SOFTWARE-DEFINED NETWORKING BASED CONTROL FLOW OPTIMIZATION FOR MULTI-CLOUD ENVIRONMENT

A Thesis submitted in fulfillment of the requirement for the award of the degree of

DOCTOR OF PHILOSOPHY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Rajat Chaudhary

(Registration No. : 901603012)

Under the guidance of:

Dr. Neeraj Kumar
Professor, CSED

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR INSTITUTE OF ENGINEERING & TECHNOLOGY,
PATIALA – 147004

February 2021

CERTIFICATE

I, Rajat Chaudhary, Regn. No. 901603012, hereby declare that the thesis entitled **"Software-Defined Networking Based Control Flow Optimization For Multi-Cloud Environment"** submitted to the Computer Science and Engineering Department at Thapar Institute of Engineering & Technology, Patiala, Punjab, India is an authenticated record of my own work for the award of the degree of "Doctor of Philosophy" under the supervision of Prof. (Dr.) Neeraj Kumar. This report has not been submitted to any other institution for award of any other degree.

Rajat Chaudhary
Regn. No. 901603012

Place: Patiala, Punjab (India)
Date: 23 February, 2021

This is to certify that the above statement made by the candidate is correct to the best of our knowledge.

Verified by:

Prof. (Dr.) Neeraj Kumar
(Supervisor)
Professor, CSED
Thapar Institute of Engg. & Technology
Patiala, Punjab (India)

ABSTRACT

In recent years, the huge expansion of Datacenters (DC) to execute billions of end-user applications in real-time leads to a large amount of energy consumption across the globe. So, the traditional TCP/IP-based networks which are being used for DC inter-connections are facing challenges of managing stringent Quality-of-Service (QoS) requirements of different applications of the end-users and service providers. Moreover, the existing solutions rely on distributed architecture and do not scale for large scale data centers. The issue of high power consumption at DC arises with the increase in the number of nodes and links in the network. Also, it becomes problematic on the DC whenever the underlying network resources (switches, routers) are not efficiently utilized at the time of peak data traffic resulting in high operational cost of energy utilization. However, a single controller due to its limited capacity and resources may not handle heavy load traffic generated from various smart devices. In order to handle this, multiple controllers need to be deployed at the control plane so as to ensure improved efficiency and scalability of the network. The data flow by the distributed controllers fluctuates frequently which results in an uneven load distribution amongst different controllers.

Software-Defined Datacenters (SDDC) have been widely used for load-aware data management for different applications across the globe. Due to its centralized architecture, the issues of scalability along with resilience (to overcome the failure of single or multiple controllers) are still challenging because of an exponential increase in the data generated from different smart devices. Most of the solutions reported in the literature for this problem use a single controller which may not address the scalability issues. However, the issues of scalability and resilience in SDDC can be solved by deploying multiple distributed controllers at the control plane. However, the primary concern in a network having various controllers is the optimal Controller Placement Problem (CPP) to resolve the issues of fault-tolerance, latency among controllers, availability, and placement.

Software-Defined Networking (SDN) emerges as one of the leading technologies to address the aforementioned issues using the programmable switches and controllers. The decoupling of control functionality from the forwarding devices to the control plane in SDN provides a unique platform to design a reconfigurable network. In the aforementioned challenges, the research work focused on three problems: (i) load balancing at the control plane, (ii) energy efficiency and fast flow forwarding for the data plane, and (iii) scalability and resilience at the control plane. This task has been accomplished in this research work with three different

approaches.

The first approach presents a Load Optimization and Anomaly Detection Scheme (LOADS) is proposed. Using LOADS, the probability of switch selection is determined using the following two factors (i) distance from the switch to the controller, and (ii) resource consumption ratio of the switch to its controller. Also, an IP flow-based network anomaly detection module has been designed to classify the traffic as malicious or normal. In order to address the network anomaly, the *LOADS* scheme uses Access Control Policies (ACPs) on the user's behavior in the network. The proposed scheme is evaluated on the Mininet emulator using the POX controller with datasets of Internet Topology Zoo from the BTNorthAmerica zone.

The second approach formulated the Energy-Aware Routing (EAR) problem of DCs as a Mixed Integer Non-Linear Programming (MINLP) for which an Energy-Efficient Fast Flow Forwarding (EnFlow) scheme is designed. The *EnFlow* scheme uses the power-saving mode of the network to solve the EAR problem. It has three modules namely– *priority scheduling*, *routing*, and *re-routing*. The first module works according to the First-in-First-Out Push Out Priority (FIFO-POP) scheduling using the multiple OpenFlow switches. The FIFO-POP is designed to save the energy usage of multiple switches by reducing the average waiting time of incoming packets in the queue buffers. The second module is based upon an efficient flow re-routing for a new node and link adaptation to provide the maximum bandwidth to the wired links. The third module is based upon the meta-heuristic Ant Colony Routing (ACR) to execute the stochastic decision policy on the network controller for computation of the shortest path of the forwarding nodes. The proposed EnFlow scheme is simulated using the data traces of 34 cities of NorthAmerica zone with Omnet++ 5.1 using various performance evaluation metrics.

The third approach proposes Placement Availability Resilient Controller (PARC) scheme. The PARC scheme works in the following four phases: (i) stable network partitioning (ii) localization of controllers using the cooperative game theory (iii) computation of an optimal number of multiple controllers and (iv) computation of minimal extra backup controllers to improve the overall network cost. The numerical results of the PARC scheme are evaluated on Internet2 OS3E topology using POCO-toolset simulated in Matlab. Moreover, the PARC scheme outperforms the existing state-of-the-art schemes (POCO-SA, POCO-MOALO, and CNCP) for inter-controller as well as switch-to-controller latency.

ACKNOWLEDGMENT

Before discussing my journey of Ph.D., I would like to thank the almighty God who gave me strength and courage to overcome all the obstacles and complete this endeavour. The aim of my life, to be called by the salutation of a 'Doctor', seems to become a reality, when I got admission in Doctorate of philosophy in Thapar Institute of Engineering & Technology. Research initiated with this startup in my life. Without acknowledging the people who supported me throughout this journey, this task would be incomplete. I know words are never enough to express the gratitude; I am just delivering the phrase for the acceptance of regards.

Firstly, I would like to express my sincerest thanks to my parents. With their consent, support, and motivation, I thought to accept this biggest challenge in my life. They have been the true source of real inspiration for me. Secondly, I would like to thank my supervisor, Prof. (Dr.) Neeraj Kumar, who has supported me throughout my Ph.D. work with his patience and knowledge; while providing me with the room to work in my own way. Apart from providing me with excellent supervision, active cooperation, and constant encouragement throughout this journey, he also shared their invaluable experiences with me to succeed in life. I will always remain indebted to him.

I am very thankful to the examiners Prof. (Dr.) Sanjay Kumar Singh from Indian Institute of Technology (IIT), Banaras Hindu University (BHU), Varanasi, India and Prof. Geyong Min from University of Exeter, England, United Kingdom for taking out their time from their busy schedules and evaluating my thesis.

I am also grateful to the head of the department, Prof. (Dr.) Maninder Singh, Associate Head and Dean of Student Affair Prof. (Dr.) Inderveer Chana, Ph.D. Coordinator, Dr. Sushma Jain, and members of my doctoral committee, Prof. (Dr.) Anil Kumar Verma, and Prof. (Dr.) Kulbir Singh for their constructive suggestions and ensuring the correct pace of my work. I am also obliged to the Director, Prof. (Dr.) Prakash Gopalan, Dean (RSP), Prof. (Dr.) Rafat Siddique, and the management of Thapar Institute of Engineering and Technology, who provided me with all the necessary resources and facilities to complete my work.

The chain of my gratitude will definitely be incomplete if I forget to thank my complete family, my father Shri. Rajveer Singh, my mother Mrs. Rajesh, my elder sister Ranjana Chaudhary, my younger brother Rohan Chaudhary and my wife Ritu Singh, for their unconditional love, support, and encouragement in every phase of my life. It was due to my father's, sister, and my wife confidence and vision that motivated me to overcome every obstacle during the research. Since then, the journey of a Ph.D. has been a sweet and bitter ride at times which

leads to a special mention for my mother who stood by me through thick and thin and gave me courage at the times when I felt really low. Her constant motivation showed me the silver lining in the dark clouds.

I would also like to express heartfelt thanks to Dr. Sudeep Tanwar and Dr. Sudhanshu Tyagi, who always believed in me and whose blessings have truly played the role of game-changer in my life. I would also like to pay my sincere regards to all my relatives and cousins for their constant motivation and support. They made this journey more comfortable with words of encouragement which helped me in finishing my work.

I would also like to thank my friends and colleagues with whom I have traveled this journey of research. A special thanks to my research group, Dr. Gagangeet Singh Aujla, Dr. Anish Jindal, Dr. Aaisha Makkar, Dr. Sahil Garg, Dr. Kuljeet Kaur and budding doctors, Mr. Ishan Budhiraja, Mr. Vivek Dabra, Ms. Shubhani Aggarwal, Ms. Arzoo Miglani, and Mr. Himanshu Sharma. These people have made my research journey all the more memorable and pleasant. As one cannot mention the names of all well-wishers, friends, and beloved ones, I would like to pay my regards to one and all who supported me during this journey of knowledge.

(**Rajat Chaudhary**)

List of Publications

Journal Publications (SCIE):

1. Rajat Chaudhary, Neeraj Kumar, “LOADS: Load Optimization and Anomaly Detection Scheme for Software Defined Networks", *IEEE Transactions on Vehicular Technology*, vol. 68, no. 12, pp. 12329-12344, Dec. 2019. (IF 5.379-Q1)

2. Rajat Chaudhary, Neeraj Kumar, “EnFlow: An Energy-Efficient Fast Flow Forwarding Scheme for Software Defined Networks", *IEEE Transactions on Intelligent Transportation Systems*, pp. 1-17, Jun. 2020. (Early Access, DOI: 10.1109/TITS.2020.2999134) (IF 6.319-Q1)

3. R. Chaudhary, N. Kumar, “PARC: Placement Availability Resilient Controller Scheme for Software-Defined Datacenters", *IEEE Transactions on Vehicular Technology*, vol. 69, no. 8, pp. 8985-9001, Aug. 2020. (IEEE, IF 5.379-Q1)

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviations	Definitions
ACK	Acknowledgement
ACO	Ant Colony Optimization
AC-OFER	Ant Colony based on Online Flow-Based Energy-Efficient Routing
ACP	Access Control Policies
ACR	Ant Colony Routing
ADS	Anomaly Detection System
AGAR	Analytic Game Aware Routing
ALFE	Adaptive Least Frequently Evicted
ALTO	Application Layer Traffic Optimization
AODV	Adhoc On-Demand distance Vector
ARCM	Adaptive Resource Capacity Management
ASP	Adaptive Suspicious Prevention
BDAEAH	Big Data Application in Emotion-Aware Healthcare
BGP	Border Gateway Protocol
BSN	Body Sensor Networks
CaaS	Control-as-a-Service
CAFTS	Controller Assignment in Fault-Tolerant SDN
CAGR	Compound Annual Growth Rate
CAIDA	Center for Applied Internet Data Analysis
CAMA	Context-Aware Mobile Approach
CAPEX	Capital Expenditure
CARPO	Correlation-Aware Power Optimization
CDF	Cumulative Distribution Function
CDS	Connected Dominating Sets
CE	Control Elements
CORNET	Controller-based Robust Network
CNCP	Capacitated Next Controller Placement
CPP	Controller Placement Problem
CPS	Cyber Physical System
CPU	Central Processing Unit
D2D	Device-to-Device
DARPA	Defense Advanced Research Projects Agency
DC	Datacenters
DCAN	Devolved Control of ATM Networks

DCN	Data Center Networks
DDoS	Distributed Denial-of-Service
DHA	Distributed Hopping Algorithm
DISCO	Distributed Flow Consolidation Scheme
DPS	Deterministic Path Selection
DSR	Dynamic Source Routing
E2E	End-to-End
EaaS	Everything-as-a-service
EAR	Energy-Aware Routing
ECMP	Equal Cost Multi-Path routing
EE-DADR	Energy Efficient and Delay-Aware Distributed Routing
EV	Electric Vehicles
EVPN	Ethernet Virtual Private Network
ESR	Energy Efficient Sensor Selection and Routing
EXR	Exclusive Routing
FC	Fog Computing
FE	Forwarding Elements
FFD	First Fit Decreasing
FFHA	First-Fit Heuristic Algorithm
FIFO	First-In-First Out
FIFO-PO	First-in-First-Out Push Out
FIFO-POP	First-in-First-Out Push Out Priority
FML	Field Manipulation Language
FN	False Negative
ForCES	Forwarding and Control Element Separation
FP	False Positive
FPR-RE	Flow Preemption Routing with Redundancy Elimination
FPTA	Fully Polynomial Time Approximation
FSR	Fair Sharing Routing
GB	Giga Bytes
GBEP	Gigabit Ethernet port
GPRS	General Packet Radio Service
GPSR	Greedy Perimeter Stateless Routing
GRE	Generic Routing Encapsulation
GSMP	General Switch Management Protocol
GT	Game Theory
GT-HWDS	Game Theory and Holt-Winters for Digital Signature
GTP	GPRS Tunneling Protocol
HADMM	Hybrid Alternating Direction Method of Multipliers
HAN	Home Area Network
HCN	Healthcare network
HD	High Definition
HOTMLP	Hash Offset Tree Match on Longest Prefix
HVAC	Heating, Ventilation, and Air Conditioning
ICMP	Internet Control Message Protocol
ID	Identifier
IDS	Intrusion Detection System
IETF	Internet Engineering Task Force

ILP	Integer Linear Programming
ILP-EAR	Integer Linear Program-based Energy-Aware Routing
IoT	Internet-of-Things
IPS	Intrusion Prevention System
LARAC	Lagrange Relaxation based Aggregated Cost
LB	Load Balancing
LFB	Logic Function Block
LOADS	Load Optimization and Anomaly Detection Scheme
LRU	Least Recently Used
MBF	Multiple Bloom Filter
MEC	Mobile Edge Computing
MHSN	Mobile Healthcare Social Networks
MINA	Multi-Network Information Architecture
MIP	Mixed Integer Programming
MINLP	Mixed Integer Non-Linear Programming
MOALO	Multi-Objective Ant Lion Optimization
MPLS	Multi-Protocol Label Switching
MPR	Multi-Path Routing
NBS	Nash Bargaining Solution
NCP	Network Control Point
NETCONF	Network Configuration
NFV	Network Functions Virtualization
NIC	Network Interface Card
NIST	National Institute of Standards and Technology
NOS	Network Operating System
NSDP	Network Security Defense Patterns
NV	Network Virtualization
NVGRE	Network Virtualization using Generic Routing Encapsulation
OF	OpenFlow
OLT	Optical Line Terminal
ONU	Optical Network Units
OPEX	Operational Expenses
OS3E	Open Science, Scholarship and Services Exchange
OSPF	Open Shortest Path First
OVSDB	Open Virtual Switch Database
P2P	Peer-to-Peer
PARC	Placement Availability Resilient Controller
P-BC	Primary and Backup switch-Controller
PCAR	Priority-based Congestion-Avoidance routing
PCE	Path Computation Element
PDR	Packet Delivery Ratio
PFG	Partition Form Games
PHI	Personal Health Information
POCO	Pareto Optimal COntroller
POCO-MOALO	Pareto Optimal COntroller Multi-Objective Ant Lion Optimization
POCO-SA	Pareto Optimal COntroller based on the Simulated Annealing
POF	Protocol-Oblivious Forwarding

PM	Physical Machine
POF	Protocol-Oblivious Forwarding
PS	Priority Shifting
QNSD	Queue-length based Service Distribution
QoS	Quality-of-Service
QVIA-SDN	QoS-aware Virtual Infrastructure Allocation on SDN
QWS	Quality of Web Service
RCP	Routing Control Platform
RENEMA	Residential Network Management Application
REST	Representational State Transfer
RE-FPR	Redundancy Elimination Flow Preemption Routing
RL	Reinforcement Learning
RQFA	Requirement First Assignment
RTT	Round-Trip Time
SA	Simulated Annealing
SAR	Specific Absorption Rate
SDDC	Software-Defined Datacenters
SD-EC	Software-Defined Edge Computing
SD-FC	Software-Defined Fog Computing
SD-IoT	Software-Defined Internet of Things
SD-IoV	Software-Defined Internet of Vehicles
SDN	Software-Defined Networking
SD-HCN	Software-Defined Healthcare Network
SDVRAN	SDN and Virtualized Radio Access Networks
SD-WAN	Software-Defined Wide Area Network
SDWSN	Software-Defined Wireless Sensor Networks
SENAD	Secure Network Application Deployment
SFC	Service Functions Chaining
SICCQ	SDN-based Incast Congestion Control via Queue-based monitoring
SITL	System-in-the Loop
SM	Symptoms Matching
SMS	Systematic Mapping Study
SNDlib	Survivable Network Design library
SOM	Self Organizing Map
SPON	Software defined Passive Optical Network
SR	Segment Routing
SSR-DCCR	Search Space Reduction Delay-Cost-Constrained Routing
STT	Stateless Transport Tunneling
SVM	Support Vector Machine
SYN	Synchronization
TAEO	Thermal Energy-Aware Routing Algorithm
TCAM	Ternary Content Addressable Memory
TCP/IP	Transmission Control Protocol/Internet Protocol
TE	Traffic Engineering
TN	True Negative
ToR	Top-of-Rack

TP	True Positive
TWh	TeraWatt-hour
VMM	Virtual Machine Monitor
VoIP	Voice over IP
VONs	Virtual Optical Networks
VPN	Virtual Private Networks
VRF	Virtual Routing and Forwarding
VxLAN	Virtual Extensible Local Area Network
VZ	Virtual Zones
W	Watt
WARM	Workload-Aware Revenue Maximization
WBAN	Wireless Body Area Networks
WDM	Wavelength Division Multiplexing
WOA	Whale Optimisation Algorithm
WSN	Wireless Sensor Network
ZB	Zetta Bytes

Chapter 1

Introduction

In recent years, there has been an exponential increase in the usage of Internet-enabled smart devices in different applications like vehicular networks, datacenter, and Internet-of-Things (IoT). The communication among the smart devices is used for the computation and communication across the globe which may result in heavy Internet data traffic generation. For example, the vehicular network which is an emerging paradigm of a smart transportation system generates a large amount of data traffic [1]. To handle this large amount of data traffic, a distributed control mechanism called as SDN is widely used to ensure the proper delivery of services, infrastructure, and applications to the end-users' smart devices [2]. The large traffic generated in such an environment managed by the datacenter is posing various issues (latency, load imbalance, network congestion, and the malicious traffic flow) to the Internet-enabled smart devices in large-scale networks [55].

1.1 Traditional TCP/IP Networks

The conventional Transmission Control Protocol/Internet Protocol (TCP/IP) architecture may have a challenge with respect to the scalability preservation and malicious flow detection [4]. It is because of the strong coupling of the software control part inside the data forwarding nodes which limits the scalability of the network in the conventional architecture. Moreover, the design of the conventional architecture is based on a distributed approach which makes it complex to monitor the user's behavior and to detect malware threats. An attacker can gain full access control of the network infrastructure by either impersonating as a server or compromising the network nodes and links to gain access to the devices [5]. In such scenarios, it is a challenging task to modify network configuration if any node gets compromised by an adversary. However, Software-Defined Networks (SDN) is an efficient approach to address the aforementioned challenges of traditional networks.

SDN is an emerging technology that has been used in a wide range of applications over the past many years. It is a 'softwarization' solution that provides network programmability by

removing the network intelligence from the forwarding devices to perform smooth execution of data traffic and anomaly detection [6]. In comparison to the existing infrastructure, SDN architecture provides better services to manage enterprise, Wide Area Networks (WAN), and data centers. According to the CISCO report [7], the volume of the Cloud Datacenters (DC) traffic will expand by 27% Compound Annual Growth Rate (CAGR) by the end of 2021. Today, SDN supports 23% of the Cloud DC traffic which might grow at a rapid pace to 44% by 2021 (almost two-third of the traffic). It also allows elephant flows (e.g., High Definition video streaming or cloud gaming) without degrading the mice flows (e.g., query data, updates) transported through the DC clusters. It overcomes most of the problems in the conventional network infrastructure such as– flexibility, ease of network configuration, malicious flow detection, and agility [8], [9]. Although the functionalities of SDN are efficient and reliable in preventing malicious threats and vulnerabilities, the issue of scalability persists due to a single logically centralized controller. A single network controller has limited resources such as– bandwidth, computing power, and memory for effective network management [10]. Therefore, in order to improve network scalability, it is essential to implement multiple distributed controllers [11].

In a distributed controller architecture, switches are mapped to their reference controllers. During the peak load, some of the switches receive mice flow while others may receive an elephant flow [12]. This unpredictable and dynamic nature of the incoming traffic flow creates an issue of load imbalance at the control plane [13]. Therefore, Load Balancing (LB) is required in order to migrate the load among different controllers resulting in a balanced state of the network [14]. Also, it is required to design a secure and dependable access control policies on the forwarding nodes (OpenFlow switches) such that they are capable of detecting anomalous behavior of the incoming flow. The flow statistics may reveal anomalies on the OpenFlow switches which may result in various malicious threats [153]. For example, CISCO report [16] suggests that around 37%, 38%, and 14% group of malicious file extensions were suspected in email documents in archive files (zip, jar, 7z) between January and September 2017. These malicious files just needed user interaction for malware activation. In the era of the IoT, one of the threats, i.e., botnets can generate Distributed Denial-of-Service (DDoS) attacks.

With an exponential increase in the usage of Internet-enabled applications on mobile devices, there has been a considerable increase in the energy consumption of these devices across the globe [17]. These devices communicate with each other using various intermediate network devices such as–servers, switches, routers, which are an integral part of the modern Datacenters (DC) [18]. As per the study in [19], DC will be gobbling up to 20% of the overall world's electricity consumption by 2025 and may become the cornerstone of the global economy. According to statistics given in 2017 [20], approximately 8 million DC across the globe are managing the data traffic workloads, which result in the consumption of 416.2 Terawatt-hour (TWh) of electricity. The total global electricity usage was 2% in 2017, which is estimated to increase by 5% by the end of 2020. It is predicted that if only 10% of the vehicles of the United States are converted to autonomous driverless cars, it would generate approximately 40 ZB of data

annually, i.e., around 4 TB of data daily [21]. Further, High Definition (HD) video streaming is one of the significant reasons that adversely affects the energy footprint due to its massive operations. As per the Cisco report [22], the video streaming will exponentially increase upto 82% of the Internet traffic by the end of 2021 which was only 72% in 2012. Netflix alone accounts for almost one-third of the Internet traffic in the United States by using Amazon web services as it does not have its own DC. By the end of 2025, it is expected that energy consumption by DC will be the largest in the world.

So, the traditional TCP/IP-based networks used for DC inter-connections are facing challenges of managing stringent Quality-of-Service (QoS) requirements of different applications of the end-users and service providers [23]. Furthermore, the existing techniques use concepts such as virtualization, lightweight containerization, and hybrid scheme to minimize the energy consumption of cloud DC [25]. The challenges faced by the traditional approach is that it follows distributed architecture where the control functionality is embedded in the forwarding devices and do not scale for large scale DC. The issue of flow scheduling and high power consumption at DC arises with the increase in the number of nodes and links in the network. The limitation with the distributed architecture is that the optimal decision of each flow is local, as a result, the distributed system is less power efficient while the centralized approach has the global information of flow in its database and is more energy-efficient. Hence, it becomes problematic on DC whenever underlying network resources (switches and routers) are under-utilized at the time of peak Internet traffic, resulting in high operational cost of energy utilization [24].

1.2 Software-Defined Networks (SDN)

To handle the aforementioned issues, SDN can be a viable solution as the control plane is completely isolated from the data forwarding plane in contrast, to the traditional vendor-specific network infrastructure [26], [28]. The feature of SDN being network programmable adds flexibility in network reconfiguration during traffic congestion [29]. The centralized architecture of the network controller helps to maximize the redundant utilization of links and nodes in order to route the delay-sensitive traffic through the optimized route as per their flow type [30]. The network controller in SDN consolidates the traffic by shutting down or hibernating the idle resources and thus, minimizes the total operational cost [31]. An example of an energy-efficient SDN-enabled OpenFlow switch is the Cisco Nexus 9000 series. The Nexus series is designed mainly for DCs to support 16 line cards, two chassis, three fans, six fabric with the bandwidth of 10 and 40 Gigabit Ethernet port (GBEP). The power consumption of each 10 and 40 GBEP is less than 3.5 Watt (W) and 14 W respectively [32]. However, the traditional Cisco catalyst 6500 series switch, which is mainly designed for enterprise and campus networks consumes more electricity. The catalyst 6500 switch consists of 2 line cards, each having 48 GBEP. The energy consumption of the fixed part of the switch is 472 W, and each port consumes 3 W [26].

Thus, in order to process the huge amount of data traffic generated between different DC, there is a requirement of efficient flow scheduling and energy-efficient mechanisms.

From the last few years, there has been an exponential increase in the Software-Defined Datacenters (SDDC) across the globe. SDDC provides flexibility for the data storage as compared to the traditional datacenters. They have three major components, i.e., storage, computation, and networking [33]. The integration of Network Functions Virtualization (NFV) along with Software-Defined Networking (SDN) leads to the popularity of SDDC. The hardware infrastructure equipment, i.e., storage, network, and the server, can be virtualized to deliver Infrastructure-as-a-Service (IaaS) which is the building block of SDDC [34]. It improves elasticity and agility within the traditional IaaS datacenter because the virtualized datacenters are automated and provisioned by intelligent policy-based software [35]. SDN is widely implemented in the datacenters to manage the workloads in cloud automation and cloud operations with a unified data management software [36]. According to the Allied Research report [37], the market of the SDDC is expected to grow annually by 32% on an average and is expected to reach worth $139 billion by 2022.

The limitations of the traditional datacenters are that they provide limited services to the end-users in terms of bandwidth and storage facility, high latency, manual device configuration, and traffic congestion. SDDC is a network programmable to provide intelligence analytics for infrastructure and workload provisioning. Thus, it efficiently addresses all the issues of traditional datacenters. Due to their centralized architecture, the issue of scalability alongwith resilience (to overcome the single point of failure) is still major concerns, which need new solutions from the research communities. The issues mentioned above of scalability and resilience in SDDC can be addressed by deploying multiple controllers at the control plane such that the data plane works smoothly in case of failure of a single network controller. It can be used to achieve a high-performance gain in large-scale networks. It extends the control plane functionality by improving scalability, reliability, efficiency, latency, redundancy, and data consistency [38]. However, the roles of a controller are as follows (i) to define and install the flow rules proactively on the data plane (ii) to perform traffic engineering by preventing network failure based on periodic traffic monitoring and flow management (iii) to virtualize the network functionalities such as multiple virtual controllers, load balancers, virtual switches, and gateways, and (iv) to monitor end-to-end data traffic through network links, and feedback collection from the forwarding plane. In addition, the responsibilities of the OpenFlow switches are as follows (i) to store flow tables in Ternary Content Addressable Memory (TCAM) (ii) efficiently handle flow table entries in order to perform routing of the incoming requests to the final destination (iii) follow forwarding decision installed by the controller and send a notification back to the controller, and (iv) encapsulation and tunneling of the packets [39].

Though the distributed controllers deployment is the most suitable technique to enhance the efficiency of a system, it may have challenges for its design principles. For example, one of the biggest challenges is the Controller Placement Problem (CPP), which results in

the generation of the issues of fault-tolerance, latency among controllers, availability, and their placement [40], [41]. The CPP deals with three important issues (i) *volume* (the number of distributed controllers required in the network topology) (ii) *placement* [42], and (iii) *resilience* (backup controllers). Hence, latency and cost are the two parameters that can affect network performance. Since there are few limited latency-sensitive applications so deploying too many controllers may cause an increase in the overall network cost [43], [44] [45]. In this paper, we propose a near-optimal solution for the CPP for SDDC.

1.3 Thesis Organization

The thesis would be organized as given below along with a brief description of what the chapter would represent.

Chapter 1: Introduction
This chapter introduces the limitations of the traditional datacenters of limited services to the end-users in terms of bandwidth and storage facility, high latency, manual device configuration, and traffic congestion. However, Software-Defined Networks (SDN) is a promising technology to address the issues (latency, load imbalance, network congestion, and the malicious traffic flow) to the Internet-enabled smart devices in large-scale networks. This chapter mainly focused on three problems: (i) load balancing at the control plane, (ii) energy efficiency and fast flow forwarding for the data plane, and (iii) scalability and resilience at the control plane. Moreover, the research challenges are also discussed with brief solutions.

Chapter 2: Literature Review
This chapter details the comprehensive literature review to resolve the problem of load imbalance among distributed controllers on the control plane. In addition, the existing solutions that rely on energy-aware routing to achieve energy-efficiency in SDN switches at the data plane are analyzed. Finally, the existing schemes to ensure the issues of scalability, resilience, and consistency on the control plane in large-scale networks are elaborated.

Chapter 3: Load Optimization and Anomaly Detection Scheme for Software-Defined Networks
In this chapter, we formulated the load balancing problem as a Mixed Integer Non-Linear Programming (MINLP) to minimize the average response time between the switch and the controller. Then to solve this problem, a load-balancing algorithm is designed to manage the flow traffic between OpenFlow switches so as to have a trade-off between switch migration cost and the controller's load variance. An SDN framework is proposed, in which an OpenFlow-based anomaly detection scheme is designed to identify the suspicious activities on the forwarding nodes. An anomaly mitigation scheme is also designed based on access control policies defined

for each user's behavior.

Chapter 4: An Energy-Efficient Fast Flow Forwarding Scheme for Software-Defined Networks
In this chapter, the Energy-aware Routing (EAR) problem is formulated by using an MINLP for which we proposed First-In-First-Out Push out Priority (FIFO-POP) scheduling schemes to minimize the average waiting time. In addition to energy-efficiency, an efficient flow re-routing and link utilization scheme are proposed. Finally, to build the shortest path between source and destination, an ant colony routing algorithm is proposed.

Chapter 5: Placement Availability Resilient Controller Scheme for Software-Defined Datacenters
In this chapter to resolve the issues of the Controller Placement Problem (CPP), a problem is formulated in the form of MINLP. The PARC scheme is proposed which divides the CPP into different sub-problems, i.e., network partitioning, the controller's availability, the minimal number of primary active controllers, and the minimal number of backup controllers required for resilience. The proposed scheme uses cooperative game theory to compute a stable network partitioning and an optimal position of the distributed controllers in each sub-network. The overall network cost of the proposed scheme is minimized by finding the minimal number of distributed controllers along with the extra backup controllers using a heuristic approach. Moreover, the PARC scheme performs global data consistency of the updated events among multiple controllers based on timestamp at the control plane, which provides synchronization among different events. The Pareto Optimal COntroller (POCO) toolset is used to simulate the numerical results in Matlab. The performance analysis compares the proposed PARC scheme with the other existing state-of-the-art schemes such as Pareto Optimal COntroller based on the simulated annealing (POCO-PSA), Pareto Optimal COntroller Multi-Objective Ant Lion Optimization (POCO-MOALO), and Capacitated Next Controller Placement (CNCP) scheme.

Chapter 6: Conclusion and Future Scope
This chapter concludes the thesis by highlighting the contributions made using the proposed schemes. Moreover, this chapter provides future directions in the security of SDN. In the future, we will explore the resilience of the POCO scheme by increasing the resilience level upto three or more controllers' failure to analyze the control plane reliability.

1.4 Summary

In this chapter, the challenges related to Software-Defined Networking based control flow optimization are discussed. The research work focused on three problems: (i) load balancing at the control plane, (ii) energy efficiency and fast flow forwarding for the data plane, and (iii)

scalability and resilience at the control plane. This task has been accomplished in this research work with three different approaches.

Chapter 2

Literature Review

The development and deployment of smart applications in a smart city environment have brought multi-facet benefits and challenges. In order to provide a high quality of experience to the end-user domain, the concept of data locality has anticipated the emergence of edge computing. In this potential edge-of-things ecosystem, the need to provision the smart services at a location closer to the end-user has led to the exchange of large volumes of data generated from the smart devices and sensors deployed at different geo-located sites. Such a massive amount of data generated from the smart terminals need to be collected, transmitted, analyzed, and processed using edge servers. This requires the seamless exchange of data among geo-separated nodes, which can degrade the performance of any designed solution and technology. Therefore, a dynamic, agile, and programmable network management paradigm is essential in the above discussed edge-of-things ecosystem. However, the traditional network technology, which relies on configurable black boxes ends up in increased computational overhead, misconfiguration, congestion, inefficient utilization of bandwidth, and lower data rates. To handle the above challenges, SDN has gained wide attention from academia, researchers, and the industrial sector. Shifting the computational load from forwarding devices to a logically centralized controller has realized the long term dream of every network operator who wanted complete control and global visibility of the network. In this chapter, the deployment of SDN in smart applications and infrastructural components of SDN and OpenFlow are explored for a better understanding of the underlying concepts.

