and services into the existing architecture in rapid response to changing business needs or user demands. However, vendor's equipment product cycle generally takes three years or more. This time consuming process is an impediment to deploying new capabilities and services into the existing architecture. Lack of standard and open interfaces limits the ability of network operators to tailor the network to their individual

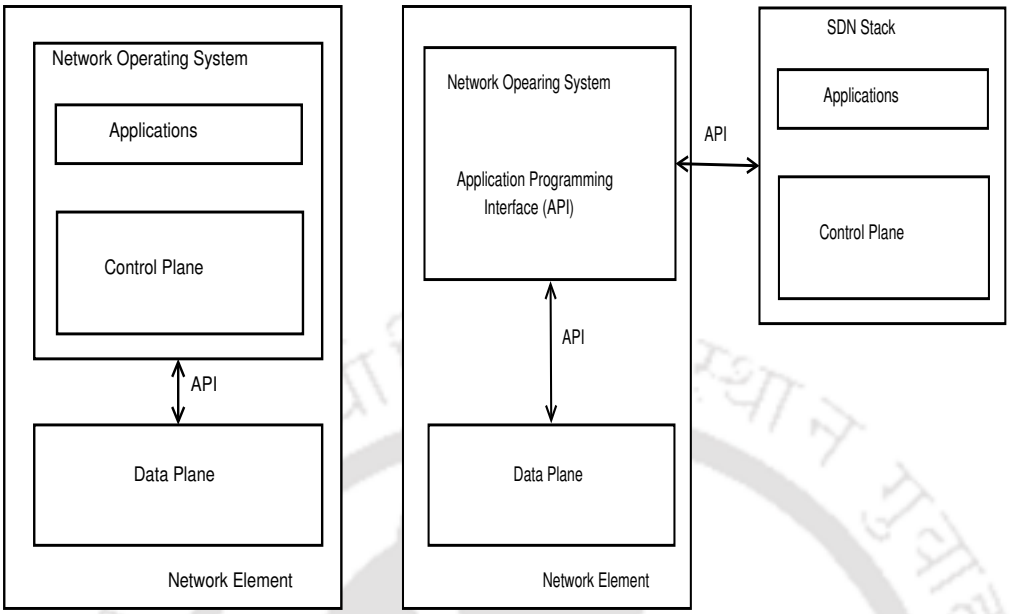
environments [34].

2.3 Software Defined Network

Software Defined Networking (SDN) is a network paradigm that separates the control plane of the network devices from the data plane. The control plane, where routing and signaling decisions are made, is offloaded to a separate entity called the controller. The switches/routers in SDN are of no intelligence, simply accept rules from the controller and forward packets. Additionally, SDN supports programmability of the control plane through open northbound and southbound interfaces. SDN architecture offers numerous benefits over the traditional networking architecture such as increased network infrastructure utilization, centralized network management and so on. Programmability of the control plane allows us to experiment and deploy new ideas and applications without disturbing the existing network. Fundamental structure of a network element in both traditional networking and SDN is shown in Fig. 2.1. The traditional architecture keeps both the control and data planes within the network element. We can observe from Fig. 2.1b that the control plane is not part of the network element in SDN architecture.

The reference architecture of the software defined network is shown in Fig. 2.2. It consists of three layers namely an infrastructure layer at bottom, a control layer in middle, and an application layer at top.

- **Infrastructure layer:** It consists of forwarding elements such as routers, switches and wireless access points etc. These forwarding devices are responsible for collecting network status information and sending it to the controller. Furthermore, they are also responsible for forwarding packets using the rules received from the controller. Forwarding devices communicate with the controller using the northbound interface. Every switch maintains a flow



(a) Traditional network (b) SDN network

Figure 2.1: Basic structure of a network element

table whose entries are supplied by the controller. Each flow table entry consists of three fields namely entry identifier, action, and statistics. Upon a packet arrival, the header of the packet is matched with the flow table entries. If there is a match, then the action corresponding to that entry, such as forward packet on to a port or drop the packet, is performed. If there is no match, then the packet is sent to the controller for processing. The statistics portion of an entry keeps track of information such as the number of packets of each flow, and time since the last packet matched a flow and so on.

- **Control layer:** It consists of one or more controllers for controlling the infrastructure elements. These controllers are responsible for maintaining the global view of the network using the information collected by forwarding devices. Furthermore, they are also responsible for sending forwarding rules to switches. These controllers communicate with the forwarding devices using the northbound interface. When there are multiple controllers in the control layer,

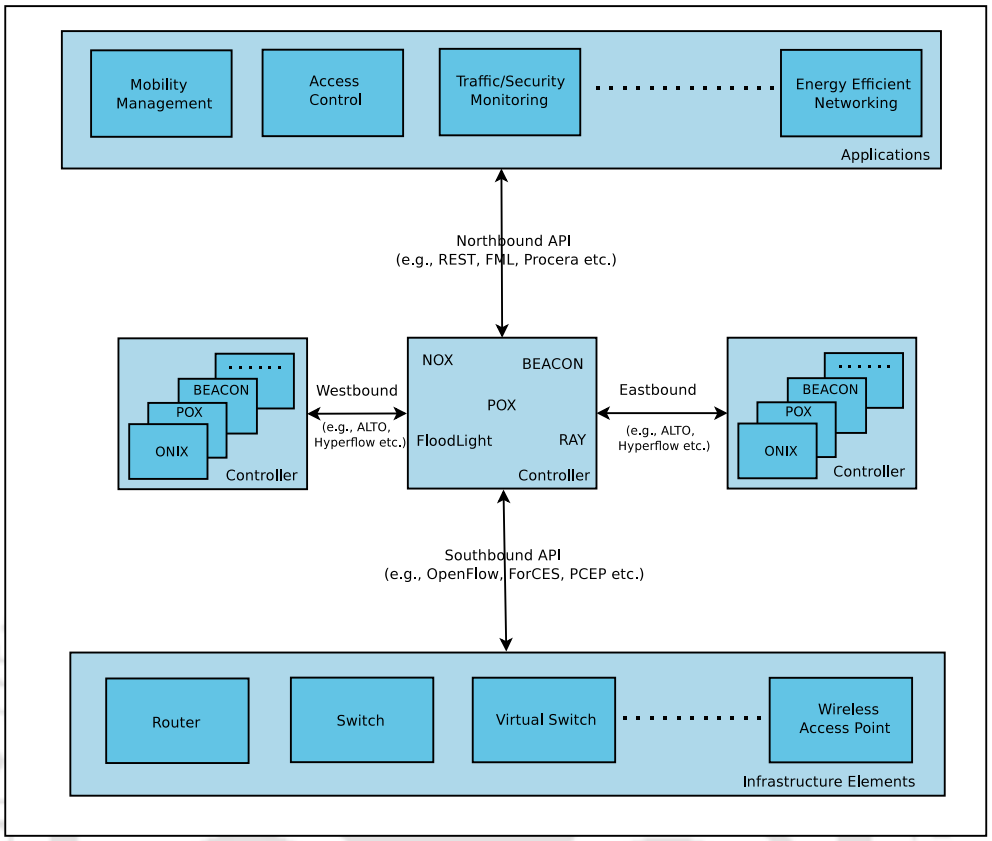


Figure 2.2: Software Defined Network architecture [2]

they communicates with each other using east and west bound interfaces.

- **Application layer:** As we can see from Fig. 2.2, the application layer is present above the control layer. SDN eliminates the usage of middle boxes such as load balancers, firewalls, and NAT by implementing their functionality as an application in software on a server. These applications communicates with the controllers using the northbound interface. Mobility management [36, 37], access control [38], energy efficient networking [39], VM migration [40], adaptive routing, load balancing [41, 42], multicast [43, 44], network virtualization [24], and security monitoring [45, 46] are some of the functionalities that can be implemented in the application layer.

2.4 Precursors to SDN

Neither programmability of the control plane nor decoupling control from forwarding is new to the networking community. Whats new in SDN is, programmability of the control plane is achieved by decoupling data and control planes. In this section, we present the precursors to SDN that supported programmability of the control plane or decoupling control and data planes.

2.4.1 Programmability of the control plane

Active networking solution such as SwitchWare and software routing suites such as Click and BIRD are released prior to SDN that supports programmability of the control plane.

Active networking solutions:

Active networking architecture, emerged in mid 1990s, proposed the idea of programmable networks to expedite the network innovation. SwitchWare is an active networking solution which supported programmable switches that perform customized computations (i.e. per user or application basis) on packets flowing through them [3]. Capsule approach is another active networking solution in which the packets carry procedures along with embedded data.

Software routing suites:

Software routing suites on PC hardware such as Click [4], BIRD [5] and Quagga [47] attempted to create extensible software routers by making network devices programmable. Behavior of these network devices can be modified by loading different or modifying existing routing software. Quagga Routing Suite supports routing protocols such as Routing Information Protocol (RIP), Intermediate System to In-

termediate System (IS-IS), Open Shortest Path First (OSPF) and Optimized Link State Routing Protocol (OLSR), and label distribution protocols such as Multi Protocol Label Switching (MPLS). BIRD is a portable,efficient, and modular Internet routing daemon which supports IPv6, multiple routing tables, multiple routers on a single system and most of the routing protocols.

2.4.2 Control plane and data plane separation

Routing Control Platform (RCP), Forwarding and Control Element Separation (ForCES) framework released by IETF, 4D design of network architecture, Path Computation Element (PCE) architecture, Ethane and OpenFlow are released prior to SDN that supports separation of control and data planes.

Routing Control Platform:

It offloads the task of computing the best Border Gateway Protocol (BGP) route for a destination prefix from router to an external entity known as Route Control Server (RCS) to overcome the limitations of full mesh Internal Border Gateway Protocol (IBGP) and Route Reflector (RR) [6]. RCP constitute three modules which includes the Interior Gateway Protocol (IGP) Viewer, the BGP Engine and the RCS. The RCS computes, on behalf of each router, the best BGP route for each destination prefix using the IGP topology information and the routes to external destinations that are provided by the IGP Viewer and the BGP Engine respectively. RCP achieves simultaneously the scalability benefits of route reflectors and correctness of a full mesh IBGP.

Forwarding and Control Element Separation Framework:

This framework is released by the IETF which aimed at standardizing the communication between forwarding and control elements [7, 8]. Routing protocols

such as RIP, OSPF, and BGP and signaling protocols such as Resource Reservation Protocol (RSVP) and Label Distribution Protocol (LDP) for MPLS can be implemented in the control plane. Functions such as Longest Prefix Match (LPM) forwarder, classifiers, traffic shaper, meter, NAT, etc can be implemented in forwarding devices. This standardization allows for the separation of Control Elements (CEs) from the Forwarding Elements (FEs). Furthermore, it also allows CEs and FEs from different component vendors to inter-operate with each other, which results in increased design choices.

4D architecture:

In order to overcome the limitations of traditional networks being fragile and difficult to manage, Greenberg *et al.* proposed 4D architecture for the network control and management which separates the network control and management into four components (i.e. decision, dissemination, discovery, and data planes) without changing the format of data packets [9]. Replacing today's management plane, the decision plane makes all decisions driving network control, including reachability, load balancing, access control, security, and interface configuration. The dissemination plane provides a robust and efficient communication substrate which is used to move the control information created by the decision plane to the data plane and state identified by the discovery plane to the decision plane. The discovery plane is responsible for discovering neighbors, identifying box level characteristics such as the number of interfaces on a router, the number of Forwarding Information Base (FIB) entries a router can hold, and identifying link characteristics such as capacity of the interfaces. The data plane handles packets based on the state, which includes forwarding table, packet filters, link scheduling weights, and tunnel and network address translation mappings, that is sent by the decision plane. The three major design principles of 4D architecture includes

network-level objectives, network-wide views, and direct control. Furthermore, it also considers the design principles of traditional systems such as scalability, reliability and consistency.

Path Computation Element architecture:

This architecture offloads the task of computing the paths that satisfies specific constraints to an entity known as the Path Computation Element (PCE) [10, 11]. The PCE is an application running either within the network element or on an external entity (out-of-network). The scenarios in which the PCE entity is offloaded to an external server are listed below:

- If constraints based path computation is highly CPU intensive, then the network element which has limited computing power may not be able to compute the paths.
- When the network element may not have the complete topology information to compute the path to a destination. This happens when the destination resides in a different domain.
- If the network elements do not have the control plane functionality.
- If the Traffic Engineering Database (TED) may require a lot of memory, then the network element with limited memory is not a viable option to perform constraints based path computation etc.

Ethane architecture:

Casado *et al.* proposed a network architecture known as Ethane that makes the enterprise networks more manageable and more secure [12]. Ethane follows the lead of 4D architecture [9] and adopted the centralized control of the network rather than distributed control. Ethane offloads control of the network to a centralized

component called the controller. The switches in the networks are not allowed to communicate directly without controller permission. The switches in the network are dumb entities which takes the instructions from the controller to forward the packets. Generally the controller is replicated for fault-tolerance and scalability purposes.

2.5 OpenFlow

Nick McKeown *et al.* proposed OpenFlow that allows researchers to run experiments on the networks they use everyday [13]. OpenFlow enables the administrators to partition the traffic into research and production traffic. While the production traffic can be processed in the same way as before, researchers can control the way research traffic to be processed. In this way, researchers can try new routing protocols, security models, addressing schemes and even alternatives to IP. OpenFlow consists of controller, OpenFlow enabled switch and a protocol for communication between controller and switch. Controller is responsible for adding and removing entries from the flow table of the switch. The list of commodity switches from various vendors that respect the OpenFlow standard are presented in Table. 2.1. The list of controller implementations from various organizations that respect the OpenFlow standard are presented in Table. 2.2. Most of the controller implementations are open source.

Fig. 2.3 depicts the block diagram of an OpenFlow switch. It comprises of three parts namely flow table(s) which is a collection of entries, a secure channel for connecting switch and controller and OpenFlow protocol for communication between controller and switch. Each entry of flow table has three fields namely header, action and statistics. The packet header section of a flow table entry is used to define the flow. The structure of header is shown in Table 2.3:

The action field of a flow table entry specifies how packets should be processed.

Table 2.1: List of commodity switches that respect OpenFlow standard

Vendor	**Switch Model**
Hewlett-Packard	8200zl, 6600, 6200zl, 5400zl, and 3500/3500yl
Brocade	NetIron CES 2000 Series
IBM	RackSwitch G8264
NEC	PF5240 PF5820
Pronto	3290 and 3780
Juniper	Junos MX-Series
Pica8	P-3290, P-3295, P-3780 and P-3920

Table 2.2: List of controller implementations that respect OpenFlow standard

Controller	**Platform**	**Open Source**	**Developer**
NOX [48]	Python, C++	Yes	Stanford
POX [49]	Python	Yes	Stanford
Ryu [50]	Python	Yes	NTT, OSRG group
Beacon [51]	Java	Yes	Stanford
Maestro [52]	Java	Yes	Rice University
Floodlight [53]	Java	Yes	BigSwitch
Jaxon [54]	Java	Yes	Independent Developers
Trema [55]	C	Yes	NEC
OpenDaylight [56]	Java	Yes	Linux Foundation
Helios [57]	C	No	NEC
ovs-controller [58]	C	Yes	Independent Developers

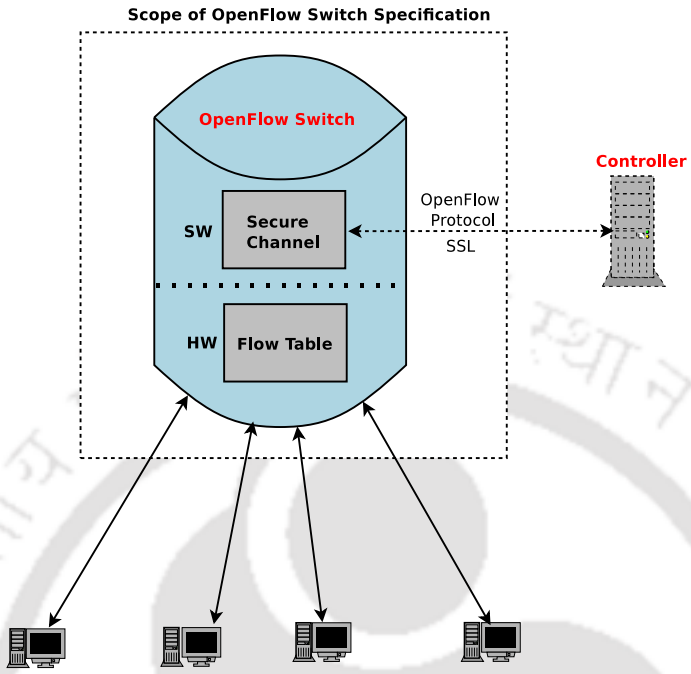


Figure 2.3: Block diagram of an OpenFlow switch

Table 2.3: Header fields used to define the flow

In	VLAN	Etherenet			IP			TCP	
Port	ID	SA	DA	Type	SA	DA	Protocol	Src	Dst

The possible actions for each flow entry are one of the following:

- Forward packets corresponding to the flow to a port(s).
- Forward packets corresponding to the flow to controller.
- Drop packets corresponding to the flow.
- Forward packets corresponding to the flow through normal switch operation.

The statistics part of a flow entry keeps tracks of information such as the number of packets of each flow, and time since the last packet matched a flow etc.

OpenFlow switches can be classified into two types based on the functionality.

- **Dedicated OpenFlow switches:** Does not support normal Layer 2 or Layer 3 functionality. It only supports OpenFlow functionality.
- **OpenFlow enabled general purpose commercial switches:** It supports both normal Layer 2 or Layer 3 functionality in addition to OpenFlow functionality.

2.6 Related Work

In this section, we briefly discuss the survey of related works on controller placement and hypervisor placement in SDNs.

2.6.1 Controller Placement

The controller placement problems in literature can be classified into different categories based on the controller capacity, network traffic characteristics, reliability of network elements, solution approach, and performance metrics. A controller placement strategy can be either uncapacitated or capacitated based on the controller capacity. If a controller placement strategy assumes that the capacity of controllers is limited, then it is capacitated; otherwise it is uncapacitated. Similarly, a controller placement problem can be either static or dynamic depending on the network traffic characteristics. In a dynamic controller placement strategy, the number of controller instances and switch to controller assignment changes with the varying load of switches, whereas in a static controller placement, the number of controllers and switch to controller assignment is fixed, i.e., does not change with the varying load of switches. A controller placement strategy that considers the failure of nodes, controllers or links into account while deploying controllers is known as reliability aware controller placement. Furthermore, the existing literature on the controller placement problems in SDNs can be classified into latency aware,

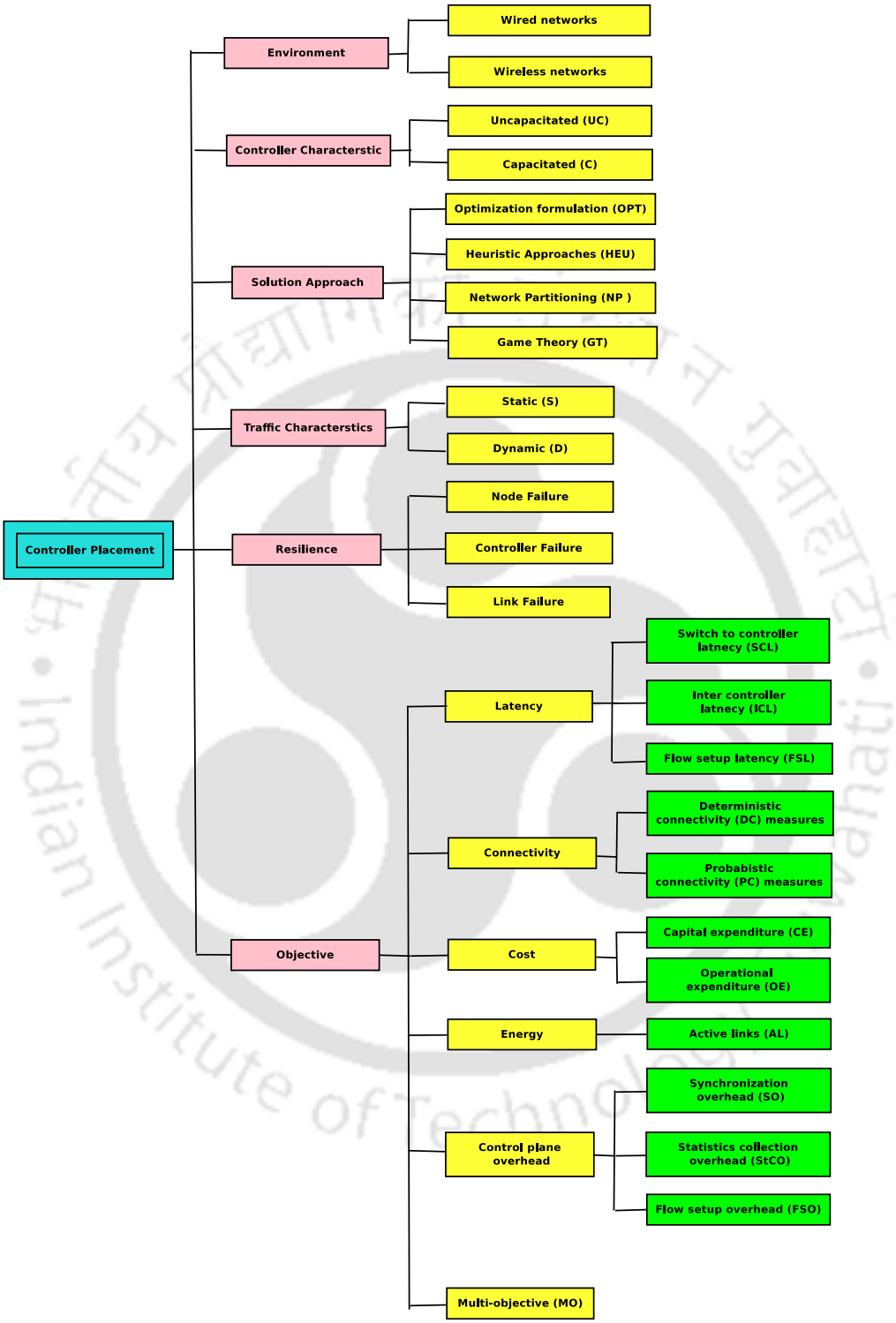


Figure 2.4: Taxonomy of controller placement problems in SDNs

connectivity aware, cost aware, energy aware, quality of service aware and/or control plane overhead aware placement depending on the objective of the problem. The complete taxonomy of controller placement strategies is depicted in Fig. 2.4.

In this section, we summarize some important works that deal with latency, connectivity, cost, QoS and/or control plane overhead aware controller placement strategies and some network partitioning based controller placement strategies in SDNs.

Latency aware controller placement:

Heller *et al.* initiated the study of the Controller Placement (CP) problem in SDNs [17]. Given a network topology, the authors investigated the answers to the following two questions: how many controllers are required and where to deploy them? They considered metrics such as the average case latency and the worst case latency which are determined using k-median and k-center problems respectively. Given a network graph G(V,E) with forwarding devices as nodes and latency as edge weights, the average case and the worst case latency for a placement P of controllers is given as

$$AvgLatency(P) = \frac{1}{n} \sum_{(v \in V)} \min_{(w \in P)} d(v, w) \tag{2.1}$$

$$WorstLatency(P) = \max_{(v \in V)} \min_{(w \in P)} d(v, w) \tag{2.2}$$

where $d(v, w)$ is the shortest path from the switch $v \in V$ to the controller $w \in P$ and n is the total number of forwarding nodes in the topology.

The performance of the placement strategy is evaluated using networks from Internet OS3E and Internet Topology Zoo. They also analyzed the following:

- The effect of placement on latency.
- The effect of the number of controllers on latency.
- Whether one controller is enough for the entire network or not?

Capacity of the controller is a crucial aspect to be considered, because latency between the switches and controllers increase with the load of controllers and heavily loaded controllers are more prone to failures. Guang Yao *et al.* considered capacity of the controllers and proposed a solution to the Capacitated Controller Placement (CCP) problem [19]. The goal is to minimize the worst case latency between switches and controllers while satisfying the capacity constrains. They also proposed a heuristic using linear relaxation and binary search to solve large scale instances of the problem. Their results show that the number of controllers required to avoid overload are lesser when capacity of controllers is taken into account than without considering capacity. Furthermore, it also avoids the load on the busiest controller.

A Pareto Optimal COntroller placements (POCO) framework that determines the pareto optimal frontier with respect to various performance metrics is presented in [59–61]. Multiple competing objectives like latency between switches and controllers, inter controller latency, controller imbalance and resilience against node and link failures are considered. In case of small and medium sized networks, the POCO framework exhaustively evaluates the entire search space to find the pareto optimal frontier (set of all pareto optimal placements). They also proposed a pareto simulated annealing heuristic for solving the problem on large sized networks. Resilience against failures is achieved by deploying controllers at locations that results in better latency in case of failure. Their method is neither optimized for latency in case of failures nor considered capacity of controllers. An extended version of the POCO framework named POCO-PLC is presented in [62] that works under dynamic conditions.

Huque *et al.* proposed a dynamic controller placement strategy that changes the number of controllers depending on the varying load of switches [63]. Since changing the location of a controller dynamically is not practically feasible, they deployed the controller modules at locations that minimize the latency using Bowyer-

Watson algorithm. Here the controller module is a set of controllers that generally run on a multi-core processor. Then they used a dynamic flow management algorithm to activate/deactivate controllers in controller modules depending on load variations.

In [64], the authors proposed a controller placement strategy to minimize the communication overhead between switches and controllers while ensuring that the communication overhead between controllers does not exceed a threshold. They also proposed a controller placement strategy to minimize the communication overhead between controllers while ensuring that the communication overhead between switches and controllers do not exceed a threshold. They formalized the problems using linear programming. Additionally, the nash bargaining game theory is used to compute a trade-off between the above mentioned objectives.

In [65, 66], the authors proposed an analytical model for the response time of controller in distributed SDNs while considering the switch to controller latency and the inter controller latency. They utilized two data-ownership models namely Single Data-Ownership (SDO) and Multiple Data-Ownership (MDO) that provides strong consistency and eventual consistency respectively. They concluded that both the switch to controller latency and the inter controller latency affects the response time of a controller in a SDO model. Further, the switch to controller latency is the only crucial factor that affects the response time of a controller in a MDO model. They proposed two mathematical models for determining the optimal placement of controllers that minimize the reaction time of controllers in SDO model and MDO model respectively. They also proposed a strategy that exhaustively checks all possible placements to determine the pareto optimal placements with respect to the switch to controller latency and the inter controller latency. To efficiently solve the problem on large networks, they presented an evolutionary method to determine pareto optimal placements. oftware Defined Networking In [67], the

authors proposed two strategies for deploying controllers in SDNs while achieving resilience against link and node failures. The first strategy, reliable controller placement with disjoint control paths, maps each switch to a unique controller, but maintains two edge disjoint paths to reach the controller. The second strategy, referred to as reliable controller placement with disjoint control replicas, assigns each switch to two controllers (primary and backup) and maintains edge disjoint paths to reach those controllers. The impact of topology and number of controllers on the average path length is analyzed using networks from SNDlib topology. They also analyzed the expected control path loss and availability by varying the link failure probability.

In [68], the authors proposed two controller placement schemes. All possible link and node failures are considered while formulating the first problem. However, only single link failures are considered while formulating the second problem. The objective is to minimize the worst case latency between switches and controllers under switch and link failures. The authors also proposed a state search algorithm and a greedy algorithm for solving the problem that gives optimal and near optimal solutions respectively. The state search algorithm exhaustively evaluates all possible network states to determine the worst case latency. It takes exponential time which is computationally intractable for larger problem instances. Therefore, they proposed a polynomial time greedy algorithm which iteratively determines the location of controllers. Furthermore, the performance of the algorithms are evaluated using topologies from the Internet Topology Zoo, Internet2, and Cernet2.

A mathematical model that takes the number of controllers as an input and produces optimal traffic distribution in terms of flow setup latency is proposed in [69]. The traffic distribution matrix comprises of fraction of switch demand assigned to a controller. The demand of a switch is served by multiple controllers. Hence, their approach ensures that some fraction of switch demand is not affected in

the event of controller failures. The same authors proposed a mathematical model to generate a short term plan that adjusts the traffic distribution fractions with the dynamic traffic conditions [70]. They also introduced a model that adjusts the traffic distribution fractions in the event of a controller failure. Further, they presented algorithms for computing long term traffic distribution, short term traffic adjustment, traffic adjustment in the event of controller failures.

To address the limitations of a static switch to controller assignment, Dixit *et al.* proposed a controller architecture that dynamically changes the number of controllers with traffic conditions [71, 72]. They also proposed a switch migration protocol and a load adaptation module to balance the load across controllers with the variation in traffic dynamics. The switch migration protocol ensures liveness, safety and serializability properties by operating in four phases. The load adaptation module computes the load on each controller by periodically collecting load measurements. Then, the module invokes re-balancing and resizing operations which evenly distribute the load across controllers and increase/decrease the number of controllers respectively.

Connectivity aware controller placement:

Zhang *et al.* are the first one to address resilience aspects of the controller placement problem [73]. The authors used the probability of disconnection between nodes as a metric to locate the controllers. They used min-cut algorithm to partition the network into domains, then a controller is deployed at the centroid of each domain. The proposed min-cut based controller placement is compared with greedy based and random controller placement algorithms.

Guo and Bhattacharya *et al.* used interdependence network analysis to improve the resilience of SDNs [74]. They analyzed the cascading failures of nodes due to link and other node failures using the interdependence graph. The authors introduced

an expected fraction of nodes that are both survived and connected to a controller in case of link and node failures as a metric to locate the controller. Since the problem is NP-hard, they used partition and selection approach to find the best placement of controllers. Greedy modular optimization technique is used to partition the networks into k clusters and a controller is placed at the centroid of the cluster. Three different topologies with 50 nodes are generated using igraph library to evaluate their model.

Muller *et al.* proposed a controller placement strategy named survivor that improves the survivability of SDNs [75]. The authors formulated the problem as an ILP model by considering capacity, connectivity, and backup controllers. Capacity of controllers is considered to avoid overload, path diversity to ensure the connectivity and survivability and backup controllers is used to recover from failures. The goal is to maximize the connectivity while satisfying the capacity and placement constrains. In order to maximize the connectivity, controllers are placed at locations having a maximum number of node disjoint paths to the switches. Two heuristics are proposed in which proximity and residual capacity of controllers is used to determine the list of backup controllers. Three different topologies from Internet Topology Zoo are used to evaluate their model. Four different metrics are used, the first two quantify resilience and other two measures overload.

Yannan *et al.* proposed a controller placement strategy that improves the reliability of the control plane [76–78]. The goal is to deploy controllers at locations that minimize the expected percentage of control path loss due to network failures. This problem is shown as NP-hard by reducing it from the dominating set problem. Different heuristics such as simulated annealing, greedy, and random placement strategies are used to solve the problem. The performance of the algorithms are evaluated using topologies from Internet2 OS3E and Rocketfuel projcet. Their results show that simulated annealing performs better than other methods. Furthermore, the results of simulated annealing are used to analyze the

impact of the number of controllers on the reliability and trade off between reliability and latency.

Cost aware controller placement:

Sallahi and St-Hilaire proposed an optimal model for the controller placement which determines optimal number, location, type of controllers, and interconnection between network elements [79]. The goal of the model is to minimize the cost of the network while satisfying capacity and latency constraints. Different factors such as the capacity of controllers, the cost of the links, the cost of switches and controllers, and the latency of the paths are considered in the model. The results shows that, for a fixed number of controllers cost of the network increases with switches, and for a fixed number of switches cost decreases as the number of potential locations for the placement increases. The same authors proposed a mathematical model to minimize the cost of expanding the existing network by adding new switches while satisfying controller capacity and switch latency constraints [80]. It produces number of controllers of a specific type to be added/removed, links of a specific type to be added/removed.

Francisco and Pedro studied Fault Tolerant Controller Placement (FTCP) problem that aims at achieving five nines of reliability between controllers and switches [81]. They formulated the FTCP as well known fault tolerant facility location problem. The goal is to determine the number and placement of controllers that minimizes the cost while satisfy the reliability constrain. As the FTCP is NP-hard, they proposed a heuristic to solve large scale instances of the problem. The proposed FTCP is evaluated on WAN topologies from the Internet Topology Zoo.

A topology independent for determining the optimal number of controllers in SDNs is proposed in [82]. It dynamically changes the number of controllers in the network with varying load conditions. The authors assumed that each controller

runs on a virtual machine thus facilitating the dynamic addition and deletion of controllers on a server. The objective is to minimize the cost of the controllers which includes CAPEX and OPEX. An upper bound on the delay and utilization of the controller are included as constraints. They solved the global optimization problem for an initial load condition to decide the optimal location of controllers. As load of the network changes, each controller decides whether to add or delete new controllers instances using a non-zero sum game. Each controller computes a payoff value which is a function of current utilization and current delay of the controller. If the current payoff value of a controller is greater than an upper bound, then it offloads all of its switches to neighbor controllers and triggers self deletion. On the other hand, if the payoff value of a controller is less than a lower bound, then it either offload some of its switches for load balancing or triggers a new controller addition.

A controller placement strategy that adjusts the number of controllers with the changing network conditions is proposed in [83]. The authors also presented a framework that contains three modules namely monitoring module, reassignment module, and provisioning module. The monitoring module uses heartbeat messages to monitor the aliveness of controllers and collects statistics from them. The reassignment module is responsible for deciding whether to reassign the switches to other controllers or not based on the statistics collected by the monitoring module. The initial controller provisioning and reassigning switches to other controllers is done by the provisioning module. The problem is formulated as an integer linear program to minimize the weighted sum of switch reassignment cost, communication cost and the flow setup cost. The authors also presented a greedy knapsack and simulated annealing heuristics to solve the problem efficiently.