Smart cities become the vision of various organizations across the globe for providing uninterrupted services and experiences to the end-users. From smartphones to bursty traffic, from social media to smart utilities, and from e-healthcare to smart infrastructure; smart technologies (Wireless sensor networks, smart grid, smart transportation, e-healthcare, and so on) have completely engulfed the geographic ecosystem. Multi-modal sensors, multidimensional data sources, powerful gadgets, and smart devices act like standalone machines to provision complicated services to a wide range of smart applications to provide quality of experience (QoE) and ambient living to the citizens of a smart ecosystem. Moreover, the evolution of the Internet of Things (IoT) and Tactile Internet have revolutionized the industrial organizations by

machine-to-machine and device-to-device communication . Modern set of standards and dynamic technologies have envisioned new horizons with data locality using edge/fog computing, cloud of things, mist computing, and osmotic computing. Smart communities (e.g., intelligent transportation, smart healthcare, smart utilities, etc.) are expanding in an exponential manner in vertical as well as horizontal directions. The enterprise model of Everything-as-a-service (EaaS) build over the underlying service-oriented architecture of smart communities rely on the five attributes listed by the National Institute of Standards and Technology (NIST). These attributes consist of: i) measured services, ii) on-demand self-service, iii) broad network access, iv) resource pooling, and v) rapid elasticity [46] [47]. Fig. 2.1 illustrates various service-oriented verticals in a typical smart city environment with smart communities.

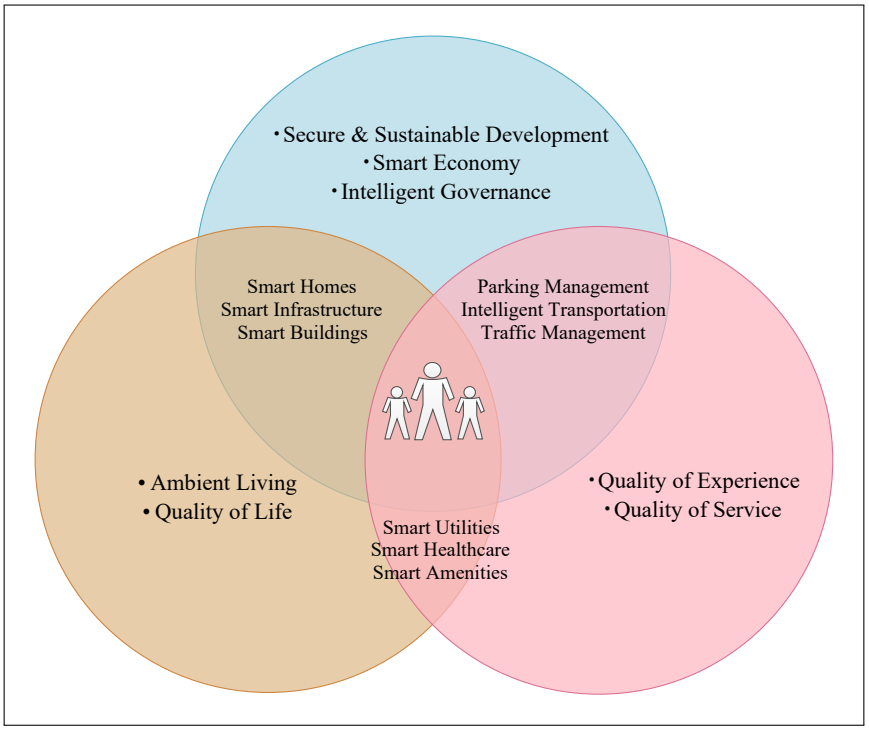

Figure 2.1: Various verticals of smart communities.

This entire service-oriented ecosystem can be viewed as a cloud of smart things, in which the data collection, analysis, and processing are handled closer to the location of the user. However, the user base of the entire smart community is huge enough to generate millions of Zettabytes (ZB) of data on a daily basis. For example, a typical smart city having a population of one million generates data of 200 million Gigabytes (GB) on a daily basis [49]. According to a survey report [50], 204 million emails are exchanged, 5 million searches are made on Google, 1.8 million Facebook users make likes, 350,000 tweets are exchanged, and the amount of 272,000 is spent on Amazon to purchase various amenities, and more than 15,000 tracks are downloaded over iTunes on a daily basis. With billions of devices embedded inside the smart city infrastructure, the entire framework is critically dependent on efficient data management.

The realization of smart cities relies on the efficient management of the large volume of data generated by various devices and sensors. For this purpose, one of the essential requirements is the seamless transmission of data across various geo-separated nodes.

However, an exponential increase in the ever-growing data traffic progressively erodes the margins of the network operators in terms of bandwidth, availability, and data rates. The tenable vision of having complete control of the network is a long dream of every network operator irrespective of the vendor of the network infrastructure. Network devices were considered as configurable black boxes until modern technological development enabled them to be programmed and configured according to users' requirements. This holistic visibility and control over the entire network through a logically centralized remote server breaks the vertical integration of traditional networks by Software Defined Networking (SDN). The provisioning of services to the end-user domain in a smart city scenario embraces various dynamic changes in the topological setup over a potential deployment of heterogeneous devices and sensors. SDN serves as a suitable candidate for providing agile, flexible, and scalable network management in smart application ecosystems. In the context of a global scenario, a lot of large-scale service providers such as–Facebook, Google, eBay, and Apple are working on the trends of softwarized data centers for the sustainable development of smart applications. Fig. 2.2 presents the taxonomy of the literature review on Software-defined Networking.

2.1 Scope of the survey

Keeping in view the above-discussed points, this survey is focused on the relevance of SDN in various smart applications. Specifically, this survey considers the deployment of SDN in four vertical domains namely; 1) Data Center networks (DCN), 2) Edge/fog computing, 3) IoT-based applications, and 4) Wireless Body Area Networks (WBAN). SDN is a promising technology that addressed the challenges of strong control and data plane coupling in traditional networks. By deploying SDN controllers in various smart applications, it is possible to dynamically change the network configurations as and when required. In this direction, various existing proposals have discussed different aspects related to SDN architecture. The potential research surrounding SDN and its perspective deployments was on zenith during the last five years (2014 to 2018). Most of the existing surveys on SDN considered diverse parameters such as software and hardware OpenFlow (OF) switches, controllers types, programming languages, emulation and simulation tools, flow table management, security, energy management, fault tolerance, network virtualization, and hypervisors' software. For example, authors in [51] discussed various aspects related to the evolution of programmable networks. Similarly, in [5], the authors investigated various aspects related to SDN from implementation points of view. Lara *et al.* [6] discussed different aspects related to OF paradigm in networking. Akyildiz *et al.* [7] investigated the role of SDN and OpenFlow for traffic engineering. In [8, 9], the authors discussed various aspects related to SDN in a comprehensive manner. In [10], the authors

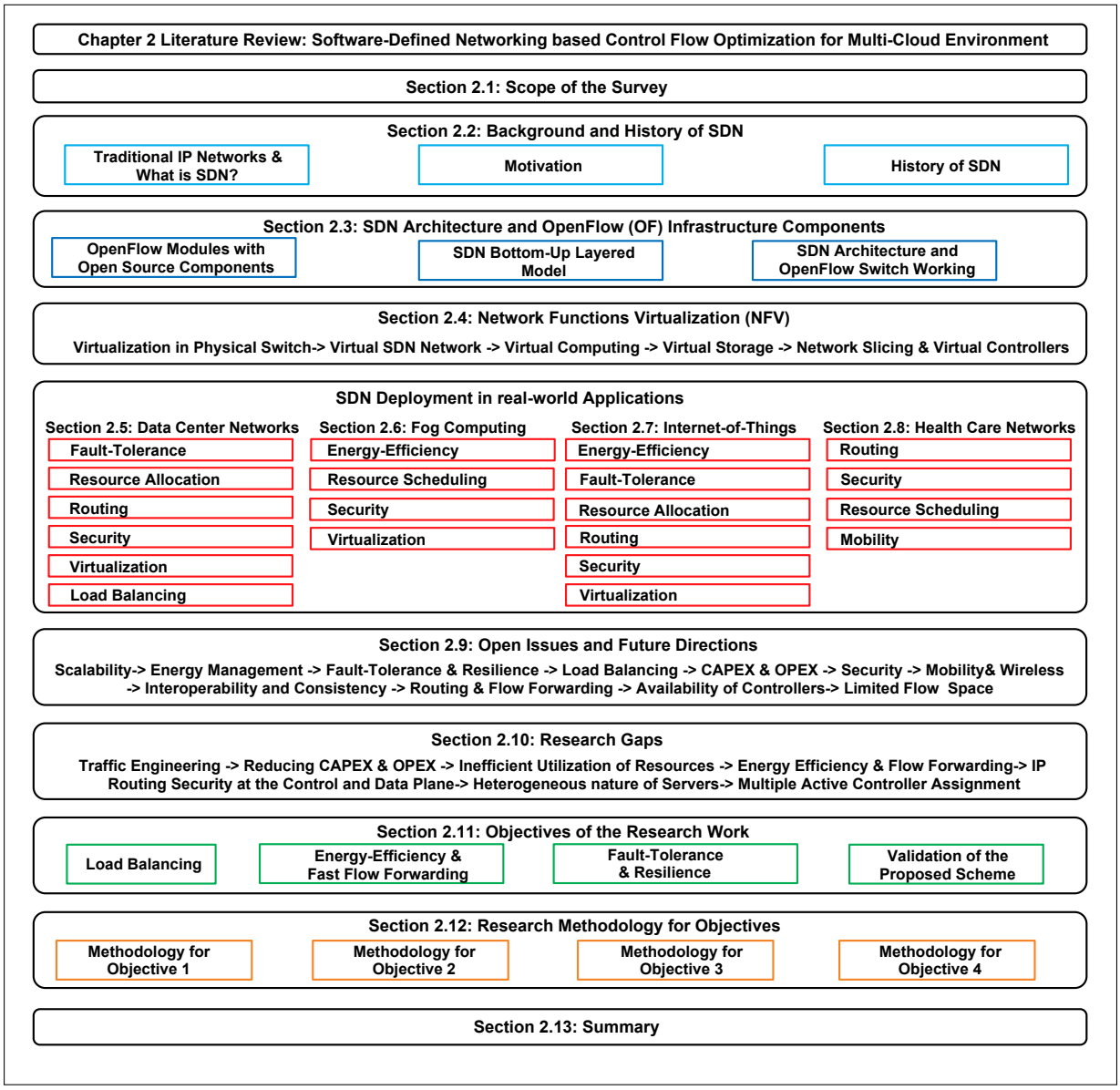

Figure 2.2: Taxonomy of the literature review on Software-defined Networking.

investigated various network virtualization hypervisors in SDN environments. In [11], the focus remained on the concepts of network function virtualization. Moving towards a different perspective, Trois *et al.* [12] surveyed various programming languages and their perspective impact on SDN. In another related work, Nguyen *et al.* [13] investigated various rule placement problems in OF networks. In [14], different unicast routing algorithms were discussed with respect to quality and performance parameters. In addition, Alvizu *et al.* [15] discussed the role of SDN in transport networks. In [16], various approaches and challenges with respect to different testbeds for large scale SDN were analyzed. Baktir *et al.* [17] discussed the potential benefits of SDN in edge computing. A major discussion in [18, 19] focused on the security challenges and possible solutions in SDN. In [20], the authors presented various SDN architectures along with a discussion on security and energy efficiency. Khan *et al.* [21]

highlighted various threats related to the topology discovery in SDN. After analysis of all the above-discussed proposals, a comparative analysis is provided in Table 2.1. It is pertinent to mention here that none of the above-discussed proposals highlighted any aspect related to the potential deployment of SDN and OF concepts for smart applications. Specifically, neither of the existing proposals have discussed the deployment of SDN and its related concepts in DCN, edge and fog computing, IoT and healthcare ecosystems. To the best of our Knowledge, this is the first survey of its kind which starts from the historic evolution of programmable networks, then move on to the architectural details and OF infrastructural components, and finally discuss different deployment models of SDN in smart applications (DCN, edge/fog computing, IoT and WBAN).

2.2 Background and History of SDN

This section presents the background and history of evolution of SDN. Moreover, the comparison between SDN technology with traditional networks is also presented in this section.

2.2.1 Traditional IP Networks and What is SDN?

The traditional networking is TCP/IP model and has significant limitations to meet the expectations of modern IT infrastructure. The limitations of the traditional networks are as follows: tightly coupled planes, distributed architecture, manual configuration, inconsistent network policies, error prone, security, and inability to scale. The control part and the packet forwarding devices are deeply integrated at the same place, embedded in the hardware devices. The strongly coupled nature makes it difficult to modify the network policies. The distributed architecture is vulnerable to security threats and it becomes extremely complicated to troubleshoot the network. The another issues is the inconsistency in the network policies as it becomes difficult to maintain consistency in modifying the network policies. With the substantial increase in the workloads in the IT infrastructure, the demands for the increased bandwidth is required. The traditional networking are statically arranged in such a manner that the growing demands of the IT infrastructure needs to redesign the complete topology. To overcome the issues faced by traditional networks, a prominent technology named as SDN based on the ideas of Openflow architecture is first introduced in 2006 at Stanford University in clean slate research project [55], [11].

The term SDN is basically a centralized software for controlling the entire network and is also called as software-based networking. SDN is ”softwarization” solution that provides network programmability by removing the network intelligence from the forwarding devices. SDN solves the issues of the traditional architecture and to accommodate the increased workloads of modern networking. The SDN architecture improves better services to manage enterprise, WAN, and data centers. The features of SDN are decoupled planes, programmable

Table 2.1: Comparative analysis of existing surveys related to Software-defined Networks.

Authors	Year	1	2	3	4	5	6	7	8	9	10	11	12	13	14	Research Challenges & Future Directions
Nunes *et al.* [51]	2014	✓	✓	✓	×	✓	×	×	×	×	×	✓	×	×	×	Controller & Switch design, Scalability, Controller-service Interfacing, Virtualization and Cloud Service Applications, Information Centric Networking, and Heterogeneous Networking with SDN.
Hu *et al.* [52]	2014	×	✓	✓	✓	×	✓	×	✓	✓	✓	✓	×	✓	×	Flow Forwarding, Scalability, Load Balancing, Resilience.
Lara *et al.* [53]	2014	✓	✓	✓	×	✓	✓	×	✓	✓	×	✓	×	×	×	Security, Availability, Scalability, Survivability.
Akyildiz *et al.* [54]	2014	×	✓	✓	×	×	×	×	✓	✓	×	✓	×	×	×	Flow Management, Fault Tolerance, Topology Update, Traffic Engineering.
Kreutz *et al.* [55]	2015	✓	✓	✓	✓	✓	✓	×	✓	✓	✓	✓	×	×	×	Resilience, Scalability, Security & Dependability, Mobility, Wireless.
Xia *et al.* [56]	2015	✓	✓	✓	✓	×	×	×	×	✓	×	✓	×	×	×	Routing, OpenFlow switch Flow Forwarding.
Blenk *et al.* [57]	2015	×	✓	✓	✓	×	✓	×	✓	✓	✓	×	×	×	×	Network Virtualization, Network Hypervisors.
Mijumbi *et al.* [58]	2016	×	×	×	×	×	✓	✓	×	✓	×	✓	×	×	×	Capital Expenditure (CAPEX) and Operational Expenses (OPEX) Compatability.
Trois *et al.* [59]	2016	×	✓	✓	✓	×	✓	×	×	✓	×	×	×	×	×	Abstractions, Programmable Forwarding modes.
Nguyen *et al.* [60]	2016	✓	×	✓	×	×	✓	✓	✓	✓	×	✓	×	×	×	Resource Allocation, OpenFlow rules placement problem.
Yan *et al.* [61]	2016	×	×	✓	×	×	✓	×	✓	✓	×	✓	×	×	×	Distributed Denial-of-service (DDoS) attacks against application level.
Hayward *et al.* [62]	2016	×	×	✓	×	×	✓	×	✓	✓	✓	✓	×	×	×	Network Security, Security Assessment framework.
Alvizu *et al.* [63]	2017	×	×	✓	✓	×	✓	×	×	✓	✓	✓	×	×	×	Control Plane Optimization, Abstraction, Scalability.
Huang *et al.* [64]	2017	✓	×	✓	×	✓	×	×	✓	✓	✓	×	×	×	×	Network Slicing, Compatability with existing network equipments.
Baktir *et al.* [65]	2017	✓	×	✓	×	×	✓	✓	×	✓	×	✓	✓	✓	✓	Protocols & Standardization, Network Functions Virtualization (NFV).
Rawat *et al.* [66]	2017	×	×	✓	✓	×	✓	✓	✓	✓	×	✓	×	✓	×	Energy Consumption on OpenFlow switches, Security attacks.
Khan *et al.* [67]	2017	×	×	✓	×	×	✓	×	×	✓	✓	✓	×	×	×	Routing, Latency and Cost Constraints.
Guck *et al.* [68]	2018	×	×	×	×	×	×	×	×	×	×	✓	×	×	×	Frequent Migration of the Virtual Machines, Authentication mechanisms.
Wang *et al.* [69]	2019	×	×	✓	×	✓	×	×	×	✓	✓	✓	×	×	×	Virtual Machine Management, Resource Scheduling.
Rafique *et al.* [70]	2020	×	×	✓	×	×	✓	×	✓	✓	✓	×	✓	✓	✓	Latency, Resource Scheduling, Security & Privacy Vulnerabilities.

Note- 1: SDN Hardware and Software Switches, 2: SDN Controllers, 3: OF-Switch Flow Entries Management, 4: Programming Languages, 5: Simulation Tools, 6: Security, 7: Energy-Efficiency, 8: Fault Tolerance, 9: NFV, 10: Hypervisor Softwares, 11: DCN, 12: Fog Computing, 13: IoT/WSN, 14: Healthcare Network/WBAN, 15: Open Issues & Research Challenges, ✓: considered, and ×: not-considered.

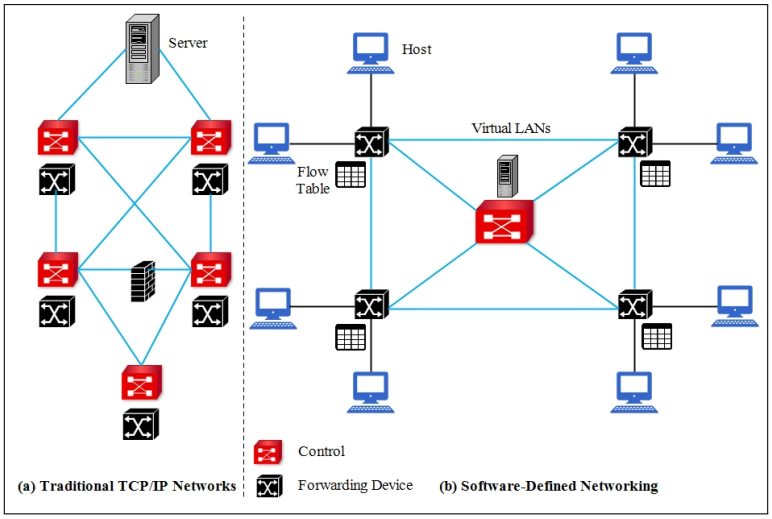

Figure 2.3: Comparison of traditional networking with SDN.

network and centralized model. The control or network brain and the data forwarding plane are decoupled, therefore, the forwarding nodes becomes control free and performs only packet forwarding functionality. Rather than manually configure all the network devices by administrator, SDN allows programmatic software control to automate the tasks and increased flexibility in modifying the network policies dynamically. An another feature is centralized architecture which helps to assign centralized policy management which helps to easily modify and audit policies. The centralized policies are used for routing, security, maintain consistency among physical and virtual workloads, Load Balancing (LB), and global monitoring of the network topology [51]. The SDN architecture provides scalability and flexibility to rebuild the topology through programmable network. Fig. 2.3 shows a comparison of the traditional networking with SDN.

2.2.2 Motivation

SDN is a promising technology that addressed the challenges of the traditional TCP/IP networking. By implementing SDN Controllers in various applications, it becomes flexible to dynamically change the network load. Table 2.1 shows a comparative analysis of the existing survey on SDN with our survey. The major work is done in last 6 years from 2014 to 2019. The existing survey on SDN considered various parameters such as- SDN software and hardware Openflow switches, SDN controllers types, SDN programming languages, emulation and simulation tools, flow table management, security in SDN, energy management, fault tolerance, network virtualization, hypervisor software's for distributed controllers and so on. Moreover, SDN is deployed in various applications in wide area networks (WAN) such as- data center networking, fog computing, Internet-of-Things, Wireless Sensor Network (WSN), healthcare network etc. Therefore, taking into account all the above discussed SDN parameters, the com-

parison of the existing survey is done with our proposed survey.

2.2.3 History of SDN

In this section, we summarize the history of SDN. The details of each design aspect of early programmable network and how network programmability is promoted year-wise by which organization is shown in Fig. 2.4. The technologies related to SDN is briefly described as below.

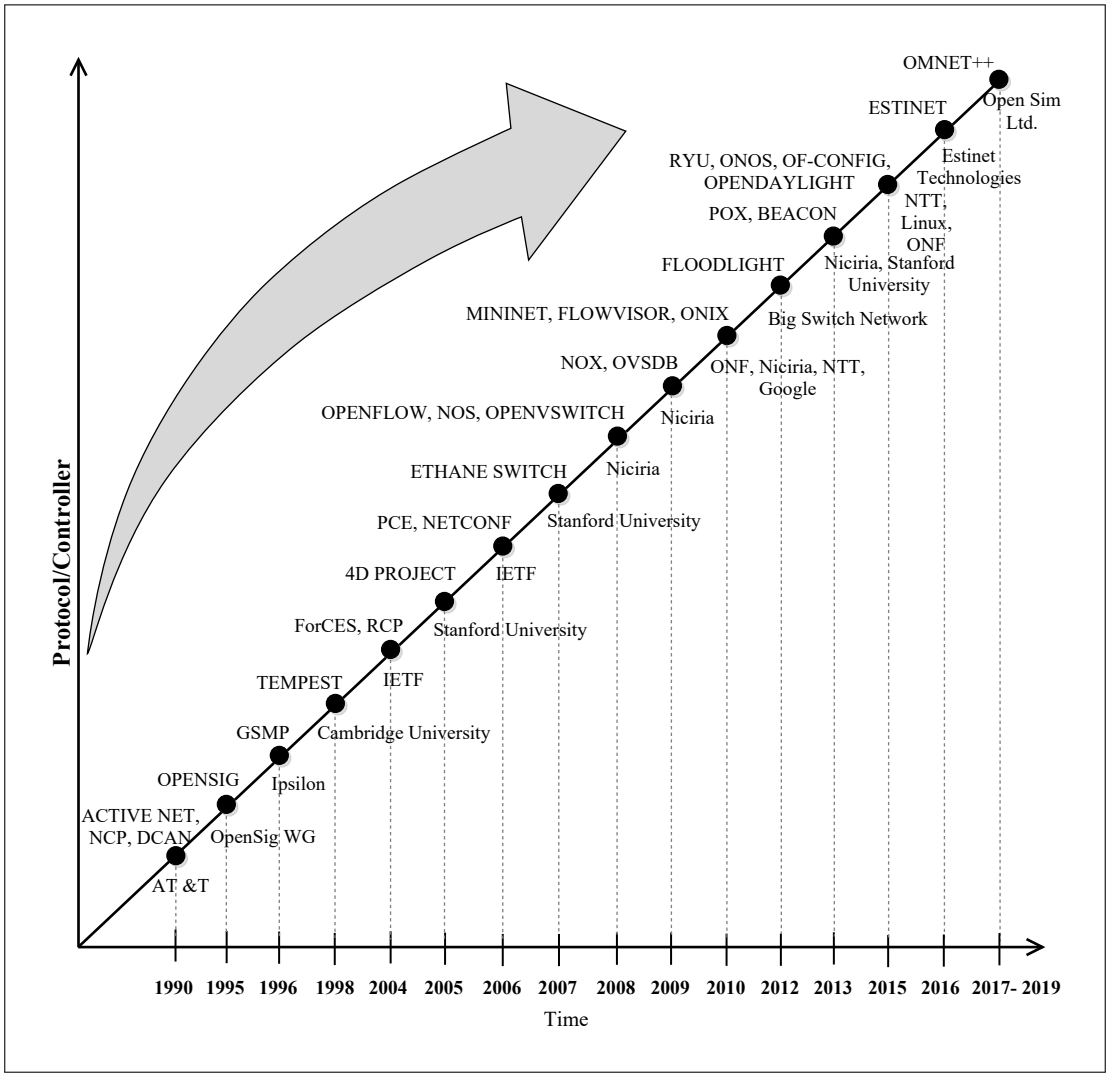

Figure 2.4: History of software defined networking.

- **Active Networking**: The active networking [55] takes the earliest initiative to promote software-based programmable network. The active networking project is introduced by US Defense Advanced Research Projects Agency (DARPA). The routing of active networking is generally based on two models: (a) programmable switches/routers, and (b) capsules [51]. The former models (programmable switches/router) simply download the

instruction sets sends from the control logic and process the flow rules for packet forwarding without any modification in the packet format. The programming model follows out-of-band signaling (use different bands or dedicated bands) to execute the control information at the network devices for managing channels. The later model (capsule), the control information to be executed are encapsulated in program fragments; where the tiny programs are enveloped in messages and are executed through the forwarding devices. Among the two models, the later became popular in Active networking as this approach provides significant impact in the management of network paths by installing a new functionality at the data plane. An example of active networking to create the software routers are SwitchWare [71, 72], Click [73], XORP [74], Quagga [75], Bird [76] etc. to control and modify the network operations dynamically on the real-time.

- **Network Control Point (NCP)**: In the mid 90s, when the Internet is becoming popular, the another initiatives is taken by AT&T telecommunication company by starting a project named as NCP [55]. The NCP targets to decoupled the control signaling of the telephone lines with the data plane. The control logic is completely isolated from the forwarding nodes to yield scalability and flexibility in the device management. The NCP improves the overall channel efficiency of the telephone signaling.

- **Devolved Control of ATM Networks (DCAN)**: Another project started in mid 90s is the DCAN [51] for scalable operations of the ATM networks. The project aims to design a protocol that split-up the control function of the physical ATM switches from the application feature and need to be handled by an external dedicated workstations entity. The external entity manages the allocation of resources to the ATM switches and provides the guarantees of improved quality-of-service (QoS).

- **OpenSig**: In 1995, an open signaling movement [51, 55] was introduced by OpenSig WG. The movement started workshops on open signaling to analyze the challenges faced by combining the control and the forwarding nodes. An Open signaling project promotes wider adoption of open, extensible, and programmable interfaces in the ATM networks and Internet. The idea of the proposal is to develop more intellectual programmable network which is possible by the separation of the switch controller with the switch fabric and the signaling between the planes is performed through a switch independent interfaces. The open signaling evolves both the levels of the planes to work independently. The open interfaces supports interoperability and new innovation in the design of the hardware equipments.

- **GSMP**: The Ipsilon Networks designed a asymmetric protocol in 1996 for WAN named as General Switch Management Protocol (GSMP)v1 [51]. Later on GSMPv2 and GSMPv3 is released in 1998 and 2003. The protocol is designed to control and manage ATM switches, frame relay or Multi-Protocol Label Switching (MPLS) through the external

controller. The various tasks performed by the protocol are monitoring and reporting statistics, supervise network switch ports, record all configuration information, insert or update switches on the multi-point connections, reporting statistics, creating and exempting switch connections via a controller etc. The protocol enables high-performance in ATM networks and grant permission to regulate both unicast and multicast switch connections.

- **Tempest**: Based on the motivation of programmable networks in ATM switches, virtualization is a new concept which laid it's foundation in early 90s. The earliest initiatives on the network virtualization in switches is started by the Cambridge University. In context with ATM networks, the tempest [77] framework is the first project started in 1998 which supports switches virtualization. The virtualization in the tempest framework helps to share single physical network resources by multiple independent switches. The multiple switches are created virtually residing on a single physical ATM switch. The multiple switches and virtual network is the substitute of control architecture into an operational ATM network.

- **Forwarding and Control Element Separation (ForCES)**: The Internet engineering task force (IETF) designed a ForCES protocol [51, 55] in 2004. The protocol is designed by the consolidation of two elements i.e. Forwarding Elements (FEs) and Control Elements (CEs). The message sharing between the CEs and the FEs is done using the master-slave architecture via the ForCES protocol. The CEs are the masters and the FEs are the slaves. The function of the FEs is flow forwarding of every packet. The CEs is the Logic Function Block (LFB) to create the control logic and installs the control information through signaling. The CEs monitors how the forwarding elements process each packets.

- **Routing Control Platform (RCP)**: In [78], the author designed a RCP protocol in 2004. The RCP is designed to reduce the complexity of the Border Gateway Protocol (BGP). The IP routers in BGP performs inter-domain routing in an autonomous system. The fully distributed path selection computation is done in each domain. The RCP overcome this limitation by separating the functionality of inter-domain routing from IP routers. Thus, the computation of the best path selection for each router in each domain is performed at the control plane. Finally, the result is intimated at the data plane for the best path in each domain and also exchanged routing information of RCP with other domains.

- **4D Project**: In 2005, Stanford University proposed a 4D approach [79] for network control. The 4D approach refers to bottom-up layered architecture. The lowermost layer is the data, middle layer comprises of discovery and dissemination, and finally the topmost layer is the decision plane [80]. The decision plane is the management layer with the primary objective to create logic for routing, LB, security, interface configuration, and

access control. It can monitor the global topological view and the logic is elevated to the forwarding nodes. The discovery layer performs the functions such as- physical device discovery, build their logical entities, association between the physical and logical entities, capacity of the interface and which is suitable for a particular switch/router, and also discover the capacity of the switches to hold the flow table entries. The dissemination layer performs the establishment of the secure connection between the decision and data plane, and in addition delivers timely and reliable multi-casting of control information. The data layer performs the functions such as- packet filtering, address translation mapping, next-hop forwarding, queue management, and tunneling.

- **PCE and NETCONF**: In 2006, IETF standardized two communication protocols namely, Path Computation Element (PCE) [81] and Network Configuration (NETCONF) to determine the network path based on a network graph. The PCP is the request/response communication protocol between the Client-Server model. The server computes the optimal route on the basis of on-demand requests from clients. The PCP protocol is suitable for the computation in ATM, Ethernet, MPLS, and BGP with the requirements of high bandwidth services. The another protocol is the NETCONF that deals with the network configuration and sends a notification whenever any change in the configuration. The NETCONF protocol [55] uses XML based data encoding which is designed to install, update, and delete the network configuration as per the dynamic network nodes.

- **ETHANE Network**: A new network architecture called Ethane network was designed by Stanford University [82] in 2007. The communication in an Ethane network between host X and host Y is established on the basis of five activities. The activities are switches and ports pairs registration, switches bootstrapping for secure channel connectivity, hosts authentication, flow setup, and forwarding. The Ethane network architecture is designed in such a way that the communication between the end hosts should not exchanged messages directly they have to follow all the five above mentioned activities. In addition to various SDN projects, a list of Openflow modules and open source components arises in the market from year 2008 to 2019 is explained in the next part.

2.3 SDN Architecture and OpenFlow (OF) Infrastructure Components

The architectural and controller deployment details related to OF hardware infrastructure is elaborated in this section.

2.3.1 OpenFlow Modules with Open Source Components

The term OpenFlow (OF) is the open source communication protocol developed by Stanford University and Niciria in 2008. The OF-protocol is the flow-oriented protocol with switches and ports abstraction to allow message sharing between the network controller and the forwarding devices. An OF-Switch comprises of three modules namely; (a) multiple flow entries tables and group table, (b) OF communication channel, and (c) OF protocol. The OF-switches connects to each other via a OF ports and each OF-switch establish a secure communication with the network controller via a OF protocol. The firmware of OF-Switch and OF-vSwitch is Pica8, Indigo and the kernel is written in linux 3.3.

In 2008, with the existence of OF protocol in the market, the idea is to design an operating system on servers that is centralized logically and control all the heterogeneous network devices via a OF protocol. The conventional operating systems (such as- Cisco IOS, Juniper JUNOS, ExtremeXOS, and SR OS) are device specific and are only compatible for it's products. Thus in 2008, an OF-based Network Operating System (NOS) (such as- NOX, ONOS, ONIX) [11] is designed to provide high-level abstraction for network resources. The SDN controller software gets installed on the PC server's NOS to configure the network policies build by an administrator. The NOS provides abstractions of essential services and interfaces to connect with the forwarding devices. The southbound interface is the Open Virtual Switch Database (OVSDB) for OF-switches. OVSDB interface is necessary to create multiple virtual OF-switches (OF-vswitch) as an instance of physical OF-switch. The virtual switches are used for efficient utilization of network resources (memory, storage, and communication links). In 2015, another southbound interface is the IF-CONFIG designed by the open network foundation (ONF).

Tables 2.2 and 2.3 shows a summarized overview of all SDN OpenFlow (OF) modules consists of OF-hardware switch fabrics, OF-software switches, logical and physical interfaces, SDN controllers, simulation, and emulation tools.

2.3.2 SDN Bottom-Up Layered Model

The SDN architecture is depicted in Fig. 2.5 and follows a bottom-up layered approach. The bottom-up layered SDN model is a three layered architecture also called as networking planes i.e. data plane, control plane, and the application or management plane. The lowermost layer is the data plane consists of network forwarding nodes like; physical OF-switches, OF-routers, and gateways. Additionally, OF-vswitch are the virtual switches and port to multiple hardware platforms. The forwarding nodes performed various operations such as- packet header lookup, forwarding, encapsulation, pipeline processing and data statistics. The SDN controller communicated with the forwarding nodes via a southbound interfaces. The southbound interfaces such as- Openflow, OVSDB, Protocol-Oblivious Forwarding (POF), ForCES, NetConf, OF-Conf and so on are used as a communication protocol [59].

Table 2.2: List of SDN hardware switches and open source components.

Type	Name	Developer	Language	Compatible OS	Model	Interfaces	Features and Config Commands
Hardware Switches	2960x, 3000x	Cisco	-	Cisco IOS, NOS	-	Openflow, OVSDB, BGP	Max. Capability- $16K^*$ mac addresses, 16 static IPv4 and IPv6 unicast routes, $1K^*$ active VLANs
	EX9200	Juniper	-	JUNOS, NOS	-	Openflow, VxLAN, OVSDB, LLDP	Max. Capacity: 64 GB DRAM, 64 GB SSD, 512000 forwarding entries, 32000 VLANs, 256000 IPv4 and Ipv6 multicast routes, bandwidth 480 Gbps/slot
	PF5820	NEC	-	OS independent	-	Rest, Nova API	Max.- 160000 flow entries, 48 ports, 4094 VLANs
	MLX Series	Brocade	-	OS independent	-	Openflow, OVSDB	Max.- 128000 flow entries, 9.5 billion pps routing, 4094 VLANs
	3920	Pica8	-	OS independent	-	Openflow, OSPFv3	Max.- 4094 VLANs, forwarding speed 960 Mpps, latency 900 ns
	7150 series	Arista Networks	-	OS independent	-	Openflow, OVSDB	latency 350 ns, routing 960 Mpps, 4096 VLANs
	X8	Extreme Networks	-	OS independent	-	Openflow, BGP	latency 2.3 microsecond, 11.4 billion pps, 4094 VLANs
Software Switches	9000x, 3000x	Cisco	Python	Cisco NX-OS, linux	centralized, distributed	Openflow, REST	enable openflow agent, configure-physical devices and interfaces
	Contrail-vrouter	Juniper	Java, Python	XtreemOS, linux	centralized, distributed	REST API	routing and switching, load balancing, security and API services
	Switchlight	Big Switch	Java	switch light OS, linux	centralized	Openflow, Rest APIs	thin switching software for physical and virtual switches
	OpenWRT	Stanford University	Java, Python	linux	centralized, distributed	Openflow	configure wireless router to an Openflow-enabled switch
	OFsoftswitch13	Ericson	C, C++	linux	centralized, distributed	Openflow, Rest APIs	provides secure channel to connect switch to controller
	OpenvSwitch	ONF	C, Python	linux	centralized, distributed	OVSDB, Openflow	virtual software switch and ported to multiple hardware platforms

Table 2.3: List of SDN hardware switches and open source components (Contd..)

Type	Name	Developer	Language	Compatible OS	Model	Interfaces	Features and Config Commands
Logical, Physical Interfaces (switch ports)	L2 (physical), virtual LANs (logical)	ONF (Open Networking Foundation)	C++, Java, Python	linux, Mac, Windows	centralized, distributed	Openflow, NetConf, OVSDB, PCEP	**Msg. (C → S)**: read, modify, packet-out, handshake, switch configuration. **Sync Msg.**: hello, echo request/reply. **Async. Msg.**: packet-in, port status
Controllers	Opendaylight (ODL)	Linux Foundation	Java	linux	distributed	Rest, OSGi	topology discovery, shortest path forwarding
	Floodlight	Big Switch	Java	linux, Mac, windows	centralized (multi-threaded)	Rest APIs	modify behavior, routing, forked from the Beacon controller
	POX	Niciria	Python	linux	centralized	adhoc API, Openflow, OVSDB	path selection, load balancing, queriable topology graph and support for virtualization.
	NOX	Niciria	C++, Python	linux	centralized	adhoc APIs	1st openflow controller, tasks- topology management, routing, switching
	Ryu	NTT	Python	linux	centralized (multi-threaded)	Rest APIs, RPC	*rest_router.py*, *rest.firewall*, *rest_topology.py*, *rest_conf_switch.py*
	Hyperflow	(virtual)	Java	linux	distributed	Rest APIs	synchronize events of multiple controllers, guarantees loop-free forwarding and is resilient to network partitioning.
	Disco	(virtual)	Java	linux, Mac, OS	distributed	Rest APIs	intra-domain and inter-domain functionalities with other controllers
	Smartlight	(virtual)	Java	linux	distributed	Rest APIs	fault tolerant hierarchal master slaves controller
Simulation Tools	Mininet	Stanford University	Python	linux	centralized	Openflow	free license (CLI): *sudomn*, *arp*, *mac*, *switch ovsk*, *sudomn − −test pingall*
	Estinet	Estinet Tehnologies	C, C++	linux, Windows	centralized	Openflow	paid license (GUI), Uses tunnel interfaces and sync simulation clock to control the order execution of events
	NS-2	LBNL	C++, Python	linux	distributed	Openflow	integrates with OpenflowSwitch, supports built-in to emulate an openflow for real-time simulations.
	OMNET++	Open Sim Ltd.	C++	linux, Mac, Windows	distributed	Openflow	use .ned files to assign addresses to nodes then reads configuration file omnetpp.ini

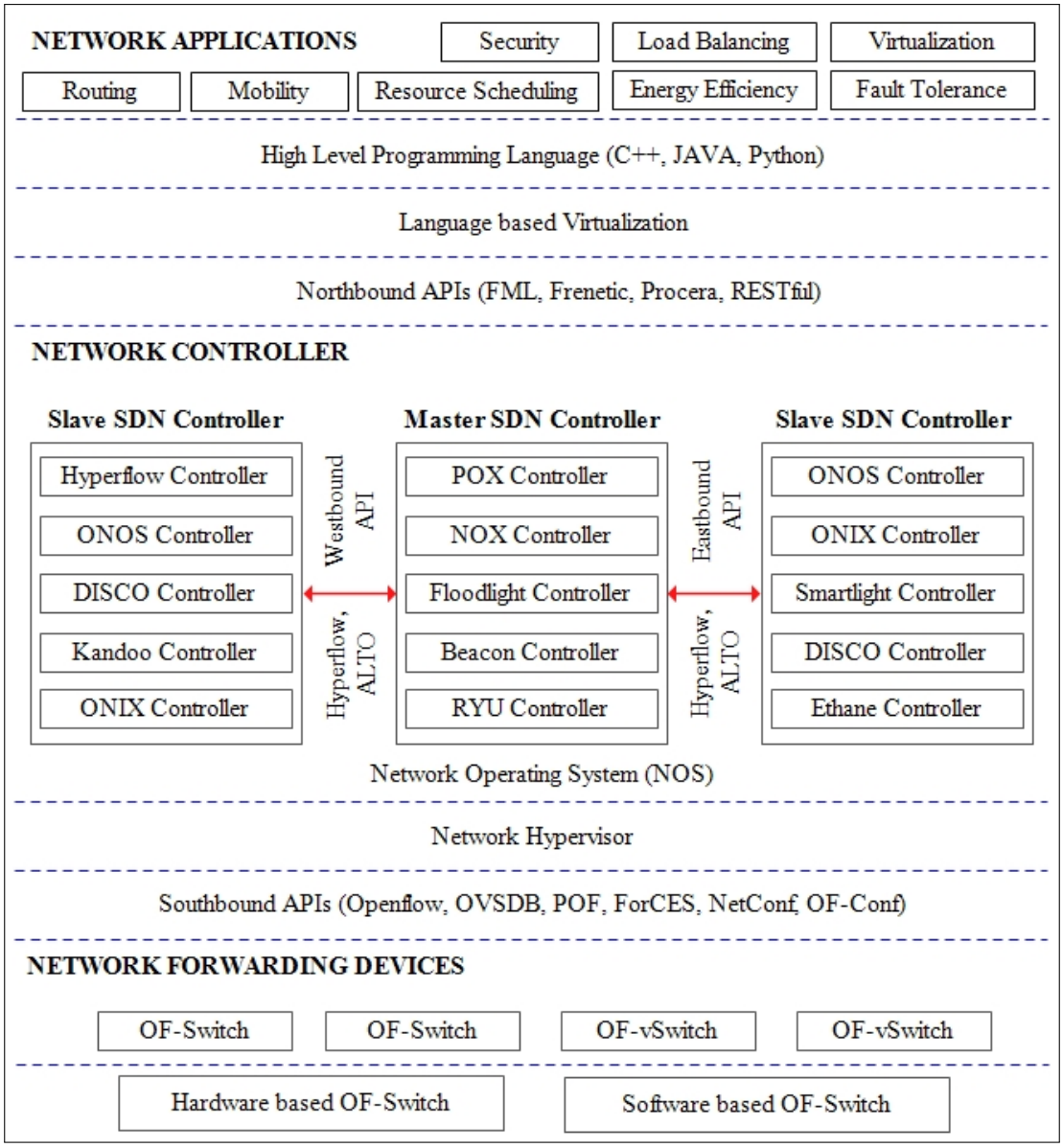

Figure 2.5: Layered architecture of SDN.

The middle plane is called as the control logic or network brain on which the SDN controller software resides. The SDN controller is installed on the NOS. With the help of hypervisor software, master-slave controller architecture is designed. The network hypervisor also creates multiple virtual switches on the data plane controlled by the SDN controller. The requirement of virtualization concept helps in failure recovery as complete back-up of the master controller is performed on the slave controllers. The efficient resource utilization is achieved through multi-threaded controllers as a slave or virtual controllers created from a master controller. The communication between the master and slave controllers is done via a eastbound and westbound interfaces such as- Hyperflow, and Application Layer Traffic Optimization (ALTO).

At the topmost layer is the application or management plane. The network applications runs on the application layer. The list of network applications are routing, security, wireless

and mobility, LB, virtualization, resource scheduling, energy efficiency, fault tolerance and so on. The programmer or expert remotely handles the network applications through programming in high-level languages (C++, Java, Python) [59] and the instructions are executed on the SDN controller to control the forwarding nodes. The SDN applications exchange message with the network controller via a northbound interfaces. The common northbound interfaces are as follows: Field Manipulation Language (FML), Frentic, Procera, and Representational State Transfer (REST) APIs [56].

2.3.3 SDN Architecture and OpenFlow Switch Working

Fig. 2.6 presents SDN Architecture and working of OpenFlow Switch. In an SDN architecture, an OF-switch has two components i.e. hardware and software. The hardware component of an OF-switch is a hardware switch fabric (application-specific integrated circuit) of multiple vendors having distinct Ternary Content Addressable Memory (TCAM) capacity to store multiple flow tables entries. The software component of an OF-switch is a programmable switch to perform necessary actions on the flow rule instructions. The software/programmable OF-switch consists of multiple flow tables (flow table 0 to N) and are connected with each other via a pipeline. Each flow table has four fields: match rule, priority, action, and a counter. The match rule checks the attributes such as- switch port number, MAC address, Ethernet type, VLAN address, IP address, and TCP address. Moreover, an another attribute is the priority field to assign the data priority as high, low or medium based on the data categories. For example, set data priority of voice and text data as high, images as medium, and video as the lowest priority based on the bandwidth consumption. Hence, the action is first performed based on the highest priority. The next is the action field for taking the necessary actions on the data packet. The actions such as- forward the packet to egress ports, forward normal in a pipeline, forward to controller or drop the packet. The last part is the counter field to count the number of packet processed in the queue along with their lifetime. The flow tables also have timeout mechanisms consisting of hard and idle timeouts. The hard timeouts is activated after a fixed amount of time and idle timeouts occurs after a period of inactivity [60].

Figure 2.5 shows how network packets are send from host A having a particular source address (Suppose, MAC source: 00-00-00-00-00-00-00-0A, IP source address: 20.0.0.1) to host B address (suppose, MAC source: 00-00-00-00-00-00-00-0B, IP source address: 20.0.0.2) through the OF-switches. The steps to be followed are as follows:

Step 1: Initially, the data packets of host A is send to the edge switch through access points (Wi-Fi, cellular base station).

Step 2: The packet is received at the ingress port 1 of an OF-switch. The programmable switch initiates the lookup process. The flow table stored in an OF-switch performs rule matching for each packet header. The matching field returns the output as no existing matching rule in an OF-switch.

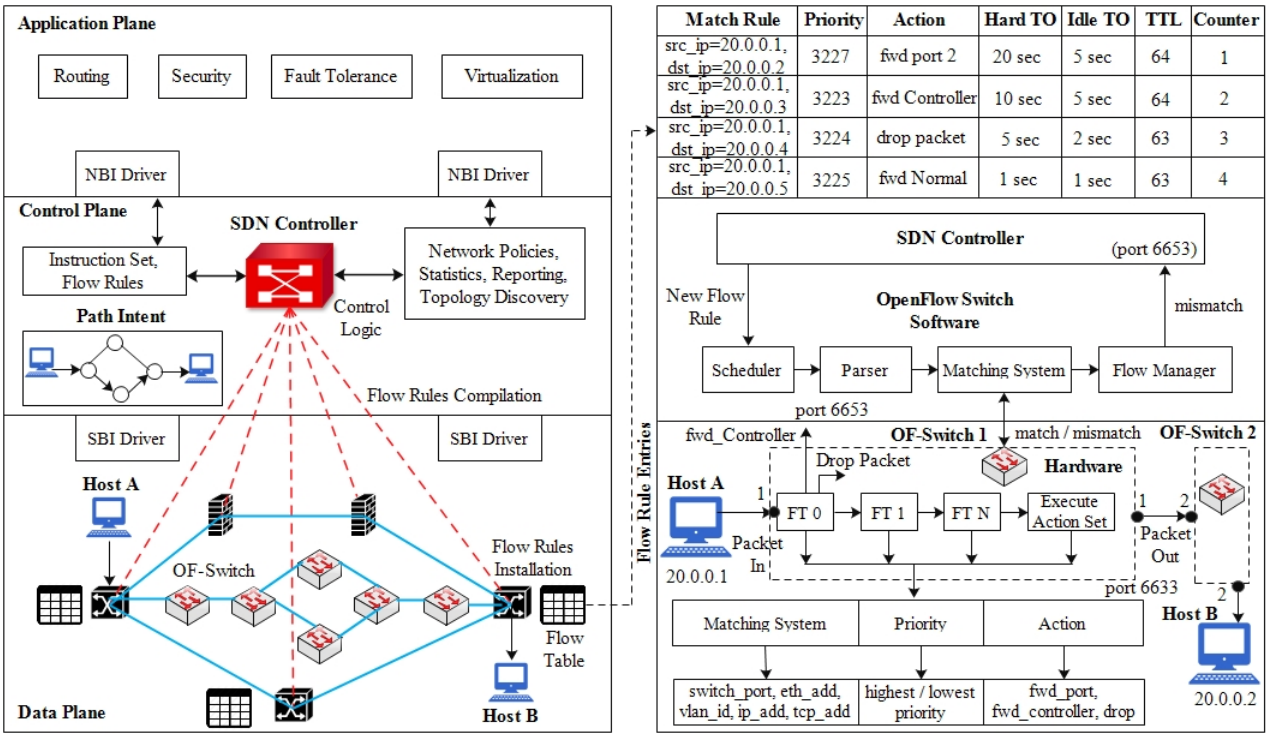

Figure 2.6: SDN architecture and working of OpenFlow switch.

Step 3: The OF-switch first encapsulates the packet and transmit a notification to the SDN controller in case the matching rule does not exist.

Step 4: The controller creates a new flow and matching rule for routing the packet to the destination.

Step 5: The controller install the matching fields on port 1 of an OF-switch and install the action field on port 2 of the switch.

Step 6: The flow table is updated on the OF-switch. The lookup process is repeated again and returns the output as flow rule is matched successfully.

Step 7: Based on priority order takes the necessary actions to forward the packet on port 2. Therefore, the packet is forward to the next switch where flow table lookup process repeats and after successful matching forwards to the destination (host B).

In case, the rule is not matched again then the action performed is to drop the packet and inform the controller to build a new flow rule. The controller generally listens on port 6653 for every request. The next part is to integrate virtualization in SDN architecture which is explained below.

2.4 Network Functions Virtualization (NFV)

With the advancement in technology, the size of the physical infrastructure expands and becomes more capable to offer innovative services to users/tenants. However, the scarcity of network resources always exists as the tenants demands rise consistently. The solution is to

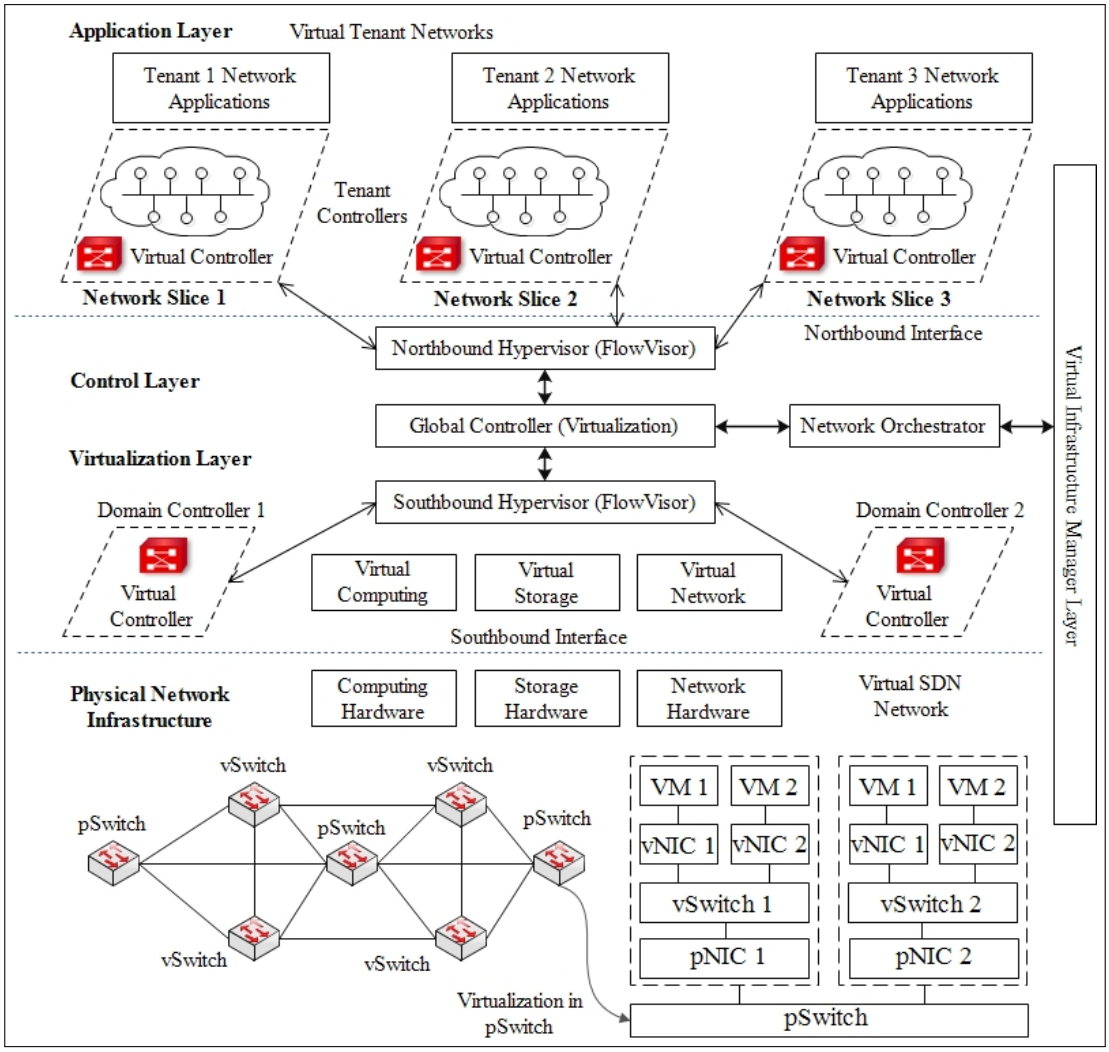

Figure 2.7: Network Hypervisor in an SDN architecture.

efficiently utilize the physical resources among tenants. Network Virtualization (NV) is an imminent solution having the physical infrastructure sharing capabilities to meet the high demands of tenants. The concept of NFV intertwined with SDN is started in 2012 at Stanford University with the development of OpenFlow architecture [56], [57]. The concept of network virtualization in an SDN architecture performs slicing of physical infrastructure into multiple logical isolated virtual SDN networks. The network slicing technique improves the utilization of hardware resources as each independent network slice has its own resources such as- link bandwidths, switch CPU resources, switch forwarding tables, network topology. The network slice is performed at each layer. For example, at the physical layer, the wavelength division multiplexing (WDM) [63] creates an instance of the physical resources. Next is the data link layer, where network slices are created by provisioning Virtual Local Area Networks (VLANs), the network slices of the forwarding tables in an Openflow switch is created by Multiprotocol Label Switching (MPLS), and Virtual Routing and Forwarding (VRFs along with the specific VLAN mapping) create virtualization at the network layer (layer 3) [58].

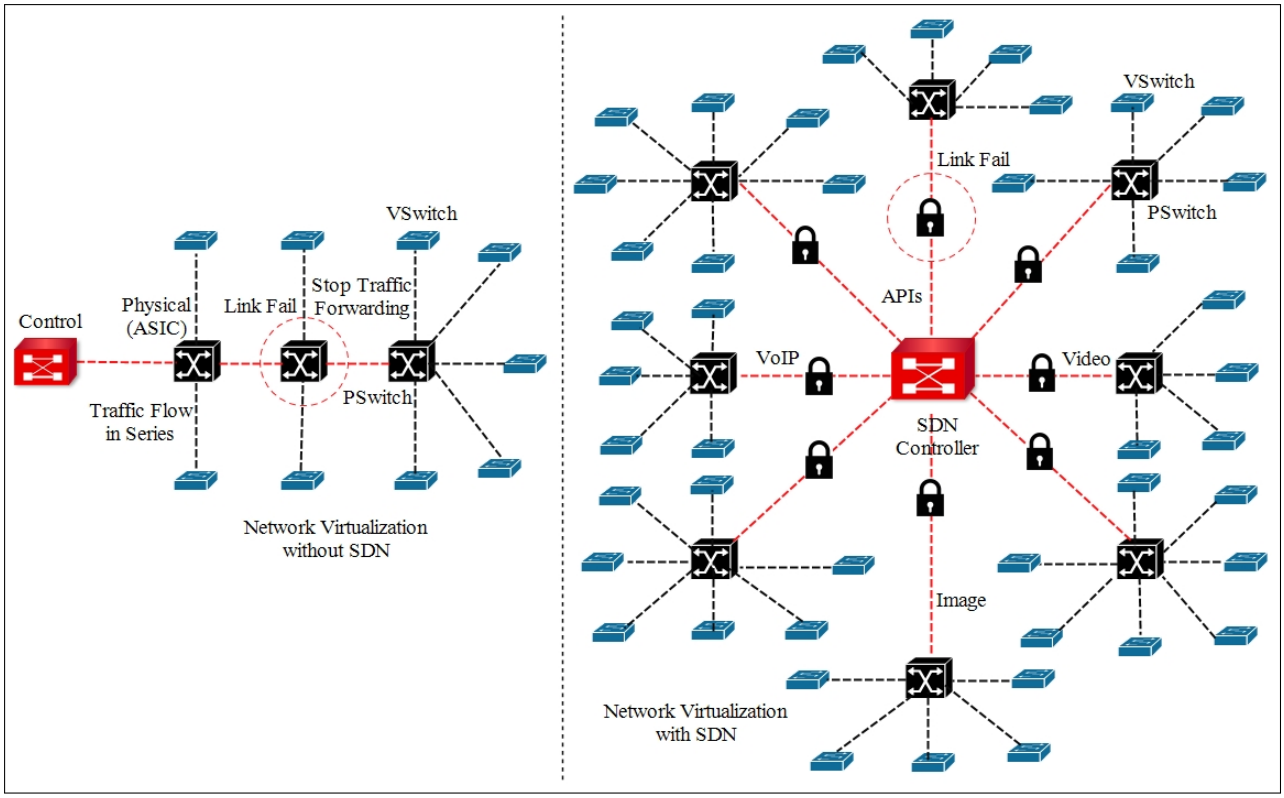

Figure 2.8: An example of Network Function Virtualization with SDN and without SDN.

The network hypervisor (Virtual Machine Monitor (VMM)) is software installed at the NOS to create network virtualization in an SDN architecture. Both the centralized and distributed architecture of hypervisors are used to monitor the virtual SDN networks. Fig. 2.7 shows an architecture of network hypervisor in an SDN architecture. The physical controller (pSDN) monitors a complete global abstracted topological view to take intelligent decisions regarding flow forwarding. The pSDN is connected via a network orchestrator and virtual manager to handles inter-domain routing. The network orchestrator acts as a centralized hypervisor resides on the pSDN. The distributed hypervisors (northbound and southbound hypervisors) are present on the top of the network orchestrator and virtualizes multi-domains. The northbound and southbound hypervisor is installed at the control plane and using the Openflow protocol, the hypervisors create logical virtual SDN networks (vSDNs) on the top of the pSDN. Fig. 2.7 shows how a physical SDN network is sliced into three virtual SDN networks and a virtual/proxy controller (vSDN) is located in every virtual network. At the application plane, tenants run multiple applications simultaneously on every vSDN controller in all virtual networks. NFV technology is useful in SDN to create Network Virtualization (NV), storage virtualization, and server virtualization. The NV runs virtual network applications such as- routing, firewall, access control, LB, and so on. The NV also creates an instance of network resources such as- virtual switch (vSwitch), virtual router (vrouter), and vSDN [58]. At the data plane, multiple vSDNs creates an instance of the physical switch (pSwitch) to create a logically isolated virtual switches (vSwitches).

The internal working of how the logical vSwitches are created from the pSwitch is explained with the concept of virtualization of the Network Interface Card (NIC) used for communication. The pSwitch consists of multiple physical NICs (pNICs) and each pNIC create a vSwitch by creating virtual NICs via the hypervisor. The hypervisor software creates virtual NICs to run several Virtual Machines (VMs) builds on top of an individual Physical Machine (PM) because each VM requires its own vNIC. Therefore, the flow table entries are executed both at the pSwitch and the vSwitches [83]. The virtualization layer is a proxy or middleware between the forwarding nodes and the controller as well as between the controller and the network applications. The FlowVisor is the most popular hypervisor software used for the efficient utilization of physical hardware infrastructure in the SDN planes. Fig. 2.8 shows an example of Network Function Virtualization with SDN and without SDN. Table 2.4 shows the list of network hypervisor software along with their description.

In the next section, we discuss the deployment of SDN in various applications.

2.5 SDN Deployment in Data Center Networking (DCN)

The SDN Controller is implemented in DCN to resolve the issues related to energy management, failure recovery, resource allocation, routing, security, and virtualization. Fig. 2.9 shows the scenario of deploying SDN controller in real-world applications. The related work on all these above-mentioned issues (energy management, fault-tolerance, resource allocation, routing, security, and virtualization) are discussed below.

A taxonomy of deployment of SDN in various applications such as– data center networking, fog computing, Internet-of-Things, and healthcare network is shown in Fig. 2.10.

2.5.1 Energy Management in DCN

The Data Centers (DC) located at distributed geographical locations perform the tasks related to storage, network, and computation. With the expansion of DCs, the huge amount of energy is consumed by the geo-distributed DCs to serve millions of user requests daily. To solve the issues of energy management, various proposals have been presented by the researchers. Goyal *et al.* [27] proposed an energy-efficient inter and intra communication in wireless sensor networks. Aujla *et al.* [28] proposed an efficient energy management scheme in Cloud DCs using SDN. To achieve optimal utilization of resources, the author integrates Renewable Energy Sources (RES) with DCs and designed two approaches for energy efficiency in DCs. The first approach is the Support Vector Machine (SVM) classifier for workload distribution of incoming jobs and the second approach is Game Theory (GT) to handle the scheduling of workload requests. Yu *et al.* [84] optimize the energy consumption cost by presenting a VM placement and power-saving routing algorithm. The process of placing several VMs on a single host PM to run various network applications concurrently in order to achieve maximum utilization of

Table 2.4: List of network hypervisor software integrated with SDN.

Hypervisor Softwares	Protocols and Standard APIs	Architecture	Support OF	Open Source	Control Plane Isolation	Data Plane Isolation	Description of the Hypervisor features
FlowVisor [57]	JavaScript Object Notation, Openflow	centralized	✓	✓	×	✓	(1) Middleware between the tenants SDN controllers and the OF-switches. (2) Isolates: bandwidth, topology, flow space, switch CPU, forwarding tables, control channel.
VeRTIGO [57]	REST, NetConf, Openflow	centralized	✓	×	×	×	(1) Provides additional capacity in FlowVisor by creating virtual links and virtual ports.
OpenVirteX [57]	JavaScript Object Notation RPC, Openflow	centralized	✓	✓	×	✓	(1) Each network slice has its complete flow space without overlapping.
FlowN [57]	Openflow, OVSDB	centralized	✓	×	✓	×	(1) Provides a MySQL database and performs container-based application virtualization.
BYOC-Visor [83]	Openflow, Generic Routing Encapsulation (GRE) tunnels	centralized	✓	✓	✓	×	(1) Provides customized and secure services to tenants through security applications.
Ad-Visor [57]	JavaScript Object Notation, Openflow, GRE	centralized	✓	✓	×	✓	(1) Improves topology abstraction and sharing of the flow space by vSDNs. (2) The tenant SDN controller inspects only endpoints of the virtual path for packet forwarding.
Slice Isolator [57]	Openflow, NetConf	centralized	✓	✓	✓	✓	(1) Allow different vSDNs controllers to exchange flow tables for similar functions (2) Allow interface isolation between the vSDNs and physical port to map the incoming data.
MobileVisor [57]	OpenFlow with GPRS Tunneling Protocol (GTP)	centralized	✓	✓	✓	✓	(1) Proxy software and acts as a mobile controller. (2) Slices the mobile packet network according to services offered by the mobile operators (3G, 4G).
Optical FlowVisor [57]	Openflow, GRE	centralized	✓	✓	×	✓	(1) Allow mobile operators to create Virtual Optical Networks (VONs) on the physical infrastructure. (2) Each isolated VON is a collection of virtual optical switches connected via a virtual optical link. (3) The services offered by virtual links depends on the wavelength channels in a single optical fiber.
Hyperflex [57]	Openflow, REST APIs	distributed	✓	✓	✓	✓	(1) Prevents network to degrade its performance by over-utilizing the hypervisor layer. (2) Allocate each tenant with a particular amount of CPU resource in advance.
AutoSlice [57]	Openflow, REST APIs	distributed	✓	✓	✓	✓	(1) Share the workloads by slicing the hardware infrastructure into multiple physical domains. (2) Assign multiple proxies controllers and set one proxy on each physical domain.
AutoVFlow [57]	NetConf, Openflow, REST APIs	distributed	✓	✓	✓	✓	(1) Perform slicing of the infrastructure into multiple domains and acts as a container for each domain. (2) The distributed administrator in multiple domains is configured by a centralized hypervisor. (3) A virtual identifiers/MAC address is used as an identifier of the flow packets and is unique for every domain.

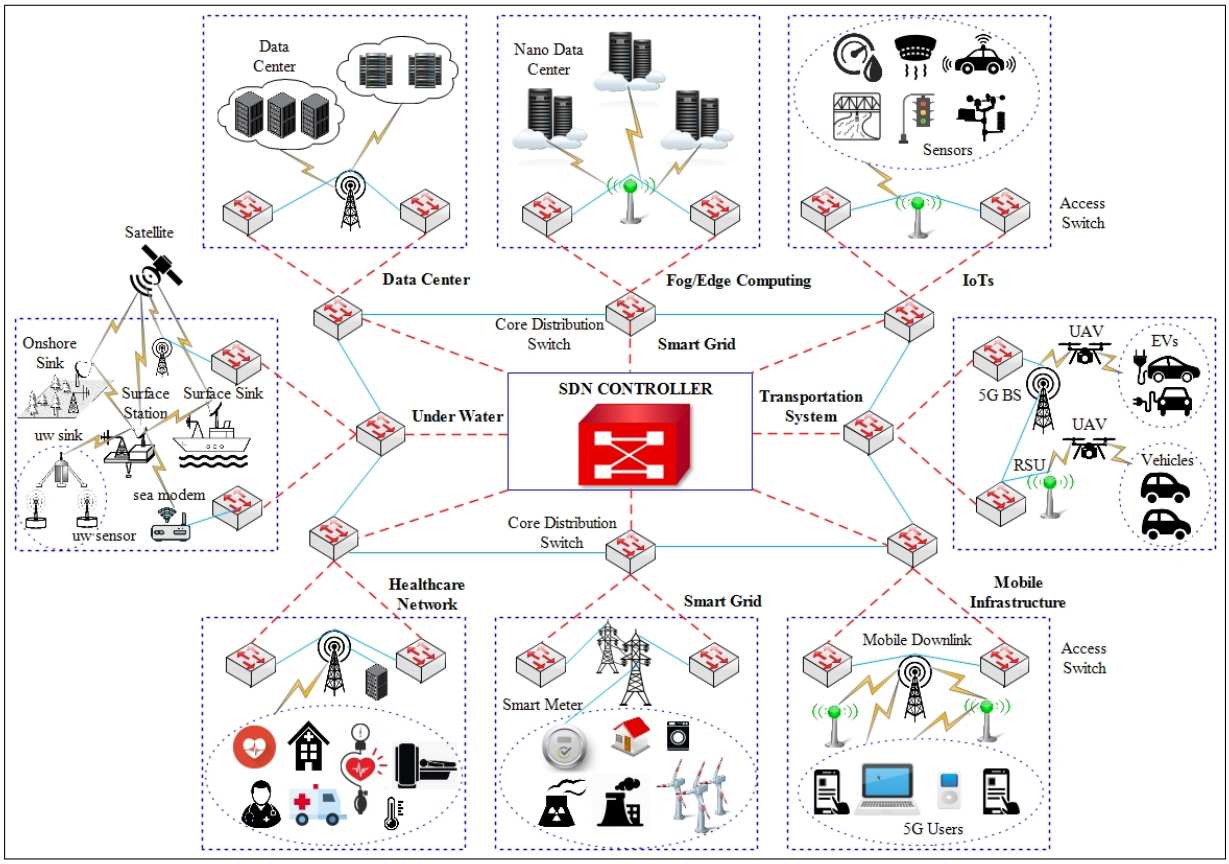

Figure 2.9: The scenario of deploying SDN controller in real-world applications.