A controller placement strategy to minimize the number of controllers in the network is formulated as a integer linear program [84]. The authors introduced

constraints for limiting the maximum inter controller latency and the maximum load imbalance across controllers and a constraint for closest assignment between switches and controllers. They also proposed a resilient controller placement strategy that assigns each switch to $1 + \gamma$ controllers for resilience against controller failures. It is formulated as a integer linear program by assuming that the failure probability of controllers is equal. In addition to minimizing the number of controllers, penalty cost for increase in the latency due to controller failures is also included in the objective.

Tanha *et al.* investigated a controller placement strategy to minimize the cost of deploying controllers and the expected cost of routing the traffic from controllers to switches while satisfying controller capacity constraints [85]. It is mathematically modeled as an integer linear program. It corresponds to the capacitated reliable fixed-charge location problem [86]. Reliability against controller failures is achieved by assigning each switch to one primary controller and multiple levels of backup controllers. Their approach is evaluated on various networks from Internet Topology Zoo by assuming that the controllers fail independent of each other. Results demonstrated that increasing the cost of the network increases with the increase in resilience level, i.e., number of backup controllers for each switch. Additionally, the load imbalance across controllers also increases with the increase in resilience level.

The authors in [87] proposed a controller placement strategy that minimizes the number of controllers in the network while limiting the maximum switch to controller latency and inter controller latency. The authors, initially, formulated it as a integer linear program by taking capacity and reliability of controllers into account and extended for single link failures also. They assigned each switch to r controllers for resilience against controller failures. Since assigning switches to the closest controllers results in load imbalance among controllers, they have relaxed the

closest assignment constraint, i.e., the primary controller of a switch is the closest one in terms of propagation latency. Further, they proposed two heuristics that takes exponential and polynomial time respectively. Given an input graph G and parameter r, the polynomial time heuristic works as follows: constructs a complete graph G_o of G as a overlay, remove edges in G_o that violates the inter controller latency constraint, compute all cliques A of size r and $r+1$, for each switch i, determine subset of cliques computed in previous step that contain i and satisfies switch to controller latency constraints, sort the switches in increasing order of the number of associated cliques and finally it determines a feasible maximal clique of an element of A.

Energy aware controller placement:

Alejandro Ruiz-Rivera *et al.* proposed an energy aware controller association algorithm for SDNs named GreCo using Binary Integer Program (BIP) [88]. The goal is to assign switches to the controllers so as to switch off the maximum number of links while satisfying latency, capacity, and connectivity constraints. As the problem is NP-hard, they proposed a heuristic with $O(|v|^5)$ time complexity to solve large scale instances. Four topologies from Internet Topology Zoo are used to evaluate their heuristic and BIP. Results shows that GreCo saves up to 55% of energy during peak hours and the heuristic uses 20% more links compared to the optimal solution.

QoS aware controller placement:

The authors of [89] investigated the controller placement problem while considering topology, the load of switches, and the response time of controftware Defined Networkingollers. They analyzed the impact of the number of switches and the load of switches on the flow setup time and the number of controllers required to reduce the queuing. Finally, they concluded that the controller placement

problem can be transformed to a controller selection problem that adaptively selects controllers depending on the varying load of switches, topology while satisfying the QoS constraints of switches.

In [90], the authors proposed a strategy to minimize the number of controllers with an upper bound on the QoS [90]. The mean response time of a switch is used as a QoS parameter. The latency perceived by a switch is the sum of two times propagation delay between the switch and its controller and the service time of the controller. The service time of a controller is modeled using M/M/1 queuing system. The authors also proposed incremental greedy, primal-dual, and network partitioning heuristics to solve the problem.

In [91], the authors proposed a controller placement strategy that minimizes the average response time and control plane overhead by dynamically changing the number of controllers with varying traffic conditions. It is mathematically formulated as a variant of well known, NP-hard, Multi-objective Generalized Assignment Problem (MGAP) [92]. To solve the problem efficiently on large scale networks, they transformed the variant of MGAP problem to a two phase problem which uses the idea of stable matching in its first phase and uses the cooperative game theory in the second phase. The authors proposed an algorithm with $O(MNlog(N))$ complexity for the first phase using a variant of deferred acceptance algorithm. Further, they proposed a coalition formation algorithm with $O(MN)$ complexity for the second phase. Here, M and N are the number of controllers and switches respectively. The two phase algorithm is evaluated on fat-tree and VL2 topologies using trace based simulations.

Controller plane overhead aware controller placement:

Xu Li *et al.* proposed a zone-based distributed network optimization (Z-DNO) to deploy controllers in very large scale SDNs [93]. To minimize the control

plane overhead, they partitioned the entire network into regions and zones and assigned a dedicated controller to each of them. Note that a region itself may comprises of one or more zones. That is, zone is the smallest logical division of the network. The overhead of the network involves state collection overhead in each zone, coordination overhead for each zone, and interaction overhead between controllers in different zones. However, only the state collection overhead and the interaction overhead is considered in the objective. The authors modeled the Z-DNO using a binary integer programming model which is NP-hard. Therefore, they proposed clustering, partitioning, and assignment heuristics to efficiently solve the problem. The assignment heuristic allows the trade-off between load of controllers and interaction overhead. Su and Hamdi proposed a measurement aware distributed controller placement strategy for SDNs [94]. The objective is to minimize the sum of statistics collection cost and synchronization cost. The authors formulated it as a quadratic integer programming problem. They also presented an approximate algorithm with 1.61 approximation factor.

Network partitioning based controller placement:

Xiao *et al.* considered controller placement problem for the WAN topology [95]. They investigated the answers to the following two questions: given a WAN topology, how to partition the network into domains, and where should the controller go in each domain? They considered the min-max cut function (Mcut) as the metric, which minimizes the inter domain similarity and maximize the intra domain similarity, to partition the network into domains using spectral clustering. The network graph is represented by $G(V, E, W)$, where V represents forwarding devices in the network, E represents latencies, and the weight matrix W represents the bandwidth of the links. The Mcut metric for partitioning the network into k domains

is defined as follows

$$Mcut(SDN_1, SDN_2, \ldots, SDN_k) = \frac{1}{2}\sum_{i=1}^{k}\frac{x_i^T(D-W)x_i}{x_i^T W x_i} \tag{2.3}$$

where D is the degree matrix and x is the n dimensional indicator vector.

They have considered the average case latency shown in (2.1) as a metric to determine the placement. The authors solved the facility location problem to determine the location of the controller within the domain. Their method is evaluated on WAN networks from Internet OS3E, and Internet Topology Zoo.

In [96], the authors proposed an approach for partitioning the network into multiple domains while considering density and latency. They defined the density of a switch i as the number of switches that are less than a threshold distance d_c from i. Additionally, the latency metric of a switch i is defined as the minimum distance between switch i and other switches with higher density. Given an input graph and threshold distance, their method determines the number of clusters using the density and latency metrics of switches. Then, it selects the switches with large latency value as controllers and assigns remaining switches to the same cluster as its nearest neighbor with higher density. Further, they proposed a strategy for partitioning the network into multiple domains by taking controller capacities into account. The performance of their density based approaches are compared with the mincut based method for partitioning [73] and POCO [60]. Disconnections between switches and controllers in case of link failures is used to measure the network reliability.

Table 2.4 compares various controller placement strategies presented in this subsection. For brevity, we used UC, C, S and D to denote Uncapacitated, Capacitated, Static, and Dynamic respectively. The solution approaches OPT, HEU, NP and GT denote the OPTimal, HEUristic, Network Partitioning, and Game Theory respectively. Further, we used SCL, ICL, and FSL to denote Switch to Controller Latency, Inter Controller Latency, and Flow Setup Latency respectively.

Table 2.4: COMPARISON OF CONTROLLER PLACEMENT STRATEGIES

Strategy	Controller Characteristics		Traffic		Solution Approach				Objective
	UC	C	S	D	OPT	HEU	NP	GT	
[17]	✓		✓		✓				SCL
[19]		✓	✓		✓	✓			SCL
[59]	✓		✓		✓				SCL, ICL and Load imbalance
[60]	✓		✓		✓				SCL, ICL and Load imbalance
[61]	✓		✓		✓	✓			SCL, ICL and Load imbalance
[62]		✓		✓					SCL, ICL and Load imbalance
[63]		✓		✓	✓				SCL
[64]		✓	✓		✓			✓	SCL and ICL
[65]	✓		✓		✓	✓			SCL and ICL
[66]	✓		✓		✓	✓			SCL and ICL
[67]	✓		✓		✓			✓	Average path length
[68]	✓		✓		✓	✓			SCL
[69]		✓	✓		✓				FSL
[70]		✓	✓	✓	✓	✓			FSL
[71]		✓		✓		✓			Load balancing
[72]		✓		✓		✓			Load balancing
[73]	✓		✓			✓			Probability of disconnections
[74]	✓		✓				✓		Expected fraction of connections
[75]		✓	✓		✓	✓			Number of node disjoint paths
[76]	✓		✓		✓	✓			Expected fraction of disconnections
[77]	✓		✓		✓	✓			Expected fraction of connections
[78]	✓		✓		✓	✓			Expected fraction of connections
[79]		✓	✓		✓				Capital expenditure
[80]		✓	✓		✓				Capital expenditure
[81]	✓		✓		✓	✓			Capital and operational cost
[82]		✓		✓	✓			✓	Capital and operational cost
[83]		✓		✓	✓	✓			Capital and operational cost
[84]		✓	✓		✓				Capital expenditure
[85]		✓	✓		✓				Capital and operational cost
[87]		✓	✓		✓	✓			Capital expenditure
[88]		✓	✓		✓	✓			Number of active links
[89]		✓	✓		✓				FSL and cost
[90]		✓	✓				✓		Capital expenditure
[91]		✓	✓					✓	Response time and overhead
[93]		✓	✓				✓		Control plane overhead
[94]	✓		✓		✓	✓			Control plane overhead
[95]	✓		✓				✓		Min-max cut
[96]		✓	✓				✓		Control plane overhead

2.6.2 Hypervisor Placement in SDNs

Blenk *et al.* proposed a hypervisor placement problem (HPP) that determines the optimal locations for deploying hypervisors in VSDNs [32, 97]. They also presented the multi-controller switch deployment problem wherein switches support the multi-controller functionality. That is, switches can be assigned to multiple hypervisors/controllers in order to reduce the control plane latency. The performance of HPP is analyzed by considering various controller placement strategies such as deploying at random locations, locations that minimize the worst case latency and locations that minimize the average case latency. They assumed that the controllers are deployed in each virtual network before determining the location of hypervisor(s) which results in sub optimal performance.

2.7 Summary

In summary, we discussed the need for a network architecture and the limitations of the traditional network architecture. We also discussed the architecture of SDN and some of the precursors to SDN that supported programmability of the control plane or decoupling control and data planes. We classified the controller placement problem in SDN also reviewed important literature related to controller placement and hypervisor placement in SDNs. Most of the works in the existing literature did not considered controller failures. Few works considered backup controllers, but they are not planned ahead to optimized the worst case latency in case of failures. Although the existing literature addressed the controller placement problem using network partitioning, to the best of our knowledge, there is no work on controller placement that used cooperative game theory for network partitioning. Furthermore, there is no work in literature that jointly optimizes the placement of hypervisors and controllers in VSDNs.

Chapter 3

Failure Foresight Capacitated Controller Placement in SDNs

The standard Capacitated Controller Placement (CCP) problem selects p locations for installing controllers while taking switch demands and controller capacities into account. More precisely, given the number of controllers to be deployed, it determines simultaneously the location of controllers and the assignment of switches to them. Each switch receives forwarding rules from a unique controller at any point of time. The objective is to minimize the maximum, for all switches, of the sum of the latency from the switch to the nearest controller. However, CCP does not take the reliability of controllers into account while deploying the controllers.

3.1 Motivation

The worst case latency of the CCP without and with failures when three controllers are deployed in Internet 2 OS3E topology [98] is shown in Fig. 3.1. The switches assigned to a controller are shown with the same color and shape as that of the controller. The worst case latency path between the switch and its controller

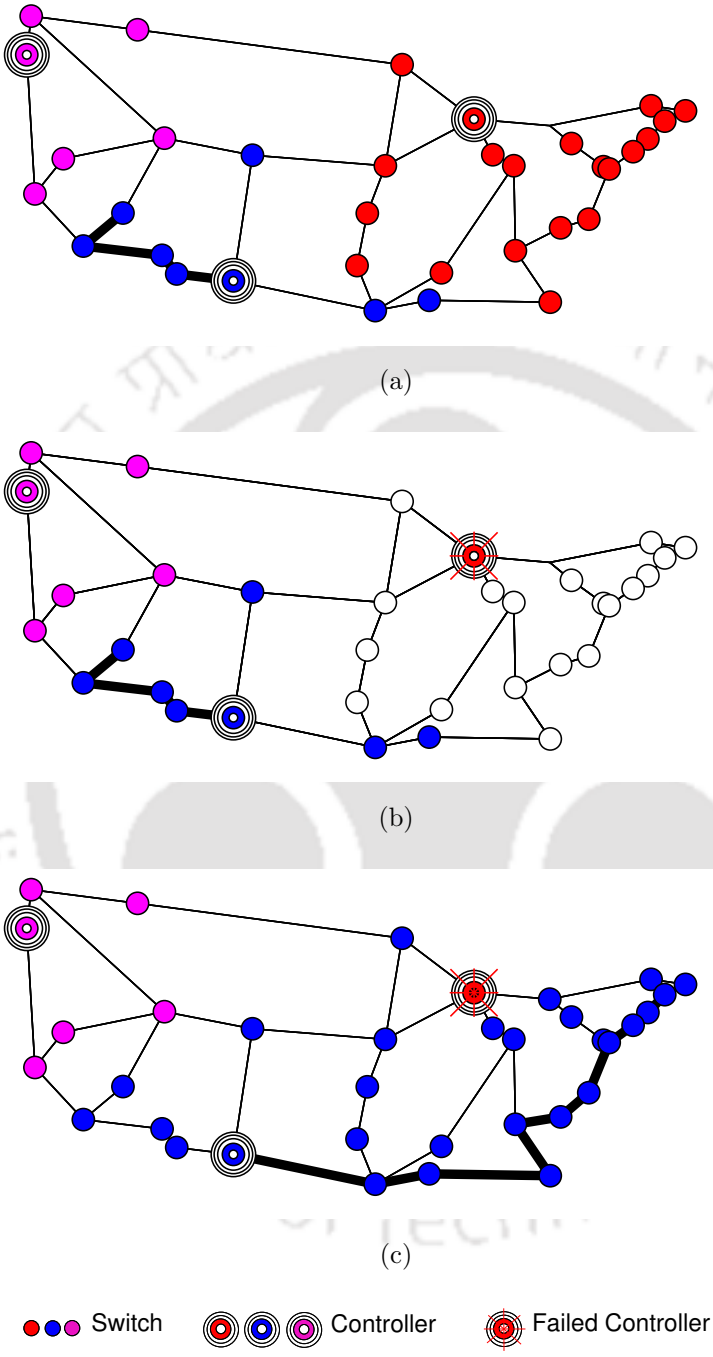


Figure 3.1: Worst case Latency of Internet 2 OS3E topology without planning a) Without failure. b) Disconnections after a failure c) Drastic increase in latency after reassignment.

is highlighted in both cases. The worst case latency in failure free case is 7.68 ms. Failure of a controller results in disconnections between the controller and the switches that are assigned to it as shown in Fig. 3.1b. The disconnected switches need to be reassigned to one of the working controllers with enough capacity in case of failures. This reassignment can be done by solving the assignment problem presented in Section 1.1, which generally requires administrative intervention. However, the reassignment of switches result in significant upsurge in the worst case latency from 7.68 ms to 19.66 ms which can be observed in Fig. 3.1c. Therefore, we think there is a need for planning ahead to avoid disconnections, repeated administrative intervention, and drastic increase in the worst case latency in case of controller failures. To the best of our knowledge, this is the first work that plans ahead for the failure of controllers to avoid a drastic increase in the worst-case latency and disconnections.

In this chapter, we propose three mathematical models for the controller placement problem in SDNs that plans ahead for the controller failures. We avoid disconnections due to controller failures by maintaining a list of $\mu(> 1)$ reference controllers for every switch, where μ is a constant fixed by the network designer. The objective of our first model is to minimize the worst-case latency between switches and their second reference controllers while satisfying the capacity and closest assignment constraints. This helps in reassigning switches of the failed controllers to next reference controllers without a drastic increase in latency. This problem is named as Failure Foresight Capacitated Controller Placement (FFCCP) problem. This problem is a generalization of the Capacitated Second p-center problem [99]. We extend our model to multiple controller failures that minimizes the worst-case latency between switches and their μ^{th} reference controllers while satisfying the capacity, closest assignment and multi level closest assignment constraints. We also introduce a variant of FFCCP that minimizes the sum of worst-case latencies from

switches to their 1^{st}, 2^{nd} ,..., μ^{th} reference controllers.

The remainder of the chapter is organized as follows: In Section 3.2 we present the assumptions, input parameters, and various optimization formulations of the FFCCP problem. Section 3.3 introduces various performance metrics. Numerical results demonstrating the advantages of the proposed models over the existing ones are reported in Section 3.4. We conclude the chapter in Section 3.5.

3.2 Problem Formulation

In this section, we first present the input parameters and state the assumptions of the models. Next, we present the mathematical formulations of the FFCCP problem and also present the performance metrics.

3.2.1 Input parameters

In this section, we define the input parameters used in the formulation. Table 3.1 lists all the input parameters in the model.

We denote the network by a graph $G = (S, E)$ where $S = \{1, 2, \ldots, n\}$ is a finite set of switches/routers and E is the set of physical links between the switches. Let $C = \{c_1, c_2, \ldots, c_p\}$ be the set of controllers to be installed and P be the potential locations for installing controllers. We denote the capacity of c_j by U_j. We denote the minimum propagation latency from switch i to j by d_{ij}, $1 \leq i, j \leq n$, and the number of PACKET_IN[1] messages generated by the i^{th} switch by L_i.

3.2.2 Assumptions

The following assumptions are used in the model.

[1]Packets whose header fields do not match with any of the flow table entries trigger a PACKET_IN message to the OpenFlow controller

Table 3.1: Notations used in FFCCP

Input Parameter	Description
$G(S, E)$	Physical network
S	Set of network elements
E	Physical links between the network elements
p	Number of controllers to be deployed in the network
$C = \{c_1, c_2, \ldots, c_p\}$	Set of controllers for deploying in the network
P	Set of potential locations for deploying controllers
L_i	Number of PACKET_IN messages generated by the switch i
U_j	Capacity of controller c_j
μ	Parameter that decides level of planning
d_{ij}	Latency of the shortest path between i and j

- We assume that all the switches are OpenFlow capable, thus every switch acts as a potential location for deploying a controller.

- We assume that the number of controllers to be deployed in the network is known and given as an input parameter.

- We assume that there can be at most $\mu - 1$ controller failures at a time where μ is a constant fixed by the network designer which depends on the level of planning.

- We assume that the switches know in advance the unavailability of controllers, that is, the switches have failure foresight.

- The controllers are homogeneous in terms of capacity, that is, the number of requests they can process are same.

Table 3.2: Decision variables used in FFCCP formulation

Variable	Description
y_j	=1, if a controller is deployed at location j; =0, otherwise
x_{ij}	=1, if j is the first reference controller of switch i; =0, otherwise
w_{ij}	=1, if j is the second reference controller of switch i; =0, otherwise

3.2.3 FFCCP for single controller failure

Here, we formulate the FFCCP by assuming that at most one controller can fail at any time. Therefore, the goal is to select a subset $Q \subseteq P$ of locations of size p to deploy controllers and assign switches to them so as to minimize the worst-case latency from any switch to its second reference controller while satisfying the capacity constraint and the closest assignment between switches and their first reference controllers.

$$\min_{\substack{Q \subseteq P \\ |Q|=p}} \{z\} = \min_{\substack{Q \subseteq P \\ |Q|=p}} \left\{ \max_{i \in S} \delta(i,j) \right\} \tag{3.1}$$

where $\delta(i,j)$ represents the minimum delay from switch i to its nearest second reference controller at j.

Let the decision variables used in the formulation are y_j, x_{ij}, and w_{ij}, where $i \in S, j \in P$. The variable y_j specifies whether a controller is installed at location $j \in P$. Note that the variable is set to one if a controller is deployed at location j. For a switch i, the variable x_{ij} is set to one if j is the first reference controller of switch i, otherwise set to zero. The variable w_{ij} is set to one if j is the second reference controller of switch i, otherwise set to zero. Table 3.2 lists all the decision variables used in the formulation.

The FFCCP is formulated as follows:

$$\min \quad z$$

Subject to the following constraints:

$$\sum_{j \in P} x_{ij} = 1 \quad \forall i \in S \tag{3.2}$$

$$\sum_{\substack{j \in P \\ j \neq i}} w_{ij} = 1 \quad \forall i \in S \tag{3.3}$$

$$\sum_{j \in P} y_j = p \tag{3.4}$$

$$x_{ij} + w_{ij} \leq y_j \quad \forall i \in S, j \in P \tag{3.5}$$

$$\sum_{\substack{a \in P \\ d_{ia} \leq d_{ij}}} x_{ia} \geq y_j \quad \forall i \in S, j \in P \tag{3.6}$$

$$\sum_{i \in S} L_i x_{ij} + \sum_{i \in S} L_i w_{ij} \leq U_j y_j \quad \forall j \in P \tag{3.7}$$

$$z \geq \sum_{j \in P} d_{ij} w_{ij} \quad \forall i \in S \tag{3.8}$$

$$y_j \in \{0, 1\} \quad \forall j \in P \tag{3.9}$$

$$x_{ij}, w_{ij} \in \{0, 1\} \quad \forall i \in S, j \in P \tag{3.10}$$

Constraints (3.2) and (3.3) ensure that each switch maintains exactly one first reference controller and second reference controller respectively. Notice that the first reference controller to switch i need not be different from i, if the controller is at location i. Whereas for switch i, the second reference controller must be different from i. Constraint (3.4) guarantees that the number of controllers deployed is equal to p. Usage of the constraint (3.5) is twofold. Firstly, it ensures that j is either the first reference controller or second reference controller to a switch i. Secondly, it avoids assigning switches to controllers that are not active. Constraint (3.6) ensures that the first reference controller of a switch i must be the controller that is closest

to the switch i [100]. It works as follows: if switch i is neither assigned to the controller located at j nor to the controller that is not farther from i than j, i.e., $\left\{\sum_{a\in P,\ d_{ia}\leq d_{ij}} x_{ia} = 0\right\}$, then there can not be a controller at j[2]. Constraint (3.7) ensures that the sum of loads of all switches for which j is either the first or second reference controller does not exceed j's capacity. Constraint (3.8) ensures that the objective value is greater than or equal to the minimum delay from any switch to its second reference controller. Constraints (3.9) and (3.10) enforce the decision variables to be binary.

The above formulation can be extended to the average latency objective by replacing the inequality in (3.8) by an equality as shown below:

$$z = \sum_{i\in S}\sum_{j\in P} d_{ij}w_{ij} \tag{3.11}$$

3.2.4 FFCCP for multiple failures

We extend the formulation presented in the previous subsection to multiple controller failures. The goal here is to select a subset $Q \subseteq P$ of locations of size p to deploy controllers and assign switches to them so as to minimize the worst-case latency from any switch to its μ^{th} reference controller while satisfying the capacity constraint and the closest assignment between switches and their other reference controllers.

$$\min_{\substack{Q\subseteq P\\ |Q|=p}} \{z\} = \min_{\substack{Q\subseteq P\\ |Q|=p}} \left\{\max_{i\in S} \delta_\mu(i,j)\right\} \tag{3.12}$$

where $\delta_\mu(i,j)$ represents the minimum delay from switch i to its nearest μ^{th} reference controller at j.

Let the decision variable r_{ij}^l, $\forall i \in S, j \in P, l = 1, 2, \ldots, \mu$, be defined as follows. For a switch i, the variable r_{ij}^l is set to one if j is the l^{th} reference controller of switch

[2]Note that, all the delay constraints used in this thesis are based on distance. However, the smallest delay is not always with the closest node due to link speed, high volume of traffic, etc.

i, otherwise set to zero.

The FFCCP for multiple failures is formulated as follows:

$$\min \quad z$$

Subject to (3.4), (3.9) and the following constraints:

$$\sum_{j \in P} r_{ij}^{1} = 1 \quad \forall i \in S \tag{3.13}$$

$$\sum_{\substack{j \in P \\ j \neq i}} r_{ij}^{l} = 1 \quad \forall i \in S, l = 2, 3, \ldots, \mu \tag{3.14}$$

$$\sum_{l=1}^{\mu} r_{ij}^{l} \leq y_j \quad \forall i \in S, j \in P \tag{3.15}$$

$$\sum_{\substack{a \in P \\ d_{ia} \leq d_{ij}}} r_{ia}^{1} \geq y_j \quad \forall i \in S, j \in P \tag{3.16}$$

$$\sum_{\substack{a \in P \\ d_{ia} \leq d_{ij}}} r_{ia}^{l} + (r_{ij}^{l-1} + r_{ij}^{l-2} + \cdots + r_{ij}^{1}) \geq y_j, \quad \forall i \in S, \ \ j \in P, \ \ l = 2, 3, \ldots, \mu \tag{3.17}$$

$$\sum_{l=1}^{\mu} \sum_{i \in S} L_i r_{ij}^{l} \leq U_j y_j \quad \forall j \in P \tag{3.18}$$

$$z \geq \sum_{j \in P} d_{ij} r_{ij}^{\mu} \quad \forall i \in S \tag{3.19}$$

$$r_{ij}^{l} \in \{0, 1\} \quad \forall i \in S, j \in P, l = 1, 2, \ldots, \mu \tag{3.20}$$

Constraints (3.13) and (3.14) ensure that each switch maintains exactly one l^{th} reference controller. Notice that the first reference controller to switch i need not be different from i, if the controller is at location i. Whereas for switch i, the l^{th} reference controller, for $l > 1$, must be different from i. Usage of the constraint (3.15) is twofold. Firstly, it ensures that j is only one of the l^{th} reference controllers,

for $l = 1, 2, \ldots, \mu$, to a switch i. Secondly, it avoids assigning switches to controllers that are not active. Constraint (3.16) ensures that the first reference controller of a switch i must be the controller that is closest to the switch i [100]. It works as follows: if switch i is neither assigned to the controller located at j nor to the controller that is not farther from i than j, i.e., $\left\{\sum_{a \in P,\ d_{ia} \leq d_{ij}} r_{ia}^{1} = 0\right\}$, then there can not be a controller at j. Constraint (3.17) guarantees that the l^{th} reference controller, for $l > 1$, of switch i is the l^{th} closest controller to switch i [101]. It works as follows: if a controller located at j is not the $1st, 2nd, \ldots, (l-1)th$ closest controller to a switch i, then either j or some controller closer than j is the l^{th} closest controller to the switch i. Constraint (3.18) ensures that the sum of loads of all switches for which j is the l^{th} reference controller, for $l = 1, 2, \ldots, \mu$, does not exceed j's capacity. If $y_j = 1$, the load on the controller located at j due to the switches having it as l^{th} reference controller, is $\sum_{i \in S} L_i r_{ij}^{l}$. The controller at j can be l^{th} reference controller, for $l = 1, 2, , \ldots, \mu$, to a switch. Hence, the total load on the controller at j is $\sum_{l=1}^{\mu} \sum_{i \in S} L_i r_{ij}^{l}$, which should not exceed j's capacity. If $y_j = 0$, $r_{ij}^{l} = 0$ for $l = 1, 2, \ldots, \mu$ from (3.15). The equation (3.18) is trivially satisfied. Constraint (3.19) ensures that the objective value is greater than or equal to the minimum delay from any switch to its μ^{th} reference controller. Constraint (3.20) enforce the decision variables, r_{ij}^{l}, to be binary.

3.2.5 FFCCP with Combined Objective

CCP minimizes the worst-case latency in failure free case at the expense of increasing the worst case latency in case of failures. FFCCP minimize the worst-case latency in case of failures at the expense of increasing the worst case latency in failure free scenario. Since the network will operate without failures for the major portion of the time, we proposed a variant of FFCCP that is not optimized for failures alone, but optimized for the worst-case latencies with and without failures together. More

specifically, we introduced a variant of FFCCP that minimizes the sum of worst-case latencies from switches to their l^{th} reference controllers, $\forall l = 1, 2, \ldots, \mu$. It minimizes the sum of worst-case latencies from switches to their 1^{st}, 2^{nd} ,..., μ^{th} reference controllers. Since it minimizes the combined worst case latencies of all μ reference controllers, it is named as Combined Objective FFCCP (CO-FFCCP). CO-FFCCP minimize the worst-case latency in failure free and failure cases together. It is formulated as

$$\min_{\substack{Q \subseteq P \\ |Q|=p}} \left\{ z^1 + z^2 + \cdots + z^\mu \right\} = \min_{\substack{Q \subseteq P \\ |Q|=p}} \left\{ \sum_{l=1}^{\mu} \max_{i \in S} \delta_l(i, j) \right\}, \tag{3.21}$$

where $\delta_l(i, j)$ represents the minimum delay from switch i to its nearest l^{th} reference controller at j and z^l, for $l = 1, 2, \ldots, \mu$, is the worst case latency between switches and their l^{th} reference controllers.

Subject to (3.4), (3.9), (3.13)-(3.18), (3.20) and

$$z^l \geq \sum_{j \in P} d_{ij} r_{ij}^l \quad \forall i \in S, l = 1, 2, \ldots, \mu \tag{3.22}$$

3.2.6 Controller Failover

OpenFlow v1.2 and above allows each controller to be in one of the three roles for a switch: master, equal, or slave. Controllers that are in either master or equal role for a switch receive the PACKET_IN messages sent by the switch. Initially, the first reference controller of a switch is in master role and all other reference controllers of that switch are in slave role. Detecting failure of controllers is an essential part in controller failover, which can be done using the methods proposed in [1, 102–110]. Controller failover can be performed using the pre-partitioning failover method proposed in [109]. The following controller failure detection and controller failover method can be used for FFCCP. Each controller in the network periodically exchange heartbeat messages with all other controllers to keep track of liveness. These heartbeat messages are useful in checking the liveness of the

controllers. Failure of a controller is detected by all other working controllers if they do not receive three consecutive heartbeat messages from the controller. Two controllers j and k are said to be peers if they are the l^{th} and m^{th} reference controllers respectively for a non empty subset of switches where $1 \leq l \neq m \leq q$. The l^{th} reference controller ($2 \leq l \leq \mu$) sends a role request message to change its role from slave to master to all the switches in the peering domain when it detects the failure of other $(l-1)$ peering controllers. It also periodically sends failover messages to other $(l-1)$ peering controllers. When one of the $(l-1)$ peering controllers recovers from failure, it sends a role request message to all the switches in the peering domain to change its role from slave to master. Note that whenever a switch receives a role request message from a controller to change its role from slave to master, it does not only changes the role of the requesting controller from slave to master but also changes the role of other peering controllers from master to slave if any. If all μ reference controllers of a switch fails, then it can fall back to traditional local control. However, the switch should support both OpenFlow pipeline and traditional Ethernet operation, i.e., the switch should be hybrid.

3.3 Performance metrics

We introduce two metrics for comparing FFCCP and CCP in failure free case and failure case, which we refer as worst case latency and maximum worst case latency for brevity, in (3.23) and (3.24) respectively. The worst case latency defined in (3.23) is the maximum, for all switches, of the latency from the switch to its nearest controller. The maximum worst case latency defined in (3.24) is the maximum, for all possible failure scenarios, of the worst case latencies. Hence, we have to consider all possible failure scenarios while determining the maximum worst case latency.

$$L_{worst} = \max_{i \in S} \left\{ \min_{j \in C} d_{ij} \right\} \tag{3.23}$$

$$L^{F}_{max\_worst} = \max_{f \in F} \left\{ \max_{i \in S} \left\{ \min_{k \in C} d^{f}_{ik} \right\} \right\} \tag{3.24}$$

Here, F is the set of all possible failure scenarios, n is the number of switches in the network, and d^{f}_{ik} is the latency from switch i to controller k under a failure scenario f.

Moreover, we also introduce two metrics for comparing the inter controller latency of FFCCP and CCP in (3.25) and (3.26). The maximum inter controller latency presented in (3.25) is the maximum, for every pair of controllers, of the inter controller latencies. The average inter controller latency defined in (3.26) is the average, for every pair of controllers, of the inter controller latencies.

$$ICL_{max} = \max_{j,k \in C} d_{jk} \tag{3.25}$$

$$ICL_{avg} = \frac{1}{|C|} \sum_{j,k \in S} d_{jk} \tag{3.26}$$

3.4 Numerical Results

In this section, we solve the proposed FFCCP models using real-world network topologies and compare the performance obtained with the standard capacitated controller placement problem from the literature.

3.4.1 Evaluation Setup

We evaluated our proposed formulations on AT&T and GEANT networks of Internet Topology Zoo [33]. Table 3.3 describes the number of nodes, the number of edges, minimum degree, maximum degree, average degree, and diameter of input networks used for evaluation. Longitude and latitude of the nodes are used to determine the propagation latencies. The demand of the switches is randomly generated between 100K and 400K requests/second [19, 111]. We assumed that each controller runs on a single server with 10 Gbps access bandwidth. Since the size of a PACKET_IN

Table 3.3: Characteristics of input networks

Parameter/Network	AT&T	GEANT
Nodes	25	40
Edges	57	61
Min degree	2	1
Max degree	10	10
Avg degree	4.56	3.05
Diameter	24.1 ms	28 ms

message according to OpenFlow $v1.2$ is 160 bytes, the capacity of controllers is set to 7800 Kilo packets/s. We considered all possible failure scenarios while determining the worst case latency in case of failures [99], [61]. The integer linear programming input file is generated from MATLAB [112] and sent to the CPLEX optimizer 12.6.2 [113].

3.4.2 Results

The worst case latency defined in (3.23) is used to compare the performance of CCP, FFCCP, and CO-FFCCP in failure free case. We used the maximum worst case latency defined in (3.24) as a metric to compare CCP, FFCCP, and CO-FFCCP in case of controller failures. The maximum worst case latency of our proposed method in case of controller failures refers to the maximum, for all switches, the latency from the switch to its second reference controller when $\mu = 2$ and the latency from the switch to its third reference controller when $\mu = 3$ and so on. We evaluated the worst case latency for all possible failure scenarios (i.e failure of each individual controller when $\mu = 2$ or all possible combinations of two controller failures when $\mu = 3$ and so on) and the maximum among these is taken as the maximum worst case latency in case of failures. The same procedure is applied to CCP while determining the worst

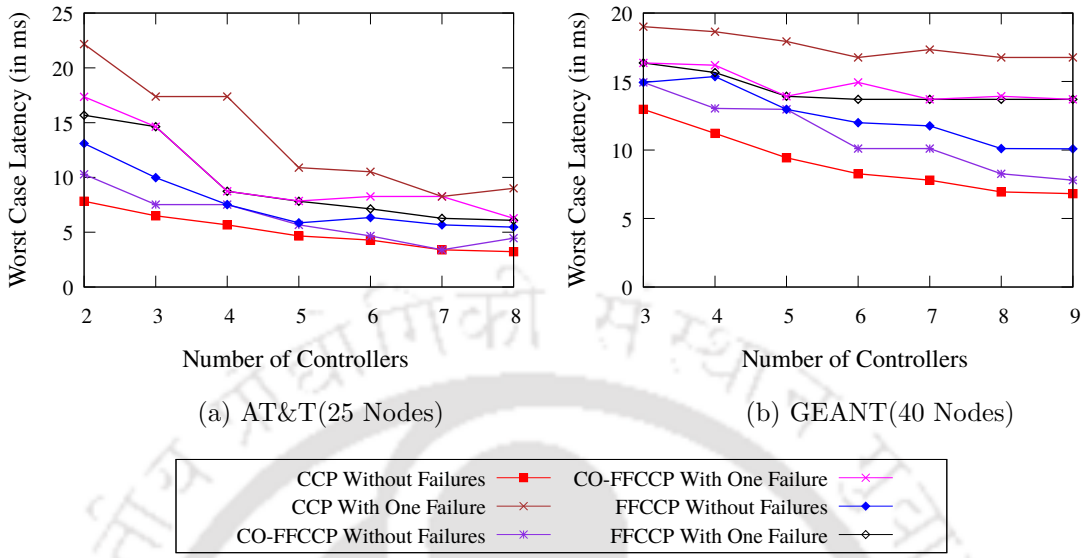


Figure 3.2: Worst case switch to controller latency of CCP, FFCCP, and CO-FFCCP with one controller failure

case latency in case of failures. Since exhaustive search of all failure scenarios is sufficient to determine the maximum worst case latencies in case of failures, we did not explicitly considered the spatial and temporal distribution of controller failures. Hence, the results we obtain are deterministic in nature.

The comparison between the performance of the CCP, FFCCP, and CO-FFCCP with one controller failure (i.e. $\mu = 2$) for different p values starting from the minimum number of controllers required is shown in Fig. 3.2. Please note that the minimum number of controllers for topology depends on the number of nodes and the number of reference controllers for each node. In case of controller failure, there is a drastic increase in the worst-case latency of CCP and it is much higher than that of FFCCP. Due to lack of backup controllers and planning ahead for the failures in CCP, requests from switches of failed controllers need to be serviced by other controllers with enough capacity, which need not be next nearest. This in turn increases the worst-case latency drastically. However, the increase in the worst-case latency of FFCCP is much lower, because of planning ahead for the failures. FFCCP

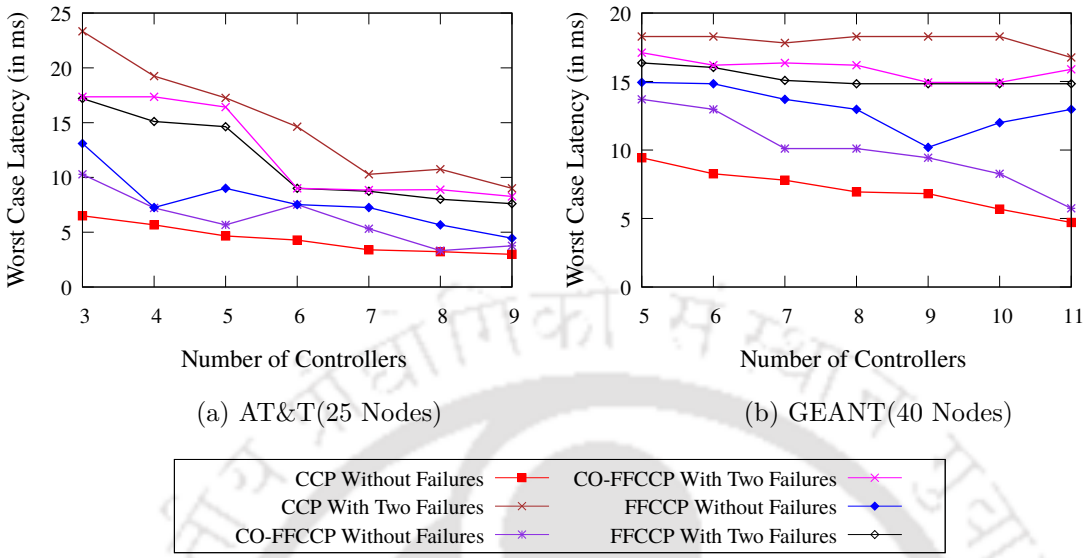
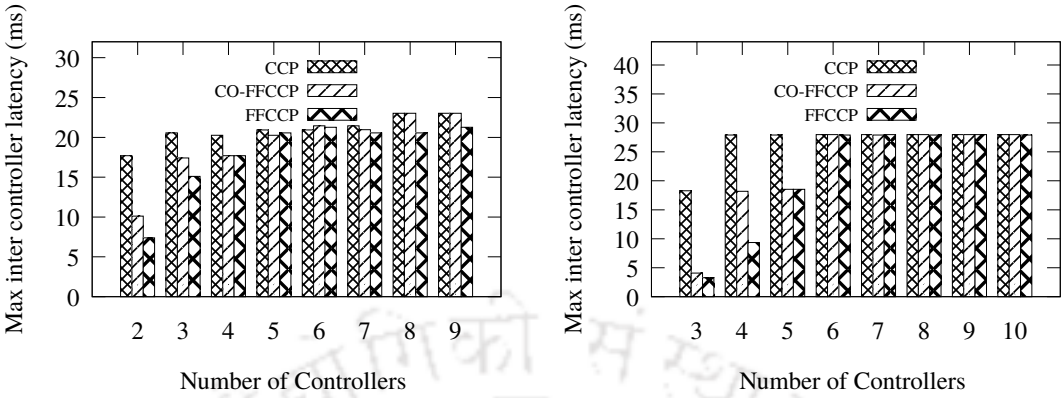


Figure 3.3: Worst case switch to controller latency of CCP, FFCCP, and CO-FFCCP with two controller failures

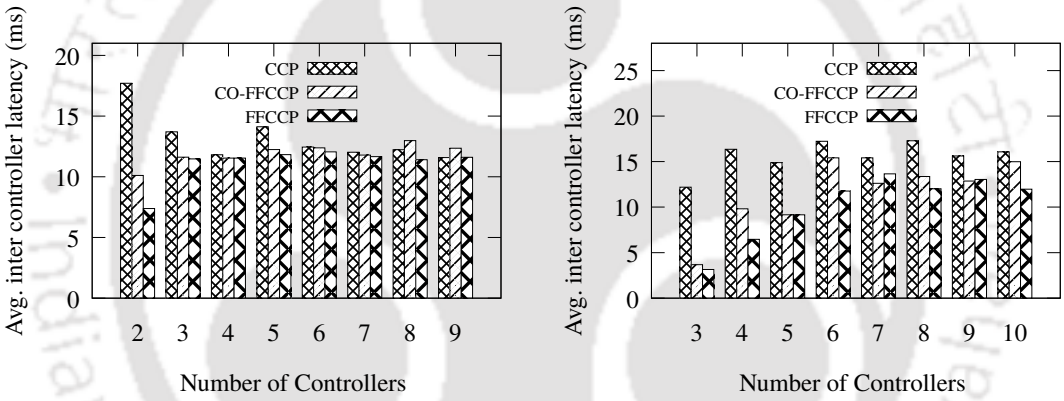
performs better than CO-FFCCP also, because FFCCP is optimized for failures, whereas CO-FFCCP is not optimized for failures alone, but it is optimized for the worst-case latencies with and without failures together. The worst-case latency of FFCCP is higher than CCP, when there is no failure, because FFCCP is not optimized for the worst-case latency between switches to first reference controllers. The results for two controller failures (i.e $\mu = 3$) are given in Fig. 3.3, which follow the similar performance of CCP, FFCCP, and CO-FFCCP. When the number of controllers is much more than the minimum number, there are many working controllers with enough spare capacity to serve the switches of the failed controllers. This leads to decrease in the worst case latency of CCP. That is, the gap between CCP and FFCCP decreases in case of failures as shown in Fig. 3.2a, Fig. 3.3a. However, the same trend can be realized in GEANT when the number of controllers is increased up to 15 because they have more number of nodes.

We can observe that CO-FFCCP neither performs best in failure free case nor in case of failures because it is optimized for neither of these two cases. But,



(a) AT&T(25 Nodes) (b) GEANT(40 Nodes)

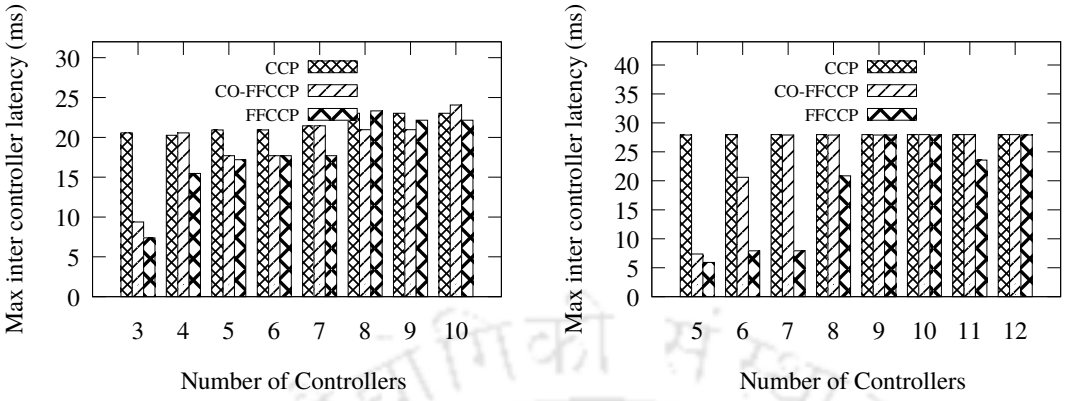
Figure 3.4: Worst case inter controller latency of CCP, FFCCP, and CO-FFCCP with one controller failure



(a) AT&T(25 Nodes) (b) GEANT(40 Nodes)

Figure 3.5: Average inter controller latency of CCP, FFCCP, and CO-FFCCP with one controller failure

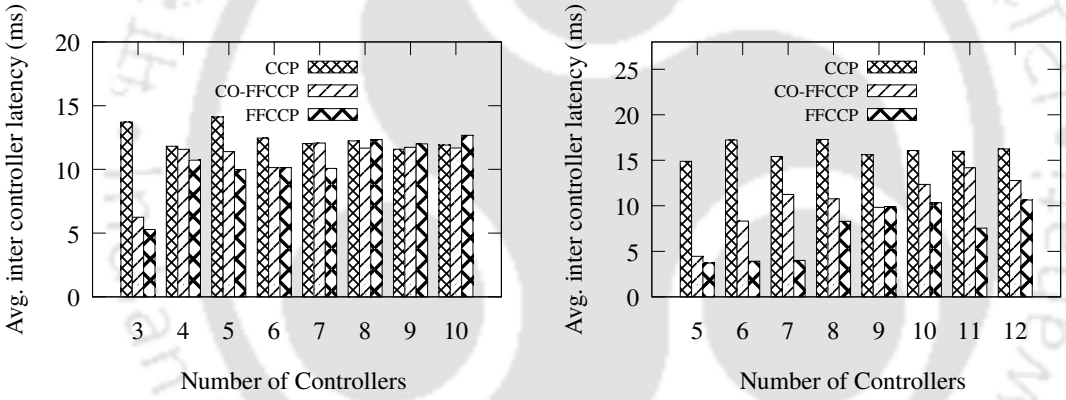
CO-FFCCP performs better than FFCCP in failure free case and better than CCP in case of failures because it minimizes the worst-case latencies with and without failures together. We can also observe that the worst-case latency of CO-FFCCP is close to CCP in failure free case and it is close to FFCCP in case of failures. Hence, CO-FFCCP performs better than FFCCP and CCP when we consider both failure free and failure cases. Thus, FFCCP and CO-FFCCP are generally beneficial than CCP and it is better to plan ahead for the failures. In turn, the number of controllers required by FFCCP and CO-FFCCP is more than CCP because they



(a) AT&T(25 Nodes) (b) GEANT(40 Nodes)

Figure 3.6: Worst case inter-controller latency with two controller failures

maintain μ reference controllers for every switch whereas CCP maintains only one.



(a) AT&T(25 Nodes) (b) GEANT(40 Nodes)

Figure 3.7: Average inter-controller latency with two controller failures

In distributed SDNs, all the controllers must be communicated with each other to maintain a consistent global view of the network so as to ensure proper network operation. The worst inter controller latency and the average inter controller latency characterizes the latency between the farthest controllers in the network and the distribution of controllers across the network respectively. Therefore, the maximum and average inter controller latencies of CCP and FFCCP with single controller failure (i.e. $\mu = 2$) for different p values starting from the minimum number of controllers required is shown in Fig. 3.4 and Fig. 3.5 respectively. We

can observe that the FFCCP results in lesser maximum and average inter controller latencies when compared to CCP and CO-FFCCP. The latency between the farthest controllers increases with the number of controllers in the network. Therefore, the maximum inter controller latency of FFCCP is lower than the CCP and CO-FFCCP when fewer number of controllers are deployed in the network and it is close to CCP and CO-FFCCP when more than the required number of controllers are deployed in the network which is evident from Fig. 3.4. However, the cost of the network increases when much more than the required number of controllers are deployed in the network. Therefore, the designer of the network typically avoids deploying more than the required number of controllers. We can observe similar trends with respect to the maximum and average inter controller latencies of CCP and FFCCP for two controller failures (i.e. $\mu = 3$) from Fig. 3.6 and Fig. 3.7 respectively.

3.4.3 Complexity analysis

The complexity of CCP, FFCCP for single failure, FFCCP for multiple failures and CO-FFCCP is presented in terms of the number of decision variables, and the number of inequality and equality constraints in Table 3.4. Since $P = S$, the number of decision variables in CCP, FFCCP, and CO-FFCCP is asymptotically equal to $O(|S| * |S|)$. The number of equality constraints and inequality constraints in CCP, FFCCP, CO-FFCCP are asymptotically equal to $O(|S|)$ and $O(|S| * |S|)$ respectively. However, the actual number of decision variables and constraints in FFCCP for single failures are nearly two times more than the number of decision variables and constraints in CCP. Further, the actual number of decision variables and constraints in FFCCP for multiple failures and CO-FFCCP are nearly μ times more than the number of decision variables and constraints in CCP. Therefore, FFCCP and CO-FFCCP demands more physical memory for large scale networks when compared to CCP.

Table 3.4: Complexity analysis of CCP, FFCCP and CO-FFCCP

CCP	Number of Decision Variables	$1 + \|P\| + \|S\| * \|P\|$
	Number of equality constraints	$1 + \|S\|$
	Number of inequality constraints	$\|S\| + \|P\| + \|S\| * \|P\|$
FFCCP for single failure	Number of Decision Variables	$1 + \|P\| + 2 * \|S\| * \|P\|$
	Number of equality constraints	$1 + 2 * \|S\|$
	Number of inequality constraints	$\|S\| + \|P\| + 2 * \|S\| * \|P\|$
FFCCP for multiple failures	Number of Decision Variables	$1 + \|P\| + \mu * \|S\| * \|P\|$
	Number of equality constraints	$1 + \mu * \|S\|$
	Number of inequality constraints	$\|S\| + \|P\| + (1 + \mu) * \|S\| * \|P\|$
CO-FFCCP	Number of Decision Variables	$\mu + \|P\| + \mu * \|S\| * \|P\|$
	Number of equality constraints	$1 + \mu * \|S\|$
	Number of inequality constraints	$\mu * \|S\| + \|P\| + (1 + \mu) * \|S\| * \|P\|$

3.5 Conclusion

In this chapter, we investigated the failure foresight capacitated controller placement problem in SDNs that avoid disconnections, repeated administrative intervention, and drastic increase in the worst case latency in case of controller failures. We designed an optimization model, for a single controller failure using two-indexed decision variables. The objective is to minimize the worst-case latency between switches and their second reference controllers. We extended our optimization model for multiple controller failures. The objective is to minimize the worst-case latency between switches and their μ^{th} reference controllers while satisfying the capacity and closest assignment constraints. We also designed a variant of FFCCP that minimizes the sum of worst-case latencies from switches to their 1^{st}, 2^{nd} ,..., μ^{th} reference controllers. The proposed formulations are evaluated on various networks from the Internet Topology Zoo. Our results show that the proposed FFCCP and CO-FFCCP perform better than CCP in case of failures. Simulation results also demonstrated that the CO-FFCCP perform better than FFCCP and close to CCP

in failure free case.

In this chapter, we assumed that the switches know the status of controllers, which is not always possible. In the next chapter, we investigate a controller placement strategy that not only considers capacity and reliability of controllers, but also plans ahead for controller failures without assuming the failure foresight of switches. We seek to minimize the maximum, for all switches, of the sum of two latencies, the latency from the switch to the nearest controller and the latency from the nearest controller to its closest controller.

Chapter 4

Capacitated Next Controller Placement in SDNs

In previous chapter, we assumed that the switches have failure foresight. That is, the switches know the status of controllers, which is not always possible. Here, we relax the failure foresight assumption of switches. Hence, the switch where a controller is deployed knows the availability of the corresponding controller whereas other switches do not know the status of this controller. To the best of our knowledge, this is the first work that plans ahead for the failure of controllers, without assuming failure foresight of switches, to avoid a drastic increase in worst-case latency and disconnections.

In this chapter, we propose a controller placement problem in SDNs that not only considers capacity and reliability of controllers but also plans ahead for controller failures without assuming that the switches have failure foresight. Note that the switch where a controller is deployed knows the availability of the corresponding controller whereas other switches does not know the status of this controller. Thus, the switches always forward the PACKET_IN messages to their respective nearest controller irrespective of whether it is available or not. In addition

to the first reference controller, we also maintain a second reference controller for every switch. The objective is to minimize the maximum, for all switches, of the sum of the latency from the switch to the nearest controller with enough capacity (first reference controller) and the latency from the first reference controller to its closest controller with enough capacity (second reference controller). Henceforth, it is referred as Capacitated Next Controller Placement (CNCP) for brevity. The CNCP corresponds to the capacitated version of the next p-center problem [114,115]. We formulated the CNCP as ILP using two-indexed binary variables. However, it is not straightforward to extend the formulation to the average latency objective. Hence, we formulated the problem as ILP using three-indexed binary variables which can be easily extended for the average latency objective. We also extended the formulation to include multiple controller failures. Furthermore, we presented a simulated annealing heuristic that efficiently solves the problem on large scale networks.

The remainder of the chapter is organized as follows: In Section 4.1 we present the assumptions, input parameters, and various ILP formulations of the CNCP problem. Section 4.2 introduces various performance metrics. A simulated annealing heuristic to efficiently solve the problem on large scale networks is presented in Section 4.3. Numerical results demonstrating the advantages of the proposed model over the existing ones are reported in Section 4.4. We conclude the chapter in Section 4.5.

4.1 Problem Formulation

In this section, we first present the input parameters and state the assumptions of the models. Next, we present the ILP formulations of the CNCP problem and also present the performance metrics.

4.1.1 Input Parameters

The input parameters used in the problem formulation are same as those used in previous chapter. For better readability and completeness, we present the input parameters in this subsection.

The network is represented by a graph $G(S, E)$, where $S = \{s_1, s_2, \ldots, s_n\}$ be the set of switches and E be the set of physical links between switches. Let $C = \{c_1, c_2, \ldots, c_p\}$ be the set of controllers to be deployed and P be the potential locations for deploying controllers. We denote the load of s_i by L_i and the capacity of c_j by U_j. Please note that the load of a switch refers to the number of messages whose header does not match with any of the flow table entries. All unmatched messages are forwarded to the corresponding controller. Let d_{ij} be the minimum propagation latency between switch s_i and controller c_j, for $i \in S, j \in P$.

4.1.2 Assumptions

We have relaxed the failure foresight assumption of switches. All other assumptions from Chapter 3 are used in this problem formulation. For better readability and completeness, we present the list of assumptions in this subsection.

- We assume that all the switches are OpenFlow capable, hence, acts as potential locations for deploying controllers, i.e., $P = S$.

- We assume that the number of controllers to be deployed in the network is known and given as an input parameter.

- We assume that there can be at most $\mu - 1$ controller failures at a time where μ is a constant fixed by the network designer which depends on the level of planning. If the designer plans ahead for two controller failures, then μ is set to 3.

- The controllers are homogeneous in terms of capacity, i.e, the number of requests they can process are same.

4.1.3 CNCP Formulation with two-indexed variables

In this subsection, we assume that at the most one controller fails at any time, i.e., $\mu = 2$. The controller deployed at switch j means the controller is immediately connected to switch j. Thus, the latency from the switch j to the controller deployed at j is negligible. Therefore, this switch is the only entity that forwards the incoming requests and outgoing replies to and from the controller. Thus, the switches always forward the PACKET_IN messages to their nearest controller with enough capacity, which we refer as the first reference controller for brevity, irrespective of whether it is available or not. Note that if a controller is deployed at location j, then it is the first reference controller to the switch j. When PACKET_IN messages arrives at the switch, where the corresponding first reference controller is deployed, the switch forward packets to the first reference controller if it is available else to a controller nearest to the first reference controller, which we refer as the second reference controller for brevity. Here, the controller nearest to the first reference controller j is nothing but the controller other than j that is nearest to the switch to which j is immediately connected and also satisfies the capacity constraint. Hence, the objective is to minimize the maximum, for all switches, of the sum of the latency from the switch to the first reference controller and the latency from the first reference controller to its second reference controller.

$$\min_{\substack{Q\subseteq P\\|Q|=p}} \{z\} = \min_{\substack{Q\subseteq P\\|Q|=p}} \max_{i\in S} \left\{ \min_{j\in P} d_{ij} + \min_{j' \in arg\min_{j\in P} d_{ij}} \min_{\substack{k\in P\\k\neq j'}} d_{j'k} \right\} \tag{4.1}$$

Let the two-indexed decision variables used in CNCP formulation are y_j, r^1_{ij}, r^2_{ij} and w_{jk}, where $i \in S, j \in P$. The variable y_j specifies whether a controller is installed at location $j \in P$. Note that the variable is set to one if a controller is

Table 4.1: Decision variables used in CNCP formulation

Variable	Description
y_j	=1, if a controller is deployed at location j =0, otherwise
r_{ij}^1	=1, if j is the first reference controller of switch i =0, otherwise
r_{ij}^2	=1, if j is the second reference controller of switch i =0, otherwise
w_{jk}	=1, if controllers are deployed at both j and k =0, otherwise

deployed at location j. For a switch i, the variable r_{ij}^1 is set to one if j is the first reference controller of switch i, otherwise set to zero. The variable r_{ij}^2 is set to one if j is the second reference controller of switch i, otherwise set to zero. The variable w_{jk} determines whether controllers are deployed at locations $j, k \in P$. Note that the variable is set to one if controllers are deployed at both the locations j and k.

The CNCP is formulated as follows:

$$\min_{Q \subseteq P, |Q|=p} \{z\}$$

Subject to:

$$\sum_{j \in P} y_j = p \tag{4.2}$$

$$\sum_{j \in P} r_{ij}^1 = 1 \quad \forall i \in S \tag{4.3}$$

$$\sum_{\substack{j \in P \\ j \neq i}} r_{ij}^2 = 1 \quad \forall i \in S \tag{4.4}$$

$$r_{ij}^1 + r_{ij}^2 \leq y_j \quad \forall i \in S, \forall j \in P \tag{4.5}$$

$$w_{jk} d_{jk} \leq \gamma G_d \quad \forall j, k \in P \tag{4.6}$$

$$w_{jk} \geq y_j + y_k - 1 \quad \forall j,k \in P \tag{4.7}$$

$$w_{jk} \leq y_j \quad \forall j,k \in P \tag{4.8}$$

$$w_{jk} \leq y_k \quad \forall j,k \in P \tag{4.9}$$

$$y_j + \sum_{\substack{h \in P \\ d_{ih} > d_{ij}}} r_{ih}^1 \leq 1 \quad \forall i \in S, \forall j \in P \tag{4.10}$$

$$\sum_{i \in S} L_i r_{ij}^1 + \sum_{i \in S} L_i r_{ij}^2 \leq U_j y_j \quad \forall j \in P \tag{4.11}$$

$$z \geq (d_{ij} + d_{jk})(r_{ij}^1 + r_{ik}^2 - 1) \quad \forall i \in S, j,k \in P \tag{4.12}$$

$$y_j \in \{0,1\} \quad \forall j \in P \tag{4.13}$$

$$r_{ij}^1, r_{ij}^2 \in \{0,1\} \quad \forall i \in S, j \in P \tag{4.14}$$

$$w_{jk} \in \{0,1\} \quad \forall j,k \in P \tag{4.15}$$

Constraint (4.2) guarantees that exactly p controllers are deployed in the network. Constraints (4.3) and (4.4) ensures that each switch has unique first and second reference controllers respectively. Constraint (4.4) also specifies that the second reference controller of switch i must be different from the one that is deployed at i. However, the first reference controller of switch i need not be different from the one that is deployed at i. Constraint (4.5) ensures that j is either the first or the second reference controller to a switch i. That is, the first and second reference controllers of a switch must be different. It also checks the validity of assignments. If no controller is deployed at j, i.e., $y_j = 0$, then the assignment of switches to j are not allowed, i.e., $r_{ij}^1 = 0$ and $r_{ij}^2 = 0$. Constraint (4.6) guarantees that the latency between any pair of active controllers is less than γG_d, i.e., the maximum allowable inter controller latency. Here G_d is the diameter of the network graph and γ is a parameter supplied by the designer of the network. Please note that the maximum allowable inter controller latency is expressed as a fraction of the graph diameter to maintain consistency across various network topologies. The variable

$w_{jk} = 1$ if both $y_j = 1$ and $y_k = 1$, i.e., $w_{jk} = y_j y_k$ which makes the formulation non linear. Therefore, we use the constraints (4.7)-(4.9) to avoid the non linear term in the formulation and make it linear. Constraint (4.7) ensures that the variable w_{jk} takes a value one when y_j is one. Constraint (4.8) guarantees that the variable w_{jk} takes a value one when y_k is one. Constraints (4.7)-(4.9) together ensures that the variable w_{jk} takes a value one when both y_j and y_k are one.

Since all the PACKET_IN messages of a switch are forwarded to the first reference controller when there are no failures, we have to ensure the closest assignment between switches and their first reference controllers. We adopted the closest assignment constraint introduced by Wagner and Falkson in [116]. The original constraint is proposed to ensures the closest center behavior in a location-allocation model. Therefore, we can safely use it to ensure that the first reference controller of a switch i is the one that is closest to it as shown in (4.10). It works as follows: if a controller is deployed at j ($y_j = 1$), then the assignment of switch i to any controller farther from i than j is not allowed ($\sum_{h \in P; d_{ih} > d_{ij}} r^1_{ih} = 0$). The closest assignment between a switch and its second reference controller is implicitly ensured by the objective function. Constraint (4.11) is the demand constraint which guarantees that the total demand of the switches served by a controller j does not exceed it's capacity U_j. Here the total demand includes the demand of switches for which j is the first or second reference controller. Constraint (4.12) ensures that, for any switch $i \in S$, the objective value is greater than or equal to the sum of latency from switch i to its nearest controller l and the latency from l to its nearest controller j. When both $r^1_{ij} = 1$ and $r^2_{ik} = 1$, the term $r^1_{ij} + r^2_{ik} - 1$ on the right hand side of (4.12) is positive. Therefore, the right hand side of (4.12) effectively reduces to $(d_{ij} + d_{jk})$ which is equivalent to the sum of latency from switch i to its first nearest controller l and the latency from l to its nearest controller j. In all other cases, the right hand side of (4.12) is non positive. Constraints (4.13) and (4.14) are

integral constraints which ensures that all the decision variables take values either 0 or 1.

The objective of the above formulation is to minimize the worst case latency in case of controller failure. If the objective is to minimize the average latency in case of controller failure, then the straightforward extension of the above formulation, replacing the inequality in (4.12) by equality, does not function correctly. This is due to the fact that the right term $(r^1_{ij} + r^2_{ik} - 1)$ of (4.12) is negative when both r^1_{ij} and r^2_{ij} are equal to zero. However, we can extend the above formulation to the average latency objective by replacing the term $r^1_{ij} + r^2_{ik} - 1$ with $r^1_{ij} r^2_{ik}$. However, this replacement makes the formulation non linear. Therefore, to circumvent the non linear term and to extend the above formulation for the average case objective, we need to introduce a set of additional variables r_{ijk}, $\forall i \in S, \forall j,k \in P$, defined as follows:

$$r_{ijk} = \begin{cases} 1, & \text{if the first and second reference controllers to} \\ & \text{switch } i \text{ are } j \text{ and } k \text{ respectively.} \\ 0, & \text{otherwise.} \end{cases}$$

The CNCP formulation for the average case objective is given as follows:

$$\min_{Q \subseteq P, |Q|=p} \{z\}$$

Subject to (4.2)-(4.11), (4.13)-(4.15) and

$$z = \frac{1}{|S|} \sum_{i \in S} \sum_{j \in P} \sum_{k \in P} (d_{ij} + d_{jk}) r_{ijk} \tag{4.16}$$

However, the variables r^1_{ij} and r^2_{ik} are redundant if we introduce variable r_{ijk}. Therefore, we formulate the CNCP using only three indexed variables r_{ijk} to avoid both non linear terms and redundant variables in the next formulation.

4.1.4 CNCP Formulation with three-indexed variables

In this subsection we formulate the CNCP as ILP using three-indexed binary variables. Let the decision variables r_{ijk}, w_{jk} and y_j $\forall i \in S, j, k \in P$, are defined as above. The CNCP formulation with three-indexed variables is given as follows:

$$\min \quad z$$

Subject to (4.2), (4.6)-(4.9), (4.13), (4.15) and

$$\sum_{j \in P} \sum_{\substack{k \in P \\ k \neq i,j \\ d_{ij} \leq d_{ik}}} r_{ijk} = 1 \quad \forall i \in S \tag{4.17}$$

$$\sum_{\substack{k \in P \\ k \neq i,j \\ d_{ij} \leq d_{ik}}} r_{ijk} + \sum_{\substack{k \in P \\ k \neq j \\ d_{ik} \leq d_{ij}}} r_{ikj} \leq y_j \quad \forall i \in S, \forall j \in P \tag{4.18}$$

$$y_j + \sum_{\substack{h \in P \\ d_{ih} > d_{ij}}} \sum_{k \in P} r_{ihk} \leq 1 \quad \forall i, j \in S, i \neq j \tag{4.19}$$

$$\sum_{i \in S} \sum_{k \in P} L_i r_{ijk} + \sum_{i \in S} \sum_{k \in S} L_i r_{ikj} \leq U_j y_j \quad \forall j \in P \tag{4.20}$$

$$z \geq \sum_{j \in P} \sum_{k \in P} (d_{ij} + d_{jk}) r_{ijk} \quad \forall i \in S \tag{4.21}$$

$$r_{ijk} \in \{0, 1\} \quad \forall i \in S, \forall j, k \in P \tag{4.22}$$

Constraint (4.17) guarantees that each switch has unique first and second reference controllers, which are ensured by (4.3) and (4.4) in the earlier formulation. Constraint (4.18) corresponds to (4.5), which guarantees that j is either the first or the second reference controller to a switch i. Constraint (4.19) corresponds to (4.10), which ensures the closest assignment between switches and their respective first reference controller. It works as follows: if a controller is deployed at j ($y_j = 1$), then any controller that is farther from the switch i than j cannot be the first reference

controller of the switch i, i.e., the second term on the left hand side of (4.19) must be zero. Constraint (4.20) is demand constraint corresponding to (4.11). The first and second terms on the left hand side of (4.20) corresponds to the sum of demand of the switches for which j is the first and the second reference controller respectively. Constraint (4.21) corresponds to (4.11) which ensures that the objective value is greater than or equal to the sum of latency from any switch i to its nearest controller l and the latency from l to its nearest controller j. Constraints (4.13) and (4.22) are domain constraints which guarantees that all the decision variables take values either 0 or 1.