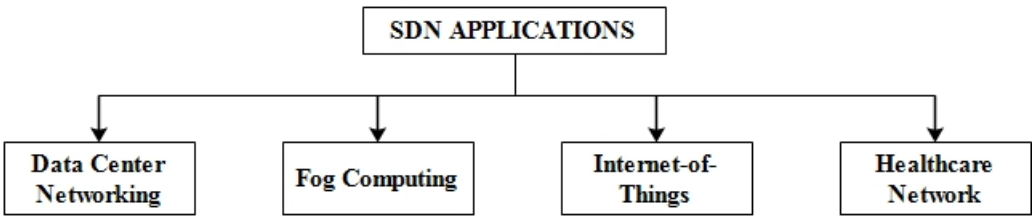

Figure 2.10: Deployment of SDN in various applications.

resources is called VM placement. Zeng *et al.* [85] examined the issue of energy management and the phrase the issue using an integer linear programming (ILP). The author presented a Multi-Path Routing (MPR) scheduling and computation-efficient heuristic algorithm to save the energy consumed by the activated OF-switches. In [86], the author designed a VM migration algorithm with a dynamic overbooking strategy instead of an existing fixed booking strategy to handle the dynamic workloads. Amokrane *et al.* [87] formulates the issue of online-flow routing as an ILP problem in order to save the overall energy consumption at DC. The author solves the ILP problem using Ant Colony based on the Online Flow-Based Energy-efficient Routing (AC-OFER) approach. The experimental result shows that the AC-OFER approach reduced the overall energy cost up to 35%, 44%, and 49% compared to minimal residual capacity (MRC), shortest path routing (SPR), and LB approach. There is a large number of proposals exist in

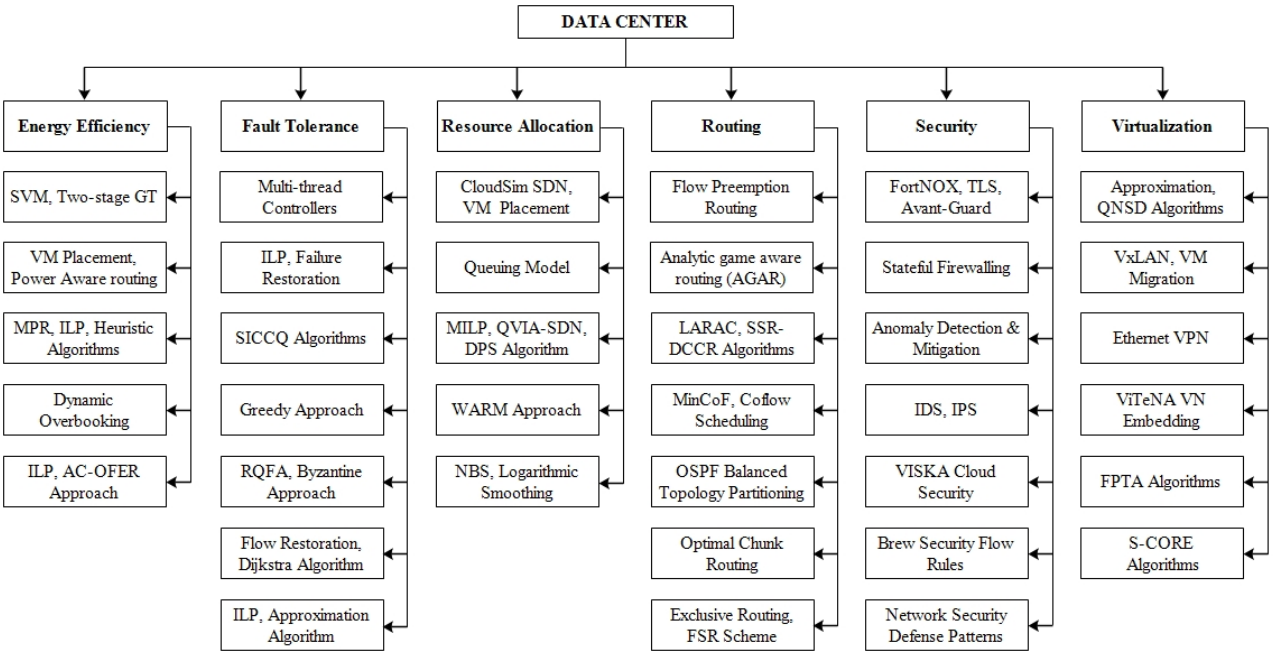

Figure 2.11: A Taxonomy of SDN in data center networking.

the literature which addresses the aforementioned issues. For example, Li *et al.* [26] proposed an Exclusive Routing (EXR) algorithm to avoid flow scheduling by overlapping links based on the time dimensions and compared the EXR scheme with existing Fair Sharing Routing (FSR) scheme. The EXR scheme executes the data transmission of flow schedule after completion of ongoing flow while the FSR scheme performs flow scheduling through overlapping links. The results proved that the EXR outperforms the FSR scheme both for the flow completion time and switches energy savings. Also, Zhang *et al.* [88] proposed a Redundancy Elimination Flow Preemption Routing (RE-FPR) scheme having an improvement in EXR approach which preserves the flow priority during the peak time. In [89], the authors proposed a heuristic-based path selector scheme to achieve energy-efficiency in SDN switches. Francois *et al.* [90] proposed a time-driven link sleeping optimization scheme for energy saving. The performance analysis shows that the proposed scheme minimized energy consumption by 28.3%.

Heller *et al.* [91] proposed a prototype model named as *ElasticTree* for power optimization in DC. The proposed approach is based on a distributed optimizer for flow consolidation based on identifying the traffic patterns. Likewise, Wang *et al.* [92] proposed a Correlation-Aware Power Optimization (CARPO) algorithm for flow consolidation to minimize the energy consumption of DC. The CARPO algorithm first identifies the correlation among the traffic patterns such that the different flows can be consolidated through a small set of links and nodes. Secondly, the CARPO approach merges the correlation among the flows with the link rate adaptation approach in order to minimize the power consumption. The shortcomings faced by both the existing state-of-the-art approaches is that the computation time increases exponentially as the DC scalability increases. Javed *et al.* [93] resolved the power optimization issue using a Thermal Energy-Aware Routing Algorithm (TAEO). The proposed TAEO scheme addresses

two issues: (i) identifies and avoids hotspot nodes using Specific Absorption Rate (SAR), and (ii) improves the stability period and network lifetime. Likewise, Awan *et al.* [94] worked on the EAR issue for emergency data traffic flowing from delay-sensitive applications. The proposed approach is named as Priority-based Congestion-avoidance routing (PCAR) scheme. The PCAR scheme classifies the incoming flow into emergency and normal data traffic. For emergency data traffic, PCAR approach is used for the minimum delay and enhance the Packet Delivery Ratio (PDR), while for the normal data traffic, data aggregation and filtration method is used to reduce the traffic load and minimize the energy consumption. In [95], the authors manage the orchestration and flow consolidation of the underlying network resources. The proposed flow consolidation algorithm computes a trade-off between power consumption and hardware processing modules to save 80% of the energy consumption. Lian et al. [96] proposed an energy conservation scheme in wireless sensor networks to improve the data capacity.

Maaloul *et al.* [97] proposed the decision-making EAR solution to maintain a balance between power saving and resource scheduling. It uses the First-Fit Heuristic Algorithm (FFHA) in OpenFlow switches by considering EAR with rule space constraints. Kannan *et al.* [98] designed a compact Ternary Content Addressable Memory (TCAM) for energy-saving without switching-off the network devices. The proposed approach by the authors uses small tags to store and route the data packets, which results in an energy saving of approximately 80%. Likewise, Giroire *et al.* [99] addressed the issue of capacity constraint with respect to power-hungry flow table TCAM memory. The proposed scheme is a heuristic approach named, as an Integer Linear Program-based Energy-Aware Routing (ILP-EAR) that computes a feasible routing solution based on a greedy approach. The optimization algorithm uses the smart space allocation method for power optimization in OpenFlow switches and routers. In [100], Huang and Youn address the capacity constraint problem of the TCAM memory by installing a proactive approach instead of a reactive approach using a hidden Markov model. Similarly, in [101], the authors proposed different sleep scheduling schemes in wired networks to have green routing. Kurdi [102] proposed an Ant Colony Optimization (ACO) technique to construct the shortest path routing based on the pheromone level of ants. Also, Miao *et al.* [103] designed the preemptive scheduling scheme to minimize the network energy consumption cost.

From the existing schemes, it has been observed that fast flow forwarding plays a significant role in green routing. For example, Kaur *et al.* [104] proposed a heuristic approach for load balancing and energy minimization in Software-Defined Internet of Vehicles (SD-IoV) paradigm. Zheng *et al.* [105] proposed a Distributed Flow Consolidation Scheme named as DISCO algorithms for power optimization in DC. The DISCO algorithm improves the scalability of the DC that performs correlation analysis of each flow separately for flow consolidation. Similarly, Kaur *et al.* [106] designed a flow consolidated framework using efficient resource utilization for energy-minimization in SDN-DC. The proposed scheme uses the First Fit Decreasing (FFD) technique in order to design a consolidated bandwidth allocation and VM deployment model that saves almost 27.9% energy consumption as compared to the existing schemes.

Table 2.5: Summary of the existing state-of-the art schemes.

Schemes	①	②	③	④	⑤	⑥
EXR	✓	✓	✗	✗	✗	$\frac{m_1}{C}+\frac{m_2}{C}+\frac{m_2+m_3}{C}$
FSR	✓	✓	✗	✗	✗	$\frac{m_1}{C}+\frac{2m_2}{C}+\frac{m_2+m_3}{C}$
RE-FPR	✓	✓	✗	✗	✗	$\frac{0.5(m_1+m_2)}{C}+\frac{0.5m_2}{C}$
ElasticTree	✓	✗	✗	✓	✗	1000 servers: 7.3 min
CARPO	✓	✗	✗	✓	✓	1000 servers: 8.6 min
TAEO	✓	✗	✗	✓	✗	-
PCAR	✓	✓	✗	✓	✗	-
ILP-EAR	✓	✗	✗	✗	✗	-
DISCO	✓	✓	✗	✓	✗	320 switches: 21.6 min

①: energy-efficiency; ②: packet scheduling; ③: flow re-routing cost; ④: end-to-end delay; ⑤: link adaptation rate; ⑥: flow completion time; m: messages; C: bandwidth capacity.

Motivated from the existing schemes, it is mandatory to design an energy-efficient flow forwading scheme, which operates on the forwarding nodes (switches). The comparative research gaps found in the existing schemes are summarized in Table 2.5. The existing schemes achieved the energy-efficiency on switches by using the optimized algorithms for flow routing. None of the existing proposals considered the packet scheduling and re-routing on OpenFlow switches. The *packet-in* messages arrived in the switch queue buffer where they had to wait for a long duration, till the *packet-out* messages got scheduled. However, the power consumption on active links of the switches increases, if the waiting time of scheduling the *packet-in* messages increases. Moreover, the re-routing cost also increases the additional energy consumption in the case; any node is deleted or added in the network. Hence, an optimal re-routing scheme is required so that during the addition or deletion of a new node, there is no need to change the entire routing scheme.

2.5.2 Fault Tolerance in DCN

The failure recovery is an another issue of centralized controller connected with geo distributed DCs. In [107], the author presented a congestion control framework and solves the congestion events in DCs using SDN-based Incast Congestion Control via Queue-based monitoring (SICCQ) algorithm. The SICCQ algorithm implements two major events i.e. packet arrivals to the switch ports, and incast detection timer expiry. The incast congestion happens when many-to-one traffic pattern is followed. It means multiple synchronized requests is send to a single DCN. In addition, a SICCQ hypervisor algorithm is presented to handle the incoming packets of different types i.e. incast ON, incast OFF, and TCP Acknowledgment (ACK). Liu *et al.* [108] address an important issue related to Controller Placement Problem (CPP) in terms of network reliability. The CPP issue is resolved by two algorithms named as clustering based global optimization algorithm and controller placement algorithm based on greedy approach.

The K-mean clustering algorithm splits the complete domain into multiple sub-domains and each sub-domain consists of multiple OF-switches controlled by a single controller. The greedy approach finds the optimal solution with the case of multi-paths to compute the links reliability between the node and the controller in each cluster. In [109], the author considered the issue of CPP as a Controller Assignment in Fault-Tolerant SDN (CAFTS). The solution of CAFTS is resolved by presenting a byzantine approach using a cost-efficient Requirement First Assignment (RQFA) algorithm. The RQFA algorithm maintain a set of controllers as candidate for assignment. Initially, the candidates is assigned with minimum residual capacity to accommodate the switches and check whether the performance is satisfactory. In case, the performance degrades, a new set of candidate is assigned. Astaneh *et al.* [110] a dynamic approach named as optimal Dijkstra-Like Path Cost algorithm is used. The algorithm solves the issue of flow restoration and the result shows that the proposed approach minimize the number of operations upto 50% with minimum links failure. Xie *et al.* [111] address the fault-tolerant issue in multiple controllers in DCN. The author designed an optimal solution by choosing an approximation algorithms to minimize the overall network overhead on controllers. The proposed technique supports failure recovery on distributed controllers and reduces the communication overhead generated from state synchronization.

To solve the above-discussed issues of CPP, several proposals exist in the literature. For example in [112], a framework named as Pareto Optimal COntroller (POCO) has been presented by the authors for the CPP. The POCO framework covers four aspects of CPP (i) inter-controller latency (ii) controller failure (iii) network disruption (controller-less nodes) and (iv) load imbalance. Also, Lange *et al.* [113] designed a Pareto Optimal solution based on the Simulated Annealing (POCO-SA). Similarly, in [114], a POCO Multi-Objective Ant Lion Optimization (POCO-MOALO) algorithm was designed as a Pareto-Optimal solution to select the optimal position for the controller placement. The proposed algorithm uses the concept of load capacity factor and propagation latency simulated on POCO toolset for the Internet2 Open Science, Scholarship and Services Exchange (OS3E) topology. Ksentini *et al.* [115] proposed a bargaining game for CPP. Similarly, in [116], the authors proposed the distributed approach based on non-zero sum game theory for CPP. The proposed approach initially computes the payoff on each controller and then takes the decision about insertion, deletion or load migration among the controllers by comparing the payoff with the adjacent controllers. Killi *et al.* [117] adopted a Capacitated Next Controller Placement (CNCP) scheme to minimize the average latency between the switch to it's closest reference controller and between two neighbouring reference controllers with sufficient residual capacity. It ensures the sustainability of the network in case of any node or link failure in the network. The term *resilience* means deploying the extra number of backup controllers for fault tolerance in case, the primary active controller fails.

Tanha *et al.* [118] used the approximation algorithm for CPP to prolong the network reliability and to mitigate the issues of any node or link failure in the network. Moreover, in [119], the authors have extended their research work to address the issues of latency and a capacity-

aware guarantee of controllers. The proposed scheme uses a clique-based technique in the network topology to ensure the recovery of both the node and the link failure. Zhang *et al.* [120] formulated the CPP by using multi-objective function and proposed an optimization approach named as adaptive bacterial foraging to have the network reliability. In [121], authors proposed *Compare* and *Drop* algorithms to compute the average and worst-case latency. The experimental results have been tested on Internet topology Zoo to evaluate the impact on the number of dropped controllers.

In [123], the authors designed a primary-backup controller scheme to ensure fault tolerance. The results illustrated that the proposed scheme minimizes the number of controllers up to 50%. Moreover, the controller assignment problem is addressed by using a genetic algorithm and analytic hierarchy approach in [124]. Similarly, Li *et al.* [125] proposed a quality-of-service (QoS) aware resource scheduling scheme by integrating the SDN approach and instance placement optimization algorithm for virtual networks. The proposed scheme designed a Pareto-optimal solution based on non-cooperative game theory to improve the service-level agreement. In addition to the aforementioned issues in CPP, data inconsistency is still one of the crucial issues using the multiple controllers. Data consistency means synchronization of the messages based on the ordering of events among multiple controllers. In [126], authors proposed a consensus mechanism for CPP to perform coordination of events among different controllers, each having its own database. The proposed scheme uses a leader-follower approach on the control plane to maintain global data consistency.

Akyildiz *et al.* [54] presented a survey on Traffic Engineering (TE) approach in SDN. The author focused on four parameters such as- flow scheduling, fault tolerance, topology update for new or existing policies, and network traffic analysis. The concept of deploying distributed controllers, multi-threaded controllers, and generalized controller is introduced for failure recovery. In [127], the author formulates an Integer Linear Programming (ILP) problem for virtual network resource allocation issue. The Openflow-based failure restoration algorithm is designed to dynamically divert the network traffic for link failure restoration and is compared with the existing pro-active path computation algorithm. The result is implemented in Mininet simulator using POX controller to link 20 end-points with 44 bi-directional network links. The performance of the proposed approach is capable to restore around 60% of damaged links while the pro-active algorithm restores only 32.6% of damaged links. Motivated from the above proposals, it is an essential requirement to design an efficient fault-tolerance SDN framework to cope up the issues of scalability, resilience, and consistency on the control plane. The existing proposals worked on the issues of CPP in large-scale networks. However, most of the existing proposals resulted in finding the least number of controllers required and their placement with minimum focus on maximizing the fault-tolerance on the control plane. In addition, none of the authors has addressed the issue of synchronization of messages among the distributed controllers with scalability and resilience.

2.5.3 Resource Allocation in DCN

The resource management in SDN architecture is another issue faced by the DCN. Generally, resource allocation consists of CPU, memory, storage, bandwidth, and channel allocation. Son *et al.* [128] solves the resource management issue using a CloudSimSDN simulator. The author compares the performance of Mininet simulator and CloudSimSDN simulator for resource allocation in DCs. The Mininet simulator is suitable to configure and test SDN controller resources but on the other hand, the cloud-unique characteristics such as- workload assignment schedulers, VM placement, VM migration rules are difficult to simulate on Mininet emulator. The CloudSimSDN simulator is a Graphical User Interface (GUI) tool and allows the simulation of all cloud features. Li *et al.* [129] address the issue of flow-level scheduling in SDN environment. The author implement a queue scheduling scheme to provide QoS guarantees for flow scheduling on SDN switches. The numerical results shows that the scheme minimizes the overall latency by upto 28% and the average throughput is increased to 90.17% at the same time when the interface has adequate feasible bandwidth.

In [130], the author formulate the virtual allocation of resources in DCs as a Mixed Integer Programming (MIP) problem. The virtual allocation specifies the necessary requirement to configure the virtual resources. For example, the configuration of virtual switches requires the number of flow table entries, memory, and processing power while the configuration of VMs requires memory, storage, and virtual CPUs. The author presented a QoS-aware Virtual Infrastructure Allocation on SDN-data centers (QVIA-SDN) scheme to identify the number of physical resources available for hosting virtual infrastructure. In addition, the author implement a Deterministic Path Selection (DPS) algorithm to minimize the number of switches by mapping flow entries and checks whether a physical path exists to hosts virtual links. In [131], the author proposed a scheme named as Workload-Aware Revenue Maximization (WARM) to increase the net profit of service providers. The simulation results proved that the WARM approach performs efficiently utilization of VMs to compute the best routing path for all applications. The proposed scheme results to decrease the Round-Trip Time (RTT) of jobs as compared to round robin algorithm and open shortest path first approach. In [132], the author presented a Nash Bargaining Solution (NBS) that improves the network bandwidth to providers and saves the delay time. The author considers the resource allocation issue as an optimization problem and implement a request allocation scheme based on logarithmic smoothing to solve the optimization problem.

2.5.4 Routing in DCN

The energy minimization problem in DCs can be resolved by applying an efficient routing scheme. Zhang *et al.* [88] designed a routing scheme named as Flow Preemption Routing with Redundancy Elimination (FPR-RE) to manage flow scheduling and eliminates the data redundancy in OF-routers. The scheme inspects the flow execution time and energy consumption

of traffic flow. In order to choose the optimal routing path for cloud gaming network, Amiri *et al.* [133] designed a bi-objective optimization scheme. In addition, the author designed an Analytic Game Aware Routing (AGAR) approach to deliver high-quality experience for online players by reducing the delay and improves the bandwidth utilization. The experimental results shows that the proposed approach minimized the End-to-End (E2E) latency by upto 19%, 17% and 14% compared to Dijkstra, Equal Cost Multi-Path routing (ECMP), and the Hedera routing algorithm. Guck *et al.* [68] presented a survey on routing algorithms for QoS in SDN. The author identify four dimensions (4D) suitable for QoS routing approach. The first dimension represents the topology category, while the second and third dimension represents the scaling of the topology into horizontal and vertical axis i.e. (X-axis, Y-axis), finally the fourth dimension represents delay constraints. The author analyzed two best algorithms such as- Lagrange Relaxation based Aggregated Cost (LARAC) and Search Space Reduction Delay-Cost-Constrained Routing (SSR-DCCR) to fit all the 4D space. Chiu *et al.* [134] reduces the communication time in DCN by designing a routing schemes namely, coflow scheduling and Minimal Coflow scheduling and routing (MinCoF). The implementation results of the proposed scheme is compared with the OF-scheduling and routing algorithm and achieves better performance by reducing the average execution time around 12.94%.

Caria *et al.* [135] presented an approach named as Open Shortest Path First (OSPF) balanced topology partitioning to control prioritized data traffic. The designed scheme performs the partitioning of network topology into multiple sub-domains and shows that the network control capabilities in each sub-domains performs better routing compared to deploy complete single network topology. Wu *et al.* [136] addressed the issue of massive data migration in inter-DCs to achieve maximum available bandwidth utilization. The author proposed a bandwidth-reserving algorithm to reserve the channel links and bandwidth for sending blocks of newly arrived tasks. In [137], the problem of high power consumption by DCs is solved by adopting an energy-aware flow scheduling using an exclusive routing scheme in time dimension for each flow. The result computes that the average flow completion time decreases if the highest priority is selected to minor flows comparable to Fair-Sharing Routing (FSR).

2.5.5 Security in DCN

Security is an another issue faced in the DCN environment. It becomes crucial to build consistent security policies and analyze the traffic patterns in SDN and NFV platform. In [61], the authors performed a comprehensive survey on the security challenges across SDN-based cloud DC environment. The survey mainly focused on the Distributed Denial-of-Service (DDoS) vulnerabilities occurred by an adversary on each SDN plane. Additionally, the safeguard mechanisms is also discussed in order to mitigate against the DDoS attacks. Yan *et al.* [138] focused on the DDoS attacks may launch by an adversary on the application, control and the infrastructure plane. An adversary can launch the DDoS attack on the following methods: attacking

network applications, northbound API, attacking controller, westbound and eastbound API, attacks on switches, and southbound API. Lorenz *et al.* [139] designed a fine grained security policies to defend from security threats by implementing stateful firewalling with SDN and NFV integrated enabled architecture. The three approaches are used for the placement patterns of software-based stateful firewall. The first approach is controller-centric approach based on placing firewall in controller software. The second approach is VNF-centric approach placing virtualized firewall deployed on the cloud infrastructure and connection setup at the data plane. The third approach is the hybrid SDN/NFV approach to implement both controller-centric at the controller for intensive connections and VNF centric at the data plane for connections setup. Giotis *et al.* [140] identified malicious events (typically DDoS attacks) in hardware and software based OF-devices. In [141], the author designed an anomaly detection technique using an artificial neural network and NSL-KDD dataset. The proposed scheme analyzed all the collected flow statistics uses time window of 30 seconds to monitor the abnormal activities in the traffic patterns. Furthermore, once the malicious events and the entropy values of the IP address and port address (layer 1-4 attacks) is detected, the author implemented an anomaly mitigation technique using OF to drop the corresponding malicious traffic. In [142], the author introduced an Intrusion Detection System (IDS) and Intrusion Prevention System (IPS) in order to mitigate from the malicious attacks. The suspicious activity is identified by deploying IDS and IPS at all the gateway nodes so that the SDN controller may block such malicious traffic from the control flow. In [143], the author presented a cloud security service named as VISKA to identify malicious switching nodes at the forwarding nodes. The proposed service mechanism partitions the data plane into virtual networks using virtualization rather than detecting the malicious activity on the entire physical network. The dynamically isolated partition creates virtualized network views of the data plane and easily localizes the abnormal forwarding behaviors at the switch level.

In literature, the researchers discussed the proof-of-concept prototype of a conventional layer-3 firewall in SDN but none of the researcher addresses the issue of flow rule conflicts at any layers. Pisharody *et al.* [144] addressed the problem of flow rule conflicts at the control and data layer in distributed SDN controller like ODL controller. The author designed a brew security policy mechanism, a novel visualization approach to resolve conflicts in flow rules. Initially, the proposed scheme applies global flow rule prioritization method for the distributed SDN controllers. Next, the cross-layer checks the flow rule conflicts by extending the conventional firewall policies of conflict classification. Finally, the author implemented a visualization method to monitor the conflicts graphically by a network administrator. In [145], the author mainly focus on building an adequate security solution in Service Functions Chaining (SFC). SFC is a technique used to guide data traffic flows using network functions in SDN and NFV architecture. The author presented a defense mechanism called as Network Security Defense Patterns (NSDP) to provide optimal placement of virtual security functions while deploying SFC in SDN. The security functions are transparently added into the traffic flows to perform

security checks at each level in different applications types.

2.5.6 Virtualization in DCN

Some of the existing proposals related to virtualization in DCN are summarized as follows: For example, Feng *et al.* [146] identify three important issues faced in network virtualization i.e. (i) optimal placement of virtual network functions, (ii) flow consideration in limited available resources, and (iii) the allocation of network resources. The author resolve all these three issues using the fast approximation algorithm named as Queue-length based Service Distribution (QNSD) algorithm to reduce the Operational Cost (OPEX). Zhao *et al.* [147] discuss the limitations of VLAN to build large multi-tenants DCN. The VLAN provides inefficient use of available network links, limited scalability (uses 12-bit segment ID that supports maximum 4094 tenants ID) to address layer 2 segments, and inefficient placement of multi-tenants segments. The author adopted a SDN based Virtual Extensible Local Area Network (VxLAN) architecture [64]. VxLAN uses 24-bit segment ID that supports more than 16 million tenants has greater extensibility to address the issues of VLAN. In [148], the author enhanced the SDN functionalities for Ethernet Virtual Private Network (EVPN). The author designed a SDN based DCs framework to automate EVPN deployment and dynamic routing policy for EVPN.

In [149], the author identify the performance degradation problem of isolated virtual network in a DCN environment. The author designed a scalable solution for DCs called as ViTeNA approach based on virtual network embedding algorithm to achieve high bandwidth guarantee to tenants. The algorithm monitors the global network outlook to allocate virtual nodes. The proposed solution consolidates the virtual networks on limited physical resources and enforce incremental consolidation for every request instead of doing a periodic consolidation. Wang *et al.* [69] formulate two problems i.e. (a) VM migration time when multiple VMs are migrated, and (b) bandwidth requirement for each migration. The author considers the problem as a MIP problem and solves the problem by using a Fully Polynomial Time Approximation (FPTA) algorithm. The designed approach is compared with the existing primary programming problem and the state-of-the-art algorithms. The result shows that the designed scheme decrease the service downtime around 20% and the overall VM migration time is decreased around 40%.

In [150], the author accomplish a framework for live VM management inorder to increase the throughput and reduces the overall network-wide communication cost. The proposed scheme is S-CORE which is a combination of three orchestration algorithms namely; Round-Robin, Best-Fit and Lookahead algorithm. The scheme uses two DC network topologies i.e., canonical tree and fat tree topology and is implemented in NS-3 simulator. The results shows that the aggregate throughput is increased to six times based on 12 live VM migrations and by migrating 50% VMs, the VM-to-VM communication cost is reduced upto 70%.

2.5.7 Load Balancing and Anomaly Detection in SDN

There exists various research proposals in the literature to resolve the problem of load imbalance among distributed controllers. For example, Aujla *et al.* [4] proposed a data offloading scheme using a Stackelberg game theory for load balancing. The proposed scheme chooses an optimal network where the selected load can be migrated. In [10], the authors proposed a game-switch migration algorithm for load balancing in order to maximize resource utilization. In an another work, Kuo *et al.* [12] proposed a segmented wildcard forwarding approach to perform load balancing which reduces the flow table modification cost by 70%. To improve the scalability in a large-scale network, Zhang *et al.* [13] designed an online load balancing mechanism for the multiple controllers to minimize the average response time on controllers which works in two phases, i.e., load detection and load execution. On the other hand, an elastic distributed controllers prototype model named as *ElastiCon* is proposed by Dixit *et al.* [151] in an OpenFlow-enabled SDN model. The *ElastiCon* model performs load balancing using switch migration protocol which handles three tasks, (i) load detection, (ii) adaptation decision, and (iii) adaptation action. Similarly, Cheng *et al.* [152] designed a Distributed Hopping Algorithm (DHA) based on the Markov Chain process in order to address the issues of load imbalance on multiple controllers. The proposed scheme computes an optimal solution for switch migration using a logarithmic-sum-exponential function in which each controller performs the tasks independently. Shang *et al.* [154] designed a load balancing algorithm to compute the load status on each controller using flow-requests deviation mean. It is used to migrate the load to the idle controller from the overloaded controllers by ensuring least flow-requests deviation and propagation delay. Similarly, Trestian *et al.* [155] proposed an OpenFlow-based the dynamic load balancing scheme to compute an optimal path configuration using multiplicative weighted routing.

Some of the existing proposals also introduced large flows on the network due to flooding attacks which may lead to load imbalance. Moreover, the large flows statistics on the controllers may generate overhead on the control plane. For example, Giotis *et al.* [140] proposed an entropy-based Intrusion Detection System (IDS) scheme to resolve the issue of the DDoS attacks. The IDS scheme classifies malicious flow by analyzing entropy changes. An adversary initially hacks the flow rules installed by the controller and then performs modification in the flow tables. It results in flow mismatch which in turn incurs a long delay. Phan *et al.* [156] proposed a scheme named as OpenFlowSIA in order to mitigate DDoS attacks by using the machine learning algorithm. Two features of the traffic namely as '*number of packets*' and '*flow time-duration*' are extracted in the proposed scheme. Then, the flow behavior is categorized based on Support Vector Machine (SVM) classifier and the output is returned to the Idle-timeout adjustment algorithm in case of normal traffic behavior, otherwise, the suspicious traffic is handled by using the policy enforcement module. In an another work, Carvalho *et al.* [157] identified the effects of anomalies during worm and DDoS attacks. Likewise, Ha

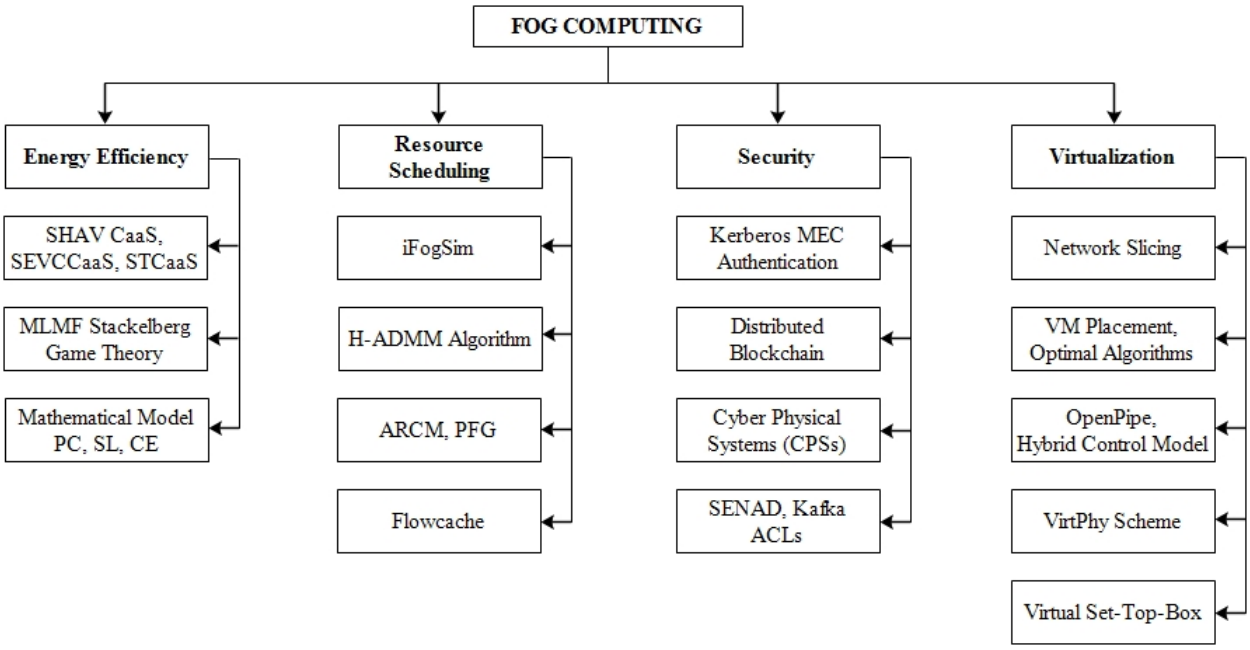

Figure 2.12: A Taxonomy of SDN in fog computing.

et al. [158] examined the suspicious traffic in a large-scale network and introduced a traffic sampling rate scheme to compute a suitable sampling rate for forwarding devices. Similarly, Amaral *et al.* [153] proposed an IDS algorithm using Tsallis entropy.