4.1.5 CNCP Formulation for multiple failures

In this subsection, we relax the assumption that at the most one controller fails at any time. We extend the three-indexed CNCP formulation to multiple controller failures. Let the decision variables r^l_{ijk}, $\forall i \in S$, $\forall j, k \in P$, $l \in \{1, 2, \ldots, \mu\}$ be defined as follows:

$$r^l_{ijk} = \begin{cases} 1, & \text{if the controller nearest to the } (l-1)^{th} \text{ reference} \\ & \text{controller } j \text{ of switch } i \text{ is } k \\ 0, & \text{otherwise.} \end{cases}$$

When $l = 1$, the $(l-1)^{th}$ reference controller of a switch refers to the zeroth reference controller which is meaning less. Thus, to maintain consistency in notation the zeroth reference controller of a switch is defined as itself. The first reference controller of a switch i is the controller nearest to i and the l^{th} $(l > 1)$ reference controller of a switch i is the controller that is nearest to the $(l-1)^{th}$ reference controller of i. Please note that the PACKET_IN messages of a switch is served by its l^{th} reference controller whenever the first $(l-1)^{th}$ reference controllers of that switch fail. Therefore, the objective is to minimize the maximum, for all switches

$i \in S$, of the sum of latency from switch i to its first reference controller and the sum of latencies from $(l-1)^{th}$ $(2 \leq l \leq \mu)$ reference controller of switch i to its l^{th} reference controller of i. Here, μ is a parameter supplied by the designer of the network, whose value is decided by the level of planning.

$$\min_{\substack{Q \subseteq P \\ |Q|=p}} \left\{ \max_{i \in S} \{ d_{ij} r^1_{iij} + \sum_{l \in \{2,3,\ldots,\mu\}} \sum_{k \in P} d_{jk} r^l_{ijk} \} \right\} \tag{4.23}$$

The CNCP formulation for multiple failures is given as follows:

$$\min \quad z$$

Subject to (4.2), (4.6)-(4.9), (4.13), (4.15) and

$$\sum_{j \in P} r^1_{iij} = 1 \quad \forall i \in S \tag{4.24}$$

$$\sum_{j \in P} \sum_{\substack{k \in P \\ k \neq i, k \neq j}} r^l_{ijk} = 1 \quad \forall i \in S, l \in \{2,3,\ldots,\mu\} \tag{4.25}$$

$$\sum_{j \in P} r^{l-1}_{ijk} - \sum_{\substack{w \in P \\ w \neq i, w \neq k \\ d_{ik} \leq d_{iw}}} r^l_{ikw} = 0 \quad \forall i \in S, \forall k \in P, l \in \{2,3,\ldots,\mu\} \tag{4.26}$$

$$r^1_{iik} + \sum_{l \in \{2,3,\ldots,\mu\}} \sum_{j \in P} r^l_{ijk} \leq y_k \quad \forall i \in S, k \in P \tag{4.27}$$

$$\sum_{\substack{h \in P \\ d_{ih} \leq d_{ij}}} r^1_{iih} \geq y_j \quad \forall i \in S, j \in P \tag{4.28}$$

$$\sum_{j \in P} \sum_{\substack{h \in P \\ h \neq i, h \neq j \\ d_{jh} \leq d_{jk}}} r^l_{ijh} + \sum_{a \in P} r^{l-1}_{iak} + \cdots + \sum_{a \in P} r^2_{iak} + r^1_{iik} \geq y_k$$

$$\forall i \in S, k \in P, k \neq i, l \in \{2,3,\ldots,\mu\} \tag{4.29}$$

$$\sum_{i \in S} L_i r^1_{iik} + \sum_{l \in \{2,\ldots,\mu\}} \sum_{i \in S} \sum_{j \in P} L_i r^l_{ijk} \leq U_k y_k \quad \forall k \in P \tag{4.30}$$

$$z \geq \sum_{j \in P} d_{ij} r^1_{iij} + \sum_{l \in \{2,\ldots,\mu\}} \sum_{j \in P} \sum_{\substack{k \in P \\ k \neq i, k \neq j \\ d_{ij} \leq d_{ik}}} d_{jk} r^l_{ijk} \quad \forall i \in S \tag{4.31}$$

$$r^l_{ijk} \in \{0,1\} \quad \forall i \in S, \forall j,k \in P, l \in \{1,2,3,\ldots,\mu\} \tag{4.32}$$

Constraints (4.24) and (4.25) together guarantees that each switch has exactly one l^{th} reference controller, for $l = 1, 2, \ldots, \mu$. Constraint (4.26) works as follows: if the PACKET_IN messages of the switch travels from the controller at j to the controller at k in case of $(l-1)$ controller failures, then the packets must move from k to some controller w in case of l^{th} controller failure. That is, if the l^{th} reference controller of switch i is k, then some controller w must be assigned as the $(l+1)^{th}$ reference controller of switch i. Constraint (4.27) ensures that j can be one of the μ reference controllers of switch i. That is, it guarantees that all the μ reference controllers of a switch are different from each other. Constraint (4.28) assures the closest assignment between switches and their first reference controllers. It was adopted from the Church and Cohon constraint presented in [117]. It works as follows: if a controller is deployed at j ($y_j = 1$), then either j or some other controller closer than j is the first reference controller of i. Constraint (4.29) is known as multi level closest assignment constraint which is adopted from the generalized Church and Cohon constraint presented in [101]. It assures the closest assignment between switches and their l^{th} reference controllers, for $l = 2, 3, \ldots \mu - 1$. It works as follows: if a controller is deployed at k ($y_k = 1$), j is the $(l-1)^{th}$ $(2 \leq l \leq \mu - 1)$ reference controller of switch i, and k is not the 1st, 2nd, ..., $(l-1)^{th}$ reference controller of i, then either k or some other controller closer than k from j is the l^{th} reference controller of i. Please note that the closest assignment between switches and their last (μ^{th}) reference controller is implicitly ensured by the objective function. Constraint (4.30) is the capacity constraint which guarantees that, for any controller, the sum of demands of the switches assigned to it is less than or equal to its capacity. The first term on the left hand side of (4.30) corresponds to the sum of demand of the

switches for which j is the first reference controller. The second term on the left hand side of (4.30) corresponds to the sum of demand of the switches for which j is l^{th} reference controller, for $l = 2, 3, \ldots, q$. The first term on the right hand side of (4.31) corresponds to the latency from the switch to its first reference controller. The second term on the right hand side of (4.31) corresponds to the sum of latencies from the $(l-1)^{th}$ reference controller of the switch to l^{th} reference controller of the switch, for $l = 2, 3, \ldots, \mu$. Therefore, constraint (4.31) ensures the correct objective value. Constraint (4.32) guarantees that all the decision variables are binary.

4.1.6 Controller Failover

CNCP not only generates the initial controller locations and the switch to controller assignment as output but also generates all the μ reference controllers for every switch. This entire information is stored at each controller. OpenFlow v1.2 and above allows each controller to be in one of the three roles for a switch: master, equal, or slave. Controllers that are in either master or equal role for a switch receive the PACKET_IN messages sent by the switch. Initially, the first reference controller of a switch is in master role and all other reference controllers of that switch are in slave role. The l^{th} reference controller of a switch becomes master whenever the first $(l-1)$ reference controllers of the switch fail, for $2 \leq l \leq \mu$. Detecting failure of controllers is an essential part in controller failover, which can be done using the methods proposed in [1,102–110]. The switch that is immediately connected to a controller can also detect the failure of the controller using the switch discovery mechanism proposed in [109]. Controller failover can be performed using the greedy failover method proposed in [109]. If all μ reference controllers of a switch fails, then it can fall back to traditional local control. However, the switch should support both OpenFlow pipeline and traditional Ethernet operation, i.e., the switch should be hybrid.

4.2 Performance metrics

We introduce two metrics for comparing CNCP and CCP in failure free case and failure case, which we refer as worst case latency and maximum worst case latency for brevity, in (4.33) and (4.34) respectively. The worst case latency defined in (4.33) is the maximum, for all switches, of the latency from the switch to its nearest controller. The maximum worst case latency defined in (4.34) is the maximum, for all possible failure scenarios, of the worst case latencies. Hence, we have to consider all possible failure scenarios while determining the maximum worst case latency.

$$L_{worst} = \max_{i \in S} \left\{ \min_{j \in C} d_{ij} \right\} \tag{4.33}$$

$$L^{F}_{max\_worst} = \max_{f \in F} \left\{ \max_{i \in S} \left\{ \min_{k \in C} d^{f}_{ik} \right\} \right\} \tag{4.34}$$

Here, F is the set of all possible failure scenarios, n is the number of switches in the network, and d^f_{ik} is the latency from switch i to controller k under a failure scenario f. Note that the way in which the term d^f_{ik} is defined for CCP is different from CNCP. It is defined for CCP and CNCP in (4.35) and (4.36) respectively.

$$d^{f}_{ik} = d_{ik} \tag{4.35}$$

$$d^{f}_{ik} = d_{ij} + d_{jk} \tag{4.36}$$

In case of controller failures, CCP reassigns the switches of failed controllers to other working controllers with enough spare capacity that is nearest to the switch. Hence, the term d^f_{ik} in CCP is the latency between switch i and controller k to which it is reassigned. However, the term d^f_{ik} in CNCP is the sum of the latency from a switch i to its first reference controller j and the latency from the first reference controller j to the second reference controller k. Moreover, we also used the maximum inter controller latency and the average inter controller latency metrics presented in (3.25) and (3.26) for comparing the inter controller latency of CNCP and CCP.

4.3 Heuristic Solution

The optimal formulation with two indexed decision variables presented in the previous section consists of quadratic number of constraints and variables. Similarly, the formulation with three indexed decision variables consists of cubic number of constraints and variables. The memory consumed by an ILP is proportional to the (number of constrains $\times$ number of decision variables). The number of decision variables and constraints increase with the network size. Hence, the running time and memory requirement of an ILP increases with the network size. In this section, we apply a heuristic technique to provide viable and faster solutions on large scale networks. Different heuristic solutions have been applied, in literature, on problems such as controller placement [61, 75, 78, 118], network embedding [119–121] etc. Simulated annealing is a popular choice for finding a global optimum of the problems that have large search space and many local optimums.

Simulated annealing is a probabilistic method proposed independently by Kirkpatrick, Gelatt and Vecchi [122], and Cerny [123]. It was motivated from the metropolis algorithm that deals with cooling of materials by slowly lowering the temperature. The main characteristic of the algorithm is to accept solutions that are worse than the current one with some probability in order to avoid getting caught at a local optimum. This can be done by using a control parameter known as temperature. The probability of accepting worse solutions decreases with the temperature. Hence, it allows the algorithm to explore the search space at higher temperatures and helps in the convergence at lower temperatures. Furthermore, the probability of accepting worse solutions decreases with the difference between objectives of current and new solution. The algorithm takes as input the network graph $G(S, E)$ and the annealing schedule and returns the best solution encountered during the traversal. The annealing schedule contains parameters starting temperature (T_0), number of iterations at each temperature

(I_{max}), temperature decrement (α) and ending temperature (T_e).

Algorithm 1: Simulated Annealing

Input: $G(S, E)$, T_0, T_e, I_{max}, α

Output: S^*, v^*

1: T=T_0, $v^* = \inf$, $i = 1$;
2: S=GenerateRandomInitialSolution () ;
3: v=EvaluateObjective (S) ;
4: **while** $T \geq T_e$ **do**
5: **if** Feasibility (S)==1 and $v < v^*$ **then**
6: $S^* = S$, $v^* = v$
7: **end if**
8: $S^{'}$=GenerateRandomNeighbour (S) ;
9: $v^{'}$=EvaluateObjective ($S^{'}$) ;
10: Δ=$v^{'} - v$;
11: r=Generate a random number between 0 and 1
12: **if** $P(S^{'}, \Delta, T) \geq r$ **then**
13: $S = S^{'}$, $v = v^{'}$
14: **end if**
15: $i = i + 1$;
16: **if** I_{max} iterations are performed at T **then**
17: $T = T\alpha$, $i = 1$;
18: **end if**
19: **end while**
20: **return** S^*, v^*

The algorithm begins with an initial temperature and gradually decrements until it reaches an ending temperature. The starting temperature must be selected very carefully i.e., it should not be too high or too low. Generally, the starting

temperature is set to 1 and ending temperature is set to a small value such as 0.00001. The ending temperature need not be zero because the probability of accepting the worst moves approaches zero as the temperature approaches zero. The algorithm picks a randomly chosen solution as the starting point in step 2 and computes its objective value in step 3. Nevertheless, we can also use a solution obtained by greedy or greedy randomized strategy as a starting point. Each solution is a triplet $S =< A, C_1, C_2 >$, where A is the set of active controllers and C_l, $l \in \{1, 2\}$, is the vector representing the l^{th} reference controller of switches, i.e., $C_l(i) = j \Leftrightarrow r^l_{ij} = 1$. Step 5-7 keeps track of the best feasible solution seen so far and its objective value. A solution is said to be feasible if it satisfies (4.17), (4.18) and (4.20). In each iteration, a random neighbor $S^{'}$ of the current solution S is generated in step 8 and its objective value is computed in step 9. The new solution is accepted and becomes the current solution for the next iteration with a probability, $P(S^{'}, \Delta, T)$. If it is not accepted then another neighbor of the current solution is generated and the process continues until the temperature reaches an ending temperature. The probability of accepting a solution is a function of temperature and change in objective value as shown in (4.37). It is evident from (4.37) that the new solution that is better than the current one is always accepted because $exp(-\frac{\Delta}{T})$ is greater than 1. On the other hand, the new solution that is worse than the current one is accepted if $exp(-\frac{\Delta}{T}) \geq r$. Where Δ is the change in objective value, i.e., difference between objectives of new and current solutions and r is a random number generated between 0 and 1. Two solutions are said to be neighbors if they differ by at most one active controller. Furthermore, two solutions which are having the same set of active controllers, but differs in at least one switch assignment are also treated as neighbors. At each temperature the algorithm must run long enough to reach the stable state. Typical number of iterations at each temperature level are in [100,1000]. After every I_{max} iterations, the temperature is modified according to the step 17 of Algorithm 1.

Typical value of α is in $[0.8, 0.99]$. Please note that the higher the value of α the more time it takes for the algorithm to terminate.

$$P(S^{'}, \Delta, T) = exp(-\frac{\Delta}{T}) = exp(-\frac{v^{'} - v}{T}) \geq r \tag{4.37}$$

4.4 Numerical Results

In this section, we evaluate the CNCP models using real-world network topologies and compare the performance obtained with the standard capacitated controller placement problem from the literature. We also compared the performance of the simulated annealing heuristic with the ILP.

4.4.1 Evaluation Setup

The proposed CNCP formulation is evaluated on various networks from Internet Topology Zoo [33] and compared its performance with CCP. The networks include: AT&T (25 nodes) and GEANT (40 nodes). All the selected networks have latitude and longitude information which are used to calculate the delay between the nodes. The demand of switches are randomly generated between 100 to 400 kilo requests per second [111], [19]. We assumed that the controller software runs on a server with maximum access bandwidth of 10Gbps. The capacity of each controller is set to $7.8 * 10^6$ packets/second because the size of a packet is 160 bytes according to the OpenFlow v 1.2 specification. The input file of the ILP is generated using MATLAB [112] and sent it to CPLEX optimizer [113].

4.4.2 Results

The worst case and maximum worst case latencies of CNCP and CCP in failure free and failure scenarios while considering a single controller failure are presented in Fig. 4.1. While evaluating for one controller failure, we need at least two controllers

so that the controller that survives can serve the switches. When the number of nodes in the network is less than or equal to 30, two controllers are sufficient for CCP and CNCP in failure free and failure cases because on average a controller can serve up to 30 nodes. Hence, we varied the number of controllers from 2 to 9 for AT&T network and the results are presented in Fig. 4.1a. CNCP requires at least 3 controllers when the number of nodes are greater than 30. Although the CCP in failure free case works well with two controllers, failure of a controller leaves the network with a single controller which cannot serve all the switches. Therefore, we varied the number of controllers from 3 to 10 for GEANT network and the results are presented in Fig. 4.1b. These results do not include the time required to reassign the switches of the failed controllers in the performance of CCP.

The worst case latency defined in (4.33) is used to compare the performance of CNCP with CCP in failure free case. When there are no failures, CCP results in lower worst case latency than CNCP which can be observed from Fig. 4.1 because it is optimized for the worst case latency. However, the worst case latency of CCP drastically increases in case of controller failures due to lack of planning for failures.

We used the maximum worst case latency defined in (4.34) as a metric to compare CNCP with CCP in case of controller failures. The maximum worst case latency of our proposed method in case of controller failures refers to the maximum, for all switches, the latency from the switch to its second reference controller when $\mu = 2$ and the latency from the switch to its third reference controller when $\mu = 3$ and so on. Note that the latency from a switch to its l^{th} reference controller is the sum of latency from the switch to its first reference controller and sum of the latencies from $(m-1)^{th}$ reference controller of the switch to its m^{th} reference controller, for $2 \leq m \leq l$. We evaluated the worst case latency for all possible failure scenarios (i.e failure of each individual controller when $\mu = 2$ or all possible combinations of two controller failures when $\mu = 3$ and so on) and the maximum among these is

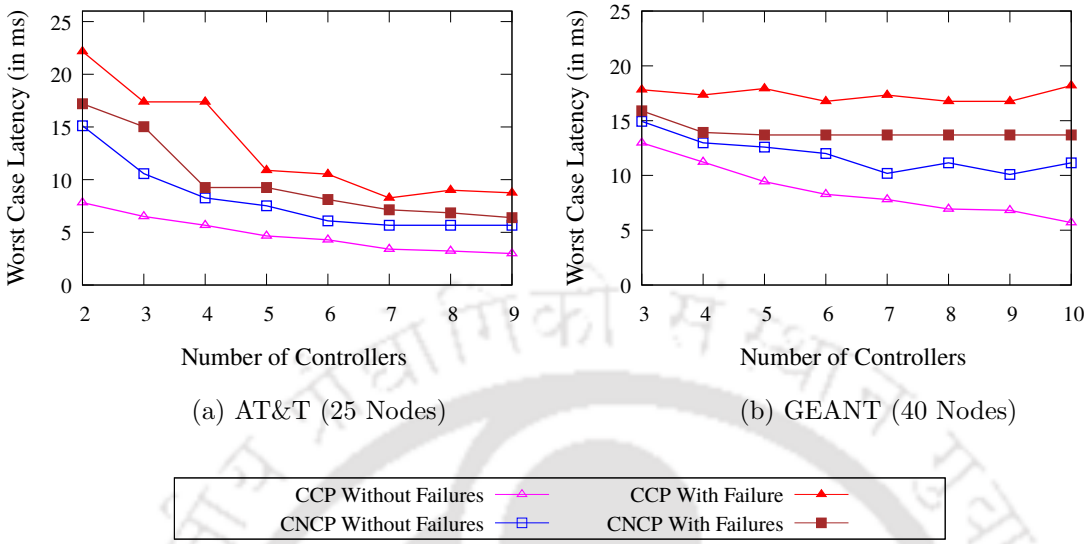
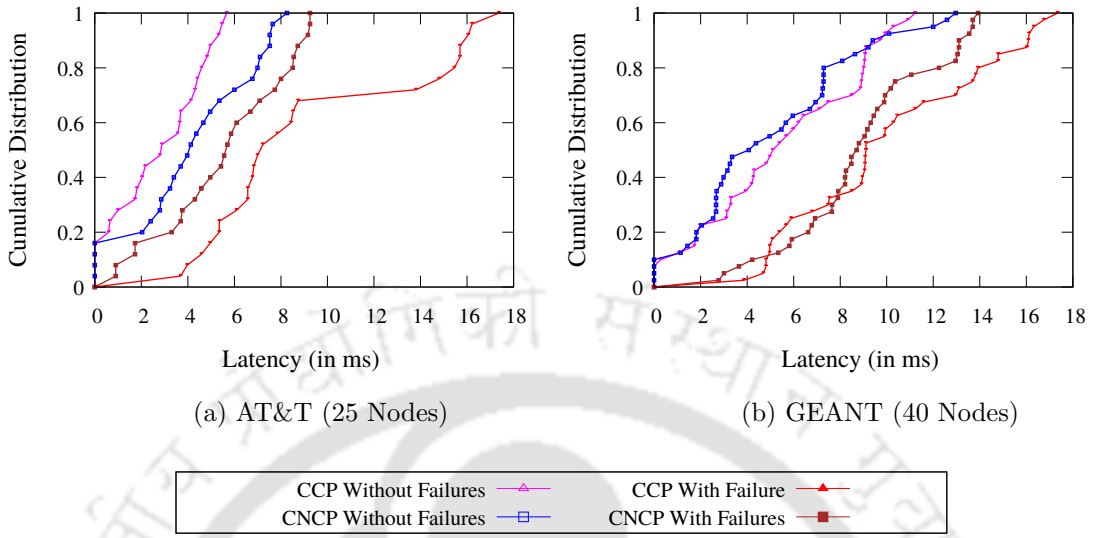


Figure 4.1: Worst case latency and maximum worst case latency of CCP and CNCP on various networks.

taken as the maximum worst case latency in case of failures. The same procedure is applied to CCP while determining the worst case latency in case of failures. Since exhaustive search of all failure scenarios is sufficient to determine the maximum worst case latencies in case of failures, we did not explicitly considered the spatial and temporal distribution of controller failures. Hence, the results we obtain are deterministic in nature. It is evident from Fig. 4.1 that CNCP performs better than CCP in case of controller failure because it is optimized for the maximum worst case latency.

In case of controller failures, the maximum worst case latency of CNCP decreases as the number of controllers in the network increases because its objective is to minimize the maximum worst case latency. However, the worst case latency of CNCP does not follow this trend in failure free case which can be observed in Fig. 4.1b because deploying controllers at locations that minimize the maximum worst case latency does not ensure that the worst case latency decreases as the number of controllers increases. Please note, deploying controllers at locations that

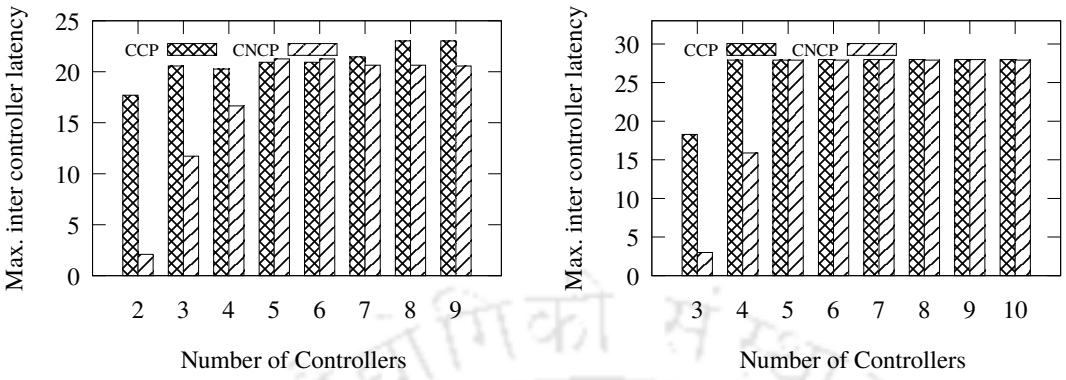


(a) AT&T (25 Nodes) (b) GEANT (40 Nodes)

Figure 4.2: Switch to controller latency distribution of CCP and CNCP on various networks when p=4.

minimize the worst case latency guarantees that the worst case latency decreases as the number of controllers increases. Since the objective of CCP is to minimize the worst case latency, CCP follows the above specified trend in failure free case. However, deploying controllers at locations that minimize the worst case latency does not ensure that the maximum worst case latency decreases as the number of controllers increases. Therefore, CCP does not follow the trend in case of failures which can be observed in Fig. 4.1a and Fig. 4.1b.

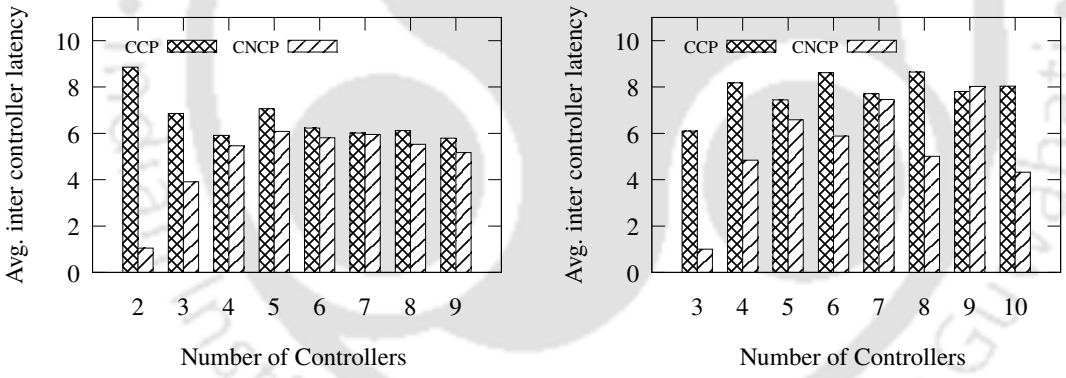
The worst case latency does not characterize the distribution of switch to controller latencies in the network. Therefore, we presented the cumulative distribution of switch to controller latencies in Fig. 4.2. We can observe from Fig. 4.2a that the maximum latency experienced by each switch in case of failures is lower in CNCP when compared to the CCP. This trend is not followed by all the switches in GEANT network which can be observed from Fig. 4.2b. But majority of the switches in GEANT network follows this trend. However, the switch to controller latencies of CCP, in failure free case, are lower than the CNCP. Therefore, we argue



(a) AT&T (25 Nodes) (b) GEANT (40 Nodes)

Figure 4.3: Maximum inter controller latency of CCP and CNCP on various networks.

that the average latency of CNCP is better than CCP in case of failures even though it is optimized for the maximum worst case latency.



(a) AT&T (25 Nodes) (b) GEANT (40 Nodes)

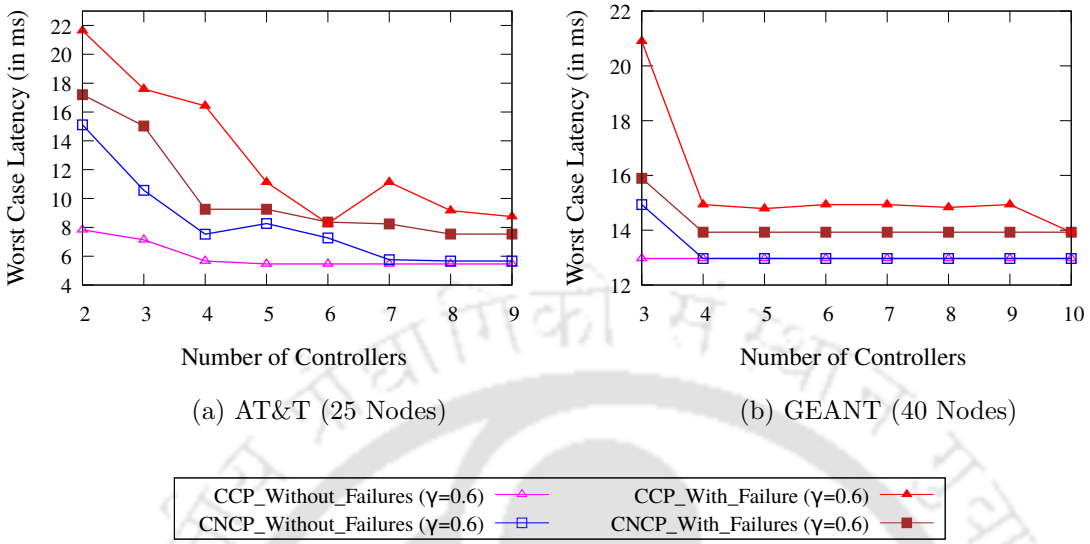
Figure 4.4: Average inter controller latency of CCP and CNCP on various networks.

In distributed SDNs, all the controllers must be communicated with each other to maintain a consistent global view of the network so as to ensure proper network operation. Since the worst inter controller latency characterizes the latency between the farthest controllers in the network, we presented those comparison results in Fig. 4.3. Since the second reference controller of a switch is the controller that is nearest to the first reference controller of the switch and the objective of CNCP

is to minimize the worst case latency between switches and their second reference controllers, CNCP forces the first and second reference controllers of a switch to be deployed close to each other. However, this does not force the controllers of different switches to be deployed near to each other. Furthermore, the latency between the farthest controllers increases with the number of controllers in the network. Therefore, the maximum inter controller latency of CNCP is lower than the CCP when fewer number of controllers are deployed in the network and it is close to CCP when more than the required number of controllers are deployed in the network. However, the cost of the network increases when much more than the required number of controllers are deployed in the network. Therefore, the designer of the network typically avoids deploying more than the required number of controllers. Since the average inter controller latency characterizes the distribution of controllers across the network, we presented those results in Fig. 4.4. It is evident that the average inter controller latency of CNCP is lower than the CCP.

We can observe that the maximum inter controller latency of CNCP and CCP increases with the number of controllers and it eventually reaches the diameter of the network. The latency between the farthest switches, i.e., the diameter of AT&T and GEANT networks is nearly 24 ms and 28 ms respectively. Therefore, we solved the problem by limiting the inter controller latency using constraints (4.6)-(4.9) with γ set to 0.6. That is, we restricted the maximum inter controller latency of AT&T and GEANT networks at 14.4 ms and 16.8 ms respectively. The results comparing the worst case latency and maximum worst case latency of CCP and CNCP when the maximum inter controller latency is restricted to 60% of the network diameter are presented in Fig. 4.5. It is evident that CNCP performs better than CCP in case of controller failure and CCP performs better than CNCP in failure free case.

The switch to controller latency and the inter controller latency are two competing objectives. Therefore, we presented the impact of restricting the

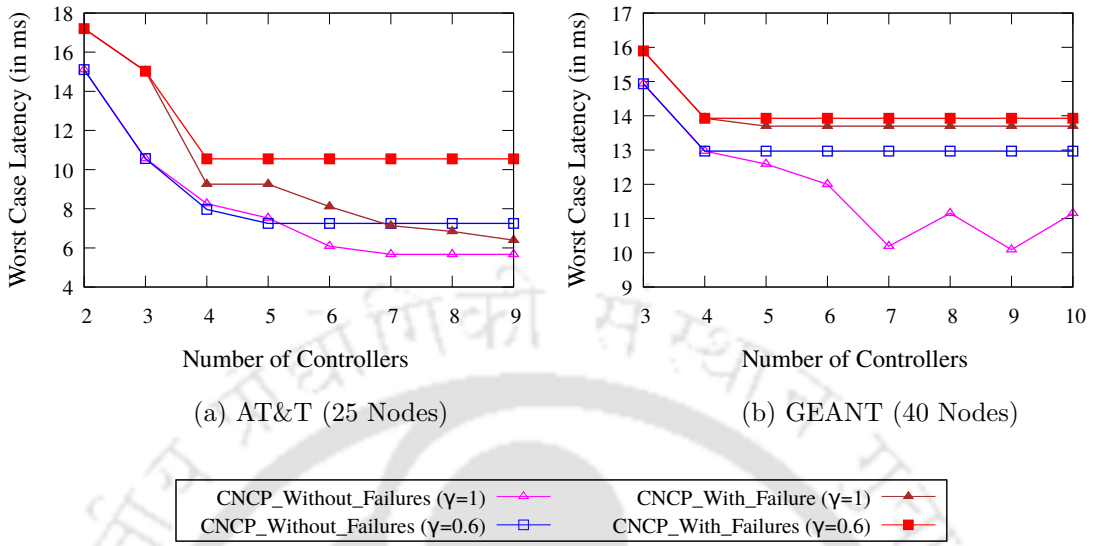


(a) AT&T (25 Nodes) (b) GEANT (40 Nodes)

Figure 4.5: Worst case latency and maximum worst case latency of CCP and CNCP while restricting the maximum inter controller latency.

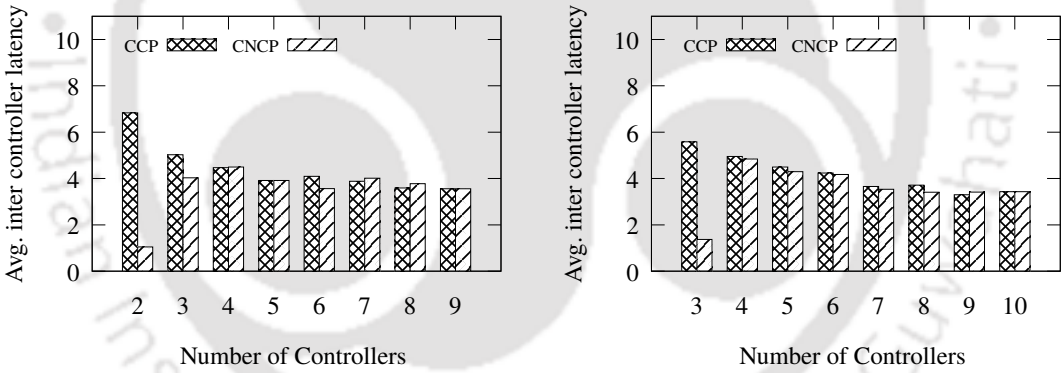
maximum inter controller latency on worst case latency and maximum worst case latency of CNCP in Fig. 4.6. There is an upsurge in the maximum worst case latency of CNCP when we restrict the maximum inter controller latency. The maximum inter controller latency of CNCP without restricting the inter controller latency ($\gamma = 1$) for ATT network is more than the upper bound (14.4 ms) when the number of controllers is greater than or equal to 4 which can be observed from Fig. 4.6a. Therefore, the upsurge in the maximum worst case latency of CNCP in Fig. 4.6a started at p=4. Similarly, for GEANT network it is more than 16.8 ms when the number of controllers is greater than or equal to 5. Therefore, the upsurge in Fig. 4.6b started at p=5. The worst case latency and maximum worst case latency of CCP and CNCP with inter controller latency constraints are higher when compared to the CNCP and CCP without these constraints.

The average inter controller latency of CNCP and CCP are presented in Fig. 4.7. It is evident that the average inter controller latency of CNCP is lower than the CCP when fewer number of controllers are deployed in the network and



(a) AT&T (25 Nodes) (b) GEANT (40 Nodes)

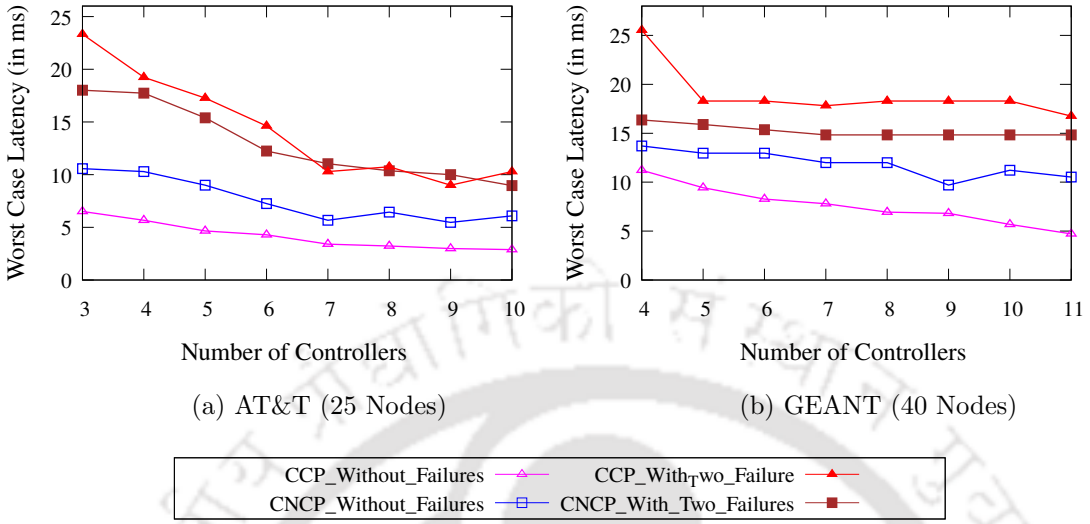
Figure 4.6: Impact of restricting the inter controller latency on worst case latency and maximum worst case latency of CNCP on various networks.



(a) AT&T (25 Nodes) (b) GEANT (40 Nodes)

Figure 4.7: Average inter controller latency of CCP and CNCP while restricting the maximum inter controller latency.

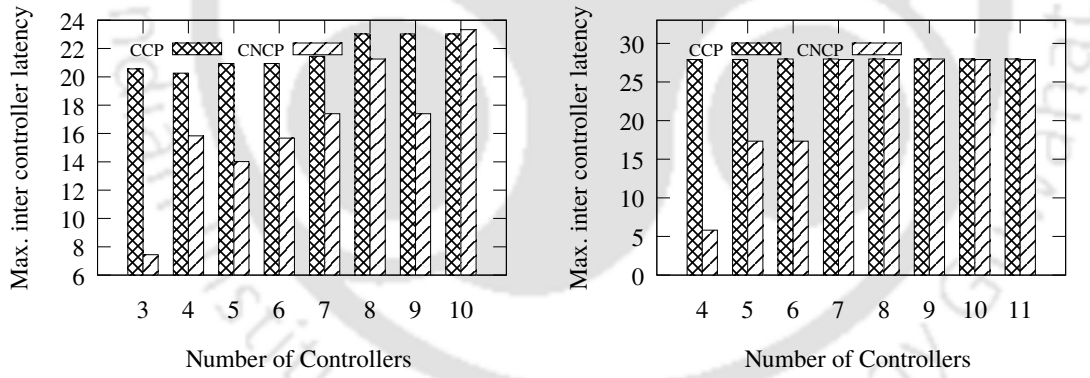
it is close to CCP when more than the required number of controllers are deployed in the network. The designer of the network typically avoids deploying more than the required number of controllers because the cost of the network increases when much more than the required number of controllers are deployed in the network. The maximum inter controller latency comparison between CNCP and CCP is not



(a) AT&T (25 Nodes) (b) GEANT (40 Nodes)

Figure 4.8: Worst case latency and maximum worst case latency of CCP and CNCP while evaluating for two controller failures.

presented because it is bounded by 60% of the network diameter.

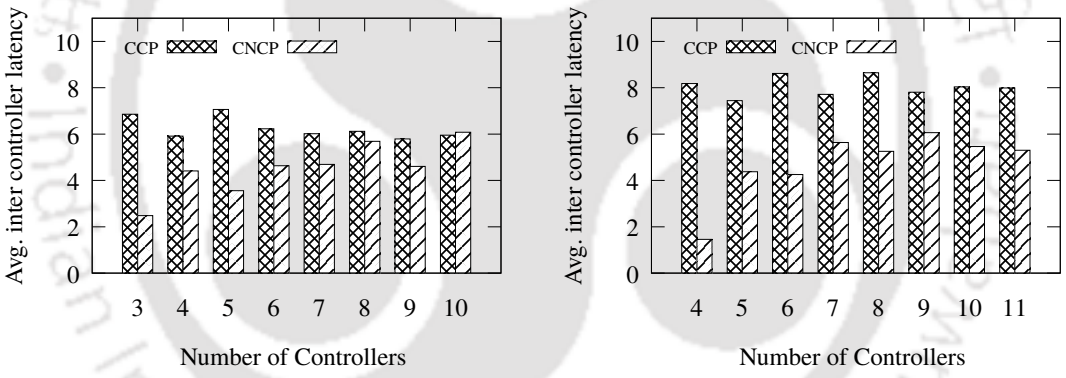


(a) AT&T (25 Nodes) (b) GEANT (40 Nodes)

Figure 4.9: Maximum inter controller latency of CCP and CNCP while evaluating for two controller failures.

Figure 4.8 illustrates the worst case latency and maximum worst case latency of CNCP and CCP while considering two controller failures. While evaluating for two controller failures, we need at least three controllers so that the controller that survives can serve the switches. When the number of nodes in the network is less

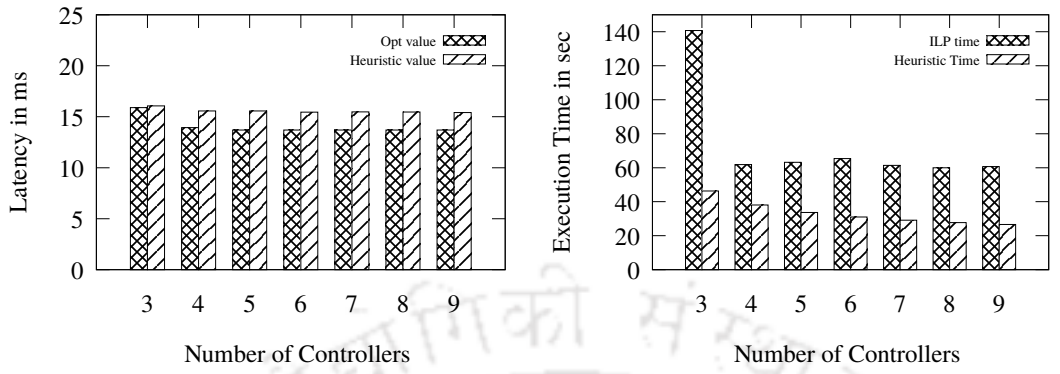
than or equal to 30, three controllers are sufficient for CCP and CNCP in failure free and failure cases because on average a controller can serve up to 30 nodes. Hence, we varied the number of controllers from 3 to 10 for AT&T network and the results are presented in Fig. 4.8a. CNCP requires at least 4 controllers when the number of nodes are greater than 30. Although the CCP in failure free case works well with three controllers, failure of a controller leaves the network with a single controller which cannot serve all the switches. Therefore, we varied the number of controllers from 4 to 11 for GEANT networks and the results are presented in Fig. 4.8b. We can observe that both CNCP and CCP follow trends similar to that of the single controller failure case. That is, CNCP performs better than CCP in case of failures and CCP performs better than the CNCP in failure free case.



(a) AT&T (25 Nodes) (b) GEANT (40 Nodes)

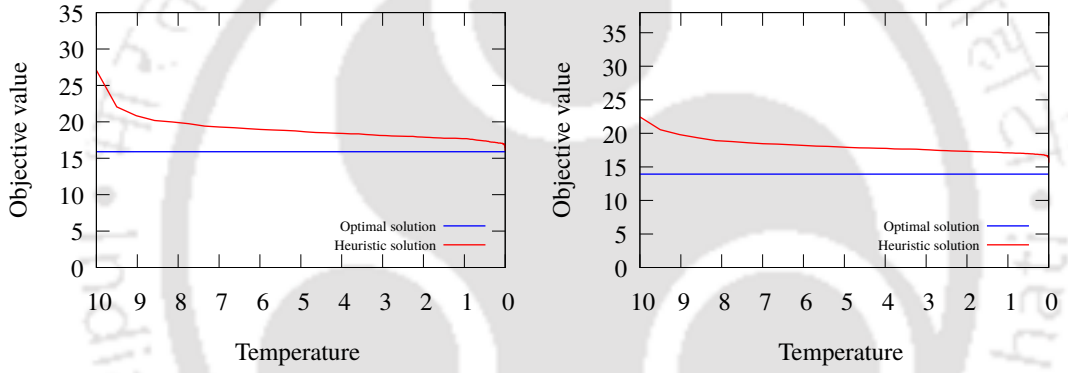
Figure 4.10: Average inter controller latency of CCP and CNCP while evaluating for two controller failures.

When the number of controllers deployed are much more than the required, CCP reassign switches of the failed controllers to nearest working controller with enough spare capacity without much increase in the worst case latency because there are many working controllers with enough spare capacity which can serve the switches of the failed controllers. We can observe that the worst case latency of CCP in Fig. 4.8a is slightly less than CNCP when p=7 and p=9 because the



(a) Objective comparison (b) Running time comparison

Figure 4.11: Performance of Simulated Annealing heuristic on GEANT topology



(a) p=3 (b) p=4

Figure 4.12: Progress of simulated annealing heuristic on GEANT topology

number of controllers are more than required and we did not included the time required to reassign the switches of the failed controllers in the performance of CCP. The maximum inter controller latency and average inter controller latency of CNCP and CCP while considering two controller failures were presented in Fig. 4.9 and Fig. 4.10 respectively. We can observe that CNCP performs better than CCP when fewer number of controllers are deployed in the network and it is close to CCP when much more than the required number of controllers are deployed in the network.

The simulated annealing heuristic is implemented in MATLAB and its performance is compared with the ILP. The starting and ending temperatures are

set to 10 and 0.0001 respectively. Similarly, the number of iterations at each temperature level, I_{max}, is set to 500 and the temperature decrement α is set to 0.95. Each instance is executed for 100 times and the average results are presented for statistical reliability. Figure 4.11a illustrates the performance of the simulated annealing heuristic on GEANT topology for different p values. It shows that the gap between the optimal value and the value returned by the heuristic is very less. This gap can be further reduced by increasing I_{max} and α values. However, it increases the running time of the heuristic. Figure 4.11b illustrates the running times of the ILP and the simulated annealing heuristic on GEANT network. We can observe that the simulated annealing heuristic takes less than the half of the time taken by ILP. The progress of simulated annealing heuristic on GEANT network for p=3 and p=4 were presented in Fig. 4.12a and Fig. 4.12b respectively. The gap between the heuristic objective value and optimal value is more at higher temperatures because the simulated annealing heuristic explores the search space by accepting worse solutions with high probability at higher temperatures. Since the probability of accepting worse solutions decreases with the temperature, the simulated annealing heuristic converge towards the optimal value at lower temperatures.

4.4.3 Complexity analysis

The complexity of CCP, CNCP for worst case latency, CNCP for average latency and CNCP for multiple failures is presented in terms of the number of decision variables, and the number of inequality and equality constraints in Table 4.2. Since $P = S$, the number of decision variables in CCP and CNCP for worst case latency is asymptotically equal to $O(|S| * |S|)$. CNCP for average latency and CNCP for multiple failures has $O(|S|*|S|*|S|)$ decision variables which is asymptotically larger than the number of decision variables in CCP and CNCP for worst case latency. The number of equality constraints in CCP, CNCP for worst case latency and

CNCP for average latency are asymptotically equal to $O(|S|)$. CNCP for multiple failures has $O(|S| * |S|)$ equality constraints which is asymptotically larger than the number of equality constraints in other three approaches. The number of inequality constraints in CCP, CNCP for average latency, and CNCP for multiple failures are asymptotically equal to $O(|S|*|S|)$. CNCP for worst case latency has $O(|S|*|S|*|S|)$ inequality constraints which is asymptotically larger than the number of inequality constraints in other three approaches. Therefore, CNCP demands more physical memory for large scale networks when compared to CCP.

Table 4.2: Complexity analysis of CCP and CNCP

CCP	Number of Decision Variables	$1 + \|P\| + \|S\| * \|P\|$
	Number of equality constraints	$1 + \|S\|$
	Number of inequality constraints	$\|S\| + \|P\| + \|S\| * \|P\|$
CNCP for worst case latency	Number of Decision Variables	$1 + \|P\| + 2 * \|S\| * \|P\| + \|P\| * \|P\|$
	Number of equality constraints	$1 + 2 * \|S\|$
	Number of inequality constraints	$\|P\| + 2 * \|S\| * \|P\| +$ $4 * \|P\| * \|P\| + \|S\| * \|P\| * \|P\|$
CNCP for average latency	Number of Decision Variables	$1 + \|P\| + \|P\| * \|P\| + \|S\| * \|P\| * \|P\|$
	Number of equality constraints	$1 + \|S\|$
	Number of inequality constraints	$\|S\| + \|P\| + \|S\| * \|S\| +$ $\|S\| * \|P\| + 4 * \|P\| * \|P\|$
CNCP for multiple failures	Number of Decision Variables	$1 + \|P\| + \|P\| * \|P\| +$ $\mu * \|S\| * \|P\| * \|P\|$
	Number of equality constraints	$1 + \mu * \|S\| + (\mu - 1) * \|S\| * \|P\|$
	Number of inequality constraints	$\|S\| + \|P\| + (\mu + 5) * \|S\| * \|P\|$

4.5 Conclusion

We investigated the capacitated next controller placement problem in SDNs that not only considers capacity and reliability of controllers but also plans ahead for

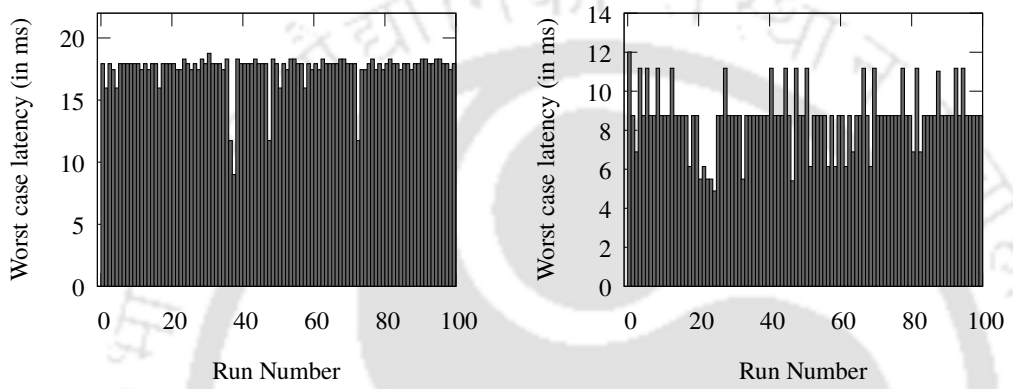
controller failures without assuming that the switches have failure foresight. We designed an optimization model, for a single controller failure, using two-indexed decision variables. The objective is to minimize the maximum, for all switches, of the sum of the latency from the switch to the nearest controller and the latency from the nearest controller to its closest controller. We also designed an optimization model using three-indexed decision variables which can be easily generalized for the average latency objective and extended it to multiple controller failures. Further, we presented a simulated annealing heuristic to efficiently solve the problem. The proposed formulations are evaluated on various networks from the Internet Topology Zoo. Simulation results show that our proposed method performs better than CCP in case of failures. Results also show that the heuristic is able to achieve near optimal solutions in less than half of the time required by the optimized formulations.

Chapter 5

Cooperative Game Theory based Network Partitioning for Controller Placement in SDNs

A single centralized controller may satisfy response time requirements of a small and medium scale network. However, a single controller does not satisfy the fault tolerant requirements of any network as it is the single point of failure. Therefore, the control plane is logically centralized, but physically distributed across multiple controllers to satisfy both the response time and fault tolerant requirements. In last two chapters, we proposed LP based solutions for the controller placement problem in SDNs. Another potential solution is to divide the network into domains and assign a controller to each of these domains. The standard k-means strategy is used to partition the Chinanet and Interoute networks and the worst case latency for 100 different runs is depicted in Fig. 5.1. It can be observed from Fig. 5.1a that the worst case latency of Chinanet topology varied from 9 ms to 18.76 ms when the network is partitioned into four subnetworks. The worst case latency of Interoute topology varied from 4.9 ms to 12.1 ms when the network is partitioned into six subnetworks

as shown in Fig. 5.1b. Moreover, the average worst case latency, computed over 100 runs, of Chinanet and Interoute networks is 17.45 ms and 8.7 ms respectively. However, the optimal worst case latency obtained by solving the optimal k-center problem [124] is 8.47 ms and 3.8 ms respectively. Therefore, we argue that k-means algorithm with random initialization results in solutions that are far from optimal.



(a) Chinanet topology with four partitions. (b) Interoute topology with six partitions.

Figure 5.1: Worst case latency of standard k-means on various topologies when executed for 100 times.

Motivated by this, we propose a controller placement strategy that partition the network using k-means algorithm with cooperative game theory based initialization. Henceforth, we refer it as cooperative k-means for brevity. The partitioning of the network into subnetworks is modeled as a cooperative game with the set of all switches as the players of the game. The switches try to form coalitions with other switches so as to maximize their value. We also propose two variants of cooperative k-means strategy that tries to produce partitions that are balanced in terms of size. The performance of our proposed strategy is evaluated on networks from Internet 2 OS3E topology and Internet Topology Zoo and compared it with the standard k-means algorithm.

The remainder of the chapter is organized as follows. In Section 5.1, we present

the background concepts of cooperative game theory and introduce our cooperative game theory based k-means algorithm for controller placement in Section 5.2. We report our evaluation results in Section 5.3. We finally conclude the chapter in Section 5.4.

5.1 Background

In this section, we present the background concepts of the cooperative game theory that are used in formulating the problem.

5.1.1 Cooperative game

A cooperative game is defined as a pair (N, v), where N is the set of all players in the game and v is value function or characteristic function. $v(F), F \subseteq N$ is known as the value of the coalition F and $v(N)$ is known as the value of the grand coalition A cooperative game can be analyzed using two solution concepts namely the core of the game and Shapley value.

5.1.2 Core of the game

Let $x = (x_1, x_2, \ldots, x_n) \in \mathbb{R}^n$ be the payoff allocation with x_i being the payoff of switch i. A payoff allocation $x = (x_1, x_2, \ldots, x_n)$ is said to be individually rational if $x_i \geq v(\{i\}), \quad \forall i \in N$. A payoff allocation x is said to be coalitionally rational if $\sum_{i \in F} x_i \geq v(F), \quad \forall F \subseteq N$. Additionally, a payoff allocation x is said to be collectively rational if $\sum_{i \in N} x_i = v(N)$. The core of a cooperative game is set of all payoff allocations that are individually rational, coalitionally rational, and collectively rational. Every coalitionally rational payoff allocation is an individually rational allocation. Thus, the set of all coalitionally rational and collectively rational

payoff allocations constitutes the core of a cooperative game.

$$\mathbb{C}(N,v) = \Big\{(x1, x2, \ldots, x_n) \in \mathbb{R}^n : \sum_{i \in N} x_i = v(N); \sum_{i \in F} x_i \geq v(F), \forall F \subseteq N \Big\} \tag{5.1}$$

5.1.3 Shapley value

The Shapley Value is a solution concept that fairly allocates the gains obtained by cooperation among the players while considering the relative importance of players in the game. Let $\phi(N,v) = (\phi_1(N,v), \phi_2(N,v), \ldots, \phi_n(N,v))$ and $\phi_i(N,v)$ be the Shapley value of the cooperative game and player i respectively. $\phi_i(N,v)$ specifies the expected payoff to player i and is given by,

$$\phi_i(N,v) = \sum_{F \subseteq N-i} \frac{|F|!(n-|F|-1)!}{n!}\{v(F \cup \{i\}) - v(F)\} \tag{5.2}$$

5.1.4 Convex game

A cooperative game (N,v) is said to be convex if the marginal contribution of a player is higher in larger coalitions.

$$v(F \cup \{i\})) - v(F) \geq v(B \cup \{i\})) - v(B), \quad \forall B \subseteq F \subseteq N \setminus \{i\}, \ \ i \in N \tag{5.3}$$

Here, $v(F \cup \{i\})) - v(F)$ and $v(B \cup \{i\})) - v(B)$ are the marginal contributions of player i with respect to the coalition F and B respectively.

Equivalently (5.3) can be written as follows:

$$v(F \cup \{i\})) - v(B \cup \{i\})) \geq v(F) - v(B), \quad \forall B \subseteq F \subseteq N \setminus \{i\}, \ \ i \in N \tag{5.4}$$

5.1.5 Shapley value of a convex game

Let Π be the set of all orderings of $N = \{1, 2, \ldots, n\}$ and $\pi \in \Pi$ be a specific ordering of the players. The initial segments of the ordering π are given by,

$$R_{\pi,m} = \{i \in N : \pi(i) \leq m\},\ m \in \{0, 1, 2, \ldots, n\}. \tag{5.5}$$

The core for a specific ordering π can be computed by solving the following equations.

$$s_i^{\pi}(R_{\pi,m}) = v(R_{\pi,m}), m \in \{1, 2, \ldots, n\} \tag{5.6}$$

The solution to these equations returns the coordinates of the intersection of the hyperplanes $H_{R_{\pi,m}}$.

$$s_i^{\pi} = v(R_{\pi,\pi(i)}) - v(R_{\pi,\pi(i)-1}),\ i \in \{1, 2, \ldots, n\} \tag{5.7}$$

The Shapley of a convex game belongs to the core of the game and is the center of gravity of above intersection points s^{π}. Therefore, it is given by,

$$\phi_i = \frac{1}{n!} \sum_{\pi \in \Pi} s_i^{\pi}. \tag{5.8}$$

5.2 Problem Formulation

In this section, we first present input parameters of the problem. Then, we formulate the network partitioning problem using cooperative k-means strategy. Finally, we present two variants of cooperative k-means strategy that tries to produce partitions that are balanced in terms of size.

5.2.1 Input Parameters

The physical network is denoted with the graph $G(S, E)$ where $S = \{s_1, s_2, \ldots, s_n\}$ is the set of switches and E is the set of edges between the switches. We denote

all pairs shortest path latency matrix with D, where the entry $D(s_i, s_j)$ represents the shortest path latency between switch i and switch j. We use the terms $D(s_i, s_j)$ and d_{ij} interchangeably in this thesis. Let P be the finite non empty set of potential locations for deploying controllers in the network. We assume that every switch is a potential location for deploying a controller, i.e., $P = S$.

5.2.2 Network Partitioning

The goal is to partition the network $G(S, E)$ into k subnetworks $G_i(S_i, E_i)$, $i = 1, 2, \ldots, k$, subject to the following:

$$\bigcup_{i=1}^{k} S_i = S, \quad \bigcup_{i=1}^{k} E_i = E \tag{5.9}$$

$$S_i \cap S_j = \emptyset, \quad \forall i, j \in \{1, 2, \ldots, k\}, i \neq j \tag{5.10}$$

$$G_i \text{ is a connected sub graph}, \quad \forall i \in \{1, 2, \ldots, k\} \tag{5.11}$$

Eq 5.9 specifies that all the subnetworks together should cover the entire network. That is, the switches and edges of all the subnetworks together need to cover respectively the switches and edges of the entire network. Eq 5.10 indicates that each switch is part of exactly one subnetwork. Eq 5.11 specifies that the switches in a partition are connected.

5.2.3 Cooperative k-means Network Partitioning

We utilized the cooperative game theory based clustering method proposed in [125]. The partitioning of the network into subnetworks is modeled as a cooperative game (S, v) where the set of all switches $S = \{s_1, s_2, \ldots, s_n\}$ are the players of the game and $v : 2^{|S|} \rightarrow \mathbb{R}$ is the value function with $v(\phi) = 0$. The switches try to form coalitions with other switches so as to maximize their value. The value function

that we employed is as follows:

$$v(F) = \frac{1}{2} \sum_{\substack{s_i, s_j \in F \\ s_i \neq s_j}} U(d(i,j)),\ F \subseteq S, F \neq \phi \tag{5.12}$$

where $U : \mathbb{R}^+ \cup \{0\} \rightarrow (0, 1]$ is a monotonically non increasing similarity function defined in (5.13). Note that D_{max} in (5.13) is the diameter of the network which is the maximum latency between any two switches.

$$U(D(i,j)) = 1 - \frac{D(i,j)}{1 + D_{max}} \tag{5.13}$$

Let B and F be any two coalitions with $B \subseteq F \subseteq S \setminus \{s_k\}, s_k \in S$. Then $v(F \cup \{s_k\}) - v(B \cup \{s_k\})$ is given by,

$$\begin{aligned}
=&\frac{1}{2} \sum_{\substack{s_i, s_j \in F \\ s_i \neq s_j}} U(d(s_i, s_j)) + \sum_{s_i \in F} U(d(s_i, s_k)) - \frac{1}{2} \sum_{\substack{s_i, s_j \in B \\ s_i \neq s_j}} U(d(s_i, s_j)) - \sum_{s_i \in B} U(d(s_i, s_k)) \\
=&\frac{1}{2} \sum_{\substack{s_i, s_j \in F \setminus B \\ s_i \neq s_j}} U(d(s_i, s_j)) + \sum_{\substack{s_i \in F \setminus B \\ s_j \in B}} U(d(s_i, s_j)) + \sum_{s_i \in F \setminus B} U(d(s_i, s_k)) \\
=&v(F) - v(B) + \sum_{s_i \in F \setminus B} U(d(s_i, s_k)) \\
\geq&v(F) - v(B)
\end{aligned}$$

Hence, the cooperative game (S, v) with the value function defined in (5.12) is

convex. Consequently, the Shapley value can be computed using (5.8) as follows:

$$\phi_i = \frac{1}{n!}\sum_{\pi\in\Pi} s_i^{\pi}$$

it can be written, using (5.7), as

$$\phi_i = \frac{1}{n!}\sum_{\pi\in\Pi} v(R_{\pi,\pi(i)}) - v(R_{\pi,\pi(i)-1})$$

it can be written, using (5.5) and (5.12), as

$$= \frac{1}{n!}\sum_{\pi\in\Pi}\Big[\frac{1}{2}\sum_{\substack{s_g,s_h\in R_{\pi,\pi(i)}\\ s_g\neq s_h}} U(d(s_g,s_h)) - \frac{1}{2}\sum_{\substack{s_g,s_h\in R_{\pi,\pi(i)-1}\\ s_g\neq s_h}} U(d(s_g,s_h))\Big]$$

avoiding duplicate pairs, gives us

$$= \frac{1}{n!}\sum_{\pi\in\Pi}\Big[\sum_{\substack{\pi(g)\leq\pi(i)\\ \pi(h)<\pi(g)}} U(d(s_g,s_h)) - \sum_{\substack{\pi(g)\leq\pi(i)-1\\ \pi(h)<\pi(g)}} U(d(s_g,s_h))\Big]$$

$$= \frac{1}{n!}\sum_{\pi\in\Pi}\sum_{\pi(j)<\pi(i)} U(d(s_i,s_j))$$

$$= \Big[\frac{1}{n!}\sum_{\substack{\pi\in\Pi\\ \pi(i)=1\\ \pi(j)<\pi(i)}} U(d(s_i,s_j)) + \frac{1}{n!}\sum_{\substack{\pi\in\Pi\\ \pi(i)=2\\ \pi(j)<\pi(i)}} U(d(s_i,s_j)) + \cdots + \frac{1}{n!}\sum_{\substack{\pi\in\Pi\\ \pi(i)=n\\ \pi(j)<\pi(i)}} U(d(s_i,s_j))\Big] \tag{5.14}$$

There are total $(n-1)!$ orderings while fixing the position of the player i. In each of these orderings, every player other than i can occur in a position preceding i in $\frac{i-1}{n-1}$ ways. Therefore, summing over all orderings gives us:

$$\phi_i = \frac{1}{n!}\Big(\sum_{i=1}^{n}\frac{i-1}{n-1}\Big)(n-1)!\sum_{\substack{s_j\in S\\ j\neq i}} U(d(s_i,s_j))$$

$$\phi_i = \frac{1}{2}\sum_{\substack{s_j\in S\\ j\neq i}} U(d(s_i,s_j)) \tag{5.15}$$

The Shapley value of player i can be computed using (5.15). We present an algorithm for initializing the controller locations using the cooperative game

theory in Algorithm 2. It takes as an input set of potential locations for deploying controllers, all pairs shortest path latency matrix and a threshold parameter, and returns initial controller location as output. Every switch is a potential location for deploying a controller, hence, step 1 of the algorithm initializes P with S and starts with an empty set of initial controller locations C_0. The Shapley value of each location is computed in steps 2 - 4. Then, the algorithm iteratively does the following in steps 6 - 9 until the set P is empty: selects a switch m from P with the highest Shapley value (ties are broken in favor of the switch with lower index), add m to the set of initial locations for deploying controllers, assign to m, all those switches T_m that are at least α similar to m, and updates the set P. Note that the parameter α determines the number of subnetworks.

The k-means algorithm for partitioning the nodes in a physical network is described in Algorithm 3. It takes as an input set of switches in the network, all pairs shortest path latency matrix and a threshold parameter and returns final locations for deploying controllers and switch to controller assignment. Step 1 of the algorithm initializes the locations for installing controllers using cooperative game theory initialization as shown in Algorithm 2. Step 2 of the algorithm initializes the updated controller locations C_{new} to empty. Then, the algorithm iteratively does the following until the current and updated controller locations are same: update the partitions in steps 4 - 7 by assigning switches to the nearest controller and update the controller location of each partition in steps 8 - 11.

5.2.4 Load aware Cooperative k-means Network Partitioning

Algorithm 3 presented in the previous section divides the network into partitions and then deploy a controller in each of the partition. However, it produce partitions that are imbalance in terms of size which is defined in (5.16). It is the difference

Algorithm 2: Cooperative game theory initialization

Input: Set of switches: $N = S = \{1, 2, \ldots, n\}$

All pairs shortest path matrix: D

Threshold: $\alpha \in (0, 1]$

Output: Initial controller locations: C_0

```
P=S, C_0 = ∅
for (i = 1, 2, ..., n) do
    φ_i = (1/2) Σ_{j∈N, j≠i} U(d(i, j))
end
while (P ≠ ∅) do
    m = arg max_{i∈P} φ_i
    C_0 = C_0 ∪ {m}
    T_m = {i ∈ P : U(d(s_i, s_k) ≥ α}
    P = P \ T
end
```

between the number of switches in the partition with largest size and the partition with smallest size.

$$Partition\_Imbalance = \max_{c_j \in C} |A_j| - \min_{c_j \in C} |A_j| \tag{5.16}$$

Consequently, controllers deployed in large partitions are being overloaded and controllers deployed in smaller partitions are being lightly loaded. Therefore, we propose two strategies in this section which takes into account the cluster imbalance.

Capacity of controllers

We can reduce the cluster imbalance by taking the capacity of controllers into account. Henceforth, it is referred as capacitated cooperative k-means for brevity.

Algorithm 3: Cooperative k-means for network topologies

Input: Set of switches: $S = \{1, 2, \ldots, n\}$

All pairs shortest path matrix: D

Threshold: $\alpha \in (0, 1]$

Output: Final locations for deploying controllers: C

Switch to controller assignment: $A_j, \forall c_j \in C$

```
C=Cooperative game theory based initialization (S,D,α)
C_new = ∅
while (1) do
    for (i = 1, 2, ..., n) do
        m = arg min_{j:c_j∈C} D(i, c_j)
        A_m = A_m ∪ {i}
    end
    for (j = 1, 2, ..., |C|) do
        m= arg min_{i:i∈A_j} Σ_{l:l∈A_j} D(i, l)
        C_new = C_new ∪ {m}
    end
    if (C and C_new are not equal) then
        C = C_new
        A_j = ∅, ∀c_j ∈ C
        C_new = ∅
    else
        exit
end
```

This can be implemented by replacing Step 5 of Algorithm 3 with the following step.

$$m = \underset{\substack{j:c_j \in C \\ L(s_i) < R(c_j)}}{\arg\min} \; D(i, c_j) \tag{5.17}$$

Here, $R(c_j)$ is the residual capacity of controller in partition j and $L(s_i)$ is the demand of switch i. (5.17) ensures that each switch is assigned to a nearest controller with residual capacity more than the demand of switch i.

Equipartition

The alternative way to circumvent the cluster imbalance is to divide the network into equal sized partitions. Henceforth, it is referred as equipartition cooperative k-means for brevity. This can be implemented by replacing Step 5 of Algorithm 3 with the following step.

$$m = \underset{\substack{j:c_j \in C \\ |A_j| < \frac{|S|}{k}}}{\arg\min} \; D(i, c_j) \tag{5.18}$$

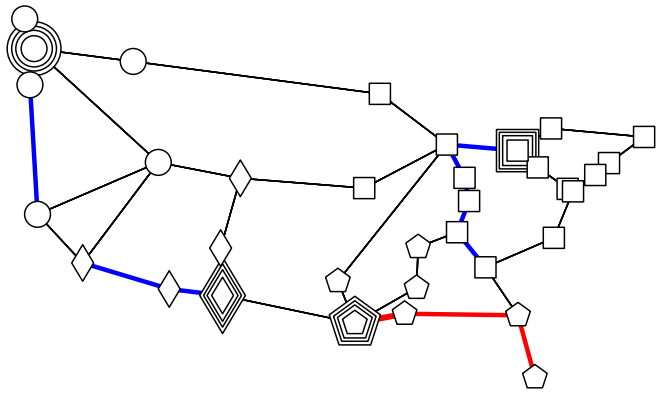
Here, $|S|$ is the total number of switches in the network and k is the number of partitions, i.e., number of initial controllers returned by the Algorithm 2. (5.18) ensures that each switch is assigned to a nearest partition whose size is less than $|S|/k$.