From the analysis of the above proposals, it was concluded that none of the aforementioned solutions provided a dynamic load balancing scheme that works in tandem with flow-based anomaly detection and mitigation scheme for SDN. As the Ternary Content Addressable Memory (TCAM) of the OpenFlow switch has limited capacity to store the number of flow entries, so a deep flow inspection is required. The proposed scheme performs mitigation from the DDoS attacks. The attacker launches DDoS flooding attacks by bombarding of unwanted requests from Internet Control Message Protocol (ICMP) messages and TCP Synchronization (SYN) messages on OpenFlow switches by establishing fake handshaking process between the source and destination machine. It creates the issues of bandwidth depletion and resource exhaustion on the control plane which may prevent the network controller for installing thousands of flow tables on the OpenFlow switches when the victim network device is under the control of DDoS attack. Motivated from these facts, it is essential to design a load balancing scheme so as to reduce the load on control plane by blacklisting the malicious requests.

The summarized overview of the performance analysis of the existing techniques on data centers networking in SDN architecture is shown in Table 2.6-2.7.

2.6 Fog/Edge Computing

Fog Computing (FC) is a new paradigms of SDN and NFV technologies. The fog nodes have the innovative added-value capabilities to reduce the E2E latency by migrating the cloud

Table 2.6: Existing work on data center networking (DCN) using SDN architecture.

Authors	Techniques	Controller	Topology	Setup Description	Coding/ Simulator	Issues Considered						Parameters Improved			
						1	2	3	4	5	6	7	8	9	10
[28]	SVM, game theory	centralized	Tree	25 servers; EVs- 16 kWh and 20 kWh saves upto 40% energy	Matlab	✓	×	✓	×	×	×	✓	✓	×	×
[84]	Power aware routing	distributed	Fat-Tree, BCube	16 servers; 20 VMs; RP_PAR (2.9kW) saves 12% energy	-	✓	×	✓	×	×	✓	×	×	✓	×
[85]	Minimum switch activation routing	centralized	Fat-Tree	10 pods; 12,500 flows takes 11sec saves around 48% energy	-	✓	×	×	✓	×	×	×	✓	×	×
[86]	Overbooking dynamic resources	centralized	Fat-Tree	16 hosts; 4 edge; 4 aggregate switches saves 30.29% energy	CloudSim	✓	×	✓	×	×	✓	✓	×	✓	×
[87]	AC-OFER approach, Ant Colony	centralized	Tree	100 APs; 27 switches; 2 router; 50 requests/hr saves 7% energy	Java	✓	×	✓	✓	×	×	✓	✓	×	×
[107]	SICCQ approach, queue monitoring	centralized	small Fat-Tree	1 core; 2 aggregate switches; 48 servers/racks takes 200 ms	NS2	×	✓	×	×	×	✓	×	✓	✓	×
[108]	Greedy approach, K-means clustering	distributed	Internet Zoo	Multi-Path: 6 controllers; 37 nodes gives 0.2 failure rate	-	×	✓	×	×	×	×	×	✓	×	×
[109]	Requirement first assignment	distributed	Fat-Tree	100 servers; 20 switches and 2 controllers takes 100 ms	Mininet	×	✓	✓	×	✓	×	✓	✓	✓	×
[111]	Approximation algorithms	distributed	Jellyfish	16000 servers; 1000 switches; each controller capacity: 10-50 reduced the time complexity from $O(C.N^2)$ to $O(N \times \gamma)$	-	×	✓	×	×	×	×	✓	✓	×	×
[128]	VM Placement	centralized	Tree	548,341 hosts; 19,999 switches saves around 23% energy	CloudSim	✓	×	✓	×	×	×	×	✓	✓	×
[129]	Queuing model	centralized (Floodlight)	Tree	3 application flows: (a) voice, (b) video, (c) generic flow. Reduced delay by 99.9%, 90.6%, 27.84%. Throughput increased by 90.1%, 76.06%, 18.5%.	Mininet	×	×	✓	✓	×	×	✓	✓	✓	✓
[131]	WARM approach	centralized	Fat-Tree	4 pod; 16 VMs; 20 switches; bandwidth between VM and each switch is 1 Gbps with execution time 2.03 $\times 10^{-5} s$ increases revenue by $50.52	Mininet	×	×	✓	✓	×	✓	×	✓	✓	×

Table 2.7: Existing work on data center networking (DCN) using SDN architecture (Contd...)

Authors	Techniques	Controller	Topology	Setup Description	Coding/ Simulator	Issues Considered						Parameters Improved			
						1	2	3	4	5	6	7	8	9	10
[132]	Logarithmic smoothing, Queuing model	centralized	Tree	5 DCs; 100 application instances; 500 end-users reduced user delay around 300 ms	-	×	×	✓	×	×	×	✓	✓	✓	×
[88]	RE-FPR algorithm	centralized (POX)	Fat-Tree	$k(k+1)$ switches; k^2 hosts; 10 Mbps each link bandwidth saves energy upto 3-5%	-	✓	×	×	✓	×	×	✓	✓	✓	×
[133]	AGAR approach	centralized (POX)	Fat-Tree	20 vswitches; 600 links; 1000 each vswitches table capacity; 20 Mbps each link bandwidth reduced delay by 9.5%	Mininet	×	×	×	✓	×	×	✓	✓	✓	×
[134]	MinCof, Coflow scheduling	centralized	spine-leaf Fat-Tree	4 spine switches; 4 leaf switches; 16 hosts; 100 Mbps each link bandwidth reduces the completion time by 12.94%	-	×	×	✓	✓	×	×	✓	✓	✓	✓
[135]	OSPF balanced topology partitioning	centralized	Cost266, JANOS-US-CA topology	39 node; 61 links; 1432 flows- the average traffic loss is 0.82% and congestion links is 0.26% increases utilization upto 50%	-	×	✓	✓	✓	×	×	✓	✓	×	×
[136]	Optimal Chunk Routing Algorithms	centralized	Tree	10 DCs; 500 average chunks; 100 MB each chunk size reduced the delay around 10s	Java	×	×	×	✓	×	×	✓	✓	✓	×
[137]	Exclusive Routing	centralized	Fat-Tree, Blocking Fat-Tree, BCube	3456 servers; 512 MB flow size; 1 Gbps capacity; 1000/s average flow rate saves 35.02% and 26.56% energy	-	✓	×	✓	✓	×	×	✓	✓	✓	✓
[140]	Anomaly detection & mitigation	centralized (NOX)	tree	Exp.1,2,3: (50, 100, 500) Mbps average traffic rate; (1/64, 1/64, 1/256) sampling rate; (50-200, 200-500, 1000-2500) attack rate pkts/s	Python	×	×	×	×	✓	×	×	✓	×	×
[144]	Brew security flow rules	distributed (ODL)	Reingold-Tilford tree	10,000-1,00,000 flow rules; 1 GB flow size reduces latency by 5%	Mininet, Java	×	×	×	×	✓	×	✓	✓	×	×
[145]	Network security defense patterns	distributed (ODL)	Fat-Tree	k= 128: 540800 nodes; 1572864 links, execution time is 15.9 s and is 2.6 times faster	-	×	×	✓	✓	✓	✓	✓	×	✓	✓
[69]	FPTA algorithm	centralized	Tree	12 DCs; 19 links; 100 VMs reduces latency by 50 %	Matlab	×	×	✓	✓	×	✓	✓	✓	✓	×
[150]	S-CORE algorithms	centralized	Canonical, Fat-Tree	12 live VM migrations; 50% VM migration- throughput increases by 6 times and cost reduced upto 70%	Mininet	×	×	✓	✓	×	✓	✓	×	✓	✓

Note- 1: Energy Efficiency, 2: Fault Tolerance, 3: Resource Allocation, 4: Routing, 5: Security, 6: Virtualization, 7: Cost, 8: Latency, 9: Bandwidth, 10: Throughput
Notations- ✓: considered, and ×: not-considered

services towards the edge nodes. The edge nodes is closer to the end users deliver innovative services without incurring any delay. The taxonomy of Software Defined-Fog Computing (SD-FC) network is shown in Fig. 2.12. Baktir *et al.* [65] presented a survey on Software Defined-Edge Computing (SD-EC). The authors highlighted the advantages of technologies such as- fog computing, cloudlet, and Mobile Edge Computing (MEC) managed by SDN Controller which improves the communication and computation cost. The SDN capabilities performs network programming and deployed the cloud resources at the edge of the network to reduce the overall service latency. The communication architecture of SD-DC and edge DC is shown in Fig. 2.13. The major issues in SD-FC and MEC are energy management, resource scheduling, security, and virtualization discussed as below in the subsection.

2.6.1 Energy Efficiency in Fog Computing

The primarily goal across SDN-based MEC devices is the energy efficiency in fog computing. Faruque *et al.* [159] analyzed the problem of energy management in homes over SD-FC platform. The author monitors home power consumption, and efficiently manage the power consumption by controlling the home appliances such as- Electric Vehicles (EV) charger, Heating, Ventilation, and Air Conditioning (HVAC) management. The home energy control issue is resolved by using three Control-as-a-Service (CaaS) schemes namely, (i) smart HVAC CaaS, (ii) smart EV charger CaaS, and (iii) smart transformer CaaS scheme. The scheme performs efficient management and controlling of home appliances and the results shows that the power consumption is decreased significantly. Aujla *et al.* [160] discussed the issue of huge energy consumption generated in inter-DC migration. The author considered the multi-edge cloud scenario and presented a workload slicing and scheduling algorithm to solve the data accelerated operations. In addition, the author proposed an energy aware flow scheduling algorithm for taking intelligent decisions by the SDN controller for traffic engineering. Finally, a Stackelberg GT based on multi-leader multi-follower is played by the author to handle the issue of inter-DC migrations. Sarkar *et al.* [161] worked on the problems related to huge power consumption, Carbon-Dioxide (CO_2) emission, service latency, and cost in cloud DCN. The author presented a fog computing model to serve the user demands and uses a mathematical approach to handle these issues. The result proves that the overall service latency is reduced upto 50.09% with 50% appliances demanding requests spontaneously for services.

2.6.2 Resource Scheduling in Fog Computing

In [162], the author developed a modeling and simulation tool named as "iFogSim" to perform resource scheduling. The iFogSim package consists of two modules namely; (i) *cloud only placement*: in which deployment of the application modules runs in the DCs, and (ii) *edge ward placement*: the deployment of the application modules runs at the edge of the network (like routers, access points). The iFogSim helps to measures the impact of resource scheduling

in terms of latency, congestion, energy consumption, and the overall cost. Yin *et al.* [163] designed a distributed resource sharing algorithm named as Hybrid Alternating Direction Method of Multipliers (HADMM). The designed scheme dynamically adjust the massive application data by transferring the cloud data to the fog nodes for pre-processing and analysis. The author formulate a maximization problem to decide the amount of data processed by the fog nodes. The proposed scheme reduced the communication overhead by increasing the participation of fog nodes and updates the variables of fog nodes in parallel. Aliyu *et al.* [164] presented a Adaptive Resource Capacity Management (ARCM) approach to optimize the resource capacity for real-time mobile edge computing applications. Moreover, a Partition Form Games (PFG) approach in SDN-based inter-cloud architecture is adopted for capacity management and LB. Ruia *et al.* [165] address the problem of increased workload on the controller as frequent requests are generated by the routers and switches. The author presented a flowcache as a software based cache for temporarily storing the contents of the recent flows table entries installed on the switches. The flowcache in an SDN architecture acts as proxy layer between the controller and switches which provides the controller with a large amount of flow table space.

2.6.3 Security in Fog Computing

The edge devices are more prone to security threats. The major security paradigm in a fog computing environment is required on the gateway nodes, controller, edge switches, and end hosts. Rawat *et al.* [66] presented a survey on the security attacks in SDN. The survey focused on the spoofing attacks, intrusion attacks, anomaly attacks, and DDoS attacks. An adversary targets the above-mentioned security attacks on the mobile cloudlet platform to break the security in SD-EC. Chaudhary *et al.* [36] mainly focused on the DDoS attacks on mobile cloudlet architecture. The cloudlet model or Mobile Edge Computing (MEC) has the capability to make the cloud resources accessible at the edge of the mobile network for mobile users. The author presented a Kerberos authentication mechanism as a reliable authentication service to defend from the DDoS attacks. In [166], the author presented a secure distributed fog nodes model using blockchain techniques in SDN architecture. The fog nodes allow the deployment of fog services at the edge of the network which improves the overall performance. The proposed model monitors the real-time attacks on the fog nodes. Additionally, the model reduces the service latency and increases throughput.

In [167], the author discussed the Cyber-Physical System (CPS) as communication is hijacked from mobile phones to laptops in fog computing. The author mainly focused on the issues of man-in-the-middle attack launched by an adversary on fog devices (gateway). The author investigates the CPU and memory consumption on fog devices that suffered from the attack. Tseng *et al.* [168] discussed on a serious security concern about the deployment of malicious application injection on the controller's runtime. The malicious applications lead to

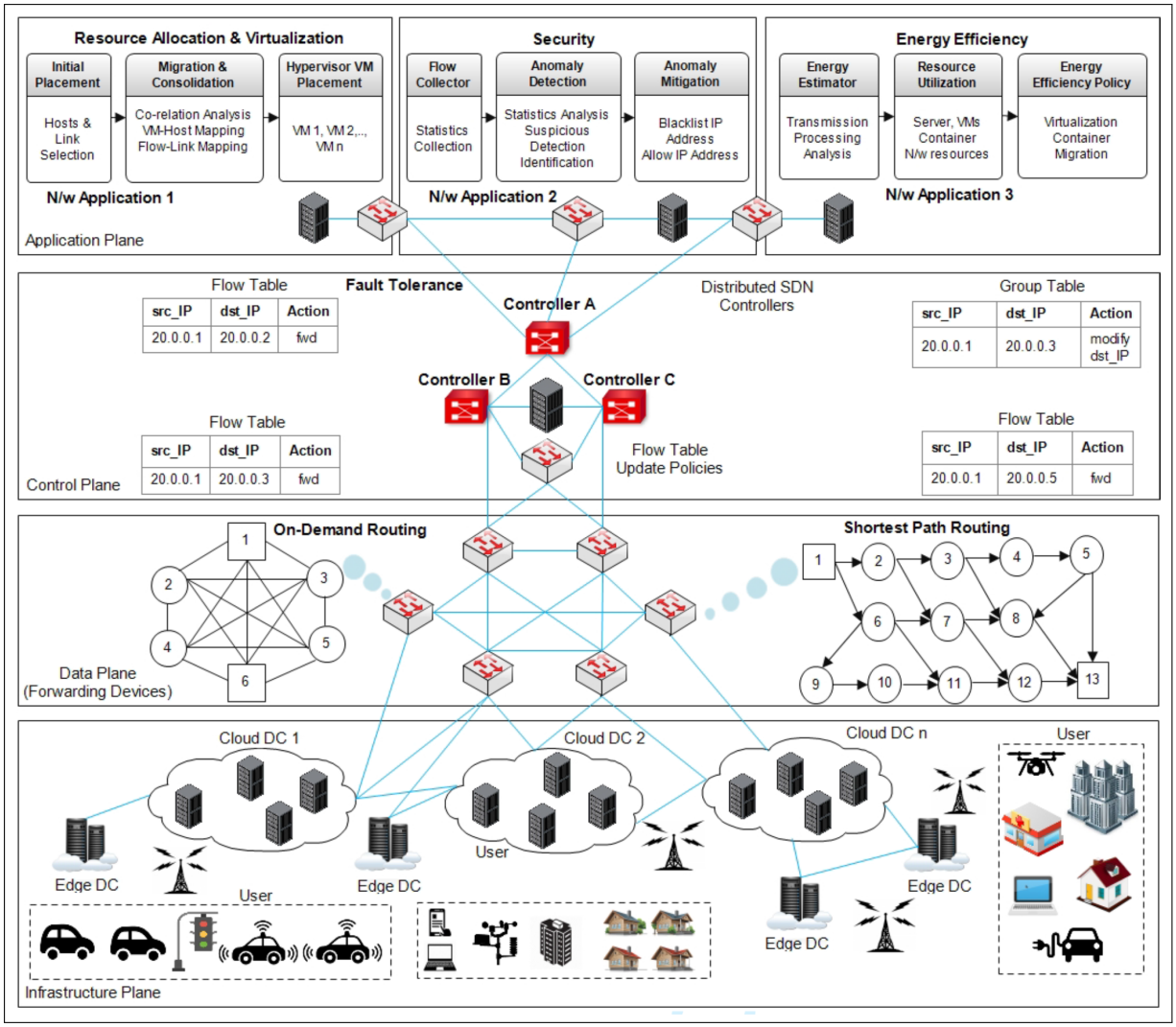

Figure 2.13: Communication architecture of SD-DC and edge DC.

the injection of app-to-control threats which exhaust the network resources. The author aims to design a Secure Network Application Deployment (SENAD) on the controller software. The proposed scheme partition the controller into a data plane controller for secure flow rules and the application plane controller to design secure applications for authentication, access control, authorization, and resource isolation.

2.6.4 Virtualization in Fog Computing

Before discussing the issues of virtualization, we first understand the scope to spread overlay network in fog computing. The concept of overlay network decouples the network services from the underlying physical infrastructure. The overlay network is built on the top of public Internet and encapsulates the packet to reach the endpoint network node where the packet gets de-encapsulated. An example of overlay network in SDN are content delivery networks, Virtual Private Networks (VPN), Voice over IP (VoIP) network, peer-to-peer network. The list

protocols supporting overlay network are as follows: VxLAN, Network Virtualization Overlays 3 (NVO3), Stateless Transport Tunneling (STT), Network Virtualization using Generic routing Encapsulation (NVGRE), GRE tunneling and so on. Bruschi *et al.* [169] worked on the issue of scalability in the fog computing environment. The author proposed a network slicing scheme to support multi-domain fog services. The network slicing scheme performs isolation of services into multiple slices. The proposed scheme offers high scalability by reducing the number of flow rules in the overlay network. The simulation result is implemented in matlab, the scheme installed the unicast forwarding rules in the overlay network and compared to the existing approaches (fully meshed and openstack approach). The performance of the overlay network shows that the amount of unicast forwarding rules decreased by one order of magnitude or four times compared to the existing approach.

Li *et al.* [170] addressed the issue of VM placement in homogeneous and heterogeneous cloudlet mesh infrastructure. The author considers same maximum bandwidth capacity and the same VM slots for each cloudlet node in homogeneous cloudlet mesh. The designed scheme is optimal algorithm to increase the amount of accepted VMs in the homogeneous cloudlet mesh. In case of heterogeneous cloudlet mesh architecture, the author considers different number of VM slots and different maximum bandwidth capacities for each cloudlet node. The author presented another optimal algorithm to increase the amount of accepted VMs in the heterogeneous architecture. The simulation results shows that the limited bandwidth resources still cope up with the growth of accepted VMs in the cloudlet mesh and increase in the number of VM slots. Liang *et al.* [171] designed a collaborative model of SDN and Virtualized Radio Access Networks (SDVRAN) with the fog computing environment. The author adopted a software-as-a-service scheme named as ”OpenPipe” (virtual resource chain) to perform network level virtualization and network operations via a network applications. In addition, a hybrid control model is designed to integrate SDN and NFV with fog computing. The hybrid model consists of two levels i.e., higher level with a SDN controller located on the cloud layer and a lower level with local controllers located at the edge of the networks (fog sub-SDNs). Moreover, the scalability of the network is increased by using a FlowVisor to creates an instance of the SDN controller and local controller as a virtualized SDN controller and virtualized local controller.

In [172], the author analyzed that the current DC are not suitable for NFV orchestration services due to the following issues such as- latency, throughput, and cost. The author proposed a VirtPhy scheme, a fully programmable architecture for NFV orchestration in fog computing with the capability to deliver fast innovative services in edge DCs. The proposed scheme uses a hypercube server-centric topology with test beds implemented using OpenStack cloud platform resulting to improve the service latency, cost, jitter, and throughput. In [173], the author presented a virtual set-up box scheme by creating virtual entities to replace the physical set-up box placed at user's home. The proposed scheme is a network softwarization approach to support personal cloud services in SDN and NFV architecture with fog computing. The performance evaluation proved that the designed approach achieves high QoS with low latency for

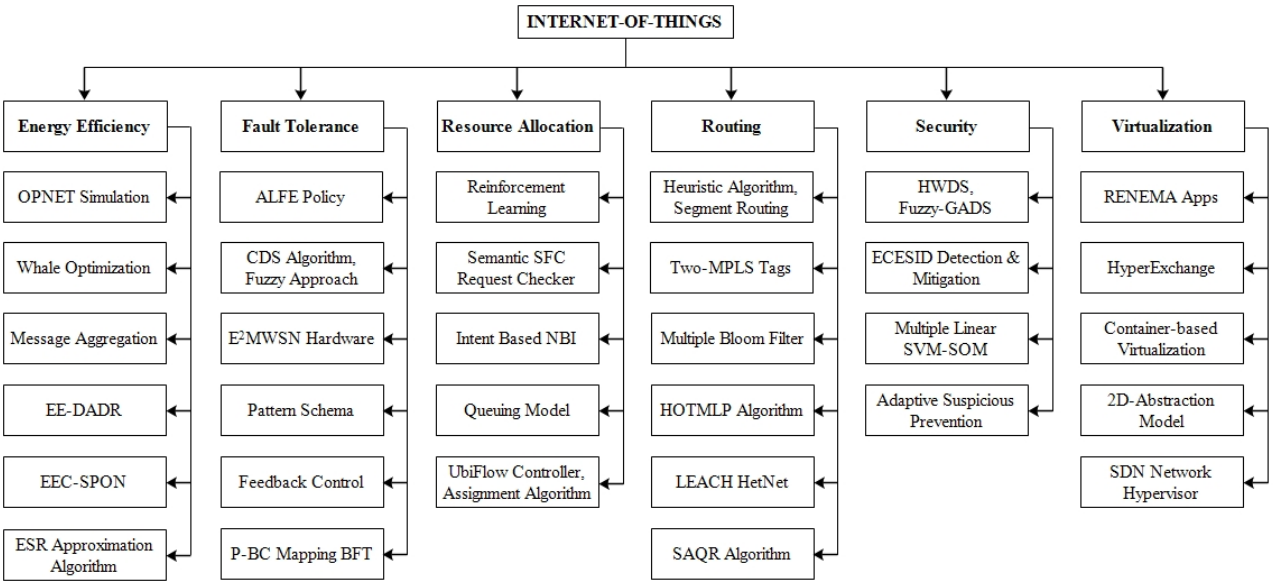

Figure 2.14: A Taxonomy of SDN in Internet-of-Things.

both static and dynamic users.

Table 2.8-2.9 shows the summarized overview of the performance analysis of the existing techniques on fog computing in terms of different network parameters.

2.7 Internet-of-Things (IoTs)

Software defined Internet of things (SD-IoT) is a new computing paradigm build to support interoperability in between heterogeneous devices. The primary objective of IoTs is sharing of information globally based on providing unique identity to each devices. Nowadays, with the expansion of smart cities in modern real-life scenario, billions of IoT devices are connected with the Internet. The taxonomy of SD-IoT network is shown in Fig. 2.14. The SDN architecture is facing some challenges to handle massive amount of IoT devices. The communication architecture of SD-IoT network is shown in Fig. 2.15. The major issues are related to energy management, failure recovery, resource allocation, routing, security, and virtualization.

2.7.1 Energy Efficiency in IoTs

The cloud DCs collect the data from IoT sensors through transmission network to perform data analytics. The transmission network mainly compromised on service latency and more energy consumption by the sensor devices. Hu *et al.* [174] addressed the issue of energy efficiency in IoTs and designed a hardware-in-the loop emulation. The designed software emulator is OPNET to build and test the controller design. The hardware-in-the loop is a cost-efficient real-time emulation model based on OPNET simulation to design and analyze the errors in the complex testbeds. The module to implement the emulator is the System-in-the Loop (SITL)

Table 2.8: Related work on fog computing using SDN architecture.

Authors	Techniques	Dataset	Controller	Setup Description	Coding/ Simulator	Issues Considered						Parm. Improved		
						1	2	3	4	5	6	7	8	9
[159]	SHAC CaaS, SEVC CaaS, ST CaaS	-	centralized	Raspberry Pi (gateway router); Tiny OS-based TelosB mote sensors (connection & computation on fog nodes)	Java	✓	×	×	×	×	✓	×	×	×
[160]	Stackelberg game theory	Google workload traces	distributed	1000 jobs; 3 Exp.: (1) cloud DCs, (2) edge devices, (3) edge cloud interplay. Saves energy consumption and reduces delay	Matlab	✓	✓	✓	×	✓	✓	✓	✓	×
[162]	iFogSim	-	centralized	Intelligent surveillance sensor: 4 fog devices; 1000 million instructions; 20,000 byte network length; the avg. inter-arrival time is 5 ms	CloudSim	✓	✓	✓	×	✓	✓	✓	✓	✓
[163]	H-ADMM algorithm	-	centralized	Large Fog nodes: 100 fog nodes reduced communication overhead on controller	Matlab	×	×	✓	×	×	✓	×	×	×
[164]	Partition form Games	-	centralized	VMs: 4096 MB; edgeIoT application: 75000; FogDCs: 4; 3,00,000 mips the service latency is less than 0.18s	CloudSim	×	×	×	×	×	✓	✓	×	×
[165]	Flow Cache	CAIDA traces	centralized (Ryu)	200,000 packets; 22 Mbps throughput generate 24,500 individual flow. Latency: (controller & switch channel) is 4 ms; (switch channel)	Python	×	×	✓	✓	×	✓	✓	✓	✓
[166]	Distributed Blockchain	real-testbed	distributed	6000 hosts; 2000 flow rate modules/s the execution time is 100 μs	TFN2K tool, Mininet	×	×	×	✓	×	✓	✓	×	✓

Table 2.9: Related work on fog computing using SDN architecture (Contd...)

Authors	Techniques	Dataset	Controller	Setup Description	Coding/ Simulator	Issues Considered						Parm. Improved		
						1	2	3	4	5	6	7	8	9
[167]	CPSs	-	centralized	(Build video tunnel on gateway) Memory Consumption: increased from 15232 KB to 15324.8 KB; CPU Consumption: increased from 16.67% to 17.92 % (almost negligible)	-	×	×	×	✓	×	✓	✓	✓	×
[168]	SENAD, Apache Kafka ACLs	-	centralized (flood-light)	10,000 packet_in; 1-10 sandbox (docker container) network application takes processing time 2.5946 ms-13.20 ms	Mininet	×	×	✓	✓	×	×	✓	×	✓
[172]	VirPhy	OpenStack Cloud	centralized (Ryu)	Add 2 extra hops: 8 server nodes; 1 NFV orchestrator; 1 network node; 1 controller; 1 Gbps link capacity. Increases jitter from 0.06 ms to 0.08 ms; packet loss is atmost 1.5% (negligible change)	Python, Mininet	×	✓	✓	×	✓	✓	✓	✓	✓
[173]	Virtual setup-box	real testbed (smart home)	distributed	7 DCs nodes; 4 edge node; 6 users home network; 10 Mbps-30 Mbps avg. bit rates. Provide network latency of 20 ms (short range) and 150 ms (long range)	Java	✓	✓	✓	×	✓	✓	✓	✓	×

Note- 1: Energy Efficiency, 2: Service Migration, 3: Resource Scheduling, 4: Security, 5: Virtualization, 6: Cost, 7: Latency, 8: Bandwidth, 9: Throughput
Notations- ✓: considered, and ×: not-considered

interface. The designed simulator is compared with the SITL emulator. The results shows that the OPNET simulation achieves energy efficiency and the delay generated by OPNET is around 0.1 second while the delay incurred in SITL emulation is around 0.4-0.5 seconds. Al-Janabi *et al.* [175] focused on energy efficiency in Software Defined Wireless Sensor Networks (SD-WSN). The author identify two major problems in sensor nodes i.e. resource limitations and arbitrary diversification in node density of regional location. The author solves both the issues and presented a new clustering scheme named as Whale Optimisation Algorithm (WOA). The scheme initially partition the complete geographical area into multiple Virtual Zones (VZ) in order to maintain the equilibrium of cluster heads in each VZs. The experimental results proved that the proposed algorithm achieves energy efficiency and improved the amount of packet received by 55% as compared to the existing clustering protocols and by 20% compared to the existing routing protocols.

Wang *et al.* [176] diagnose the issue of trust management in two major aspects i.e. energy management and efficient routing scheme to detect the malicious nodes in the SDWSNs. The author presented a centralized trusted approach on the SDN controller to monitor and decoupled the malicious nodes. Moreover, a message aggregation approach and trusted routing mechanism is designed to monitor and report the network attacks. The simulation results proves that the proposed scheme defends from the malicious threats such as- Greyhole and Blackhole attacks. Finally, the scheme improves the packet delivery ratio with low control overhead and reduces the energy consumption as compared to the existing approaches. Wen *et al.* [177] addressed the issues of energy management and E2E delay constraint in routing. The author presented a common algorithm i.e., Energy Efficient and Delay-Aware Distributed Routing (EE-DADR) to solve both the issues simultaneously. The proposed scheme is compared with the existing approach i.e. Greedy Perimeter Stateless Routing (GPSR) and the experimental results shows that the EE-DADR scheme consumes less energy and reduces the E2E delay as compared to GPSR scheme.

With the advancement in high bit rate applications, Software defined Passive Optical Network (SPON) technologies plays a crucial role in the broadband services in 4G/5G network. The SPON architecture is designed by two network modules i.e. Optical Line Terminal (OLT) and Optical Network Units (ONU). Zhao *et al.* [178] analyze that the SPON provides high bit rate to the end users but still the crisis of low bandwidth and high energy consumption is a critical issue. The author developed an energy-efficient control scheme by collaborating the optical access networks and metro network resources. The presented scheme results to improve the energy consumption in SPON and achieves high bandwidth utilization. In [179], the author observed the problem of big data processing in SD-IoT. The author formulate an optimization problem called as sensor data selection, data processing, and routing to reduce the overall energy consumption in SD-IoT. The proposed scheme is Energy Efficient Sensor Selection and Routing (ESR) based on $\alpha\ log|K|$ approximation algorithms. The experimental results finds the optimal sensor selection and reduces the total energy consumption in SD-IoT upto 56%.

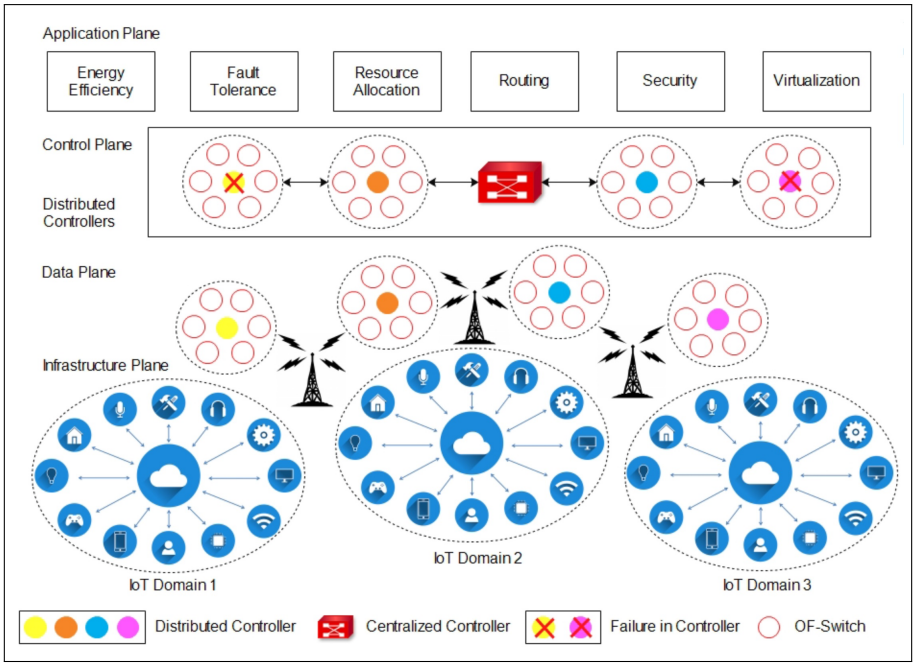

Figure 2.15: Communication architecture of SD-IoT.

2.7.2 Fault Tolerance in IoTs

The SDN architecture is facing the congestion problem due to limited memory capacity for storing the flow tables into the switches. However, the elephant flows (huge flow) over the network is handled by using an efficient cache mechanism in order to prevent the overflow of massive mice flows (tiny flows). Pan *et al.* [180] compute a solution by designing an optimal flow cache policy named as Adaptive Least Frequently Evicted (ALFE). The proposed approach is compared with the Least Recently Used (LRU) cache placement policy and is tested over 1000 cache entries. The dataset traces is achieved from Center for Applied Internet Data Analysis (CAIDA) and the simulation results shows that the ALFE approach increase the cache hit by 15% as compared to LRU approach. In [181], the authors focused on the reliability of the SD-IOT network. The reliability in terms of connectivity rate, the average link quality, hop counts, and resource management. The author introduced a dynamic approach in SD-IOT to minimize the control overhead in selection mechanism of network resources. The concept of distributed controllers enables the placement of primary controller, secondary controllers, and local controllers to control the network nodes. The selection strategy of the number of local controllers is an important concern. The appropriate scheme is Connected Dominating Sets (CDS) based on fuzzy approach helps to compute the relevant number of local controllers required to perform smooth execution of resources.

Shi *et al.* [182] primary focused on the robustness and reliability in SDWSNs. The author designed an energy-efficient and fault tolerance multi-core architecture (EE-MWSN) to achieve low energy consumption as compared to WSN platform (TinyOS and TelosB). In [122], the author presented a pattern framework as a prototype model for SDN layered architecture. The

framework consists of pattern schema (specification and language) located at the application layer to modify the network topology. The pattern language develops the pattern rules to be executed on the controller. These pattern rules are installed on the forwarding devices in order to detect the link failure and faults.