5.3 Numerical Results

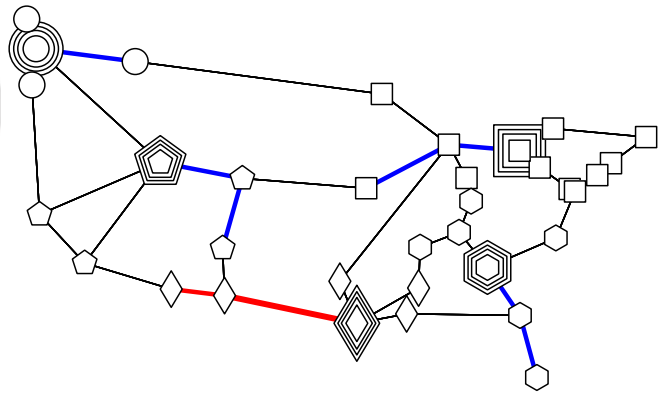
In this section, we compare and analyze the performance of the proposed algorithm with the standard k-means strategy with random initialization.

5.3.1 System Setup

The proposed cooperative k-means strategy is implemented in Python and evaluated its performance on both small scale and large scale network scenarios. We considered



(a) Four subnetworks.



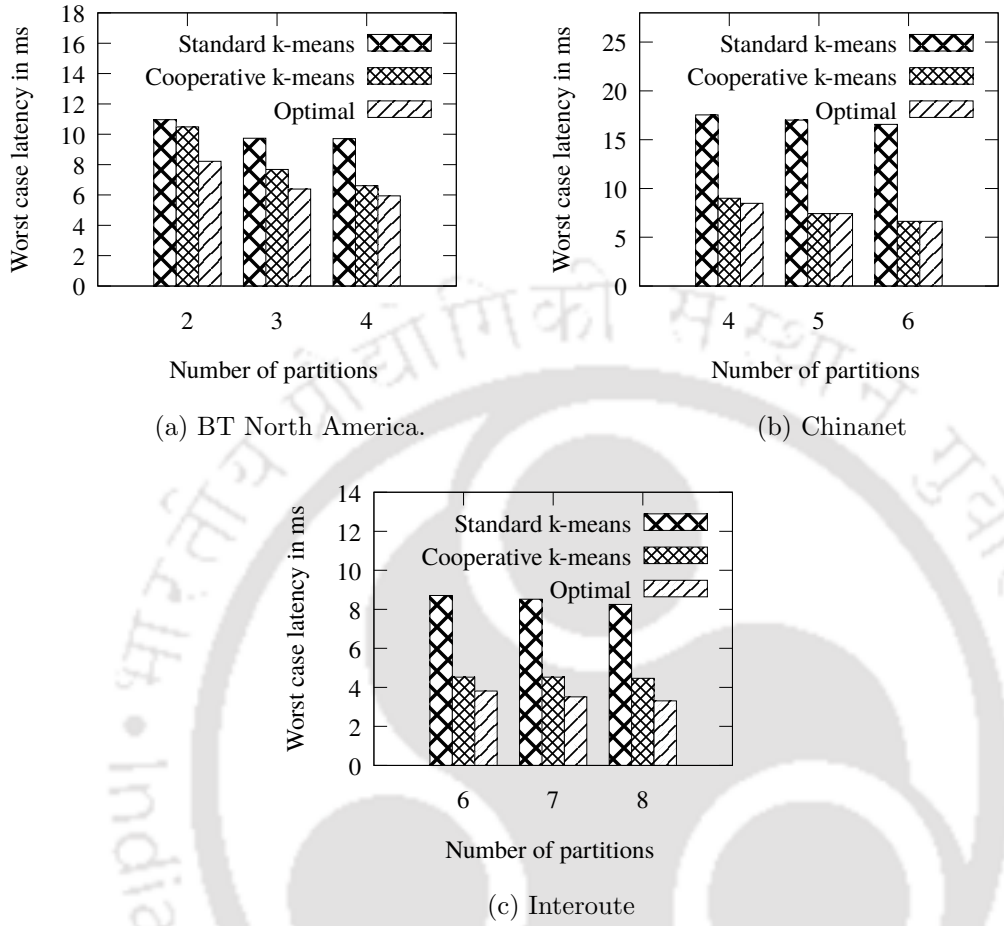
(b) Five subnetworks.

Figure 5.2: Partitioning Internet 2 OS3E topology using cooperative k-means.

Internet 2 OS3E topology [98] with 34 nodes, BT North America network with 36 nodes and Chinanet [33] network with 42 nodes for a small network scenario, and Interoute network with 110 nodes for a large network scenario.

5.3.2 Results

Fig. 5.2a and Fig. 5.2b illustrates the partitioning of the Internet 2 OS3E topology into four and five subnetworks using cooperative k-means algorithm. Nodes within a subnetwork are depicted with the same symbol and the controller of the subnetwork is highlighted by superimposing it with the same symbol multiple times. The maximum switch to controller latency path in each subnetwork is highlighted in



(a) BT North America.

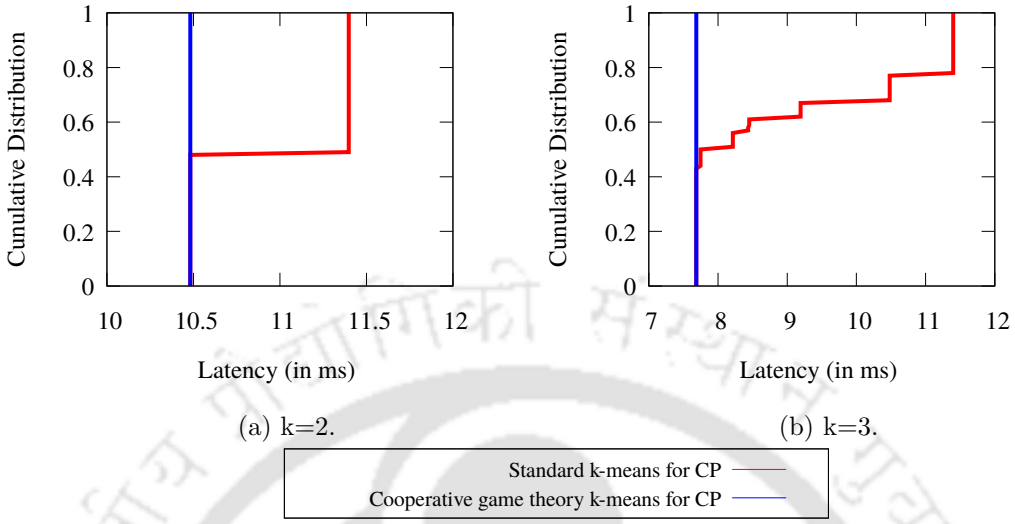
(b) Chinanet

(c) Interoute

Figure 5.3: Worst case latency of standard k-means and cooperative k-means strategies on various networks.

blue color. The worst case latency path of the network is highlighted in red color. The cooperative k-means strategy resulted in a worst case latency of 9.26 ms and 7.55 ms when the network is partitioned into four and five subnetworks respectively.

k-means algorithm with random initialization, which is also known as standard k-means, is used as a baseline method for comparison with our proposed strategy. We also compared the solution induced by our proposed strategy with the optimal solution obtained by solving the k-center problem [124]. Given an undirected graph with edge costs satisfying the triangle inequality and an integer k as input, the k-center problem selects a set of k nodes for which the maximum latency from

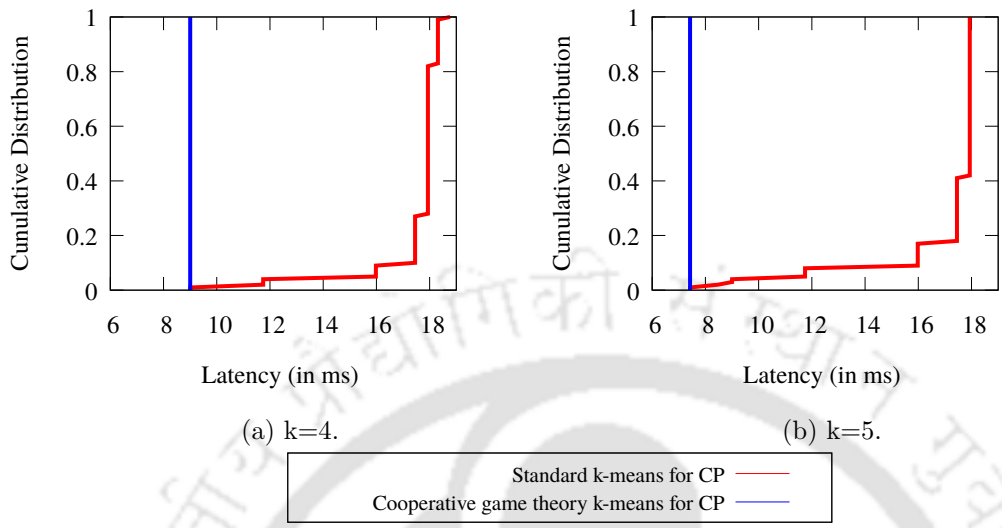


(a) k=2. (b) k=3.

Figure 5.4: Distribution of worst case latencies of cooperative k-means and standard k-means on BT North America topology when evaluated for 100 times

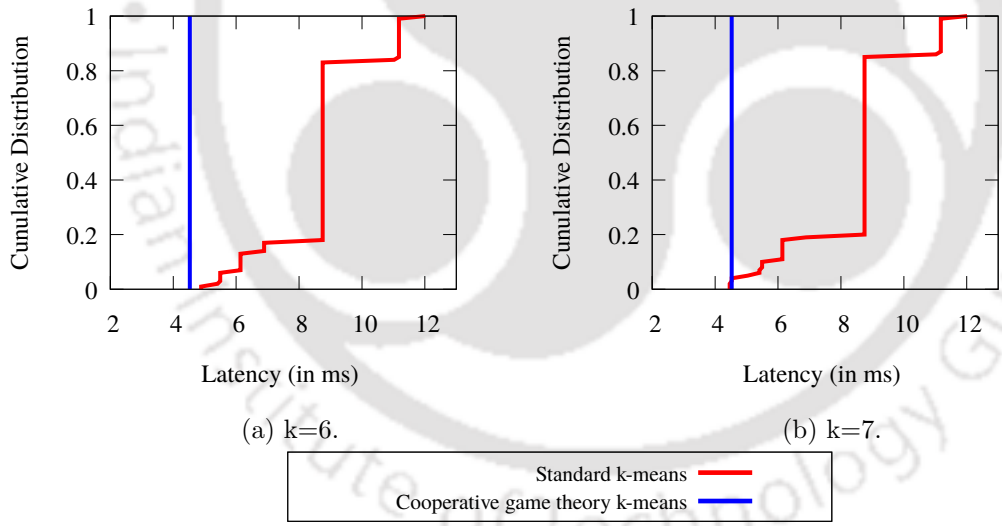
any node to its closest node in the k-set is minimum. Fig. 5.3a, Fig. 5.3b and Fig. 5.3c demonstrate the worst case latency, average computed over 100 runs, of cooperative k-means, standard k-means and optimal k-center [124] strategies on BT North America, Chinanet and Interoute networks respectively. We can observe that the worst case latency of the solution induced by cooperative k-means outperforms standard k-means and is very close to the optimal solution in all the networks.

The average worst case latency, computed over 100 runs, presented in Fig. 5.3 does not characterize the distribution of worst case latencies over these 100 execution instances. Hence, the worst case latency distribution of cooperative k-means and standard k-means when evaluated on BT North America network for 100 times is presented in Fig. 5.4. The standard k-means algorithm randomly selects initial controller locations. Therefore, the solution produced by standard k-means algorithm varies with different execution instances. On the contrary, the only step in the proposed cooperative k-means algorithm that involves some randomization is step 6. That is, selecting a switch m from P with the highest Shapley value.



(a) k=4. (b) k=5.

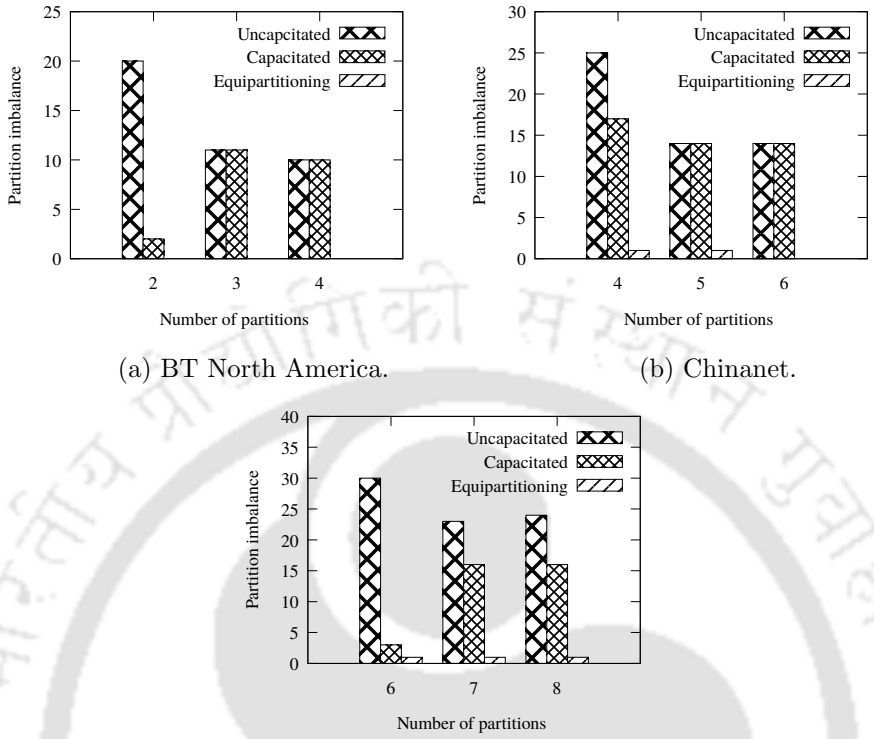
Figure 5.5: Distribution of worst case latencies of cooperative k-means and standard k-means on Chinanet topology when evaluated for 100 times



(a) k=6. (b) k=7.

Figure 5.6: Distribution of worst case latencies of cooperative k-means and standard k-means on Interoute topology when evaluated for 100 times

However, the ties are broken in favor of the switch with lower index. Hence, step 6 is also deterministic in nature. Since cooperative k-means algorithm does not involve any randomization or probability distribution, it is deterministic in nature.



(a) BT North America.

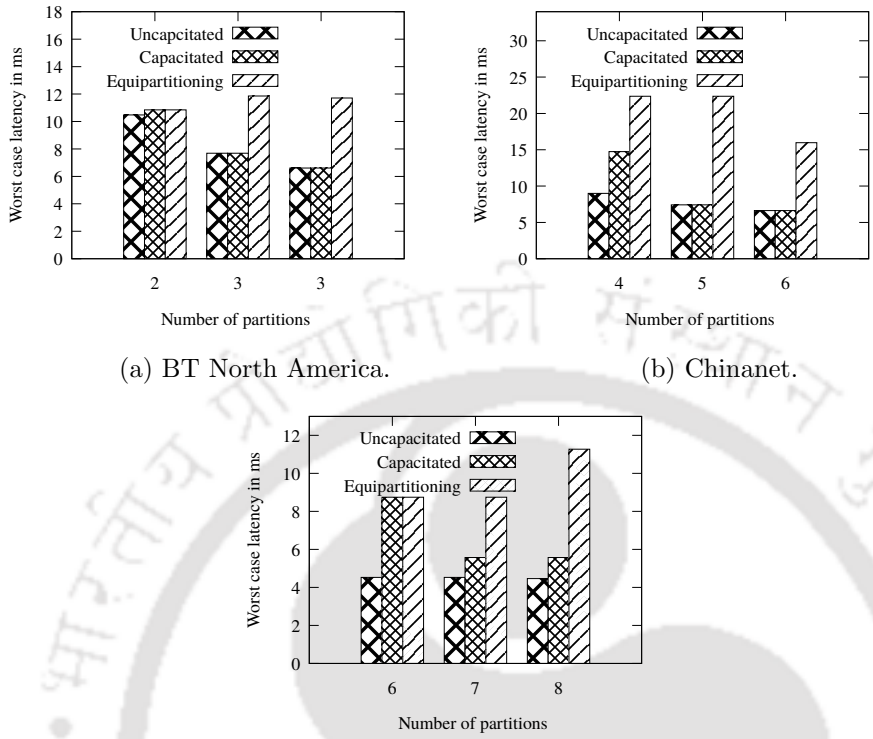
(b) Chinanet.

(c) Interoute.

Figure 5.7: Partition imbalance of cooperative k-means strategy under uncapacitated, capacitated and equipartition approaches.

The solution produced by a deterministic algorithm remains same across different execution instances. Therefore, the distribution curve of the cooperative k-means approach is a deterministic line. We can observe that the worst case latency of the solution generated by cooperative k-means is equal to the best solution generated by the standard k-means in 100 runs. Fig. 5.5 and Fig. 5.6 describe the distribution of worst case latencies of cooperative k-means and standard k-means when evaluated for 100 times on Chinanet and Interoute networks respectively. Similar trends can be observed with respect to the distribution of worst case latencies on Chinanet and Interoute networks.

We also evaluated the performance of capacitated cooperative k-means and equipartition cooperative k-means strategies on BT North America, Chinanet and



(a) BT North America.

(b) Chinanet.

(c) Interoute.

Figure 5.8: Worst case latency of cooperative k-means strategy under uncapacitated, capacitated and equipartition approaches.

Interoute networks. While evaluating the capacitated cooperative k-means, the demand of switches is set to 400 K/s [19, 111] and capacity of controllers is set to 7800 Kilo packets/s [19, 111]. That is, at most 7800/400=19 switches can be part of a partition. The partition imbalance of uncapacitated cooperative k-means, capacitated cooperative k-means and equipartition cooperative k-means strategies on different networks is presented in Fig. 5.7. We can observe from Fig. 5.7a that capacitated cooperative k-means approach performs better than the uncapacitated approach on BT North America when the number of partitions are two. However, the partition imbalance of both the approaches are same when the number of partitions are more than two. We can observe similar trends on Chinanet network with respect to the uncapacitated and capacitated strategies. It is evident from Fig. 5.7c that the

capacitated cooperative k-means approach performs better than the uncapacitated approach on Interoute network. Since equipartition approach ensures that all partition are of nearly same size, we can observe that it outperforms both capacitated and uncapacitated cooperative k-means strategies on all the networks.

The worst case latency of uncapacitated cooperative k-means, capacitated cooperative k-means and equipartition cooperative k-means strategies on different networks is presented in Fig. 5.8. We can observe from Fig. 5.8a that the uncapacitated cooperative k-means approach performs better than the capacitated approach on BT North America when the number of partitions are two. However, the worst case latency of both the approaches are same when the number of partitions are more than two. We can observe similar trends on Chinanet network with respect to the uncapacitated and capacitated strategies. It is evident from Fig. 5.8c that the uncapacitated cooperative k-means approach performs better than the capacitated approach on Interoute network. Since equipartition approach ensures that all partition are of nearly same size, we can observe that it results in higher worst case latency when compared to uncapacitated and capacitated cooperative k-means strategies on all the networks.

5.4 Summary

In this chapter, we proposed a controller placement strategy that partitions the network using k-means algorithm with cooperative game theory based initialization and deploys a controller in each of the partitions. The partitioning of the network into subnetworks is modeled as a cooperative game with the set of all switches as the players of the game. We also proposed two variants of cooperative k-means strategy that tries to produce partitions that are balanced in terms of size. The performance of our proposed strategy is evaluated on networks from Internet 2 OS3E and Internet Topology Zoo and compared it with the k-means algorithm

with random initialization. Results show that our strategy produces near optimal solutions and outperforms the standard k-means for controller placement. The load aware cooperative k-means strategies result in solutions with less partition imbalance when compared to the load unaware cooperative k-means approach.

Chapter 6

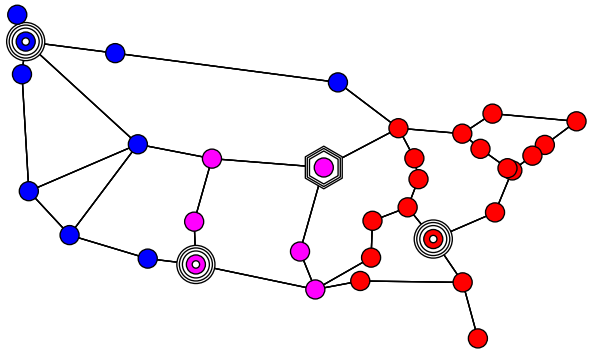
Placement of Hypervisors and Controllers in Virutalized SDNs

The virtualization of SDNs facilitates to leverage the combined benefits of SDN and network virtualization. That is, each tenant can bring their own controller to control their slice of the network. This can be realized by using the network hypervisor, which we refer as hypervisor for brevity. The hypervisor abstracts the physical substrate into multiple virtual networks and provides isolation among multiple tenants [23]. In a virtualized environment, the PACKET_IN messages of switches must pass through the hypervisor in order to reach the corresponding controller. The latency experienced by a network element is the sum of latency from the network element to the hypervisor and the latency from the hypervisor to the controller corresponding to the network element. Throughout the chapter, unless specified otherwise, latency refers to the sum of these two terms. Hence, the locations of the hypervisors and controllers together determines the latency of network elements in a virtualized environment.

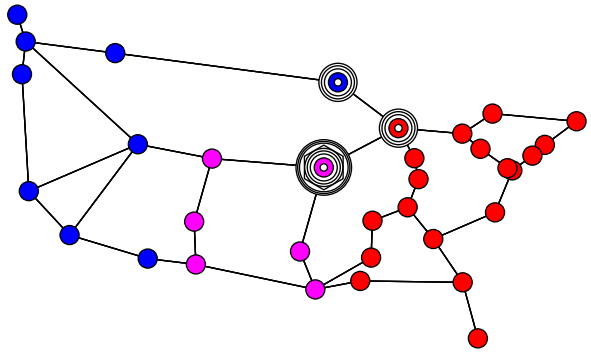
6.1 Motivation

Blenk *et al.* proposed a Hypervisor Placement Problem (HPP) that determine the optimal locations for deploying hypervisors in VSDNs while fixing controllers in each virtual network [32]. When evaluated on Internet 2 OS3E network with three virtual networks, the placement of controllers in each virtual network and hypervisor in the physical network computed using HPP strategy is shown in Fig. 6.1a. It results in a worst case virtual node to controller latency of 27.35 ms. Nodes marked with similar color belong to the same virtual network. On the other hand, determining optimal placement of controllers in each virtual network while fixing the hypervisors in the physical network is shown in Fig. 6.1b. It results in a worst case virtual node to controller latency of 20.43 ms. Therefore, determining the location of controllers in each virtual network while fixing the hypervisors is preferable than determining the location of the hypervisors while fixing controllers in each virtual network. Furthermore, jointly determining the placement of controllers in each virtual network and hypervisors in the physical network is shown in Fig. 6.1c. It results in an optimal worst case virtual node to controller latency of 18.39 ms.

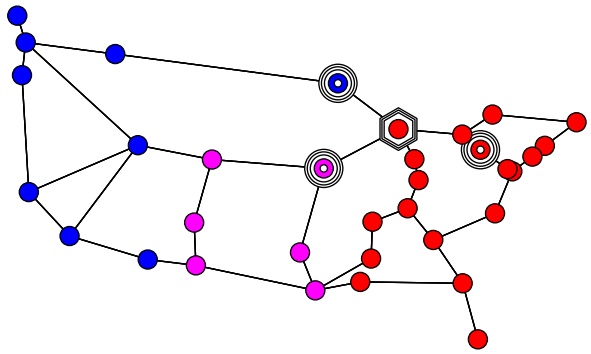
Motivated by this, we present different strategies for determining the placement of hypervisors and controllers in VSDNs. Since the packets sent by a network element pass through the hypervisor before reaching the controller in VSDNs, we present a strategy for determining the placement of controllers in VSDNs while fixing the hypervisor(s) in the physical network. It also allows the network operator to dynamically add new virtual networks on demand, therefore, beneficial in a SDN-based public cloud environment. The proposed problem, referred to as Controller Placement Problem in VSDNs (VCPP), is formulated as an ILP. We also present an approach for jointly optimizing the placement of hypervisors and controllers in VSDNs. It is beneficial in a SDN-based private cloud environment where the administrator has control over deployment of hypervisors and controllers. The



(a) Determining controllers in each virtual network followed by hypervisor in the physical network.



(b) Determining hypervisor in the physical network followed by controllers in each virtual network.



(c) Jointly determining placement of Hypervisors and Controllers.

Figure 6.1: Placement of Hypervisor and controllers in Internet 2 OS3E topology.

objective is to minimize the worst case latency between the network element and its corresponding controller. This problem, referred to as Joint Hypervisor and Controller Placement (JHCP) problem , is also formulated as an ILP. We also present a generalized ILP model for JHCP which can be used not only to optimize the worst case latency, but also to optimize other objectives such as the average latency, the maximum average latency, and the average maximum latency. Henceforth, it is referred to as generalized JHCP. To the best of our knowledge, this is the first work that analyzes various strategies for determining the placement of hypervisors and controllers in VSDNs.

The rest of this chapter is organized as follows. The formulation of the proposed problems are presented in Section 6.2. Results from a numerical evaluation of the proposed models are presented in Section 6.3. Section 6.4 concludes the chapter.

6.2 Problem Formulation

In this section, we first present the input parameters and state the assumptions of the models. Next, we present the optimization formulations of the problems and also present various other performance metrics.

6.2.1 Input parameters

In this section, we define the input parameters used in the formulation. Table 6.1 lists all the input parameters in the model.

The physical substrate is represented by $G(S, E)$, where S is the set of physical network elements and E is the set of links connecting the network elements. Let M be the set of virtual networks. We denote m^{th} virtual network by $VSDN_m$. Let V_m, $m \in M$, be the set of virtual nodes in the $VSDN_m$. Let q be the total number of hypervisors to be installed in the network. We denote the set of potential

Table 6.1: Notations

Symbol	Description
$G(S, E)$	Graph representing the substrate network
S	Set of physical nodes in the substrate
E	Set of links between the physical nodes
M	Set of virtual networks
$VSDN_m$	m^{th} virtual network
V_m	Set of virtual nodes in the $VSDN_m$
H	Set of potential locations for deploying hypervisors
q	Total number of hypervisors to be installed in the network
C_m	Potential locations for installing controllers in $VSDN_m$
p_m	Number of controllers to be deployed in $VSDN_m$
ϕ	Mapping from virtual node to physical node

locations for installing hypervisors with H. Let p_m be the number of controllers to be deployed in $VSDN_m$. We denote the set of potential locations for deploying controllers in $VSDN_m$ with C_m. Let $\phi(.)$ be a function that maps the virtual nodes to the corresponding physical nodes.

6.2.2 Assumptions

The following assumptions are used in the model.

- We assume that set of all physical nodes are the potential locations for installing hypervisors, i.e., $H = S$.

- We assume that set of all virtual nodes in $VSDN_m$ are the potential locations for installing controllers, i.e., C_m=V_m.

Table 6.2: Decision variables for VCPP

Variable	Description
y_k^m	=1, If the controller of $VSDN_m$ is deployed at $k \in V_m$ =0, otherwise
$r_{v,k}^m$	=1, If the virtual node $v \in V_m$ of $VSDN_m$ is assigned to the controller at $k \in V_m$ =0, otherwise

6.2.3 Controller Placement in VSDNs

In this subsection, we describe a strategy for determining the placement of controllers in VSDNs while fixing the hypervisor(s) in the physical network.

Decision variables

The variable y_k^m is set to one if the controller of $VSDN_m$ is deployed at $k \in V_m$, otherwise set to zero. The variable $r_{v,k}^m$ is set to one if the virtual node $v \in V_m$ is assigned to the controller at $k \in V_m$, otherwise set to zero.

Optimization Model

Given a virtual network and potential locations for deploying controllers, the goal of VCPP is to determine the locations for deploying controllers so as to minimize the worst case latency between the network element and its corresponding controller.

$$\min \left\{ \max_{\substack{v \in V_m \\ k \in V_m}} (d(\phi(v), h(\phi(v))) + d(h(\phi(v)), \phi(k))) \right\} \tag{6.1}$$

Here, $h(\phi(v))$ is the hypervisor of the physical node on which the virtual node v is embedded. Hence, the objective is to minimize the maximum latency, for all switches, the sum of latency from the switch to the hypervisor and the latency from

the hypervisor to the controller corresponding to the switch. Therefore, VCPP is formulated as follows:

$$\min z \tag{6.2}$$

Subject to the following constraints:

$$\sum_{k \in V_m} y_k^m = p_m \tag{6.3}$$

$$\sum_{k \in V_m} r_{v,k}^m = 1 \quad \forall v \in V_m \tag{6.4}$$

$$r_{v,k}^m \leq y_k^m \quad \forall v \in V_m, \forall k \in V_m \tag{6.5}$$

$$z \geq (d(\phi(v), h(\phi(v))) + d(h(\phi(v)), \phi(k)))r_{v,k}^m \quad \forall v \in V_m, \forall k \in V_m \tag{6.6}$$

$$y_k^m \in \{0, 1\} \quad \forall k \in V_m \tag{6.7}$$

$$r_{v,k}^m \in \{0, 1\} \quad \forall v \in V_m, \forall k \in V_m \tag{6.8}$$

Constraint (6.3) limits the number of controllers to be installed in a virtual network. It ensures that exactly p_m controllers have to be installed in virtual network $VSDN_m$. Constraint (6.4) guarantees that each virtual node $v \in V_m$ is assigned to exactly one controller. Constraint (6.5) ensures that each virtual node $v \in V_m$ is assigned to a valid controller. Constraint (6.6) guarantees that, for all switches, the objective value is greater than or equal to the sum of latency from virtual node to hypervisor and latency from hypervisor to controller corresponding to the virtual node. Constraints (6.7)-(6.8) force all the decision variables to take values either zero or one.

The above formulation can be extended to the average latency objective by replacing constraint (6.6) with the following constraint:

$$z = \frac{1}{V_m} \sum_{\substack{v \in V_m \\ k \in V_m}} (d(\phi(v), h(\phi(v))) + d(h(\phi(v)), \phi(k)))r_{v,k}^m \tag{6.9}$$

Note that VCPP can be applied independently on a set of virtual networks and the worst case latency and average latency across the set of virtual networks can be determined as shown in (6.10) and (6.11) respectively.

$$\min \left\{ \max_{\substack{m \in M \\ v \in V_m \\ k \in V_m}} (d(\phi(v), h(\phi(v))) + d(h(\phi(v)), \phi(k))) \right\} \tag{6.10}$$

$$\frac{1}{\sum_{m \in M} V_m} \sum_{\substack{m \in M \\ v \in V_m \\ k \in V_m}} (d(\phi(v), h(\phi(v))) + d(h(\phi(v)), \phi(k))) r^m_{v,k} \tag{6.11}$$

6.2.4 Joint Hypervisor and Controller Placement

In this subsection, we describe a strategy for jointly optimizing the placement of hypervisors and controllers in VSDNs.

Decision variables

The variable x_j specifies whether a hypervisor is installed at location $j \in H$ or not. It is set to one if a hypervisor is installed at location $j \in H$, otherwise set to zero. The variable $w^m_{v,j}$ determines the path followed by the virtual node $v \in V_m$ to reach the controller. It is set to one if the demand of the virtual node $v \in V_m$ traverses through the hypervisor deployed at location $j \in H$ to reach the corresponding controller, otherwise set to zero. For a physical node $i \in N$, the variable $h_{i,j}$ is set to one if it is controlled by the hypervisor deployed at $j \in H$, otherwise set to zero.

Optimization Model

The goal of JHCP is to jointly determine the locations for deploying hypervisors and controllers so as to minimize the worst case latency between the network element and its corresponding controller. Therefore, the objective is to minimize the maximum

Table 6.3: Decision variables for JHCP

Variable	Description
x_j	=1, If a hypervisor is deployed at potential hypervisor location $j \in H$ =0, otherwise
$w^m_{v,j}$	=1, If the demand of the virtual node $v \in V_m$ traverses through the hypervisor deployed at potential location $j \in H$ to reach the corresponding controller =0, otherwise
$h_{i,j}$	=1, If the physical node $i \in N$ is controlled by the hypervisor deployed at $j \in H$ =0, otherwise
$w^m_{v,j,k}$	=1, If the demand of the virtual node $v \in V_m$ of $VSDN_m$ traverse through the hypervisor deployed at potential hypervisor location $j \in H$ to reach the controller installed at $k \in V_m$ =0, otherwise
$t^m_{j,k}$	=1, If $x_j = 1$ and $y^m_k = 1$ =0, otherwise

latency, for all switches, the sum of latency from the switch to the hypervisor and the latency from the hypervisor to the controller corresponding to the switch.

$$\min \left\{ \max_{\substack{m \in M \\ v \in V_m}} (d(\phi(v), j) + d(j, \phi(k))) \right\}$$

where $d(\phi(v), j)$ is the minimum latency from the physical node i (where the virtual node v is embedded, i.e., $\phi(v) = i$) to the location $j \in H$ and $d(j, \phi(k))$ is the minimum latency from the hypervisor at j to the physical node $\phi(k)$. Note that the controllers of a virtual network $VSDN_m$ are deployed on virtual nodes V_m. The JHCP can be formulated as follows:

$$\min \ \ z \tag{6.12}$$

Subject to the following constraints

$$\sum_{k \in V_m} y_k^m = p_m \ \ \forall m \in M \tag{6.13}$$

$$\sum_{k \in V_m} r_{v,k}^m = 1 \ \ \forall m \in M, \forall v \in V_m \tag{6.14}$$

$$r_{v,k}^m \leq y_k^m \ \ \forall m \in M, \forall v \in V_m, \forall k \in V_m \tag{6.15}$$

$$\sum_{j \in H} x_j = q \tag{6.16}$$

$$\sum_{j \in H} w_{v,j}^m = 1 \ \ \forall m \in M, \forall v \in V_m \tag{6.17}$$

$$\sum_{v \in V_m} w_{v,j}^m \leq |V_m| x_j \ \ \forall m \in M, \forall j \in H \tag{6.18}$$

$$\begin{aligned} &w_{v,j}^m \leq h_{i,j} \\ &\forall m \in M, \forall v \in V_m, \forall j \in H, \forall i \in N : \phi(v) = i \end{aligned} \tag{6.19}$$

$$h_{i,j} \leq \sum_{m \in M} \sum_{\{v \in V_m :\, \phi(v) = i\}} w_{v,j}^m \ \ \forall i \in N, \forall j \in H \tag{6.20}$$

$$\sum_{j \in H} h_{i,j} \leq 1 \ \ \forall i \in N \tag{6.21}$$

$$z \geq (d(\phi(v), j) + d(j, k))(w^m_{v,j} + r^m_{v,k} - 1) \quad \forall m \in M, \forall v \in V_m, \forall j \in H, \forall k \in V_m \tag{6.22}$$

$$y^m_k \in \{0, 1\} \quad \forall m \in M, \forall k \in V_m \tag{6.23}$$

$$r^m_{v,k} \in \{0, 1\} \quad \forall m \in M, \forall v \in V_m, \forall k \in V_m \tag{6.24}$$

$$x_j \in \{0, 1\} \quad \forall j \in H \tag{6.25}$$

$$w^m_{v,j} \in \{0, 1\} \quad \forall m \in M, \forall v \in V_m, \forall j \in H \tag{6.26}$$

$$h_{i,j} \in \{0, 1\} \quad \forall i \in N, \forall j \in H \tag{6.27}$$

Constraint (6.13) limits the number of controllers to be installed in a virtual network. It ensures that exactly p_m controllers have to be installed in each virtual network $VSDN_m$. Constraint (6.14) guarantees that each virtual node $v \in V_m$ is assigned to exactly one controller. Constraint (6.15) ensures that each virtual node $v \in V_m$ is assigned to a valid controller. Constraint (6.16) guarantees that the total number of hypervisors installed in the network is exactly equal to q. Constraint (6.17) ensures that the demand of each virtual node $v \in V_m$ traverses through exactly one hypervisor to reach the controller. That is, each virtual node $v \in V_m$ is assigned to exactly one hypervisor. Constraint (6.18) guarantees that a hypervisor is installed at location j, if at least one virtual node $v \in V_m$ is connected to its controller via the hypervisor j. Constraint (6.19) sets a hypervisor node j to be the controller of the physical node i ($t_{i,j} = 1$) if the virtual node $v \in V_m$ is embedded at i ($\phi(v) = i$) uses the hypervisor deployed at $j \in H$ to reach the corresponding controller. Constraint (6.20) forces the variable $h_{i,j}$ to zero if none of the virtual node $v \in V_m$ uses the hypervisor deployed at $j \in H$ for sending requests to the corresponding controller. Constraint (6.21) ensures that each physical node is controlled by at most one hypervisor. Note that each virtual node resides on a physical node. If none of the virtual nodes resides in a physical node, then there is no need of assigning a hypervisor. Constraint (6.22) guarantees that, for all switches,

the objective value is greater than or equal to the sum of latency from switch to hypervisor and latency from hypervisor to controller corresponding to the switch. Constraints (6.23)-(6.27) force all the decision variables to take values either zero or one.

6.2.5 Generalized JHCP

In this subsection, we describe a generalized ILP model for JHCP which can be used not only to optimize the worst case latency, but also to optimize other objectives such as the average latency, the maximum average latency, and the average maximum latency.

The objective of the JHCP formulation presented in Section 6.2.4 is to minimize, for all virtual nodes in the network, the maximum latency from the virtual node to its controller. If the objective is to minimize the average latency of the network, the simple and direct extension of (6.22) as shown below is erroneous.

$$z = \frac{1}{\sum_{m \in M} V_m} \sum_{\substack{m \in M \\ v \in V_m}} \sum_{\substack{j \in H \\ k \in V_m}} (d(\phi(v), j) + d(j, k))(w^m_{v,j} + r^m_{v,k} - 1) \tag{6.28}$$

Because the term $(d(\phi(v), j) + d(j, k))(w^m_{v,j} + r^m_{v,k} - 1)$ in (6.28) is negative when $w^m_{v,j} = 0$ and $r^m_{v,k} = 0$. For the equation (6.28) to work correctly, the term $(d(\phi(v), j) + d(j, k))(w^m_{v,j} + r^m_{v,k} - 1)$ should be zero in all other cases except when $w^m_{v,j} = 1$ and $r^m_{v,k} = 1$. The product of the terms $w^m_{v,j}$ and $r^m_{v,k}$, i.e., $w^m_{v,j} r^m_{v,k}$ is zero in all other cases except when $w^m_{v,j} = 1$ and $r^m_{v,k} = 1$. Thus, the term $(w^m_{v,j} + r^m_{v,k} - 1)$ in (6.28) can be replaced with $w^m_{v,j} r^m_{v,k}$ as shown below:

$$z = \frac{1}{\sum_{m \in M} V_m} \sum_{\substack{m \in M \\ v \in V_m}} \sum_{j \in H} \sum_{k \in V_m} (d(\phi(v), j) + d(j, k)) w^m_{v,j} r^m_{v,k} \tag{6.29}$$

Constraint (6.29) ensures the correct objective value, i.e., the average latency of the network. However, the term $w^m_{v,j} r^m_{v,k}$ in the constraint (6.29) makes it non linear.

We can transform it to a linear constraint by introducing a new variable $w^m_{v,j,k}$ by combining $w^m_{v,j}$ and $r^m_{v,k}$. The variable $w^m_{v,j,k}$ determines the path taken by the virtual node $v \in V_m$ to reach the controller installed at $k \in V_m$. It is set to one if the demand of the virtual node $v \in V_m$ traverses through the hypervisor deployed at potential hypervisor location $j \in H$ to reach the controller installed at $k \in V_m$, otherwise set to zero.

The goal of JHCP, in this subsection, is to jointly determine the locations for deploying hypervisors and controllers so as to minimize, for all virtual nodes in the network, the average latency between the network element and its corresponding controller. That is, the objective is to minimize the average, for all virtual nodes in the network, the sum of latency from the virtual node to the hypervisor and the latency from the hypervisor to the controller corresponding to the virtual node.

$$\min \left\{ \frac{1}{\sum\limits_{m \in M} V_m} \sum_{\substack{m \in M \\ v \in V_m}} \sum_{j \in H} \sum_{k \in C} (d(\phi(v), j) + d(j, k)) w^m_{v,j,k} \right\}$$

The JHCP for average latency objective is formulated as follows:

$$\min z \tag{6.30}$$

Subject to (6.13)- (6.16), (6.21), (6.23) - (6.25), (6.27) and

$$\sum_{j \in H} \sum_{k \in V_m} w^m_{v,j,k} = 1 \quad \forall m \in M, \forall v \in V_m \tag{6.31}$$

$$\sum_{v \in V_m} w^m_{v,j,k} \leq |V_m| x_j y^m_k \quad \forall m \in M, \forall j \in H, \forall k \in V_m \tag{6.32}$$

$$\sum_{k \in V_m} w^m_{v,j,k} \leq h_{i,j} \quad \forall m \in M, \forall v \in V_m, \forall j \in H, \forall i \in N : \phi(v) = i \tag{6.33}$$

$$h_{i,j} \leq \sum_{m \in M} \sum_{\{v \in V_m : \phi(v) = i\}} \sum_{k \in V_m} w^m_{v,j,k} \quad \forall i \in N, \forall j \in H \tag{6.34}$$

$$z \geq \frac{1}{\sum_{m \in M} V_m} \sum_{\substack{m \in M \\ v \in V_m}} \sum_{j \in H} \sum_{k \in C} (d(\phi(v), j) + d(j, k)) w^m_{v,j,k} \tag{6.35}$$

$$\begin{aligned} & w^m_{v,j,k} \in \{0, 1\} \\ & \forall m \in M, \forall v \in V_m, \forall j \in H, \forall k \in V_m, \forall i \in N \end{aligned} \tag{6.36}$$

Constraint (6.31) ensures that each virtual node $v \in V_m$ is assigned to exactly one controller and the demand of v traverses through exactly one hypervisor to reach the controller. That is, each virtual node $v \in V_m$ uses exactly one pair of hypervisor and controller. Constraint (6.32) guarantees that a hypervisor is installed at location j and a controller for $VSDN_m$ is deployed at k, if at least one virtual node $v \in V_m$ is connected to its controller via the hypervisor j. It also upper bounds the number of virtual nodes connected to their controllers via the hypervisor j at V_m. Constraint (6.33) sets a hypervisor node j to be the controller of the physical node i ($h_{i,j} = 1$) if the virtual node $v \in V_m$ is embedded at i ($\phi(v) = i$) uses the hypervisor deployed at $j \in H$ to reach the controller installed at $k \in C$ ($w^m_{v,j,k} = 1$). Constraint (6.34) forces the variable $z_{i,j}$ to zero if none of the virtual node $v \in V_m$ uses the hypervisor deployed at $j \in H$ for sending requests to the controller installed at $k \in C$. Constraint (6.36) ensures that the decision variables take values either zero or one.

Note that the term $x_j y^m_k$ in the constraint (6.32) makes it non linear. We can transform it to a linear constraint by introducing an additional variable $t^m_{j,k}$ and some extra constraints. The variable $t^m_{j,k}$ is set to one if both x_j and y^m_k are one, otherwise set to zero. Thus, we can replace the constraint (6.32) with the following additional constraints.

$$\sum_{v \in V_m} w^m_{v,j,k} \leq |V_m| t^m_{j,k} \quad \forall m \in M, \forall j \in H, \forall k \in V_m \tag{6.37}$$

$$t^m_{j,k} \geq x_j + y^m_k - 1 \quad \forall m \in M, \forall j \in H, \forall k \in V_m \tag{6.38}$$

$$t^m_{j,k} \leq x_j \quad \forall m \in M, \forall j \in H, \forall k \in V_m \tag{6.39}$$

$$t^m_{j,k} \leq y^m_k \quad \forall m \in M, \forall j \in H, \forall k \in V_m \tag{6.40}$$

$$t^m_{j,k} \in \{0,1\} \quad \forall m \in M, \forall j \in H, \forall k \in V_m \tag{6.41}$$

Constraint (6.37) is the linear constraint corresponding to (6.32). Constraints (6.38)-(6.40) together ensures that the variable $t^m_{j,k}$ takes a value one when both x_j and y^m_k are one. Constraint (6.41) guarantees that the variable $t^m_{j,k}$ takes a binary value. Constraints (6.38)-(6.41) together ensures that the variable $t^m_{j,k}$ takes a value zero when either of x_j or y^m_k is zero or both of them are zero.

6.2.6 Other Objectives

Besides optimizing for the worst case latency or average latency, one can deploy hypervisors and controllers at locations that minimize the maximum average latency or the average maximum latency of the network.

Maximum Average Latency

It is the maximum over the average latencies of the individual virtual networks.

$$\max_{m \in M} \frac{1}{|V_m|} \sum_{v \in V_m} \sum_{j \in H} \sum_{k \in C} (d(\phi(v), j) + d(j,k)) w^m_{v,j,k} \tag{6.42}$$

This can be implemented by replacing the constraint (6.35) in Section 6.2.5 with the following constraint:

$$z \geq \frac{1}{|V_m|} \sum_{v \in V_m} \sum_{j \in H} \sum_{k \in C} (d(\phi(v), j) + d(j,k)) w^m_{v,j,k} \quad \forall m \in M \tag{6.43}$$

Average Maximum Latency

It is the average over the maximum latencies of the individual virtual networks.

$$\frac{1}{|M|} \sum_{m \in M} \max_{v \in V_m} \sum_{j \in H} \sum_{k \in C} (d(\phi(v), j) + d(j,k)) w^m_{v,j,k} \tag{6.44}$$

The worst case latency of each virtual network can be determined by introducing an additional decision variable for each virtual network, and the following constraints:

$$z_m \geq \sum_{j \in H} \sum_{k \in c} (d(\phi(v), j) + d(j, k)) w^m_{v,j,k} \quad \forall m \in M, \forall v \in V_m \tag{6.45}$$

Therefore, the average maximum latency objective can be implemented by replacing constraint (6.35) in Section 6.2.5 with (6.45) and (6.30) in Section 6.2.5 with the following:

$$\min \frac{1}{|M|} \sum_{m \in M} z_m \tag{6.46}$$

6.3 Numerical Results

In this section, we evaluate the proposed models and present the results obtained. We compare the proposed models with the hypervisor placement problem from the literature.

6.3.1 Evaluation Setup

The proposed VCPP and JHCP placement strategies are evaluated on the AT&T Norh America network (25 nodes and 57 links) of Internet Topology Zoo [33] and Internet 2 OS3E topology (34 nodes and 41 edges) [98]. The longitude and latitude information of nodes obtained from [33, 126] are used to compute the shortest path propagation delay between nodes in the networks. While evaluating the performance of VCPP, we considered various hypervisor placement strategies such as deploying at random locations (VCPP-RndHP), locations that minimize the worst case latency (VCPP-WorstHP), and locations that minimize the average latency (VCPP-AvgHP). The efficacy of proposed placement strategies are compared with HPP. While evaluating the performance of HPP, we considered various controller

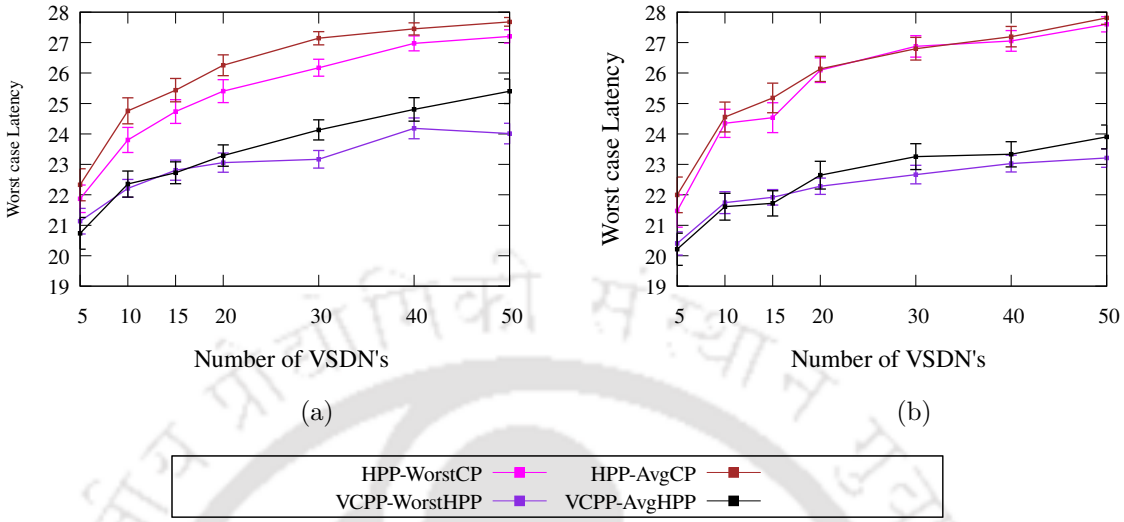


Figure 6.2: Performance of VCPP and HPP while optimized for the worst case latency. (a) AT&T network. (b) Internet 2 OS3E topology.

placement strategies such as deploying at random locations (HPP-RndCP), locations that minimize the worst case latency (HPP-WorstCP), and locations that minimize the average latency (HPP-AvgCP). The number of hypervisors in the network is set to one. We assume that each virtual network is controlled by a single controller. The number of virtual nodes in each virtual network are uniformly generated between 3 and 7. The proposed models are implemented in MATLAB [112] and solved using IBM CPLEX solver [113]. Each instance is executed for 100 times and 95% confidence intervals are presented for statistical reliability.

6.3.2 Performance analysis of VCPP in static scenario

Fig. 6.2 illustrates the performance of VCPP and HPP when optimized for the worst case latency. We can observe that VCPP outperforms HPP because VCPP first fixes the hypervisors in the network and then determines the location of controllers in each virtual network whereas HPP first deploys controllers in each virtual network

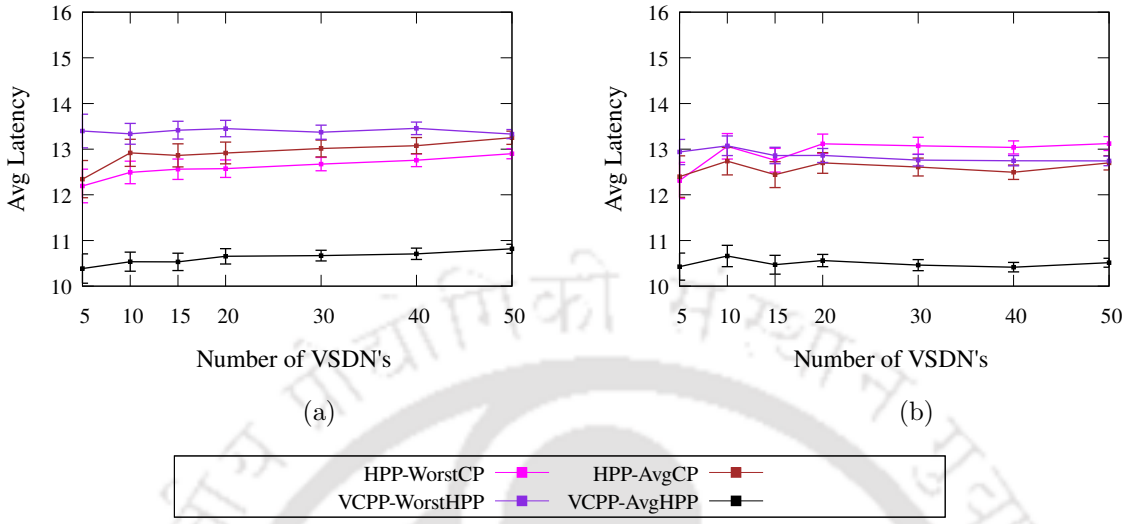


Figure 6.3: Performance of VCPP and HPP while optimized for the average latency. (a) AT&T network. (b) Internet 2 OS3E topology.

and then determine the locations of the hypervisors. We can also observe from Fig. 6.2a and Fig. 6.2b that the worst case latency of VCPP and HPP increases with the number of virtual networks. The gap between the worst case latencies of VCPP and HPP increases with the number of virtual networks. In other words, the performance of HPP deteriorates with the increasing number of virtual networks. Therefore, the network designer prefers VCPP over HPP when there are a higher number of virtual networks. VCPP-WorstHPP and VCPP-AvgHPP in Fig. 6.2a and Fig. 6.2b deploy the controllers at locations that minimize the worst case latency of the virtual network. However, they deploy the hypervisors at locations that minimize the worst case latency and the average latency between network element and hypervisor respectively. Therefore, the worst case latency of VCPP-WorstHPP is less than the VCPP-AvgHPP.

Fig. 6.3 demonstrate the efficacy of VCPP over HPP on when they are optimized for the average latency. We can observe similar trends, i.e., VCPP performs better than HPP. VCPP-WorstHPP and VCPP-AvgHPP in Fig. 6.3a and

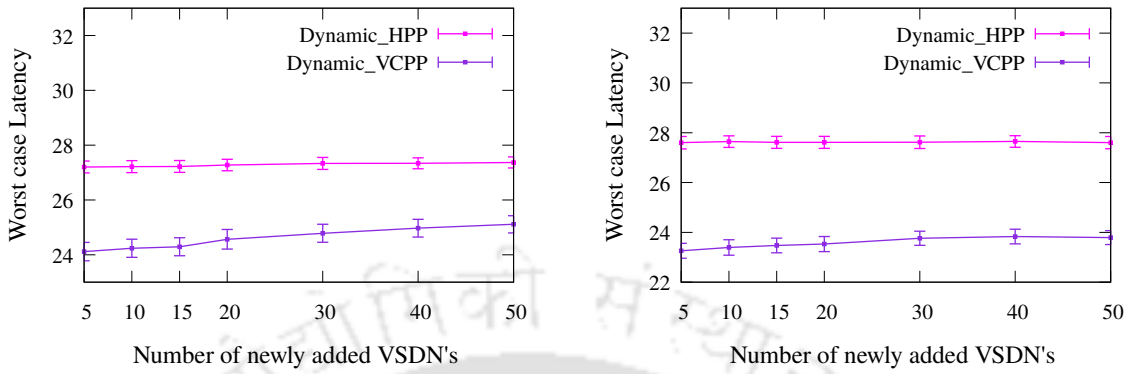


Figure 6.4: Performance of Dynamic_VCPP and Dynamic_HPP while optimized for the worst case latency. (a) AT&T network. (b) Internet 2 OS3E topology.

Fig. 6.3b deploy the controllers at locations that minimize the average latency of the virtual network. However, they deploy the hypervisors at locations that minimize the worst case latency and the average latency between network element and hypervisor respectively. Therefore, the average latency of VCPP-AvgHPP is less than the VCPP-WorstHPP.

6.3.3 Performance analysis of VCPP in dynamic scenario

HPP necessitate the number of virtual networks to be known before hand because it deploys the hypervisors after fixing the controllers in every virtual network. However, the virtual networks are being added dynamically (on demand basis) in a SDN-based public cloud environment. Hence, after deploying controllers for an initial number of virtual networks and hypervisors in the physical network, HPP must fall-back to VCPP for determining controller locations in newly added virtual networks. We computed the location of controllers and hypervisors using VCPP and HPP while fixing the initial number of virtual networks to 50. Then, we evaluated the performance of VCPP and HPP by adding new virtual networks ranging from

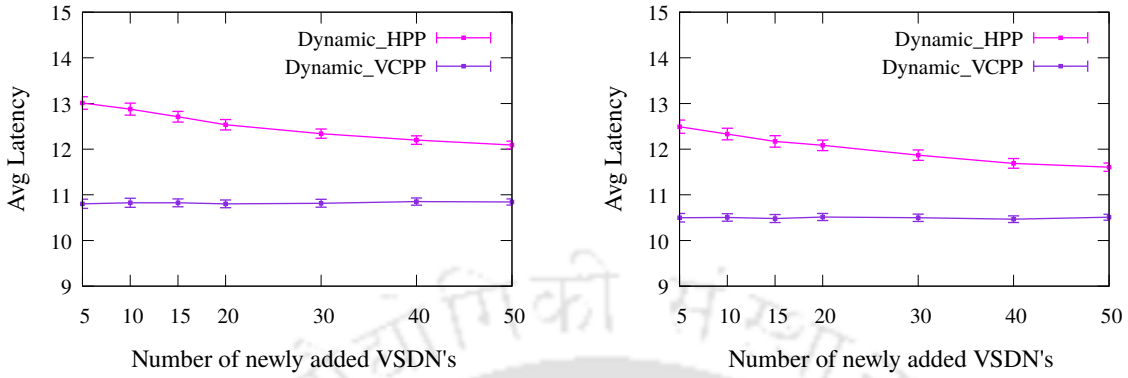
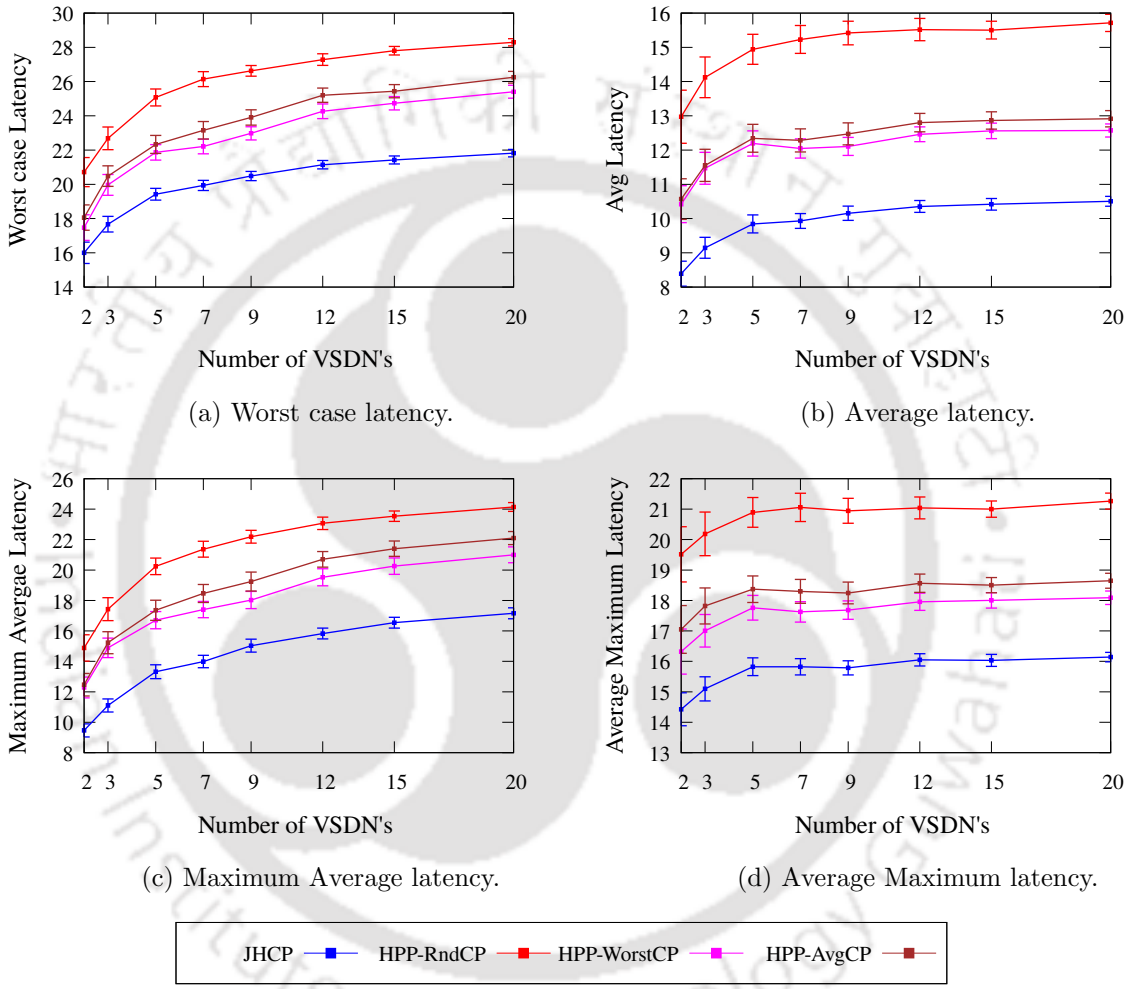


Figure 6.5: Performance of Dynamic_VCPP and Dynamic_HPP while optimized for the average latency. (a) AT&T network. (b) Internet 2 OS3E topology.

5 to 50. We use the terms Dynamic_VCPP and Dynamic_HPP to refer VCPP and HPP when the virtual networks are added on demand basis. Fig. 6.4 and Fig. 6.5 demonstrate the efficacy of Dynamic_VCPP over Dynamic_HPP when they are optimized for the worst case latency and average latency respectively. It is evident that Dynamic_VCPP performs better than Dynamic_HPP.

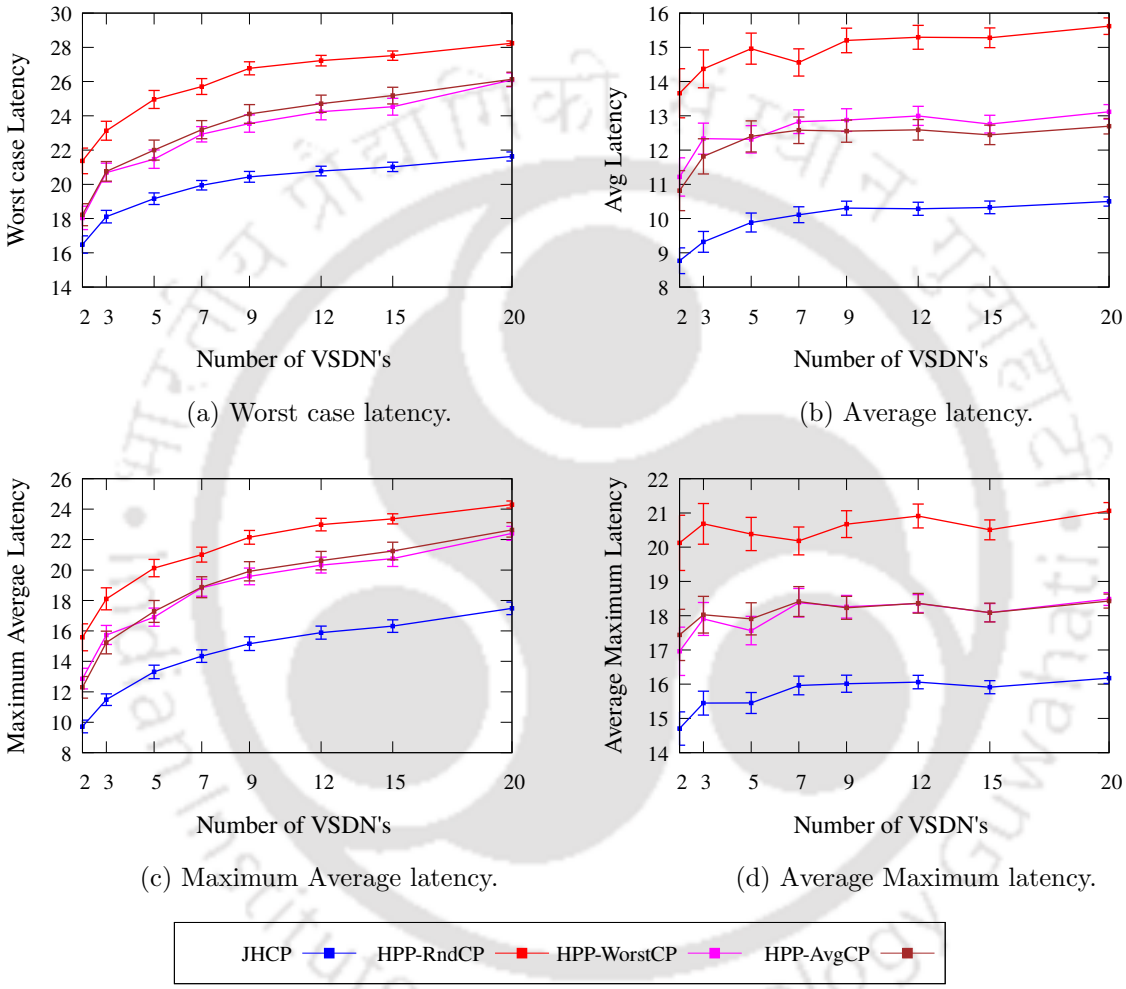
6.3.4 Performance analysis of JHCP

Fig. 6.6 and Fig. 6.7 illustrate the efficacy of JHCP over HPP on AT&T and Internet 2 OS3E networks when they are optimized for the worst case latency and average latency respectively. We can observe from Fig. 6.6a and Fig. 6.7a that JHCP results in a lower worst case latency when compared to HPP because JHCP jointly determine the locations of hypervisors and controllers whereas HPP determines the location of hypervisors in the physical network and controllers in each virtual network separately. The worst case latency of JHCP and HPP increases with the number of virtual networks. Additionally, the gap between the worst case latencies of JHCP and HPP increases with the number of virtual networks. Furthermore, the



(a) Worst case latency. (b) Average latency. (c) Maximum Average latency. (d) Average Maximum latency.

Figure 6.6: Performance of JHCP and HPP on ATT network while optimized for various metrics. (a) Worst case latency. (b) Average latency. (c) Maximum Average latency. (d) Average Maximum latency.



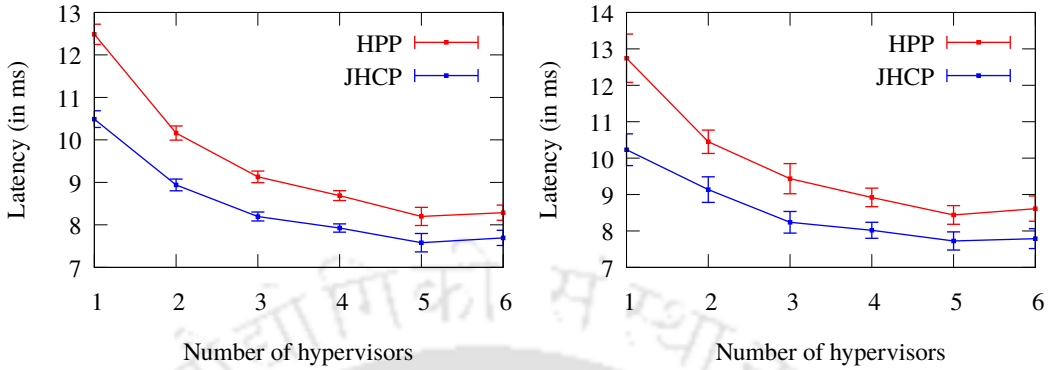
(a) Worst case latency.

(b) Average latency.

(c) Maximum Average latency.

(d) Average Maximum latency.

Figure 6.7: Performance of JHCP and HPP on Internet 2 OS3E network while optimized for various metrics. (a) Worst case latency. (b) Average latency. (c) Maximum Average latency. (d) Average Maximum latency.



(a) AT&T network (b) Internet 2 OS3E network

Figure 6.8: Effect of hypervisors on latency when optimized for the average latency

average latency, the maximum average latency, and the average maximum latency of JHCP is also less than HPP which is evident from Fig. 6.6b - Fig. 6.6d and Fig. 6.7b - Fig. 6.7d. JHCP can be used to optimally deploy hypervisors and controllers in a SDN-based private cloud environment in which the administrator has control over deployment of hypervisors and controllers. However, after deploying controllers in an initial number of virtual networks and hypervisors in the physical network, JHCP fall-back to VCPP for deploying controllers in a SDN-based public cloud environment.