In a distributed controller environment, the issue of cascading failure generally occurs when a specific controller gets failed then the network is overloaded and leads to packet loss. Hosny *et al.* [183] worked on the issue of cascading failure problem. The fault tolerance in SDN is handled by using a feedback control based on LB approach. The proposed scheme minimized the cascading failure problem on the distributed control plane by using a feedback control policy. A feedback control policy broadcast the report to all the distributed controllers and helps to automatically migrate the load to the neighbor controllers. The performance of the proposed approach helps to reduce the packet loss upto 14% and minimize the packet delay upto 17%. Although multiple controllers are deployed to achieve fault tolerance in the network but at the same time, security and the mapping from the switches to controllers and vice-versa is the major concern. Mohan *et al.* [123] identify the issue of controllers overhead and byzantine threats on control messages when an individual switch is mapped to $3f+1$ controllers. The author presented a Primary and Backup switch-controller (P-BC) mapping approach in which an individual switch is mapped to $f+1$ primary controllers and f backup controllers. The presented approach helps to mitigate from byzantine attacks on f backup controllers. In addition, the proposed approach calculate the minimum number of controllers required with each controller capacity in order to avoid delay. The results is compared with the traditional approach which proves that the scheme minimize the overall controllers requirement upto 50% and improves the network load within controllers with a fairness index upto 0.92.

2.7.3 Resource Allocation in IoTs

In highly dense smart cities, huge volumes of data is generated from smart sensors hence it becomes difficult to accomplish service requests to multiple service providers within a short span of time. The requirement of resource allocation in SD-IoT network is mandatory. Munir *et al.* [184] uses the concept of Reinforcement Learning (RL) approach and presented an intelligent agent service fulfillment algorithm to resolve the issue of delay incurred through multiple service requests. The presented scheme is implemented in fog-based controller unit and results to achieve high performance by reducing the processing and waiting delay. The proposed approach achieves the time complexity of around $O(n^2)$ and each sub-problems produce the time complexity of $O(2^n.n)$. The results shows that when the delay is 0.10 ms it achieves maximum utilization of resources. Bouten *et al.* [185] focused on the issue of limited resource capacity constraints to map VNFs with the physical infrastructure and fast routing of data transmission in virtual paths. The author find an optimal solution for mapping the VNFs or Service Function Chaining (SFC) requests in a reasonable time. The author proposed a semantic SFC validation

model to decrease the mapping time of service requests. The simulation results shows that the semantic matching on individual mapping of 100 service requests, the mapping time is reduced to 50%. Furthermore, considering the mapping of 20 SFC requests where each SFC has 5 VNFs then the average execution time comes out to be 0.85 seconds. The scheme significantly save the mapping time of an algorithm.

Cerroni *et al.* [186] observed the problem of low reliability and high latency generated due to increased in the SFC requests. The author presented a reference multi-domain SDN/NFV architecture by creating a service chain of network functions running on the hardware platforms and enhance flexibility in service deployment. Additionally, an intent based NBI architecture is presented in SDN for service orchestration. The average response time of the proposed model achieves high reliability with average latency of 31.7 ms with 95% confidence intervals and minimize the average latency upto 0.3 ms. Banaie *et al.* [187] discussed about the data sharing by adopting virtualization to create virtual sensors resources on cloud platform as an instance of physical sensors. The author uses queuing model to develop a scheduling approach to balance the load of virtual sensor resources. The proposed approach distributes the load among virtual resources and decrease the average completion time around 20s. However, in addition to resource allocation problem, ubiquitous flow control and device mobility is also an important concern in SD-IoT network. Wu *et al.* [188] presented a UbiFlow control mechanism to address the issue of mobility. The designed approach partition different geographical locations and deploy distributed controllers. The UbiFlow controller examined the flow scheduling in each partition and the results proves that the average flow transmission delay is less than 0.4s for mobile devices. The performance of the new scheme is robust in flow scheduling and achieves stability in mobility management.

2.7.4 Routing in IoTs

The network performance can be improved by designing an efficient routing scheme in SD-IoT. Lee *et al.* [189] design an efficient heuristic scheme for Segment Routing (SR) through an ordered MPLS approach to cope up with the bandwidth requirements. The performance of the designed algorithm shows an improvement in the average rejection rate and average network throughput. Kitsuwan *et al.* [190] addressed the issue of overflow in OF messages generated at the controller. The proposed scheme is two-MPLS tags to reduce the flow scheduling. The tags or label are generated on each network packets to route the packets from source to destination. The first tag is labeled to escort the network packet from source switch to the edge switch of a destination class and the second tag is labeled to escort the packet from the edge switch to another switch of a local class. The experimental results shows that the scheme reduces 81% of the flow messages and 74% reduction in the number of permanent flow entries as compared to the existing schemes. Challa *et al.* [191] address the issue of insufficient flow table memory and computes a solution for fast flow forwarding. The proposed scheme uses probabilistic data

structure named as Multiple Bloom Filter (MBF) as a space efficient technique. The proposed approach is designed column-wise to store the flow entries in SRAM rather than TCAM and returns the error probability of less than 1%. The results are compared with the LRU approach and reduces the lookup times upto 37% with flow table capacity of 2000 entries.

Zhijie *et al.* [192] focused on the similar issue of memory shortage of flow table and the routing flow forwarding speed. The author presented a scheme called as Hash Offset Tree Match on Longest Prefix (HOTMLP). The scheme performs the packet lookup and fast forwarding in SD-IoT network. The scheme apply hashing on the longest prefix rule matching to reduce the storage space and the exact prefix matching is selected based on the tree approach. The results proves that the designed algorithm performs faster lookup with space complexity of $(O(2+n/k))$. In addition, the presented routing method forward around ten millions packets per second. In [193], the author presented a low energy adaptive clustering hierarchy (LEACH) heterogeneous routing protocol to achieve both energy efficiency and optimal routing. The routing scheme is simulated in ns-2 by considering a network of 50 nodes with initial energy of 10 joule, and trust value is 5. The designed scheme is compared with the existing routing methods i.e. Adhoc On-Demand distance Vector (AODV), Dynamic Source Routing (DSR) and performs better performance in terms of energy and routing by reducing E2E delay. Lin *et al.* [194] identify the issues of packet loss, delay and bandwidth in existing routing protocols. The author proposed an efficient dynamic routing algorithm i.e. simulated annealing based QoS-aware routing (SAQR) to compute the optimal fit path by using cost function. The SAQR algorithm is compared with Multi-Network Information Architecture (MINA) layered controller approach. The results shows that the proposed scheme achieves better fitness ratios with delay reduced to 88%, packet loss reduced to 90.8%, and bandwidth requirement of packet flows is improved by 86.5%.

2.7.5 Security in IoTs

With the rapid deployment of wireless sensors in smart cities, the Device-to-Device (D2D) communication is more susceptible to security threats. As single controller is handling the complete network infrastructure, the chances of malicious distributed denial of service (DDoS) attacks may infect the heterogeneous IoT devices [67]. Hence, it becomes mandatory to provide defense mechanisms so that the controllers establish resilient and reliable connections with the switches. Gharaibeh *et al.* [195] analyzed a survey on the data management and security in smart cities SDN-IoT environment.

Assis *et al.* [196] addressed the issue of DDoS attack as enormous malicious requests sends on the controller which degrades the overall network performance. The designed approach mitigates from the DDoS attack by using Game Theory and Holt-Winters for Digital Signature (GT-HWDS). Moreover, the author uses fuzzy logic and genetic algorithm for digital signature (Fuzzy-GADS) to identify and mitigate suspicious behavior of the requesting nodes. The

performance of the designed scheme shows that the alarms got triggered automatically on the controller by the GT-HWDS and Fuzzy-GADS algorithm whenever an anomaly is detected. In [197], the author presented an architecture regarding the issue of mirai-botnet DDoS attack (malware) infecting IoT devices. The defense mechanism is the edge-centric SDN-IoT framework. The presented framework performs two tasks namely; (a) malware detection and mitigation of infected node, (b) flow handling algorithm on the controller. The simulation results shows that the designed algorithm is efficient detect and mitigate the infected node with an average detection time of 6.02 seconds.

Phan *et al.* [198] addressed the similar issue of DDoS attack in SD-IoT network by using a hybrid model. The author presented a novel hybrid flow handler mechanism based on two classification modules i.e. Support Vector Machine (SVM) and Self Organizing Map (SOM). The various linear SVM initially classifies the flow entries exist in the OF-switches. The linear SVM representation checks whether the flow entries are present in between two margin lines, if found then forward those entries to the SOM method for further judgment. The SOM module makes the final judgment of the suspected patterns based on the weights computation of neurons. The experimental results shows that the proposed hybrid model attains better accuracy in terms of the following parameters such as- True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN) errors. Dao *et al.* [199] presented a Adaptive Suspicious Prevention (ASP) mechanism to prevent the SDN controller from DDoS attacks. The proposed technique is embedded with the OF protocol to protect the DDoS attacks on the switches. The performance analysis shows that the designed technique is capable to defend from security attack by upto 38%.

2.7.6 Virtualization in IoTs

With the advent of wireless sensors, the drive for virtualization is required on the cloud with the emergence of technologies like SDN and NFV. The virtualization has the capability to dynamically allocate the resources on the applications whenever required. Moyano *et al.* [200] addressed the issue of manual configuration in residential networks. The author solved the issue of residential networks based on NFV and SDN network centric approach by configuring the residential gateways through network programmability. The author develops a Residential Network Management Application (RENEMA apps) on top of the SDN architecture as an application layer to control the network traffic and interact with the user devices. The presented framework improves the overall throughput of the Home Area Network (HAN). In [201], the author identify the problem of scalability and extensibility in SDN- NFV architecture. The author designed a protocol named as HyperExchange on the control plane to offer exchange of services in inter-domain virtual networks. The protocol provides flexibility in peering and in addition supports authentication and authorization through remote APIs for inter-domain tenants.

Drutskoy *et al.* [202] also addressed the issue of scalability in SDN-NFV architecture. The author explore FlowN architecture having the ability to modify the mapping between the physical and virtual forwarding devices. Furthermore, the author highlights the benefits of container-based virtualization to minimize the virtual network mapping overhead of deploying multiple controller applications. The FlowN architecture provides the mapping between the virtual and physical resources as each tenant has an illusion of its own available address space, network topology, controller application, and database. Duan *et al.* [203] presented a 2-D abstraction model of layer and plane architecture of SDN-NFV. The architecture discussed the layer dimension abstraction consists of five layers such as- physical, network interface, Internet, transport, application layer. The plane dimension abstraction shows the SDN three planes- data, control, and management plane. Blenk *et al.* [204] examined the issue of high latency overhead incurred on the control plane due to the existence of multiple virtual controllers. The author formulate the problem based on mixed integer programming in order to optimize the placement of virtual controller or network hypervisor instances. The author performs the trade-off strategy for optimal controller placement of virtual controllers so that maximum nodes is served. The overall network latency is reduced by upto 42% by covering the distance of 5.10^3 km to $2.9.10^3$ km by virtual controllers.

Table 2.10-2.11 shows the summarized overview of the existing work on SD-IoT.

2.8 Healthcare Network

The Healthcare network (HCN) becomes a hot spot to provide remote medical facilities at homes with the adoption of smart sensors. Using wearable devices, the clinical facilities is provided at user's home in order to reduce the overall CAPEX and OPEX rather than visiting hospitals. The healthcare industry is focusing on various parameters like; data collection through wireless sensors, data transmission via high-speed Internet, secure data storage on cloud, and data analytics on data centers. Using the underlying network technologies like SDN, NFV and 5G, the healthcare industry is improving at a rapid pace to ease the living standards of the users. The human to machine interaction between the users and doctors is possible by using wearable devices and is expected to improve the healthcare services through regular monitoring. The taxonomy of SD-HCN is shown in Fig. 2.16. However, the Software-Defined Healthcare Network (SD-HCN) is offering best QoS to the end-users but still there are some issues that affect the overall network performance of the system. The communication architecture of SD-WBAN network is shown in Fig. 2.17. The major issues in SD-HCN are routing, security, resource scheduling, and mobility. The related work in all these issues are discussed in this section.

Table 2.10: Existing work on SD-IoT.

Authors	Techniques	Controller	Setup Description	Coding/ Simulator	Issues Considered						Parameters Improved				
					1	2	3	4	5	6	7	8	9	10	11
[174]	Hardware-in-the-loop Emulator	centralized (NOX)	Area: 375mx375m; 140 sensors; 2 Mbps data rate; 40-70m sensor range; 5W source node energy reduces energy consumption and reduces delay by upto 0.1s	OPNET	✓	×	×	✓	×	×	✓	✓	×	×	×
[177]	EE-DADR Scheme	centralized	Area:1000mx1000m; 2 Mbps data rate; 300 mW power of each node; 250 m radius; 50 nodes generates 25 packets/s of 25 kb each takes the execution time of 400s	OMNET++	✓	×	×	✓	×	×	✓	✓	✓	✓	✓
[179]	ESR Scheme	centralized	33 switches; 41 links; 1 Gbps data rate; 5 mW processing power; 200 mW transmission power; 1200-2000 sensors saves energy by upto 56%	-	✓	×	×	✓	×	×	✓	✓	×	×	×
[181]	CDS, Fuzzy-logic	centralized (multi-threaded)	Area:100mx100m; 49 nodes; 1 gateway node; 0.01 node density of each node increases the node reliability	Matlab	×	✓	✓	×	×	×	×	×	×	×	✓
[122]	Pattern Schema	distributed	6 switches; 2 hosts; 6 pattern topology; 9 links achieves flow completion time of 1.3s and with redundancy degree of 3	Mininet	×	✓	×	×	×	×	×	×	×	×	✓
[123]	P-BC mapping	distributed	Controllers: f+1 primary; f backup reduces full-backup controllers upto 40% and partial controllers by 50% and reduces 50% controller load	-	×	✓	×	×	✓	×	✓	✓	×	×	✓
[184]	Reinforcement learning	centralized (multi-threaded)	8 switches; 40 hosts achieves the overall delay as 0.1ms	Mininet	×	×	✓	×	✓	×	×	✓	×	×	✓
[185]	Semantic SFC Checker	centralized	20 SFC requests; each SFC has 5 VNFs takes the execution time of 0.85s to map the requests and for 512 nodes; 100 SFC requests; the mapping time is reduced by 50%	Java	×	×	✓	×	×	✓	✓	✓	✓	×	✓
[186]	Intent based NBI	distributed (ONOS)	standard deviation is 0.28ms achieves average latency of 31.7 ms and reduces the overall latency by 0.3ms	Mininet	×	×	✓	✓	×	✓	✓	✓	×	×	✓

Table 2.11: Existing work on SD-IoT (Contd...)

Authors	Techniques	Controller	Setup Description	Coding/ Simulator	Issues Considered						Parameters Improved				
					1	2	3	4	5	6	7	8	9	10	11
[187]	Queuing model (M/M/m)	centralized	20,000-45,000 million instruction/s; 100-800 requests; 5 virtual sensor reduces the response time by 20s	CloudSim	✓	×	✓	×	×	✓	✓	✓	✓	×	×
[188]	UbiFlow Assignment Algorithm	centralized (multi-threaded)	3 servers; 3 switches; 1 Gbps link capacity of each switch; 20 access points; 100 Mbps Ethernet link to switches; 30 flows increases the avg. throughput by 42.24% and reduces latency by 62.07%	OMNET++	×	✓	✓	×	×	×	✓	✓	✓	✓	✓
[189]	Segment Routing	centralized	60 nodes; 201 links; 1 Gbps link capacity; 10-100 Mbps flow size; 100 requests improves throughput and rejection rate	Java	×	×	✓	✓	×	×	✓	×	✓	✓	✓
[193]	LEACH Protocol	centralized	Area: 920mx800m; 50 nodes; 10J initial energy; 512 packet size improves the E2E delay	ns-2	✓	×	×	✓	×	×	✓	✓	×	✓	✓
[194]	SAQR algorithm	centralized (multi-threaded)	4 core switches; 4-16 edge switches; the execution time is 3.62ms. Reduces delay and packet loss by 88% and increases bandwidth by 86.5%	Mininet	×	×	✓	✓	×	×	✓	✓	✓	✓	✓
[196]	HWDS, Fuzzy-GADS	centralized	Exp.1, Exp.2, Exp.3, Exp.4: (512, 1024, 2560, 5120) hosts- mitigate the malicious hosts and achieves accuracy more than 98%	Scorpius	×	×	×	×	✓	×	✓	×	✓	×	✓
[197]	Mirai-botnet Edge Centric	centralized	7 IoT devices; 30 wired hosts; 5s queue update interval; 10000 flow capacity; 250 ms timeout takes 6.02 s	Mininet	×	×	×	×	✓	×	✓	✓	×	✓	×
[198]	Multiple Linear SVM-SOM	centralized (POX)	CAIDA datasets; 4000 flows; 30s hard timeout computes detection rate; accuracy; and false alarm rate based upon TP, TN, FP, FN as 96.03%, 98.17%, 3.97%, 1.83%	-	×	×	✓	×	✓	×	×	×	✓	✓	✓
[199]	Adaptive Suspicious Prevention	centralized (POX)	5,957 flows; 14.58 average packets/flow; increases bandwidth from 1234 Kbps to 1,263 Kbps in switches-controller channels. Reduced flow entries by upto 38.23% and reduces packet arrival per second by upto 36.17%	Mininet	×	×	✓	×	✓	×	✓	✓	✓	×	✓
[204]	Optimal Controller Placement	distributed	34 nodes; 41 edge switches; 2 hypervisors; 17 multi-controller switches; Area: $5.10^3 - 2.9.10^3$ km reduced the overall latency by 42%	Python	×	×	✓	×	×	✓	✓	✓	×	×	×

Note- 1: Energy Efficiency, 2: Fault Tolerance, 3: Resource Allocation, 4: Routing, 5: Security, 6: Virtualization, 7: Cost, 8: Latency, 9: Bandwidth, 10: Throughput, 11: Packet Loss, **Notations**- ✓: considered, and ×: not-considered

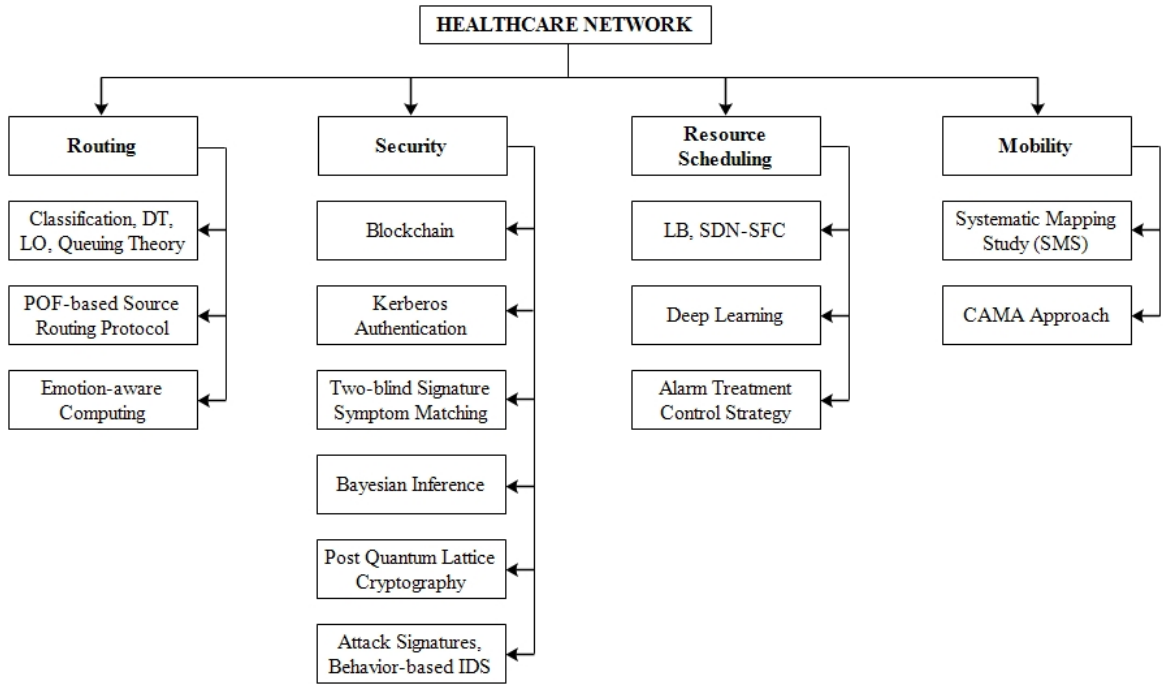

Figure 2.16: A Taxonomy of SDN in Healthcare Networks.

2.8.1 Routing in Healthcare Network

In smart communities, mostly people are health conscious so healthcare monitoring system is performed by deploying intelligent sensors in smart homes for data collection and transmission. The collected data is transmitted to the cloud for data analytics and finally stored and processed through the cloud service providers. The major issue faced in the SD-HCN is high latency as multiple requests are processed simultaneously may saturate the SDN controller. Aujla *et al.* [205] focused on the issue of routing and network overhead generated on the control plane. The author presented an integrated approach for dynamic traffic flow management. The approach is divided into three phases: (a) a dependency removal method is presented on the incoming requests at OF-switches, (b) an application-specific packet classification method is adopted on the controller by using decision tree in order to classify the application features, and (c) the congestion and starvation problem is handled in classified traffic flow on the controller by using Priority Shifting (PS) scheme based on M/G/1 queuing model. The designed PS scheme is compared with the First-In-First Out (FIFO) scheme. The simulation results shows that the average waiting time of the highest priority data is achieved with low latency and high bandwidth as compared with the medium or low priority data.

Li *et al.* [206] addressed the issue of high latency and scalability during data transmission of e-healthcare in Software Defined Wide Area Network (SD-WAN). The author presented a scheme named as Protocol-Oblivious Forwarding (POF) based source routing protocol. The proposed protocol performs processing of each packets by replacing conventional approach i.e. table lookup process by a new approach i.e. table pipeline processing for efficient routing on SDN switches. The performance of the designed scheme is better than the conventional ap-

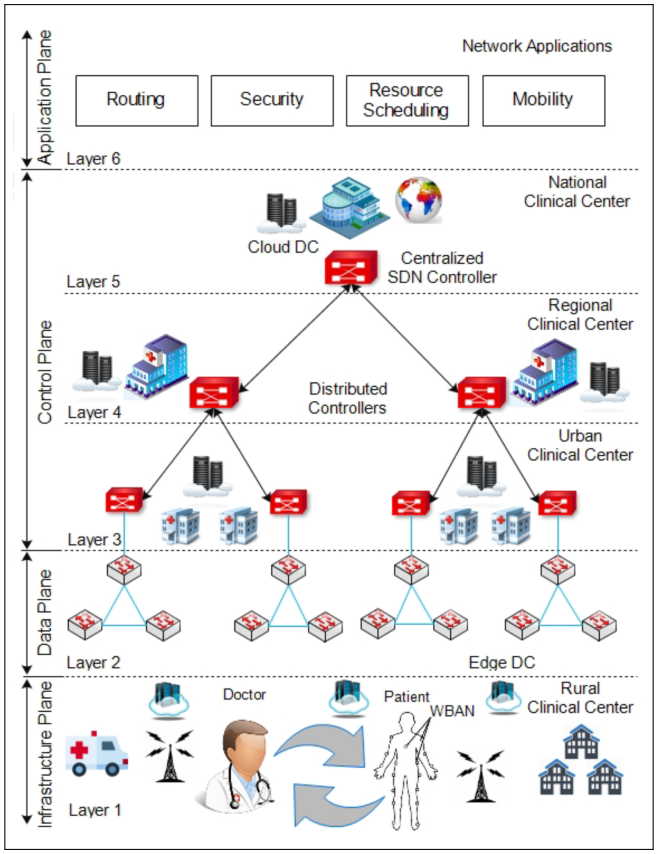

Figure 2.17: Communication architecture of SD-WBAN.

proach in large scale WAN because the controller configure only the ingress and egress edge switches while the conventional OF-protocol, the controller have to configure all the switches. The simulation results shows that the proposed scheme reduced the path setup latency to much extent in SD-WAN for huge traffic flow. Lin *et al.* [207] pay attention to improve the overall healthcare monitoring system. The author presented an architecture for Big Data Application in Emotion-Aware Healthcare (BDAEAH). The designed technique is based on emotion computing by considering user's emotions or illness as the most important factor to perform better patient treatment. The presented BDAEAH architecture performs four functions mainly; (a) collect physical signal of user's emotions through wearable sensors, (b) data transmission on data centers through SDN and 5G services, and (c) data analytics of emotion computing on data centers, and (d) display final decision to remote users or doctors. The designed architecture results to improve resource utilization and is expected to improve the overall healthcare services to remote clients.

2.8.2 Security in Healthcare Network

Hayward et al. [62] examined and presented a detailed survey on the issue of security in SD-HCN. The survey discussed various security threats occurred in the distributed controller framework. The list of security threats are as follows: unauthorized access (Controller Hijacking),

data leakage (side channel attacks), data modification (man-in-the-middle attacks), malicious compromised SDN applications (illegal OF-rules), and DoS attacks (flooding unauthorized requests on Controller-Switches communication). The smart healthcare services mainly focused on data collection, aggregation, and analysis of sensors data but security is the most important issue arises as sensitive patient's record is shared on the public insecure channel through various wireless technologies.

In [208], the author analyzed the importance of secure data storage and transmission in e-healthcare system. The author presented a SD-IoT centralized architecture in cloud and fog computing. Using network virtualization, the multiple controllers are deployed on the control plane and implement a distributed model based on blockchain technology on each controller. The presented blockchain technology maintain secure patient's records by validating every transaction and finally add each authenticated data block in the blockchain. Shayokh *et al.* [209] discussed the growth of virtual hospitals based on SDN technology to provide regular medical facilities for remote users. The major problem analyzed by the author in Wireless Body Area Networks (WBAN) is mainly insecure content delivery of patient's sensitive records. The author presented a secure authentication protocol i.e. Kerberos as a defense mechanism for secure data delivery in WBAN for virtual hospitals. The experimental results shows that the proposed protocol reduces latency and perform better security in packet inspection to verify whether it is normal or medical data.

The IoT devices plays an important role in smart healthcare system by monitoring the Personal Health Information (PHI) of end-users and displays the aggregated report on smart phones. The aggregated report is transmitted on remote DCs to check the authentication of each mobile patients. The authenticated mobile patients are allowed to share his/her report on social networking to create a group based on Symptoms Matching (SM). This process is called as Mobile Healthcare Social Networks (MHSN) and acts as a self-organizing approach to provide remote healthcare services to patients. Jiang *et al.* [210] focused on the challenges faced by sharing PHI with the unknown guest in MHSNs. In case, the unknown guest is an adversary then the privacy of sensitive information will no longer exists. To overcome the misuse of sensitive SM information, the author presented a technique known as two RSA based blind signatures SM. The proposed two blind signature technique performs both coarse grained and fine grained SM approaches. Moreover, a probabilistic data structure called as bloom filter is also used to check the similarity degree of two patients symptoms based on their weighted euclidean norm. The performance of the designed scheme is proved to be practical in maintaining privacy preserving for SM of patient records. The numerical result shows that the proposed scheme is tested for both the coarse grained and fine grained techniques by considering 50 and 100 symptoms. From the experimental results, the coarse grained SM technique achieves an average CPU loads of 16.4% for 50 symptoms and 17.3% for 100 symptoms while the average power consumption for both the symptoms are 2205.2 mw and 2502.1 mw. Likewise, the fine grained SM technique computes for 50 and 100 symptoms to produce an average Central Pro-

cessing Unit (CPU) loads of 15.7% and 17.6% while the average power consumption is 2126.3 mw and 2457.7mw.

Meng *et al.* [211] focused on the issues of malicious attacks launched by an intruder in SD-HCN. The proposed scheme is trusted Bayesian inference model deployed with Snort IDS to inspect normal devices or malicious devices inside healthcare organizations. The author surveyed on 12 medical organizations and analyze the performance of the proposed scheme. The experimental result proved that the proposed scheme is capable to decrease the trust value of malicious devices both on the controller and switch side. Chaudhary *et al.* [212] analyzed two major security threats in SD-HCN. An adversary launched eavesdropping attack during data transmission and man-in-the-middle attack at the time of data storage on the public cloud. The author presented a lattice based cryptography technique to mitigate both the security attacks. The issue related with the data exchange between the patients and doctors is solved by using a key exchange authentication mechanism based on lattice based cryptosystem. The second issue of insecure data storage on public cloud is solved by using an encryption and decryption process. Furthermore, the author applied access control mechanism to assign permissions of access rights for each patients and doctors whenever accessing any information. The performance of the designed scheme shows that the lattice cryptography technique is resilient from quantum attacks and reduced both the communication cost as well as the computation time.

In [213], the author examined the challenges related to suspicious activities occurred by the traveling patients. The author designed two approaches i.e. attack signatures and behavior based IDS to monitor and trigger an alarm to the hospital staff, in case any suspicious activity is found from the roaming patients. The designed approaches classifies the healthcare traffic from other traffic and impose a fine-grained security policies on the controller. The database of the security policies is maintained on the SDN controller to track live location of each patient and monitors the communication among patients. Finally, the performance of proposed the scheme is observed to be capable enough to figure out malicious devices and dynamic modify the security policies in order to mitigate from abnormal behavior in hospital environment.

2.8.3 Resource Scheduling in Healthcare Network

The key consideration in smart HCN is resource scheduling because the problem of network congestion arises due to lack of available resources. Li *et al.* [214] solved the problem of resource scarcity and network congestion in SD-HCN. The author presented an optimal LB technique based on SDN and SFC to eliminate the requirement of large number of resources. The SFC is build based on meta heuristic approach i.e. Simulated Annealing (SA) algorithm. The main objective of the SA algorithm is to achieve minimal transmission time and cost for content delivery on the medical teams through LB. The performance of the proposed SA algorithm is compared with the greedy algorithm. The results shows that the SA algorithm attains superior results than greedy algorithm in terms of transmission time for less number of users

(less than 50) while for the large number of users (150 or more) the greedy algorithm is more appropriate.

In [215], the author discussed about the comfort level achieved due to chemotherapy available at user's home. The patient or users in daily routines activities can perform their medical treatments such as- heartbeat checkup, diabetes, voice disorder, blood pressure and so on. The author presented a healthcare framework based on edge computing and apply deep learning method to solve the problem of voice disorder assessment. The proposed framework achieves low latency as mostly the computation is performed at the edge nodes. The results shows that the proposed approach takes 0.13 s to identify voice disorder and 0.23 s to classify voice disorder and finally, attains 98.5% accuracy with more than 99.3% sensitivity. Rego *et al.* [216] analyzed the challenges faced in smart cities during emergency situations like; accidents, terrorists activity, fire, natural disasters and so on. The major challenges of improper traffic flow management and high latency is the key consideration. The author presented a SDN-IoT based control strategy and proposed an alarm treatment algorithm to minimize the damage to humans and emergency resources. The control strategy performs efficient resource scheduling by using traffic lights, cameras, and alarm triggered management system. The proposed strategy dynamically divert the traffic flow routes from normal behavior towards emergency area by giving highest priority to emergency traffic in order to reduce the delay time in providing emergency services. The simulation result shows that the proposed algorithm reduces the average latency from 26 milliseconds (ms) to 17 ms and minimized the delay time by upto 33% in emergency traffic.

2.8.4 Mobility in Healthcare Network

The SD-HCN are adopting Body Sensor Networks (BSN) in remote or rural areas for data collection and deliver the content to doctors via a wireless networks for real-time diagnosis. The issue of mobility or signal fading generally arises in wireless networks during data transmission. The mobility in HCN may lead to packet loss and increase the delay time for data transmission. Hence, the dependability and reliability of e-healthcare system is an important concern. Junior *et al.* [217] performed a survey on mobility issue in context with mobile cloud environment. The author analyzed the Systematic Mapping Study (SMS) strategies performed by various researchers to handle the mobility in mobile cloud scenario. The mobility strategies considered by the author are as follows: hand-over algorithm, mobility management, QoS support, multi-modal support, proxy support and so on. Silva *et al.* [218] addressed the problem of mobility and poor QoS in SD-HCN. The author presented a mobility control scheme called as Context-Aware Mobile Approach (CAMA) to resolve the issue of mobility in e-healthcare services. The author considered the handover mechanism to enable the mobile nodes best connected. The proposed framework uses the Fuzzification system model for handover prediction based on current eHealth session. The performance of the CAMA approach helps to achieve

Table 2.12: Related work on SD-HCN.