6.3.5 Impact of number of hypervisors and controllers

The impact of number of hypervisors on the latency when HPP and JHCP are evaluated on AT&T and Internet 2 OS3E networks is presented in Fig. 6.8a and Fig. 6.8b respectively. We have optimized for the average latency because it characterizes the distribution of virtual node latencies. The number of virtual networks is fixed at 12 and the number of hypervisors are varied from one to six. We can observe that the average latency of HPP and JHCP decreases with the increasing number of hypervisor instances in the physical network. We can also observe that

Table 6.4: Complexity analysis of HPP, VCPP and JHCP

HPP	Number of Decision Variables	$1 + \|H\| + \|N\| * \|H\| + \sum_{m \in M} \|V_m\| * \|N\|$
	Number of equality constraints	$1 + \sum_{m \in M} \|V_m\|$
	Number of inequality constraints	$\|H\| + \|N\| + \sum_{m \in M} \|V_m\| + \sum_{m \in M} \|V_m\| * \|N\| * \|H\|$
VCPP	Number of Decision Variables	$1 + \|H\| + \sum_{m \in M} \|V_m\| + \sum_{m \in M} \|V_m\|^2$
	Number of equality constraints	$1 + \|M\| + \sum_{m \in M} \|V_m\|$
	Number of inequality constraints	$\sum_{m \in M} 2 * \|V_m\|^2$
JHCP for Worst case latency	Number of Decision Variables	$1 + \|H\| + \|N\| * \|H\| + \sum_{m \in M} \|V_m\| + \sum_{m \in M} \|V_m\|^2 + \sum_{m \in M} \|V_m\| * \|H\|$
	Number of equality constraints	$1 + \|M\| + 2 * \sum_{m \in M} \|V_m\|$
	Number of inequality constraints	$\|N\| + \|M\| * \|H\| + \|H\| * \|N\| + \sum_{m \in M} \|V_m\|^2 + \sum_{m \in M} \|V_m\| * \|N\| + \sum_{m \in M} \|V_m\|^2 * \|H\|$
Generalized JHCP for all objectives	Number of Decision Variables	$1 + \|H\| + \|N\| * \|H\| + \sum_{m \in M} \|V_m\| + \sum_{m \in M} \|V_m\|^2 + \sum_{m \in M} \|V_m\| * \|H\| + \sum_{m \in M} \|V_m\|^2 * \|H\|$
	Number of equality constraints	$2 + \|M\| + 2 * \sum_{m \in M} \|V_m\|$
	Number of inequality constraints	$\|N\| + \|H\| * \|N\| + \sum_{m \in M} \|V_m\|^2 + 5 * \sum_{m \in M} \|V_m\| * \|H\| + \sum_{m \in M} \|V_m\|^2 * \|H\|$

JHCP performs better than HPP in every scenario. Similarly, we argue that the latency experienced by the nodes in a virtual network reduces with the increasing number of controller instances in the virtual network.

6.3.6 Complexity analysis

The complexity of HPP, VCPP, JHCP for worst case latency and generalized JHCP is presented in terms of the number of decision variables, and the number of inequality and equality constraints in Table 6.4. Since $H = N$ and $|V_m| \subseteq N$, the number of decision variables in JHCP for worst case latency and HPP is asymptotically equal to $O(\sum_{m \in M} |V_m| * |N|)$. VCPP has $O(\sum_{m \in M} |V_m| * |V_m|)$ decision variables which is asymptotically smaller than HPP and JHCP for worst case latency because $V_m \subseteq N$. Additionally, generalized JHCP has $O(\sum_{m \in M} |V_m|^2 * |H|)$ decision variables which is asymptotically larger than the number of decision variables in HPP, VCPP and JHCP for worst case latency. The number of equality constraints and inequality constraints in HPP are asymptotically equal to $O(\sum_{m \in M} |V_m|)$ and $O(\sum_{m \in M} |V_m| * |H|)$ respectively. VCPP, JHCP for worst case latency, and generalized JHCP has $O(\sum_{m \in M} |V_m|)$ equality constraints which is asymptotically equal to the number of equality constraints in HPP. VCPP has $O(\sum_{m \in M} |V_m| * |V_m|)$ inequality constraints which is asymptotically smaller than the number of inequality constraints in HPP because $V_m \subseteq N$. Furthermore, JHCP for worst case latency and generalized JHCP has $O(\sum_{m \in M} |V_m|^2 * |H|)$ inequality constraints which is asymptotically larger than the number of inequality constraints in HPP and VCPP. Therefore, the amount of physical memory required by JHCP for worst case latency and k-HPP is same. Generalized JHCP demands more physical memory for large scale networks when compared to HPP. The amount of physical memory required by VCPP is less than other approaches because it has a fewer number of decision variables and inequality constraints.

6.4 Conclusion

In this chapter, we addressed two strategies, namely VCPP and JHCP, for determining the placement of hypervisors and controller in VSDNs. VCPP fixes the hypervisor(s) in the physical network and then determines the placement of controllers in each of the virtual network. It allows the network operator to dynamically add new virtual networks on demand basis. JHCP jointly determines the placement of hypervisors in the physical network and controllers in each virtual network. The performance of our proposed strategies are evaluated on AT&T network of Internet Topology Zoo and Internet 2 OS3E topology, and the results are compared with the existing HPP model. We considered different objective functions such as max latency, average latency, maximum average latency, and average maximum latency. The complexity of JHCP in terms of decision variables and constraints is asymptotically larger than HPP whereas the complexity of VCCP is asymptotically smaller than HPP and JHCP. Evaluation results show that VCPP and JHCP perform better than k- HPP with respect to the different performance metrics. Extensive evaluation using real world network topologies show that proposed models perform better than k- HPP with respect to the different performance metrics.

Chapter 7

Summary and Future Directions

The work in this thesis addressed the research problems of controller placement and hypervisor placement in SDNs. We proposed optimization models for minimizing the worst case latency between network element and controller while satisfying various constraints.

First, we addressed the failure foresight capacitated controller placement problem in SDNs that avoid disconnections, repeated administrative intervention, and drastic increase in the worst case latency in case of controller failures. We designed an optimization model for a single controller failure and extended it to multiple controller failures. The objective is to minimize the worst-case latency between switches and their μ^{th} reference controllers while satisfying the capacity and closest assignment constraints. We also designed a variant of failure foresight capacitated controller placement that minimizes the sum of worst-case latencies from switches to their 1^{st}, 2^{nd} ,..., μ^{th} reference controllers. Further, we relaxed the failure foresight assumption of switches and investigated a capacitated next controller placement strategy that not only considers capacity and reliability of controllers but also plans ahead for controller failures. We designed an optimization model for a single controller failure and extended it to multiple controller failures.

We also presented a simulated annealing heuristic to produce fast and viable solution on large networks. The proposed formulations and heuristic are evaluated on various networks from the Internet Topology Zoo. The proposed models achieve a significant improvement in the worst case latency in the event of failures and inter controller latency. Results also shows that the heuristic is able to achieve near optimal solutions in less than half of the time required by the optimized formulations.

Next, we proposed a controller placement strategy that partitions the network using k-means algorithm with cooperative game theory based initialization and deploys a controller in each of the partitions. The partitioning of the network into subnetworks is modeled as a cooperative game with the set of all switches as the players of the game. We also proposed two variants of cooperative k-means strategy that tries to produce partitions that are balanced in terms of size. The performance of our proposed strategy is evaluated on networks from Internet 2 OS3E and Internet Topology Zoo and compared it with the k-means algorithm with random initialization. Results show that our strategy produce near optimal solutions and outperforms the standard k-means for controller placement. The partition imbalance of load aware cooperative k-means strategies is less when compared to the load unaware cooperative k-means approach.

Finally, we addressed two strategies for determining the placement of hypervisors and controller in VSDNs. The first strategy fixes the hypervisor(s) in the physical network and then determines the placement of controllers in each of the virtual network. It allows the network operator to dynamically add new virtual networks on demand basis. The second approach jointly determines the placement of hypervisors in the physical network and controllers in each virtual network. The performance of our proposed strategies are evaluated on ATT network of Internet Topology Zoo and the results are compared with the existing hypervisor placement model. We considered different objective functions such as max latency, average

latency, maximum average latency, and average maximum latency. Extensive evaluation using real world network topologies show that proposed models perform better than the existing hypervisor placement model with respect to the different performance metrics.

7.1 Conclusions

From this work, we conclude that it is indeed possible to avoid disconnections, repeated administrative intervention, and drastic increase in the worst case latency in case of controller failures by planning ahead for the failures while deploying controllers in the network. Our proposed models not only performs better in terms of the worst case latency in the event of failures but also in terms of maximum and average inter controller latencies. Next, we conclude that k-means algorithm with cooperative game theory based initialization not only results in solutions that are close to optimal solution but also deterministic in nature. The load aware cooperative k-means strategies results in solutions with less partition imbalance when compared to the load unaware cooperative k-means approach. Finally, we conclude that determining the placement in each of the virtual network while fixing the hypervisor(s) in the physical network and jointly determining the placement of hypervisors in the physical network and controllers in each virtual network are efficient than determining the hypervisor(s) in the physical network while fixing the controllers in each of the virtual network.

If a network planner wants to deal with placement of controllers, our proposed FFCCP and CNCP approaches are recommended when the controller failures are more often in the network and standard CCP is recommended when the network operates under normal conditons for the major portion of time. Since the network operates under combination of normal and failure conditions together, we recommend the CO-FFCCP with appropriate weights to the network planner.

Further, we recommend our proposed cooperative k-means approach and its variants while dealing with placement of controllers in a wide area network. Our proposed JHCP approach is recommended in a SDN-based private cloud environment where the administrator has control over deployment of hypervisors and controllers.

7.2 Future Directions

There are several possible directions in which the work in this thesis can be extended. We list a few immediate extensions of our work.

- Only the failure of controllers has been considered in our work. One can extend our models to consider the failure of network links, networking elements and a combination of these.
- In this thesis, we reserved the capacity on backup controllers so that they serve the switches in the event of primary controller failure. We can extend our model to consider assigning the $\frac{1}{\mu-1}$ fraction of a switch load to μ different controllers, thereby, minimizing the backup capacity reserved on the controllers while ensuring the reliability. In failure free case, first $\mu - 1$ controllers of a switch serve its PACKET_IN messages. In case of a controller failure, the remaining $\mu - 1$ controllers of a switch serve its PACKET_IN messages.
- In this thesis, we considered latency and controller load as performance metrics for network partitioning based controller placement strategy. Considering fault tolerant requirements would be an interesting extension.
- While determining the placement of controllers and hypervisors in virtualized software defined network, capacity is not considered in our work. One can extend our models to consider the capacity of controllers, hypervisors and a combination of these.

- The models presented in this thesis can be implemented on Mininet kind of platforms to measure how the performance improvement of optimal placement of controllers is getting reflected in reality.

References

[1] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proc. Internet Network Management Conference on Research on Enterprise Networking (INM/WREN'10)*, 2010, pp. 3–3.

[2] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for sdn? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, July 2013.

[3] D. S. Alexander, W. A. Arbaugh, M. W. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith, "The switchware active network architecture," *IEEE Network*, vol. 12, no. 3, pp. 29–36, May 1998.

[4] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Trans. Comput. Syst.*, vol. 18, no. 3, pp. 263–297, Aug. 2000.

[5] The BIRD internet routing daemon. [Online]. Available: http://bird.network.cz/

[6] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, "Design and implementation of a routing control platform," in

Proc. 2Nd Conference on Symposium on Networked Systems Design & Implementation (NSDI), 2005, pp. 15–28.

[7] Forward and Control Element Separation framework. [Online]. Available: https://tools.ietf.org/html/rfc3746

[8] Forward and Control Element Separation Protocol. [Online]. Available: https://tools.ietf.org/html/rfc5810

[9] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4d approach to network control and management," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, pp. 41–54, Oct. 2005.

[10] Path Comutation Element architecture. [Online]. Available: https://tools.ietf.org/html/rfc4655

[11] Path Comutation Element Communication Protocol. [Online]. Available: https://tools.ietf.org/html/rfc5440

[12] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 1–12, Aug. 2007.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: Enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008.

[14] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proc.*

9th USENIX Conference on Operating Systems Design and Implementation (OSDI), 2010, pp. 351–364.

[15] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: Towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 3, pp. 105–110, Jul. 2008.

[16] J. Stribling, Y. Sovran, I. Zhang, X. Pretzer, J. Li, M. F. Kaashoek, and R. Morris, "Flexible, wide-area storage for distributed systems with wheelfs," in *Proc. 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009, pp. 43–58.

[17] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. First workshop on Hot topics in software defined networks (HotSDN)*, 2012, pp. 7–12.

[18] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On controller performance in software-defined networks," in *Proc. 2Nd USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, April 2012, pp. 1–6.

[19] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Communication Letters*, vol. 18, pp. 1339–1342, Aug. 2014.

[20] H. Xie, T. T., L. D., Y. H., and G. V, "Use cases for alto with software defined networks," *IETF Internet-Draft*, 2012.

[21] P. Lin, J. Bi, and Y. Wang, *East-West Bridge for SDN Network Peering*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 170–181.

[22] Z. Xue, X. Dong, S. Ma, and W. Dong, "A survey on failure prediction of large-scale server clusters," in *Proc. International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD)*, July 2007, pp. 733–738.

[23] S. Huang, J. Griffioen, and K. L. Calvert, "Network hypervisors: Enhancing sdn infrastructure," *Computer Communications*, vol. 46, pp. 87 – 96, 2014.

[24] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, "Flowvisor: A network virtualization layer," Tech. Rep., 2009.

[25] E. Salvadori, R. D. Corin, A. Broglio, and M. Gerola, "Generalizing virtual network topologies in openflow-based networks," in *Proc. IEEE Global Telecommunications Conference (GLOBECOM)*, Dec. 2011, pp. 1–6.

[26] R. D. Corin, M. Gerola, R. Riggio, F. D. Pellegrini, and E. Salvadori, "Vertigo: Network virtualization and beyond," in *Proc. European Workshop on Software Defined Networking (EWSDN)*, Oct. 2012, pp. 24–29.

[27] S. Min, S. Kim, J. Lee, B. Kim, W. Hong, and J. Kong, "Implementation of an openflow network virtualization for multi-controller environment," in *Proc. International Conference on Advanced Communication Technology (ICACT)*, Feb. 2012, pp. 589–592.

[28] A. Blenk, A. Basta, M. Reisslein, and W. Kellerer, "Survey on network virtualization hypervisors for software defined networking," *IEEE Communications Surveys Tutorials*, vol. 18, no. 1, pp. 655–685, 2016.

[29] S. Khuller and Y. J. Sussmann, "The capacitated k-center problem," *SIAM J. Discret. Math.*, vol. 13, no. 3, pp. 403–418, May 2000.

[30] C. Prehofer and C. Bettstetter, "Self-organization in communication networks: principles and design paradigms," *IEEE Communications Magazine*, vol. 43, no. 7, pp. 78–85, July 2005.

[31] W. D. Grover, "Self-organizing broad-band transport networks," *Proceedings of the IEEE*, vol. 85, no. 10, pp. 1582–1611, Oct 1997.

[32] A. Blenk, A. Basta, J. Zerwas, and W. Kellerer, "Pairing sdn with network virtualization: The network hypervisor placement problem," in *Proc. IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, Nov. 2015, pp. 198–204.

[33] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The internet topology zoo," *IEEE Journal on Selected Areas of Communication*, vol. 29, pp. 1765–1775, Sep. 2011.

[34] Open Networking Foundation White Paper. [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[35] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys Tutorials*, vol. 16, no. 3, pp. 1617–1634, Mar. 2014.

[36] K.-K. Yap, M. Kobayashi, R. Sherwood, T.-Y. Huang, M. Chan, N. Handigol, and N. McKeown, "Openroads: Empowering research in mobile networks," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 1, pp. 125–126, Jan. 2010.

[37] K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown, "Delivering capacity for the mobile internet by stitching together networks," in *Proc. ACM Workshop*

on Wireless of the Students, by the Students, for the Students, Sep. 2010, pp. 41–44.

[38] A. K. Nayak, A. Reimers, N. Feamster, and R. Clark, "Resonance: Dynamic access control for enterprise networks," in *Proc. 1st ACM Workshop on Research on Enterprise Networking*, Aug. 2009, pp. 11–18.

[39] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown, "Elastictree: Saving energy in data center networks," in *Proc. 7th USENIX Conference on Networked Systems Design and Implementation (NSDI)*, Aug. 2010, pp. 17–17.

[40] P. S. Pisa, N. C. Fernandes, H. E. T. Carvalho, M. D. D. Moreira, M. E. M. Campista, L. H. M. K. Costa, and O. C. M. B. Duarte, "Openflow and xen-based virtual network migration," in *Communications: Wireless in Developing Countries and Networks of the Future*. Berlin, Heidelberg: Springer-verlag, 2010, pp. 170–181.

[41] R. Wang, D. Butnariu, and J. Rexford, "Openflow-based server load balancing gone wild," in *Proc. 11th USENIX Conference on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE)*, March 2011, pp. 12–12.

[42] M. Koerner and O. Kao, "Multiple service load-balancing with openflow," in *Proc. IEEE 13th International Conference on High Performance Switching and Routing (HPSR)*, June 2012, pp. 210–214.

[43] Y. Nakagawa, K. Hyoudou, and T. Shimizu, "A management method of ip multicast in overlay networks using openflow," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, 2012, pp. 91–96.

[44] D. Kotani, K. Suzuki, and H. Shimonishi, "A design and implementation of openflow controller handling ip multicast with fast tree switching," in *Proc. IEEE/IPSJ 12th International Symposium on Applications and the Internet (SAINT)*, July 2012, pp. 60–67.

[45] G. Lu, R. Miao, Y. Xiong, and C. Guo, "Using cpu as a traffic co-processing unit in commodity switches," in *Proc. First Workshop on Hot Topics in Software Defined Networks (HotSDN)*, August 2012, pp. 31–36.

[46] A. Ramachandran, Y. Mundada, M. B. Tariq, and N. Feamster, "Securing enterprise networks using traffic tainting," Georgia Inst. Technol., Atlanta, GA, USA, Tech. Rep. GTCS-09-15, Oct 2009.

[47] Quagga Routing Suite. [Online]. Available: http://www.nongnu.org/quagga/

[48] NOX. [Online]. Available: http://www.noxrepo.org/

[49] POX. [Online]. Available: http://www.noxrepo.org/pox/about-pox/.

[50] Ryu. [Online]. Available: http://osrg.github.com/ryu/

[51] Beacon. [Online]. Available: http://openflow.stanford.edu/display/Beacon/Home

[52] Maestro. [Online]. Available: http://code.google.com/p/maestro-platform

[53] Floodlight. [Online]. Available: http://www.projectfloodlight.org/

[54] Jaxon:java-based openflow controller. [Online]. Available: http://jaxon.onuos.org/.

[55] Trema openflow controller framework. [Online]. Available: https://github.com/trema/trema

[56] Opendaylight. [Online]. Available: http://www.opendaylight.org/.

[57] Helios. [Online]. Available: http://www.nec.com/.

[58] ovs-controller. [Online]. Available: http://openvswitch.org/.

[59] D. Hock, S. Gebert, M. Hartmann, T. Zinner, and P. Tran-Gia, "Poco-framework for pareto-optimal resilient controller placement in sdn-based core networks," in *Proc. IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–2.

[60] 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 *Proc. 25th International Teletraffic Congress (ITC)*, Sept. 2013, pp. 1–9.

[61] 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 Trans. Netw. Service Manag.*, vol. 12, pp. 4–17, March 2015.

[62] D. Hock, M. Hartmann, S. Gebert, T. Zinner, and P. Tran-Gia, "Poco-plc: Enabling dynamic pareto-optimal resilient controller placement in sdn networks," in *Proc. IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2014, pp. 115–116.

[63] M. T. I. ul Huque, G. Jourjon, and V. Gramoli, "Revisiting the controller placement problem," in *Proc. IEEE 40th Conference on Local Computer Networks (LCN)*, Oct. 2015, pp. 450–453.

[64] A. Ksentini, M. Bagaa, T. Taleb, and I. Balasingham, "On using bargaining game for optimal placement of sdn controllers," in *Proc. IEEE International Conference on Communications (ICC)*, May 2016, pp. 1–6.

[65] T. Zhang, A. Bianco, and P. Giaccone, "The role of inter-controller traffic in sdn controllers placement," in *Proc. IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov. 2016, pp. 87–92.

[66] T. Zhang, P. Giaccone, A. Bianco, and S. D. Domenico, "The role of the inter-controller consensus in the placement of distributed sdn controllers," *Computer Communications*, vol. 113, no. Supplement C, pp. 1 – 13, 2017.

[67] P. Vizarreta, C. M. Machuca, and W. Kellerer, "Controller placement strategies for a resilient sdn control plane," in *Proc. 8th International Workshop on Resilient Networks Design and Modeling (RNDM)*, Sept 2016, pp. 253–259.

[68] S. Guo, S. Yang, Q. Li, and Y. Jiang, "Towards controller placement for robust software-defined networks," in *Proc. IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, Dec. 2015, pp. 1–8.

[69] V. Sridharan, M. Gurusamy, and T. Truong-Huu, "On multiple controller mapping in software defined networks with resilience constraints," *IEEE Communications Letters*, vol. PP, no. 99, pp. 1–1, 2017.

[70] ——, "Multi-controller traffic engineering in software defined networks," in *Proc. IEEE 42nd Conference on Local Computer Networks (LCN)*, Oct. 2017, pp. 137–145.

[71] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 7–12, Aug. 2013.

[72] A. A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Elasticon: An elastic distributed sdn controller," in *Proc. Tenth ACM/IEEE Symposium*

on Architectures for Networking and Communications Systems (ANCS), 2014, pp. 17–28.

[73] Y. Zhang, N. Beheshti, and M. Tatipamula, "On resilience of split-architecture networks," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, Dec. 2011, pp. 1–6.

[74] M. Guo and P. Bhattacharya, "Controller placement for improving resilience of software-defined networks," in *Proc. IEEE International Conference on Networking and Distributed Computing (ICNDC)*, Dec. 2013, pp. 23–27.

[75] L. F. Muller, R. R. Oliveira, M. C. Luizelli, L. P. Gaspary, and M. P. Barcellos, "Survivor: an enhanced controller placement strategy for improving sdn survivability," in *Proc. IEEE Global Communications Conference(GLOBECOM)*, Dec. 2014, pp. 1909–1915.

[76] Y. Hu, W. Wendong, X. Gong, X. Que, and C. Shiduan, "Reliability-aware controller placement for software-defined networks," in *Proc. IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2013, pp. 672–675.

[77] Y. nan HU, W. dong WANG, X. yang GONG, X. rong QUE, and S. duan CHENG, "On the placement of controllers in software-defined networks," *The Journal of China Universities of Posts and Telecommunications*, vol. 19, no. 2, pp. 92–171, oct 2012.

[78] Y. Hu, W. Wang, X. Gong, X. Que, and S. Cheng, "On reliability-optimized controller placement for software-defined networks," *China Communications*, vol. 11, pp. 38–54, Feb. 2014.

[79] A. Sallahi and M. St-Hilaire, "Optimal model for the controller placement problem in software defined networks," *IEEE Communication Letters*, vol. 19, pp. 30–33, Jan. 2015.

[80] ——, "Expansion model for the controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 21, no. 2, pp. 274–277, Feb. 2017.

[81] F. J. Ros and P. M. Ruiz, "Five nines of southbound reliability in software-defined networks," in *Proc. Third workshop on Hot topics in software defined networking (HotSDN)*, 2014, pp. 31–36.

[82] 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 *Proc. of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, June 2014, pp. 1–6.

[83] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Proc. 9th International Conference on Network and Service Management (CNSM)*, Oct. 2013, pp. 18–25.

[84] N. Perrot and T. Reynaud, "Optimal placement of controllers in a resilient sdn architecture," in *Proc. 12th International Conference on the Design of Reliable Communication Networks (DRCN)*, March 2016, pp. 145–151.

[85] M. Tanha, D. Sajjadi, and J. Pan, "Enduring node failures through resilient controller placement for software defined networks," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, Dec. 2016, pp. 1–7.

[86] L. V. Snyder and M. S. Daskin, "Reliability models for facility location: The expected failure cost case," *Transportation Science*, vol. 39, no. 3, pp. 400–416, 2005.

[87] 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*, pp. 1–1, 2018 [Accepted].

[88] A. Ruiz-Rivera, K. W. Chin, and S. Soh, "Greco: An energy aware controller association algorithm for software defined networks," *IEEE Communications Letters*, vol. 19, no. 4, pp. 541–544, April 2015.

[89] K. Sood and Y. Xiang, "The controller placement problem or the controller selection problem?" *Journal of Communications and Information Networks*, vol. 2, no. 3, pp. 1–9, Sep. 2017.

[90] T. Y. Cheng, M. Wang, and X. Jia, "Qos-guaranteed controller placement in sdn," in *Proc. IEEE Global Communications Conference (GLOBECOM)*, Dec. 2015, pp. 1–6.

[91] T. Wang, F. Liu, J. Guo, and H. Xu, "Dynamic sdn controller assignment in data center networks: Stable matching with transfers," in *Proc. IEEE International Conference on Computer Communications (INFOCOM)*, April 2016, pp. 1–9.

[92] R. Cohen, L. Katzir, and D. Raz, "An efficient approximation for the generalized assignment problem," *Information Processing Letters*, vol. 100, no. 4, pp. 162 – 166, 2006.

[93] X. Li, P. Djukic, and H. Zhang, "Zoning for hierarchical network optimization in software defined networks," in *Proc. IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–8.

[94] Z. Su and M. Hamdi, "Mdcp: Measurement-aware distributed controller placement for software defined networks," in *Proc. IEEE 21st International Conference on Parallel and Distributed Systems (ICPADS)*, Dec. 2015, pp. 380–387.

[95] P. Xiao, W. Qu, H. Qi, Z. Li, and Y. Xu, "The sdn controller placement problem for wan," in *Proc. IEEE/CIC International Conference on Communications in China (ICCC)*, Oct. 2014, pp. 220–224.

[96] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li, and T. Li, "Density cluster based approach for controller placement problem in large-scale software defined networkings," *Computer Networks*, vol. 112, pp. 24 – 35, 2017.

[97] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer, "Control plane latency with sdn network hypervisors: The cost of virtualization," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 366–380, 2016.

[98] Internet2 open science, scholarship and services exchange. [Online]. Available: http://www.internet2.edu/network/ose/

[99] I. Espejo, A. Marín, and A. M. Rodríguez-Chía, "Capacitated p-center problem with failure foresight," *European Journal of Operational Research*, vol. 247, pp. 229–244, Nov. 2015.

[100] ——, "Closest assignment constraints in discrete location problems," *European Journal of Operational Research*, vol. 219, no. 1, pp. 49–58, 2012.

[101] T. L. Lei and R. L. Church, "Constructs for multilevel closest assignment in location modeling," *International Regional Science Review*, vol. 34, no. 3, pp. 339–367, 2011.

[102] N. Katta, H. Zhang, M. Freedman, and J. Rexford, "Ravana: Controller fault-tolerance in software-defined networking," in *Proc. 1st ACM SIGCOMM Symposium on Software Defined Networking Research (SOSR)*, 2015, pp. 4:1–4:12.

[103] W. Chen, S. Toueg, and M. K. Aguilera, "On the quality of service of failure detectors," *IEEE Trans. Comput.*, vol. 51, no. 5, pp. 561–580, May 2002.

[104] R. Y. Xavier Defago, Naohiro Hayashibara and T. Katayama, "The φ accrual failure detector," *IEEE Symposium on Reliable Distributed Systems*, pp. 66–78, Oct 2004.

[105] B. Satzger, A. Pietzowski, W. Trumler, and T. Ungerer, "A new adaptive accrual failure detector for dependable distributed systems," in *Proc. ACM Symposium on Applied Computing (SAC)*, 2007, pp. 551–555.

[106] T.-W. Yang and K. Wang, "Failure detection service with low mistake rates for sdn controllers," in *Proc. Asia-Pacific Network Operations and Management Symposium (APNOMS)*, 2016, pp. 1–6.

[107] A. S. W. Tam, K. Xi, and H. J. Chao, "Use of devolved controllers in data center networks," in *Proc. IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2011, pp. 596–601.

[108] K. Kuroki, N. Matsumoto, and M. Hayashi, "Scalable openflow controller redundancy tackling local and global recoveries," in *Proc. The Fifth International Conference on Advances in Future Internet (AIFN)*, Dec. 2013, pp. 61–66.

[109] M. Obadia, M. Bouet, J. Leguay, K. Phemius, and L. Iannone, "Failover mechanisms for distributed sdn controllers," in *Proc. International Conference and Workshop on the Network of the Future (NOF)*, Dec 2014, pp. 1–6.

[110] Y.-C. Chan, K. Wang, and Y.-H. Hsu, "Fast controller failover for multi-domain software-defined networks," in *Proc. European Conference on Networks and Communications (EuCNC)*, 2015, pp. 370–374.

[111] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. ACM Internet measurement conference (IMC)*, Nov. 2009, pp. 202–208.

[112] *MATLAB version 8.5.0.197613 (R2015a)*, The Mathworks, Inc., Natick, Massachusetts, 2015.

[113] IBM ILOG CPLEX. [Online]. Available: http://www-01.ibm. com/ software/integration/optimization/cplex-optimizer

[114] M. Albareda-Sambola, Y. Hinojosa, A. Marn, and J. Puerto, "When centers can fail: A close second opportunity," *Computers & Operations Research*, vol. 62, pp. 145–156, Oct. 2015.

[115] O. Berman, D. Krass, and M. B. C. Menezes, "Locating facilities in the presence of disruptions and incomplete information*," *Decision Sciences*, vol. 40, no. 4, pp. 845–868, 2009.

[116] J. L. Wagner and L. M. Falkson, "The optimal nodal location of public facilities with price-sensitive demand," *Geographical Analysis*, vol. 7, no. 1, pp. 69–83, 1975.

[117] R. Church and J. Cohon, "Multiobjective location analysis of regional energy facility siting problems," Brookhaven National Lab., Upton, New York, United States, Tech. Rep. BNL 50567, Oct 1976.

[118] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined net-

works," in *Proc. International Conference on Network and Service Management (CNSM)*, Oct. 2013, pp. 18–25.

[119] D. Dietrich, A. Abujoda, and P. Papadimitriou, "Network service embedding across multiple providers with nestor," in *Proc. IFIP Networking Conference (Networking)*, May 2015, pp. 1–9.

[120] A. Abujoda and P. Papadimitriou, "Distnse: Distributed network service embedding across multiple providers," in *Proc. International Conference on Communication Systems and Networks (COMSNETS)*, Jan 2016, pp. 1–8.

[121] D. Dietrich, A. Abujoda, A. Rizk, and P. Papadimitriou, "Multi-provider service chain embedding with nestor," *IEEE Trans. on Netw. and Service Mang.*, vol. 14, no. 1, pp. 91–105, March 2017.

[122] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[123] V. Černý, "Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, no. 1, pp. 41–51, 1985.

[124] V. Vazirani, *Approximation Algorithms*. Springer-Verlag New York, Inc., 2001.

[125] V. K. Garg, Y. Narahari, and M. N. Murty, "Novel biobjective clustering (bigc) based on cooperative game theory," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 5, pp. 1070–1082, May 2013.

[126] Tools for analyzing the Controller Placement Problem in Software-Defined Networks. [Online]. Available: https://www.github.com/brandonheller/cpp/blob/master/src/geo/

Publications Related to Thesis

Journals

1. **Bala Prakasa Rao Killi** and S. V. Rao, "Failure Foresight Controller Placement in Software Defined Networks", IEEE Communications Letters, vol. 20, no. 6, pp. 11081111, 2016. [Chapter 3]

2. **Bala Prakasa Rao 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, 2017. [Chapter 4]

3. **Bala Prakasa Rao Killi** and S. V. Rao,"On Placement of Hypervisors and Controllers in Virtualized Software Defined Network", IEEE Transactions on Network and Service Management, vol. 15, no. 2, pp. 840-853, 2018. [Chapter 6]

4. **Bala Prakasa Rao Killi** and S. V. Rao,"Controller Placement in Software Defined Networks: A Comprehensive Survey", IEEE Communications Surveys & Tutorials. [**Under Review**]

Book Chapter

1. **Bala Prakasa Rao Killi**, Akhil Reddy Ellore, and S. V. Rao,"Game Theoretic approaches for Controller Placement in SDN", Communication Systems and Networks, Lecture Notes in Computer Science, Springer. [**Accepted**]

Conference Proceedings

1. **Bala Prakasa Rao Killi**, Akhil Reddy Ellore, and S. V. Rao,"Cooperative Game Theory based Network Partitioning for Controller Placement in SDN", Proc. of 10th International Conference on COMmunication Systems & NETworkS (COMSNETS), Jan. 2018, pp. 105-112. [Chapter 6] **[Best Paper Award]**

2. **Bala Prakasa Rao Killi** and S. V. Rao,"Controller Placement with Planning for Failures in Software Defined Networks", Proc. of IEEE International Conference on Advanced Networks and Telecommunication Systems (ANTS), Nov. 2016, pp. 1-6.

Publications Outside Thesis

1. **Bala Prakasa Rao Killi** and S. V. Rao,"Link Failure Aware Capacitated Controller Placement in Software Defined Networks", Proc. of International Conference on Information Networking (ICOIN), Jan. 2018, pp. 292-297.

2. **Bala Prakasa Rao Killi** and S. V. Rao,"Towards Improving Resilience of Controller Placement with Minimum Backup Capacity in Software Defined Networks", Elsevier Computer Networks. [**Under Revision**]