Authors	Techniques	Controller	Setup Description	Prog. Lang./ Simulator	Issues Considered				Parameters Improved				
					1	2	3	4	5	6	7	8	9
[205]	M/G/1 Queuing Model, Priority Scheme	centralized	4 nodes; 10-100 Mbps data rate; 1000 B packet size; Exp.1, Exp.2, Exp.3: voice (high), video (medium), data (low). Reduces the avg. waiting time and cost	Matlab	✓	×	✓	×	✓	✓	✓	×	×
[206]	POF-based source routing	centralized (POX)	2 hosts; 2 core switches; 2 edge switches; 1 controller. Minimizes overall path setup latency	Mininet	✓	×	×	×	×	✓	✓	×	×
[209]	Kerberos authentication	centralized	15 switches; 20 routers; 10 Mbps link capacity; 300 MB message size. Improves latency and defend from security threats	Matlab, ns-3	×	✓	×	×	×	✓	✓	✓	✓
[210]	Two-blind signature SM, Bloom Filter	centralized	1024-bit RSA; SHA-256; **Coarse Gained (50 and 100 symptoms)**: avg. CPU loads are 16.4% and 17.3%; avg. power consumption are 2205.2 mW and 2502.1 mW; **Fine Gained (50 and 100 symptoms)**: avg. CPU loads are 15.7% and 17.6%; avg. power consumption are 2126.3 mW and 2457.7 mW	Java	×	✓	✓	×	✓	×	✓	✓	✓
[211]	Bayesian inference	Distributed (ODL)	12 HC organization; 50-100 devices; 245 rules. **Controller**: Normal- avg. CPU workloads 15.3%; adversary- avg. CPU workloads 33.7%. **Switch**: Normal- avg. CPU workloads 6.8%; adversary- avg. CPU workloads 18.4%	Snort IDS	×	✓	✓	×	✓	×	×	✓	✓
[213]	Attack signature, behavior-based IDS	distributed (ONOS)	Dataset: UNSW-NB; arduino uno; voltage regulator IC; Total Power: 6000 mAH, 5V battery; ESP8266 WLAN chip. Detect abnormal behavior and avoids congestion	OpenWrt	✓	✓	×	✓	✓	×	×	✓	✓
[216]	Alarm treatment control strategy	centralized	**Emergency Traffic**: Reduced latency from 26 ms to 17 ms and minimize latency upto 33%. **Normal Traffic**: minimum latency is 0.012 ms and reduced latency by upto 50%	Mininet	✓	✓	✓	×	✓	✓	×	✓	×
[218]	CAMA, fuzzy logic	centralized	Decreases E2E delay and minimize the overall packet loss	-	×	×	✓	✓	✓	✓	×	×	✓

Note- 1: Routing, 2: Security, 3: Resource Scheduling, 4: Mobility, 5: Cost, 6: Latency, 7: Bandwidth, 8: Resilient from Security Attacks, 9: Packet loss/Accuracy

Notations- ✓: considered, and ×: not-considered

reliability in mobility issue by improving the point of attachment and handover decision. Furthermore, the proposed framework reduced the E2E delay and minimizes the packet loss at the time of content delivery.

The summarized overview of the existing work in SD-HCN is shown in Table 2.12.

2.9 Open Issues and Future Directions

The open issues and future directions related to control plane optimization on software-defined networking for a multi-cloud environment are discussed below.

- **Scalability:** Scalability for the control plane optimization is one of the issues in software-defined networking. The scalability issue is concerned with the isolation of the control and data planes. The major bottleneck with the control plane is the enormous traffic load to execute millions of flows per second. The overhead with the control plane is flow setup latency, dynamically increase the network controllers and their placement, data distribution among distributed controller instances, and large flow table size. The reason for the issue of scalability at the control plane is due to an increase in the network size by deploying virtual network resources. In addition, space limitations in forwarding tables and an increase in IP security entities for imposing the defense against security threats arise the scalability issues. A single network controller has limited resources such as– bandwidth, computing power, and memory for effective network management. Therefore, in order to improve network scalability, it is essential to implement multiple distributed controllers.

- **Energy Management:** Another issue that arises on the data plane is high energy consumption on OpenFlow switches. The energy consumption in SDN depends on the number of switches, switch ports, link utilization, traffic flow, and update in the flow table entries. The energy consumption increases due to excessive placement of virtual machines, improper utilization of resources, increase the number of CPUs, increase the level of security attacks, modifying the flow rules, and switching the nodes.

- **Fault Tolerance & Resilience:** Due to their centralized architecture, the issue of scalability alongwith resilience (to overcome the single point of failure) are still major concerns, which need new solutions from the research communities. In a distributed controller environment, the issue of cascading failure generally occurs when a specific controller gets failed then the network is overloaded and leads to packet loss. The consideration of failure recovery is important to create the backup of the network load at the time of controller failure. The issues of resilience in SDN can be addressed by deploying multiple controllers at the control plane such that the data plane works smoothly in case of failure of a single network controller. It is crucial to find a minimal number of active or

primary controllers and a minimal number of backup controllers by using some heuristic approaches. It can be used to achieve a high-performance gain in large-scale networks. It extends the control plane functionality by improving scalability, reliability, efficiency, latency, redundancy, and data consistency.

- **Load Balancing:** In a distributed controller architecture, switches are mapped to their reference controllers. During the peak load, some of the switches receive mice flow while others may receive an elephant flow. This unpredictable and dynamic nature of the incoming traffic flow creates an issue of load imbalance at the control plane. Therefore, Load Balancing (LB) is required in order to migrate the load among different controllers resulting in a balanced state of the network from an overloaded controller to an underloaded controller.

- **CAPEX & OPEX:** The capital expenditure and operational expenses are still open issues in SDDC. Thus, in order to find a minimal number of active and backup controllers, the CAPEX and OPEX is an important consideration that affects the overall network cost.

- **Security:** Also, it is required to design a secure and dependable access control policies on the forwarding nodes (OpenFlow switches) such that they are capable of detecting anomalous behavior of the incoming flow. The flow statistics may reveal anomalies on the OpenFlow switches which may result in various malicious threats. In the era of the IoT, one of the threats, i.e., botnets can generate Distributed Denial-of-Service (DDoS) attacks.

- **Mobility & Wireless:** The research challenges with mobility and wireless communication on the data plane of SDN are spectrum limitations, multipath propagation, QoS guarantee for the real-time applications (video calling, file transfer, live broadcasting), energy limitations, device mobility (area coverage, device velocity, etc). The SDN-based wireless networks are wireless local area networks (WLAN), and cellular networks. The most critical open issues with wireless communication are the increased demand for bandwidth, dynamic traffic patterns, inter-cell interference, and intra-cell interference. The issues with the device mobility are poor signal strength as connection continuity becomes difficult for moving users.

- **Interoperability and Consistency:** The control plane generally facing the research challenge of interoperability between distributed controllers and the eastbound and westbound application programming interface. Examples of programming languages are Pyretic, SDNi, ForCES interface, etc. It is essential to ensure application portability and interoperability in multiple domains, multiple layers, multiple vendors' equipment.

- **Routing & Flow Forwarding:** The network performance can be improved by designing an efficient routing scheme in SD-IoT. It is a core concern to address the issue of overflow

in OF messages generated at the controller and insufficient flow table memory and computes a solution for fast flow forwarding. The issue of flow table memory limitations can be resolved by using probabilistic data structures for space efficiency. The probabilistic data structures are designed column-wise to store the flow entries in SRAM rather than TCAM and returns the error probability of less than 1%.

- **Availability of Controllers:** Another core concern is the availability of network controllers. The nodes or switches in the network should be reachable to the network controller to process the smooth execution of traffic flow. The farthest switch linked to the network controller is facing the issue of high network latency and controllerless nodes that affects network performance. Thus, it is an important concern to decide the optimal number of controllers and their location to ensure high QoS with minimum latency.
- **Limited Flow Space:** One of the research challenges is limited memory space to store the flow table entries. The flow tables are stored in TCAM and the capacity of the chip is around 4000-32000 flow entries i.e., around 133 million flow lookup/second. However, TCAM is an expensive and power-hungry storage device. Hence, it is essential to apply compression, wildcard matching, and labeling methods to reduce the flow space entries.

2.10 Research Gaps

After the detailed analysis of the aforementioned existing proposals with respect to the current status of the DCs, the following research gaps have been identified.

- **Vision of traffic engineering at the DCs is moving at a slow pace**. With an increase in the number of cloud IP traffic and DCs IP traffic globally, the vision for enabling the network traffic to flow smoothly through the underlying network infrastructure is still not fully addressed. The traffic resides within the DCs, inter-DC traffic, and the end-users connected to the DCs through the Internet is growing at a rapid pace but the dynamic approach to handle such high volumes of traffic is moving at a slow pace. A lot of studies and projects are going on world-wide but still, a lot needs to be done.
- **Focus on reducing operational expenditure and capital expenditure rather than making DCs self-sustainable**. The focus of the majority of the existing studies was only on operational and cost minimization of DCs rather than making DCs self-sustainable. None of the existing works attempted to ease the network load in a multi-cloud environment.
- **Less focus on energy-efficient based fast flow forwarding**. Existing studies utilized flow forwarding as a primary source to perform data forwarding. None of the work tried to propose a fast flow forwarding based on an energy management scheme to sustain DCs with minimum energy consumption.

- **Inefficient utilization of resources and servers in DCs**. Even after vast research in cloud computing worldwide, the over-provisioning of resources is a common practice and there are various DCs that are under-provisioned. This leads to inefficient utilization of resources which increases energy consumption and operational cost of DCs.

- **Less focus on multiple active controller assignment**. While deploying multiple SDN Controllers, the decision of the location of the active SDN Controllers is moving at a slow pace. The optimal decision helps to maximize the number of nodes controlled by each active controller. This issue of the controller assignment will lead to an increase in the average control latency.

- **Less focus on virtual hypervisor placement to maximize resource utilization**. With the network virtualization, the problem of deciding where the hypervisors locations should be present and how many hypervisors instances are needed to give optimal performance is still not addressed in multi-controllers networks.

- **No optimal trade-off scheme for fault-tolerance at the control and data plane**. The existing proposals fail to achieve fault tolerance in a heterogeneous multi-cloud DCs. The problem of failure recovery at the data plane, control plane, and at the controller creates a big issue as SDN is a centralized architecture with distributed controllers.

- **Heterogeneous nature of servers not addressed while proving energy-QoS aware schemes for DCs**. The existing proposals considered homogeneous nature of physical servers apart from lacking in providing global QoS. But, this may not be true for each and every DC such as National DCs which allow different servers configurations apart from allowing the addition of servers if required to provide services. Hence, it implies that the minimum number of active servers does not guarantee a minimum consumption of energy.

- **No focus on IP routing security at the control and data plane**. At the control and data plane links, high risks of security attack may arise which is still not resolved.

2.11 Objectives of the Research Work

After analysis of existing proposals, and the research gaps identified following objectives are proposed in the proposal.

1. To design an optimal load balancing scheme to handle the data traffic at the SDN control plane.

2. To design an energy-efficient flow table forwarding scheme for the data plane that can perform flow balancing.

3. To design an efficient fault-tolerant framework for the controller placement problem at the SDN control plane.

4. To validate the proposed solutions by selecting various performance evaluation metrics in different scenarios.

2.12 Research Methodology for objectives

2.12.1 Methodology for objective 1

This objective aims towards designing an optimal load balancing methodology at the control plane.
The methodology adopted to achieve this objective are as follows.

- To minimize the average response time between the switch and the controller, we formulated the load balancing problem as a Mixed Integer Non-Linear Programming (MINLP). Then to solve this problem, a load-balancing algorithm is designed to manage the flow traffic between OpenFlow switches so as to have a trade-off between switch migration cost and the controller's load variance.

- An SDN framework is proposed, in which an OpenFlow-based anomaly detection scheme is designed to identify the suspicious activities on the forwarding nodes.

- An anomaly mitigation scheme is also designed based on access control policies defined for each user's behavior.

- The proposed Load Optimization and Anomaly Detection Scheme for Software-Defined Networks (LOADS) is evaluated on Internet topology Zoo simulated on Mininet Emulator using the POX Controller.

2.12.2 Methodology for objective 2

This objective aims to design an energy-efficient flow table forwarding scheme for the data plane that can perform flow balancing. DC consists of various physical servers, storage devices, and network devices. The centralized architecture of the SDN controller helps to maximize the redundant utilization of links and nodes in order to route the delay-sensitive traffic through the optimized route as per their flow type. The network controller in SDN consolidates the traffic by shutting down or hibernating the idle resources and thus, minimizes the total operational cost. The limitations of the traditional datacenters are that they provide limited services to the end-users in terms of bandwidth and storage facility, high latency, manual device configuration, and traffic congestion. SDDC is a network programmable to provide intelligence analytics for

infrastructure and workload provisioning. Thus, SDN efficiently addresses all the issues of traditional datacenters. The steps involved to accomplish this objective are listed below.

- The Energy-aware Routing (EAR) problem is formulated by using an MINLP for which we proposed FIFO-PO and FIFO-POP scheduling schemes to minimize the average waiting time.
- For energy-efficiency, an efficient flow re-routing and link utilization scheme named as Energy-Efficient Fast Flow Forwarding Scheme for Software-Defined Networks (En-Flow) are proposed.
- Then, to build the shortest path between source and destination, an ant colony routing algorithm is proposed.
- Finally, the proposed scheme is validated using the datasets of NorthAmerica on the OMNET++ platform.

2.12.3 Methodology for objective 3

This objective aims to design an efficient fault-tolerant framework for the controller placement problem (CPP) at the SDN control plane. The issue of scalability and resilience in SDDC can be addressed by deploying multiple controllers at the control plane such that the data plane works smoothly in case of failure of a single network controller.
The methodology adopted to achieve this objective are as follows.

- To resolve the issues of CPP, a problem is formulated in the form of Mixed Integer Non-Linear Programming (MINLP). Then, Placement, Availability Resilient, Controller (PARC) Scheme for Software-Defined Datacenters is proposed which divides the CPP into different sub-problems, i.e., network partitioning, the controller's availability, the minimal number of primary active controllers, and the minimal number of backup controllers required for resilience. The proposed scheme uses cooperative game theory to compute a stable network partitioning and an optimal position of the distributed controllers in each sub-network.
- The overall network cost of the proposed scheme is minimized by finding the minimal number of distributed controllers along with the extra backup controllers using a heuristic approach.
- Moreover, the PARC scheme performs global data consistency of the updated events among multiple controllers based on timestamp at the control plane, which provides synchronization among different events.

- The Pareto Optimal COntroller (POCO) toolset is used to simulate the numerical results in Matlab. The performance analysis compares the proposed PARC scheme with the other existing state-of-the-art schemes such as Pareto Optimal COntroller based on the Simulated Annealing (POCO-SA), Pareto Optimal COntroller Multi-Objective Ant Lion Optimization (POCO-MOALO), and Capacitated Next Controller Placement (CNCP) scheme.

2.12.4 Methodology for objective 4

The proposed schemes are tested for its accuracy and efficiency using realistic parameters and data traces. Since the target scheme comprises of a large number of DCs with variable user requests, so the proposed LOADS, EnFlow, and PARC schemes are simulated on the Mininet, OMNET++, and Matlab tool to demonstrate the performance evaluation. The Mininet simulator used to develop, share, and experiment with OpenFlow and Software-Defined Networking system. Further, the realistic data traces of Internet Zoo Topology from BTNorthAmerica zone, fat-tree 4-ary network topology, and Internet2 OS3E topology using POCO-toolset are collected from Quality of Web Service (QWS) dataset, Google traces, etc. The performance analysis of the proposed schemes outperforms the existing schemes in terms of the following parameters such as– execution time, migration cost, response time, energy-efficiency of the network, inter-controller latency, and switch-to-controller latency, and controller deployment cost.

2.13 Summary

Although traditional networking is widely adopted in WANs there are various challenges in network management. The major issues that occurred in traditional networking are the strong coupling of the data (forwarding devices) and the control (network brain) plane. The vertically integrated planes create a static and manual configuration of devices. Also, conventional networking is vendor-specific which creates much complexity in the management of WANs. The key features of the traditional networking lead to error-prone, data inconsistency, and huge network delay. SDN is a promising and innovative technology widely implemented in the digitalization world. The key features of SDN enabled decoupling the control part with the forwarding plane. The automation is achieved in network management through a programmable SDN network. In addition, SDN is a dynamic approach to modify the network topology as the network load increases. The major benefits of deploying SDN are scalability, reliability, and flexibility. The literature survey on SDN is different from the existing surveys as the comparative analysis is done on the existing work. The comparison of the existing survey with the proposed survey considered various parameters such as–SDN components, SDN issues, network virtualization, and SDN deployment in various real-time applications.

The literature survey on SDN is basically divided into three parts. The first part of the survey discussed the history and architecture of SDN in detail. SDN history is analyzed on the basis of year-wise improvement and innovation in SDN components. Next, the architecture components of SDN such as– SDN switches, controllers, interfaces, emulation, and simulation tools are analyzed. The second part of the survey discussed realtime deployment applications of SDN. The deployment applications considered in the survey are SD-DC, SD-FC/EC, SDIoT, SDWSNs, and SD-WBAN. The comparative analysis of the existing survey on SDN applications is performed on the basis of problem formulation, techniques proposed, and the parameters that improved the overall network performance. Finally, the last part of the survey discussed the open issues and research challenges in the SDN ecosystem with respect to smart applications.

Chapter 3

LOADS: Load Optimization and Anomaly Detection Scheme for Software-Defined Networks

The major research contributions of this chapter are summarized below.

- To minimize the average response time between the switch and the controller, we formulated the load balancing problem as a Mixed Integer Non-Linear Programming (MINLP). Then to solve this problem, a load-balancing algorithm is designed to manage the flow traffic between OpenFlow switches so as to have a trade-off between switch migration cost and the controller's load variance.
- An SDN framework is proposed, in which an OpenFlow-based anomaly detection scheme is designed to identify the suspicious activities on the forwarding nodes.
- An anomaly mitigation scheme is also designed based on access control policies defined for each user's behavior.
- The proposed LOADS scheme is evaluated on Internet topology Zoo simulated on Mininet Emulator using the POX Controller.

Chapter 3 is organized as follows. Section I presents the System model. Section II discusses in detail the proposed LOADS scheme. Section III provides the performance evaluation and finally, Section IV concludes the paper.

3.1 System model

This section is divided into two parts 1) SDN architecture and 2) A MINLP-based problem formulation.

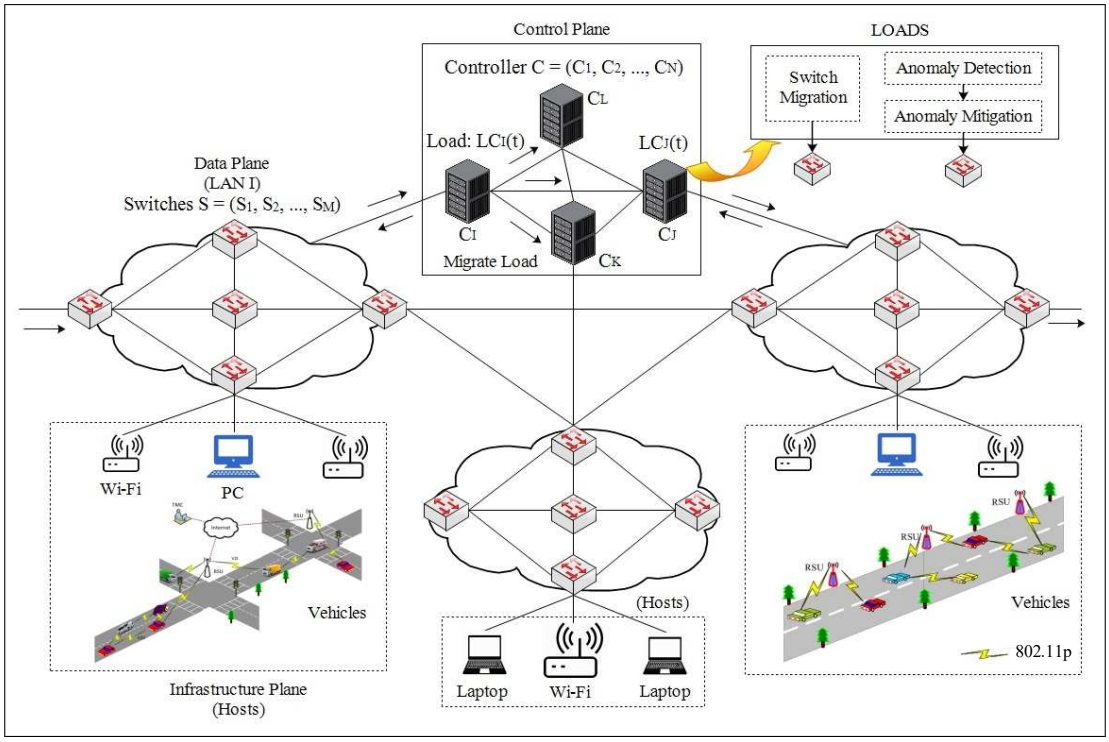

Figure 3.1: System model with multiple controllers deployed in a large-scale network.

3.1.1 SDN Architecture

SDN is modeled as a directed graph $G \in (V,E)$, where G is a collection of V nodes, and $e \in E$ are the links. Here, $V = (C,S)$ is a tuple with C as the set of controllers and S is the set of switches. Fig. 3.1 presents a multi-graph representation of multiple controllers deployed in a large-scale network consisting of three local area networks (LANs). Let $C = \{C_1, C_2, ..., C_I, ..C_N\}$ be the set of N controllers at the control plane and $S = \{S_1, S_2, ..., S_M\}$ is the set of M switches controlled by any controller C_I, wherein, each controller is having the same processing capacity Θ_C with an equal number of switches assigned to each controller.

Suppose, controller C_I controls LAN I with a set of S switches and the controllers C_J and C_K control LANs J and K respectively. Here, all the N controllers are connected in the flat model (Peer-to-Peer) in which any modification inside any controller must be broadcasted to all the controllers to maintain the global network state. The switches at the data plane serve multiple hosts (vehicles) at the infrastructure plane by using wireless connections via Wi-Fi using 802.11p or wired connections with Ethernet. The issues of collision occur as some of the links may be overloaded due to heavy traffic while the others may be underloaded. It leads to an increase in the response time for flow setup requests with the controllers.

So, to improve the response time, the proposed *LOADS* scheme is executed on the control plane to manage the load.

3.1.2 Problem Formulation

In order to address the aforementioned objectives, in this paper, the MINLP problem is formulated, which consists of input data, set of decision variables, set of constraints, and an objective function in order to execute the model. The input data is defined using the following five tuples: $\langle C, S, t,\ \Theta_C, \lambda_G(t)\rangle$, where t is the time interval of a frame, Θ_C is the threshold of C load to handle the flow setup requests, and $\lambda_G(t)$ is flow arrival rate (flows per second) at a given switch S_G ($S_G \in S$ of M switches) in time slot t which follows the Poisson process. In case, the flow arriving at switch S_G is fixed then λ is computed as: $\lambda_G(t)$ = (number of flow count/time slot) but the flow arrival follow random process whose probability (n arrivals in interval t) = $(\lambda_G(t)^n e^{-\lambda_G(t)}/n!)$ is computed using the Poisson process.

The decision variables of the model are defined as follows.

$$P_{GI}(t) = \begin{cases} 1, & \text{if controller } C_I \text{ is designated to switch } S_G, \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

The load on each controller C_I is calculated on the basis of the aggregated flow transmitted from S_G to C_I, if controller C_I is assigned to switch S_G at time t. Hence, it is represented as follows.

$$LC_I(t) = \sum_{G=1}^{M} \lambda_G(t) P_{GI}(t), \tag{3.2}$$

where $LC_I(t)$ should not exceed the processing capacity of controller Θ_{C_I} at time t. The aggregate load of the additional switch $L_{S_H}(t)$ added to the C_I controller after migration should not exceed the threshold limit of the controller Θ_{C_I}. Hence, the equation is defined as below.

$$LC_I(t) + L_{S_H}(t) \leq \Theta_{C_I}. \tag{3.3}$$

The aggregated response time of each controller C_I to serve the flow setup request is computed as below.

$$\mu_I(t) = \frac{LC_I(t)}{\Theta_{C_I} - LC_I(t)}. \tag{3.4}$$

So, the total load of all controllers $LC_{all}(t)$ is calculated as below.

$$LC_{all}(t) = \sum_{I=1}^{N} LC_I(t), \tag{3.5}$$

where $LC_{all}(t)$ should not exceed the processing capacity of all the controllers, i.e., $\Theta_{C_{all}}$. Thus, the mean response time of the controller C_I deployed to serve all the flow requests is the ratio of the aggregated response time of controller C_I to the flow arrival rate of all the switches

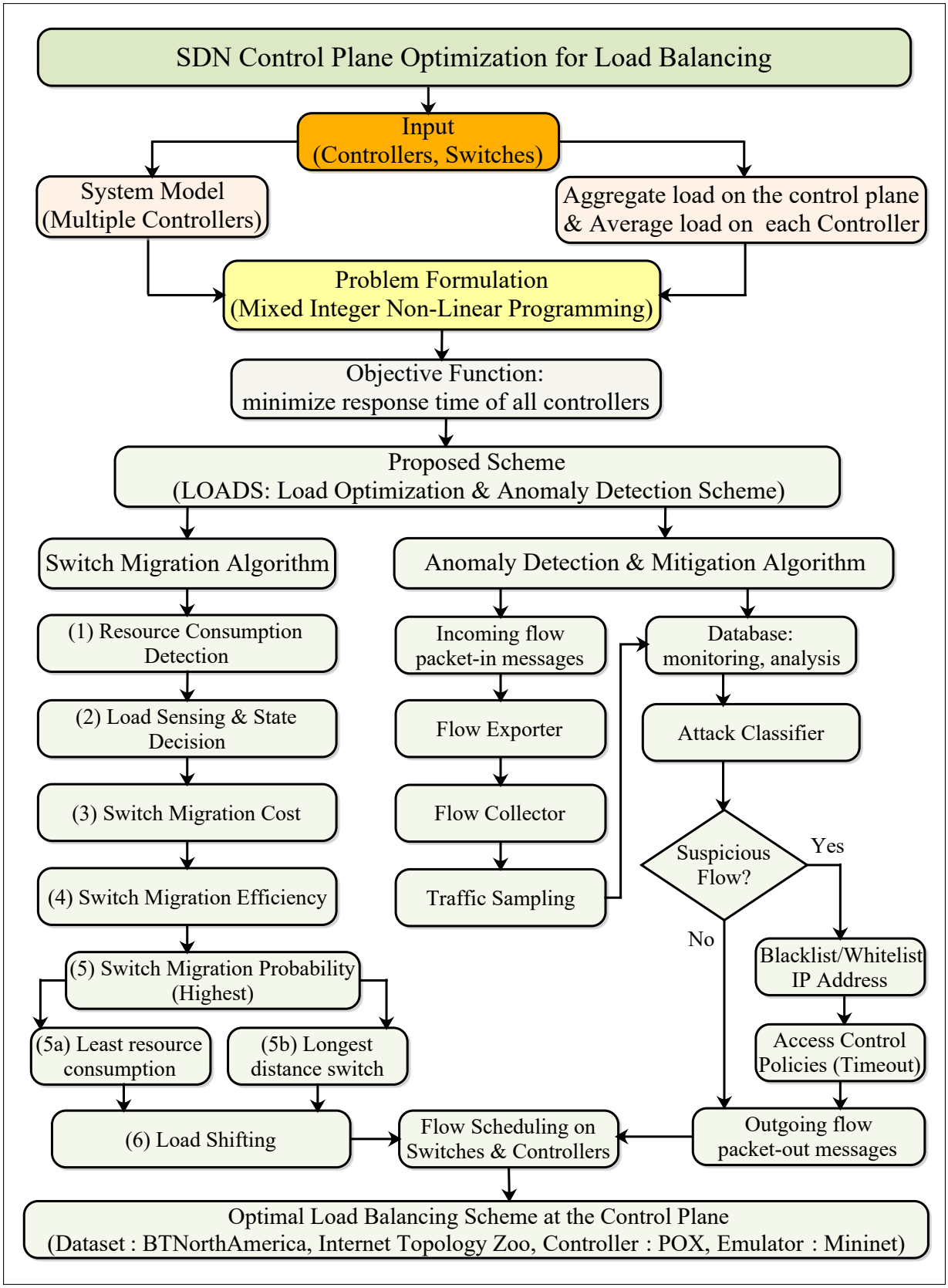

Figure 3.2: Proposed scheme LOADS: Load Optimization & Anomaly Detection Scheme for Software-defined networks.

is computed as below.

$$\overline{\mu}_I(t) = \frac{\mu_I(t)}{\sum_{G=1}^{M} \lambda_G(t)}. \tag{3.6}$$

Moreover, a flow-based anomaly detection, and mitigation scheme is used to check the behavior of each flow request served at OpenFlow switch. The traffic sampling mechanism is used to classify the behavior of flooding attacks on the network for inspection of the severity of flow traffic. Traffic sampling involves the detection and inspection of suspicious IP-flow requests such that the aggregate flow request does not exceed the processing capacity of the Anomaly Detection System (ADS). Let the capacity of the ADS be Q then the flow arriving at OpenFlow switches can be sampled in the following manner: (a) If traffic sampling is at only one switch, then the sampling is done at the rate equal to the capacity Q of the system, (b) But, if traffic sampling is at multiple switches, then the capacity is equally divided among all the switches which are an individual sampling rate at switches.

Let $d_f = [d_{f_1}, d_{f_2}, ..., d_{f_n}]$ denotes the data rate of the flows then the data rate of the switches is defined as follows: $d_s = \chi(\lambda) d_f$. Here, $\chi(\lambda)$ represents the flow status of each active OpenFlow switch which is computed by flow information matrix in Anomaly Detection System (ADS) at the Data Plane (Subsection 3.2.2.3). Suppose, β_i is the rate of sampling of switch i, $0 \leq \beta_i \leq 1$, and d_{s_i} is the data rate of switch i. Therefore, in order to efficiently utilize the ADS, the aggregate flow traffic allowed by the system should not exceed the capacity of the ADS to work efficiently. Hence, the aggregate flow traffic is defined as below.

$$\sum_{I=1}^{M} \beta_i d_{s_i} \leq Q. \tag{3.7}$$

The objective function of the *LOADS* scheme is to minimize the response time of all the controllers at the control plane. Mathematically, it can be represented as follows.

$$\min_{P_{GI}, \beta_i} \sum_{I=1}^{N} \overline{\mu}_I(t), \tag{3.8}$$

$$\begin{aligned}
s.t. \quad & \mathbf{C1} \;:\; 0 < LC_I(t) \leq \Theta_{C_I}, \;\; \forall I, \\
& \mathbf{C2} \;:\; 0 < LC_{all}(t) \leq \Theta_{C_{all}}, \;\; \forall I, \\
& \mathbf{C3} \;:\; \sum_{I=1}^{M} P_{GI}(t) = 1, \;\; \forall M, \\
& \mathbf{C4} \;:\; P_{GI}(t) \in \{0, 1\}, \;\; \forall M, \\
& \mathbf{C5} \;:\; LC_I(t) + L_{S_H}(t) \leq \Theta_{C_I}, \;\; \forall I, \\
& \mathbf{C6} \;:\; \sum_{I=1}^{M} \beta_i d_{s_i} \leq Q, \;\; \forall M.
\end{aligned}$$

Constraints **C1** and **C2** state that the load of each controller or a total load of all controllers should not exceed the threshold limit of the controller processing capacity such that no con-

troller is overloaded. **C3** defines that all the M switches assigned within a single LAN must be assigned to exactly one controller C_I at time t. **C4** represents that the value of the binary variable M is either 0 or 1 for all the switches and controllers. **C5** states that the additional load of the switch S_H added to the C_I controller after migration should not exceed the threshold limit of the controller C_I. **C6** constraint states that the aggregate flow traffic allowed by the system $\beta_i d_{s_i}$ should not exceed the processing capacity Q of the ADS in order to efficiently utilize the system. Fig. 3.2 presents the proposed scheme LOADS: Load Optimization & Anomaly Detection Scheme for Software-defined networks.

3.2 LOADS: Load Optimization and Anomaly Detection Scheme

The proposed *LOADS* consists of two components (a) switch migration, (b) anomaly detection and mitigation. The switch migration based dynamic load balancing is designed on the control plane while the anomaly detection and mitigation is deployed on the data plane. Fig. 3.2 presents the proposed scheme named as LOADS: Load Optimization & Anomaly Detection Scheme for Software-defined networks. Both of these are described in details as below.

3.2.1 Switch Migration Scheme on the Control Plane

The dynamic load balancing scheme is executed on the distributed controllers at the control plane. In the proposed scheme, the switch migration is performed from the overloaded controllers to the underutilized controllers to satisfy the **C1**-**C5** constraints. It is divided into six modules (1) resource consumption detection, (2) load sensing and state decision, (3) switch migration cost, (4) switch migration efficiency, (5) switch migration probability, and (6) load shifting. The detailed explanation of each module is discussed below.

3.2.1.1 Resource Consumption Detection

The resource consumption is known as the fraction of resources consumed at the control plane in terms of three parameters, i.e., bandwidth, CPU cycles, and memory denoted by *b, c, m* respectively. The approximate resource consumption ratio of all the controllers $\widetilde{\zeta}_{all}$ is defined as below.

$$\widetilde{\zeta}_{all} = \sum_{I=1}^{N} \widetilde{\zeta}_I, \quad \widetilde{\zeta}_I = \sum_{G=1}^{M} \widetilde{\zeta}_{GI}, \tag{3.9}$$

where $\widetilde{\zeta}_I$ is the approximate resource consumption ratio of controller C_I and $\widetilde{\zeta}_{GI}$ is the approximate resource consumption ratio of the G^{th} switch assigned on controller I.

The resource consumption for the three parameters, i.e., bandwidth, CPU cycles, and memory denoted by $\Omega_I^b, \Omega_I^c, \Omega_I^m$ of C_I is computed as below.

$$\Omega_I^b = \frac{\sum_{G=1}^{M} d_G b_G}{\eta_I^b}, \quad \Omega_I^c = \frac{\sum_{G=1}^{M} d_G c_G}{\eta_I^c}, \quad \Omega_I^m = \frac{\sum_{G=1}^{M} d_G m_G}{\eta_I^m}. \tag{3.10}$$

Here, b_G, c_G, m_G represents the approximate bandwidth, CPU, and memory usage respectively per activity of C_I generated by the G^{th} switch. Additionally, d_G is defined as the activity demand generated by the G^{th} switch on its controller, while $\eta_I^b, \eta_I^c, \eta_I^m$ represents C_I processing capability for each respective parameters.

Now, let us assume that ${f_I}^b, {f_I}^c, {f_I}^m$ are the weights on C_I for the three parameters respectively. So, as per the current state of resource usage for C_I, the combined fraction of resources consumption is always equal to 1, i.e., ${f_I}^b + {f_I}^c + {f_I}^m = 1$. Here, the weight of each parameter is inversely proportional to its individual present resource consumption ratio, which is defined as below.

$$ {f_I}^b = \frac{1/\Omega_I^b}{\sum 1/\Omega}, \quad {f_I}^c = \frac{1/\Omega_I^c}{\sum 1/\Omega}, \quad {f_I}^m = \frac{1/\Omega_I^m}{\sum 1/\Omega}, \tag{3.11} $$

where $\sum 1/\Omega = 1/\Omega_I^b + 1/\Omega_I^c + 1/\Omega_I^m$. The approximate resource consumption ratio of G^{th} switch on its controller C_I depends upon f_I and Ω_I, which is calculated as below.

$$ \widetilde{\zeta}_{GI} = lg\left[{f_I}^b \left(\frac{d_G b_G}{\eta_I^b} \right) + {f_I}^c \left(\frac{d_G c_G}{\eta_I^c} \right) + {f_I}^m \left(\frac{d_G m_G}{\eta_I^m} \right) \right]. \tag{3.12} $$

The approximate resource consumption ratio of C_I calculated above checks the $C3$ constraint. The resource consumption ratio of C_I is the aggregate value of the approximate resource consumption in terms of the three parameters.

3.2.1.2 Load Sensing and State Decision

The load sensing sub-module measures the load diversity of each controller on the control plane. The state decision provides an indication if switch migration is required or not. Load diversity is a measure of the peak load on the different controllers. Load diversity is measured on the basis of *diversification factor*.

The *diversity factor* D_F measures the efficiency of all the distributed controllers in managing the entire network devices. It is defined as below.

$$ D_F = \frac{\sum_{I=1}^{N} LC_I(t)}{\sum_{I=1}^{N} \Theta_{C_I}}. \tag{3.13} $$

Here, all the distributed controllers are connected in a peer-to-peer manner with each other. In the above computation, the term D_F is the ratio of the sum of the aggregate load of all the controllers during peak traffic to the total load capacity of all the controllers. The diversity factor is measured for the entire network to check if the aggregate load of all the controllers should not exceed the processing capacity of all the controllers (C2 Constraint). The load diversity matrix represents the load diversity between any two pairs of controllers in the entire network. The load imbalance information between two controllers is broadcasted to all the controllers by using the east/westbound application programming interfaces.

Load diversity detects the efficiency of a controller in managing the load at a particular time interval t and is checked between any two pairs of controllers C_I and C_J denoted by $D_{C_IC_J(t)}$ which is the deciding factor for switch migration. Hence, for each controller, the load diversity helps to categorize if the controller is under-utilized or over-utilized. Hence, $D_{C_IC_J(t)}$ is computed as follows.

$$D_{C_IC_J(t)} = \frac{LC_I(t)}{LC_J(t)}. \tag{3.14}$$

So, the controller load diversity matrix represents the load diversity between two controllers located at the control plane as shown below.

$[D_{C_IC_J(t)}]_{N\times N}$ =

$$\begin{bmatrix} 1 & D_{C_1C_2(t)} & \cdots & D_{C_1C_I(t)} & \cdots & D_{C_1C_N(t)} \\ D_{C_2C_1(t)} & 1 & \cdots & D_{C_2C_I(t)} & \cdots & D_{C_2C_N(t)} \\ \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\ D_{C_IC_1(t)} & D_{C_IC_2(t)} & \cdots & 1 & \cdots & D_{C_IC_N(t)} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ D_{C_NC_1(t)} & D_{C_NC_2(t)} & \cdots & D_{C_NC_I(t)} & \cdots & 1 \end{bmatrix}$$

The diagonal elements of the matrix are 1 because $LC_I(t)/LC_I(t) = 1$. Now, the condition for switch migration to occur using the load diversity between any two controllers C_I and C_J is that the load diversity of the given two controllers should be greater than the threshold Θ_0 of the load diversity.

$$if(\exists\, D_{C_IC_J(t)} > \Theta_0). \tag{3.15}$$

The state decision consists of following tuples for switch migration, i.e., (C_S, C_{im}, C_D, S_H). Here, C_S is the source out-migrating or overloaded controllers. C_{im} is the set of underutilized controllers whose load is less than the threshold value, where C_D is the destination controller selected from the set of available intermediate controllers C_{im}. The last tuple S_H is a switch which is chosen to be migrated from the controller C_I to C_J.

3.2.1.3 Switch Migration Cost

The switch migration cost includes the migration cost incurred in migrating switch S_G from C_I to C_J in the following two cases (a) information sharing cost for a specific switch for migration, and (b) the cost of the additional load on the controller. The switch migration cost is time dependent because it aims to migrate to the closest controller instead of any random controller so as to reduce the average response time. Initially, the increment in the load cost $\alpha_{LC_J(t)}$ on controller C_J is calculated using the equation.

$$\alpha_{LC_J(t)} = [LC_J(t) - LC_I(t)], \tag{3.16}$$

where $LC_J(t) = \lambda_G(t).P_{GJ}(t)$ is the existing load on C_J and $LC_I(t)$ is the new load on C_J which is migrated from C_I. Using the above equation (3.16) following holds.

$$\alpha_{LC_J(t)} = \begin{cases} LC_J(t) - LC_I(t), & (LC_J(t) > LC_I(t)), \\ 0, & \text{Otherwise.} \end{cases} \tag{3.17}$$

The switch migration cost $\alpha_{S_G,IJ(t)}$ for S_G from C_I to C_J is calculated as below.

$$\alpha_{S_G,IJ(t)} = \alpha_{LC_J(t)} + \alpha_{S_G(t)}, \tag{3.18}$$

where $\alpha_{S_G(t)}$ is the cost of information sharing for a particular migrated switch S_G.

3.2.1.4 Switch Migration Efficiency

The purpose of switch migration efficiency is to satisfy a $C2$ constraint to have a trade-off between the controller's load variance and switch migration cost. The migration efficiency improves the overall response time of all the controllers during the switch migration. The total response time before migration $\mu_{I,J}(t)$ of S_G on controllers C_I and C_J is calculated as follows.

$$\mu_{I,J}(t) = \mu_I(t) + \mu_J(t). \tag{3.19}$$

So, the total response time after migrating $\mu^{\star}_{I,J}(t)$ of S_G from controller C_I to C_J is given as below.

$$\mu^{\star}_{I,J}(t) = \mu^{\star}_I(t) + \mu^{\star}_J(t). \tag{3.20}$$

Hence, the total response time is defined as follows:

$$\Delta[\mu_{I,J}(t) - \mu^{\star}_{I,J}(t)] = [\mu_I(t) + \mu_J(t)] - [\mu^{\star}_I(t) + \mu^{\star}_J(t)]. \tag{3.21}$$

The change in the total response time before and after switch migration is given as below.

$$\begin{aligned} \Delta[\mu_{I,J}(t) - \mu^{\star}_{I,J}(t)] &= \left[\frac{LC_I(t)}{u[LC_I(t)]} + \frac{LC_J(t)}{u[LC_J(t)]}\right] \\ -\left[\frac{LC_I{}^{\star}(t) - \lambda_{GJ}(t)}{u[LC_I{}^{\star}(t)] + \lambda_{GJ}(t)}\right.&+ \left.\frac{LC_J{}^{\star}(t) + \lambda_{GJ}(t)}{u[LC_J]^{\star}(t) - \lambda_{GJ}(t)}\right], \end{aligned} \tag{3.22}$$

where $u[LC_K(t)] = (\Theta_{C_K} - LC_K(t))$ is defined as the unused controller capacity of any random controller C_K.

So, the controller's load variance before migrating S_G is computed as follows.

$$\sigma^2 = \frac{\sum_{I=1}^{G}[LC_I(t) - \overline{LC}_{all}(t)]^2}{M}, \tag{3.23}$$

where $\overline{LC}_{all}(t) = (LC_{all}(t)/N)$ is the average load of the control plane. Now, after migrating switch S_G from C_I to C_J, the updated load is defined as follows.

$$LC_I^{\star}(t) = (LC_I^{\star}(t) - \lambda_{GI}(t)P_{GI}(t)), \tag{3.24}$$

$$LC_J^{\star}(t) = (LC_J^{\star}(t) + \lambda_{GI}(t)P_{GI}(t)). \tag{3.25}$$

Hence, the load variance after migration is given as below.

$$\sigma^{\star 2} = \frac{\sum_{I=1}^{G}[(LC_I^{\star}(t) - \overline{LC^{\star}}_{all}(t)) + (LC_J^{\star}(t) - \overline{LC^{\star}}_{all}(t))]^2}{M}. \tag{3.26}$$

Therefore, the switch migration efficiency of migrating S_G to controller C_J is the ratio of difference in the controller's load variance to the switch migration cost defined as below.

$$\rho_{S_G} = \frac{\left|(\sigma^{\star 2} - \sigma^2)\right|}{\alpha_{S_G,IJ(t)}}. \tag{3.27}$$

The objective of the migration efficiency model is to achieve maximization efficiency which satisfy constraint $C4$ in order to accomodate load from C_S to C_D.

3.2.1.5 Migration Probability

The migration probability decides the possibility of choosing a specific switch S_H associated to C_S for being migrated to C_D. The outmigrating switch selection is done using the probability distribution function on the basis of two conditions as defined below.

- *Condition 1*: The resource consumption ratio generated by the switch to its controller C_I. The switch with a small load reduces the chances of controller overloading. Thus, the least resource consumption ratio is selected as the highest probability for being migrated.
- *Condition 2*: The distance from the switch to its corresponding controller plays an important role to generate the response time. The farthest switches suffer from long delay and packet loss. Thus, the highest migration probability is chosen for the long-distance switches.

Based on above conditions, the migration probability is the ratio of the probability of the H^{th} switch in a controller C_I to the probability of all the M number of switches in a controller C_I. Therefore, the migration probability is computed as below that satisfies the $C3$ constraint.

$$P(S_H) = \frac{e^{\frac{-\zeta_{HI}}{\delta_{H,I}(1-\zeta_{HI})}}}{\sum_{G=1}^{M} e^{\frac{-\zeta_{GI}}{\delta_{G,I}(1-\zeta_{GI})}}}, \tag{3.28}$$

where ζ_{HI} and ζ_{GI} represents the resource consumption ratio generated by the switch S_H

or S_G on its controller C_I. In addition, $\delta_{H,I}$ and $\delta_{G,I}$ denote the distance from the switch S_H or S_G to its controller C_I and the probability of $P(S_H)$ is calculated by choosing the maximum probability in the sample space of all S_M, i.e., $S_H = \arg\{max[P(S_G)_{S_G \in S}]\}$.

3.2.1.6 Load Shifting Phase

The load shifting phase is executed after migration probability $P(S_H, S_G)$ finalizes the switch selection S_H. In a set of intermediate controllers C_{im}, a controller C_D is chosen as a destination in-migration controller based on the maximum migration efficiency.

Suppose, $L_{S_H}(t)$ be the load of switch S_H which is to be added in each C_I of C_{im}. So, the $C5$ constraint is checked, whether the new load on C_I exceeds the threshold value after migration. Hence, $\forall C_I$ in C_{im}, it is checked that if $(LC_I(t) + L_{S_H}(t) \leq \Theta_{C_I})$, then the migration efficiency ρ_{S_H} of S_H is calculated as follows.

$$\rho_{S_H} = \frac{\left|(\sigma^{\star 2} - \sigma^2)\right|}{\alpha_{S_H, I(t)}}, \tag{3.29}$$

where $\alpha_{S_H, I(t)}$ is the migration cost of S_H on C_I.

Finally, C_D is selected for migration based on the maximum migration efficiency among all C_I using the following.

$$C_D = \arg\{max[\rho_{S_H}]\}. \tag{3.30}$$

The load shifting phase of S_H is executed from C_S to C_D using intermediate controllers C_{im} (Equation (3.28)) which completely satisfies the $C5$ constraint. Thus, the switch migration decision of the updated tuples; $\langle C_S, C_D, S_H \rangle$ are sent to all the controllers at the control plane.

In case, if one or more controller instances fail unexpectedly then the control messages containing the switch migration decision may be lost, which may result the load imbalance among the distributed controllers. So, in this situation, the Forward Error Correction (FEC) mechanism [219] can be used to overcome the burst packet loss executed on each controller. The network controller sends the message to all the OpenFlow switches that contain the original information (block length) including some redundant bits and data rate. Let us assume that in state s at time t, any controller instances fails then in order to recover the original information, the redundant bits are retrieved at time $(t-1)$. The control messages are resent to all the switches till the search process finds another nearest controller mapped to the switches.

Algorithm 1 illustrates the load balancing scheme using switch selection probability distribution. Initially, the load and resource consumption ratio of each controller C_I is calculated. In the next phase, the controller's load diversity is checked, if the load diversity exceeds the threshold capacity of each controller, then the switch migration action proceeds. This returns the set of out-migrating controllers C_S, while others are intermediate controllers from which

Algorithm 1 Algorithm for load balancing.

Input: $C = \{C_1, .., C_N\}, S = \{S_1, .., S_M\}, \Theta_C, p_{GI}(t)$.
Output: (C_S, C_D, S_H).

procedure SWITCH MIGRATION
  while $(C_I \leq C_N)$ **do**
    Calculate $LC_I(t)$ and $\widetilde{\zeta}_{kI}$ of each C_I;
    detect $D_{C_I C_J(t)}$ $\forall (C_I, C_J)$;
    if $(\exists\, D_{C_I C_J(t)} > \Theta_0)$ **then**
      $C_S \leftarrow$ out-migration controller;
      $C_{im} \leftarrow$ intermediate controllers;
      while $(\forall C_I \in C_S)$ **do**
        do Switch Migration (C_S, C_{im}, C_D, S_H);
        for $(S_I = 1; S_I \leq S_M; S_I++)$ **do**
          Compute ρ_{S_G} using $\widetilde{\zeta}$;
          Choose $S_H \leftarrow \arg\{max[P(S_G)]\}$;
          while $(\forall C_I \in C_{im})$ **do**
            if $(LC_I(t) + L_{S_H}(t) \leq \Theta_{C_I})$ **then**
              do Switch Migration();
              Compute ρ_{S_H};
              Choose $C_D \leftarrow \arg\{max[\rho_{S_H}]\}$;
              $C_D \leftarrow$ in-migration controller;
              migrate load (C_S, C_D, S_H);
            else if $(LC_I(t) \in$ overloaded controllers$)$ **then**
              Switch Migration is not possible;
            end if
          end while
        end for
      end while
    else (Switch Migration is not required);
    end if
  end while
  return (C_S, C_D, S_H);
end procedure

the destination controller is chosen. Now, for each switch that belongs to the out-migration controller, the migration probability is calculated. Next using the greedy approach of selecting the switch S_H to be migrated, the switch with maximum migration probability at each step from the set S_G is chosen. Now, the destination controller C_D among C_{im} for migrating the switch load $L_{S_H}(t)$ is selected such that the set of controllers satisfy constraint **C5**. Hence, C_D is selected based on the maximum migration efficiency. The selected destination controller C_D handles the additional load of the out-migrating controller C_S. The final output of the load balancing algorithm broadcasts the updated load, i.e., (C_S, C_D, S_H), and timestamp is sent to all the controllers to maintain global consistency.

The output returned from the switch migration operation results in the dynamic load balancing on the control plane but the problem of stability of the switch-controller mapping arises over time.

Convergence Rate: The mapping between the switch to the controller is many-to-one. From Eq. (6), the load balancing algorithm increases the stability of mapping at time t by taking into consideration the mapping at time $t-1$. Since the system response time has a lower bound due to a limited number of controllers, so the switch migration operation exits when the response time gets saturated after the execution of migration operation at time t. It shows that after a certain number of iterations of the migration process, the switch-controller assignment converges to the final state which is called a stable point. If the load balancing algorithm converges to a final switch-controller assignment then that point is called two-sided exchanged stable mapping. The stable state is achieved when in state s at time t, each switch is assigned to its reference controller in the network with minimum latency.

Complexity Analysis: The time and space complexity of Algorithm 1 is computed as below.

Time Complexity: The *while* loop (step 1-32) takes $O(n^2)$ time in the worst case. Next, the *for* loop iterates n times which takes $O(n)$ time in worst case. After this, the conditional operators takes $O(1)$ time. Therefore, the overall time complexity is computed as follows.

$T(C) = O(n^2) + O(n) + O(1) \Longrightarrow O(n^2)$.

Space Complexity: The input variable of the load balancing mechanism is fixed and cannot exceed n hence it takes $O(n)$ space in the worst case. After this, the loop takes $O(n)$ space and finally, the arithmetic operations take $O(1)$ space. Therefore, the overall space complexity is computed as follows.

$SC = O(n) + O(n) + O(1) \Longrightarrow O(n)$.

3.2.2 Anomaly Detection System (ADS) at the Data Plane

In this sub-section, our aim is to minimize the response time with respect to the constraint **C6**. As shown in Fig. 3.3, the flow-based anomaly detection scheme consists of four modules, (i) threat model (ii) flow exporter and flow collector (iii) flow traffic analysis and anomaly detection (iv) IP address-based anomaly mitigation. The detailed description of these four modules are as follows.

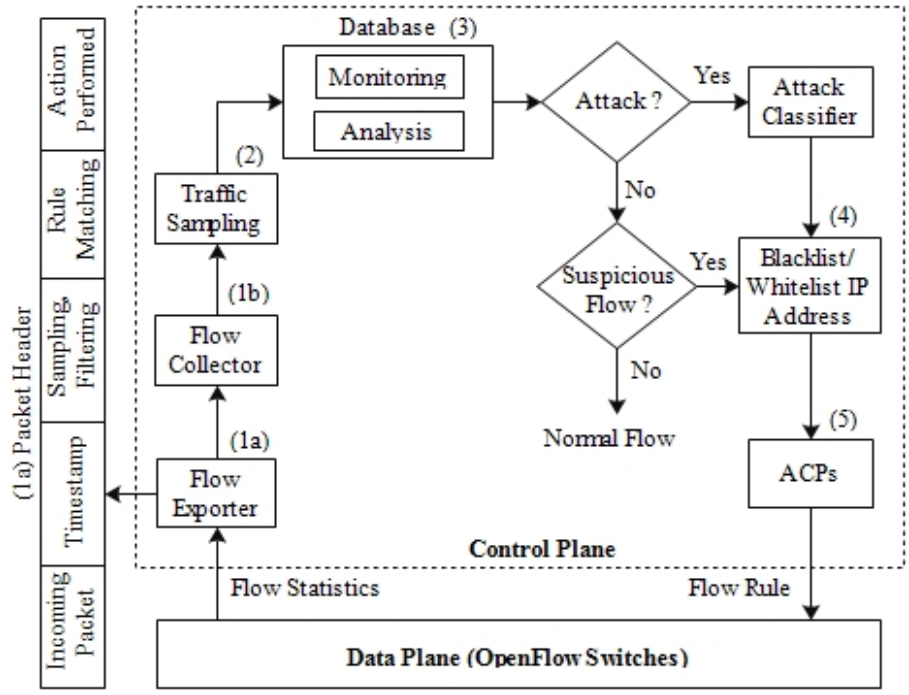

Figure 3.3: Anomaly detection system (ADS) model.

3.2.2.1 Threat Model

The proposed *LOADS* scheme only focused on flooding attacks (DDoS). The reason behind this is that the attacker performs flooding attacks by bombarding unwanted requests from ICMP messages and TCP SYN message by establishing fake handshaking process between the source and destination machine. The attacker targets to crash the victim network devices like OpenFlow switches and network controller. The flooding attacks may lead to the issue of resource exhaustion and bandwidth depletion on the control plane. The bandwidth depletion occurs when the unwanted traffic evacuates the network to prevent legitimate traffic from accessing the allocated network bandwidth. However, the resource exhaustion occurs in two cases (i) *ICMP flooding attacks*: when the attacker source IP request sends high volumes of data (elephant flow) in each flow request, (ii) *TCP SYN attack*: when the attacker source IP requests open a large number of fake connections with the victim OpenFlow switch in order to increase the flow number. Both the issues of bandwidth depletion and resource exhaustion send a large number of unwanted requests on the OpenFlow switches to be served by the corresponding reference controller to prevent installing thousands of flow tables on the OpenFlow switches when the victim network device is under the flooding attack. Thus, the DDoS attack leads to the situation of load imbalance among the distributed controllers. This issue is handled by designing a flow-based anomaly detection and mitigation scheme on the data plane of the SDN framework for reliable traffic flow management.

Fig. 3.2 shows a systematic overview of the flow-based ADS model. The proposed ADS model is divided into two phases (i) step-wise working of the packet transmission in an SDN-enabled OpenFlow switches (ii) anomaly detection and mitigation phase on the data and con-

trol plane. The ADS model is a stateful mechanism used to establish a session between the controller and the switches. The session is created to maintain the information related to logging, caching, or counting the number of packets transmitted per flow, suspicious activity, or any flow mismatch in the database on a regular basis. The packets belonging to a specific flow are captured with a flow header f_H and they consist of the attributes such as– $\langle source\_IP, destination\_IP, source\_port, destination\_port, protocol, counter \rangle$.

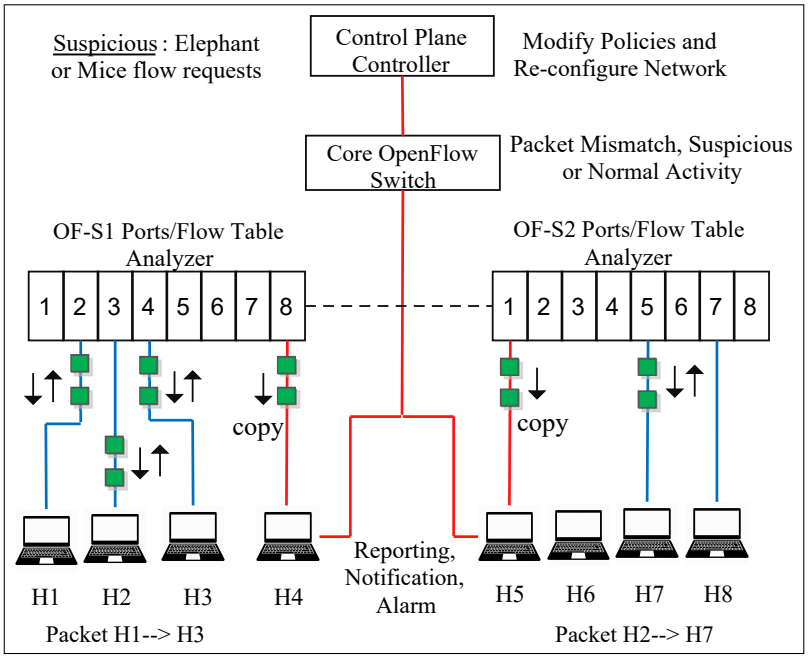

Figure 3.4: IP-flow based OpenFlow switches port mirroring.

3.2.2.2 Flow Exporter and Flow Collector

Fig. 3.4 shows the IP-flow based port mirroring process between two distribution OpenFlow switches (OF-S1, OF-S2). Each having 8-ports is based on the session created with the core OpenFlow Switch in the time interval $\delta(t)$. Suppose in OF-S1, a host *H1* transmits a packet to destination host *H3*, then the packet is first routed from *H1* to port 2 of OF-S1. The OF-S1 contains a module called as flow exporter to monitor the incoming flow requests. Flow exporter is the monitoring point interface which extracts each packet header to build a flow record from the observed traffic. The exporter captures the incoming packet and assigns a timestamp to each header. The next step is the sampling-filtering process where each flow request is sampled with a certain probability of P and then filtered according to a specific protocol. After this, the flow record is processed and forwarded to the core switch for assigning a flow rule. The incoming request is served and installs a new flow entry in the OF-S1 flow cache memory.

The OF-S1 checks the packet header and matches the flow entries in the flow table. If the flow entry is matched in the flow table then the corresponding action is performed to forward the packet to the destination host $H3$ through port 4, otherwise, the request is again forwarded

to the controller to build a new flow rule. A duplicate copy of the flow record is exported to the flow collector module. Here, a host $H4$ at port 8 on OF-S1 is working as a flow collector which stores a copy of the flow records sent from the sending device to the destination device. Similarly, the host $H2$ of OF-S1 sends a packet to the destination host $H7$ of OF-S2 and the copy is stored at port 1, i.e., host $H5$ of OF-S2. The flow record stored at the flow collector is sent to the core switch for further monitoring and analysis.

3.2.2.3 Flow-based Traffic Analysis & Anomaly Detection

The probability of each flow selected by the active OF-Switch device is calculated and this value is used to measure the flow behavior of each flow using Shannon Entropy function [158]. Finally, a lower and upper threshold tag is set to identify the flow behavior. The entropy value is compared with the threshold value to detect the flow property under three categories such as-(i) elephant flow (large flow/flooding attack), (ii) mice flow (small flow), and (iii) normal flow. Table 3.1 shows the list of symbols used for the flow-based traffic analysis.

Let, S be the number of active OF-Switches in a time interval $\delta(t)$, and n_c is the number of connections available between the switches s_a and s_b. Suppose, f_i be the IP-flow requests and $R = \{f_1, f_2, f_3, ..., f_n\}$ be the set of all IP-flow requests. For $\delta(t)$, the total number of IP-flows, packets, and bytes transmitted between s_a and s_b are computed as follows.

$$t_F = \sum_{k=1}^{n} (f_k)^{flows}, t_P = \sum_{k=1}^{n} (f_k)^{pps}, t_B = \sum_{k=1}^{n} (f_k)^{bytes}, \tag{3.31}$$

where k is a variable used for counter and pps is the number of packets transmitted per second.

Now, the probability of flows, packets, bytes transmitted between s_a and s_b are computed on the basis of number of attributes N_A in the flow header f_H. However, the value of N_A is dependent on n_F, n_P, n_B; s.t., $|n_F| = \langle f_H, n_F \rangle$, $|n_P| = \langle f_H, n_F, n_P \rangle$, and $|n_B| = \langle f_H, n_F, n_P, n_B \rangle$.

$$P_F = \frac{|n_F|}{t_F}, P_P = \frac{|n_P|}{t_P}, P_B = \frac{|n_B|}{t_B}. \tag{3.32}$$

Using Shannon entropy method [158], the degree of anomaly E for a discrete system is identified. The probability distribution $P = \{p_1, p_2, p_3, ..., p_n\}$, and $\sum_{k=1}^{n}(p_k) = 1$ is:

$$E = -\sum_{k=1}^{n} \frac{(|n_k|)}{t_k} log_2 \frac{(|n_k|)}{t_k} = -\sum_{k=1}^{n} (p_k) log_2(p_k), \tag{3.33}$$

where $0 \leq p_k \leq 1$. On the basis of E, the entropy gain value $E(f)$ of individual flow behavior is measured.

$$E(f) = \bigcup_{A=1}^{|n|} (E^{N_A}), \tag{3.34}$$

where A is a specific attribute type in f_H. Now, the flow status of each active OF-Switch $\chi(\lambda)$ can be computed using the flow information matrix given as below.

$$\chi(\lambda) = \begin{vmatrix} {E_{11}}^{N_A} & {E_{12}}^{N_A} & . & . & {E_{1\alpha}}^{N_A} \\ {E_{21}}^{N_A} & {E_{22}}^{N_A} & . & . & {E_{2\alpha}}^{N_A} \\ . & . & . & . & \\ . & . & . & . & \\ {E_{S1}}^{N_A} & {E_{S2}}^{N_A} & . & . & {E_{S\alpha}}^{N_A} \end{vmatrix},$$

where $E_{ij} = 1$, if j^{th} flow passes the i^{th} switch, otherwise, $E_{ij} = 0$. The number of rows represent the total number of active switches S, and the number of columns represent the total number of flows chosen as α.

Now, the lower L_{Th} threshold and upper threshold U_{Th} value is computed to determine the flow into two categories as (i) suspicious, and (ii) normal flows. The suspicious flow is further sub-categorized into two as: *i(a)* elephant flow, and *i(b)* mice flow. So, the flow category is determined as follows.

$$L_{Th} = \chi(\lambda) \times [N_A] \times \left([n_s - 1] \times \frac{L_P}{100} + 1 \right), \tag{3.35}$$

$$U_{Th} = \chi(\lambda) \times [N_A] \times \left([n_s - 1] \times \frac{U_P}{100} + 1 \right), \tag{3.36}$$

where n_s is the number of columns in $\chi(\lambda)$ i.e., the total number of flows, while L_P and U_P are the expected or desired lower and upper threshold (percentile). The suspicious behavior occurs if the number of flow requests exceeds the capacity of U_{Th} then the elephant flow (flooding attack) occurs. The second case occurs if the flow requests are less than L_{Th} value, then it is considered as mice flow. The last case is the normal flow, i.e., if the flow requests are within the range of L_{Th} and U_{Th}, then it is considered as a normal activity. The comparison of the flow property to detect the anomaly category is given as below.

$$\left\{ \begin{array}{c} E^{N_A} > U_{Th}, \text{ (elephant flow)} \\ E^{N_A} < L_{Th}, \text{ (mice flow)} \\ \text{Otherwise, (normal flow)} \end{array} \right\}. \tag{3.37}$$

A database $\mu(DB)$ is created which contains the number of IP-flow requests N_R that occurs in the range between L_{Th} and U_{Th}, i.e., $L_{Th} < N_R < U_{Th}$ in $\delta(t)$ time interval.

3.2.2.4 IP Address-based Anomaly Mitigation at the Data Plane

Based on the anomaly detection scheme, the output returned is the list of flow properties. Now, the traffic behavior of each IP address request is observed and if the behavior shows the sign of suspicious activity, then the average packet count of each IP address is compared. Next,

Table 3.1: List of Symbols

Notation	***Description***
f	individual IP-flow request
N_R	total number of IP-flow requests
R	set of all IP-flow requests
$\delta(t)$	time interval in seconds
s_a, s_b	OF-Switch device a and device b
n_C	number of connections between s_a and s_b
M	set of flow messages shared between s_a and s_b
$E(f)$	entropy gain computed for every flow requests
E	Shannon entropy value for a discrete time interval
T	set containing flow tags as legitimate or suspicious
t_P	total number of IP-packets sent in $\delta(t)$
t_F	total number of IP-flows sent in $\delta(t)$
t_B	total number of bytes sent in $\delta(t)$
f_H	IP-flow header
N_A	number of attributes with a particular IP-packet header
P_P, P_F, P_B	probability of the number of packets, flows, bytes
n_P, n_F, n_B	packets, flows, bytes transmitted by a switch
L_{Th}, U_{Th}	lower threshold & upper threshold value
L_P, U_P	number of expected upper and lower percentile
$\mu(DB)$	database of legitimate flow requests
$l(t)$	legitimate request table set
$n(t)$	new request table set
$l(\gamma_k)$	legitimate flow request counter
$n(\gamma_k)$	new flow request counter
L	least amount of flows per connection
$\overline{X}$	mean of the flow request coming from a device
ω	number of IP flow request in $l(t)$
β_i	rate of sampling of switch i
d_{s_i}	data rate of switch i
Q	processing capacity of the anomaly detection system

the packet count value decides that the IP address comes under the category of blacklist IP address or whitelist IP address. Finally, the ACP is decided to mitigate anomaly behavior.

4.1) Blacklist and Whitelist IP-flow Requests: The sample mean $\overline{X}$ of the average number of requests coming from a particular IP in its flow entries is computed on the basis of total number of IP-flow requests N_R and legitimate flow request counter $l(\gamma_k)$ which is given as below.

$$\overline{X} = \frac{1}{N_R} \sum_{k=1}^{N_R} [l(\gamma_k)]. \tag{3.38}$$

Algorithm 2 shows the steps followed for an IP flow-based anomaly mitigation scheme. The input is a set $M(S)$ consisting of six-tuples $\langle \mu(DB), \overline{X}, U_{Th}, L_{Th}, l(\gamma_k), n(\gamma_k) \rangle$ and the output is in the form of $\langle B[Q], W[Q] \rangle$. The output returned as $B[Q]$ consists of a queue maintained for

a list of blacklisted IP-requests and $W[Q]$ consists of queue maintained for a list of normal or whitelist requests. Initially, we consider a loop for receiving all the incoming flow requests, i.e., $f[request]$. Now, each $f[request]$ is inspected to know whether it is coming from a legitimate user or not. Here, each flow request is verified if it belongs to the database of legitimate flow requests or not. Next, if the flow requests belong to the database of legitimate flow requests, i.e., $(f[request] \in \mu(DB))$ then it is considered as a legitimate request otherwise, the flow request is considered as a new request. Next, the legitimate requests are inserted in a queue of trusted users request and the counter is increased by 1. Now, the traffic behavior of a legitimate request is observed by identifying whether the counter exceeds beyond the upper threshold limit or not. If yes, then it is a sign of suspicious behavior, otherwise not. The core switch performs an action and dispatches a message to all the distribution switches for sending the flow statistics for the packet count of that specific IP address.

The core switch now inspects the average packet number $\overline{X}$ coming from a particular IP address. Let us assume the switch has assigned a rule of $L > 7$, i.e., the least number of flows per connection. Now, we compare $\overline{X}$ with L, and if the mean value is equal to or less than 7, then the IP address is considered as blacklisted IP address, otherwise, the IP address is a whitelist. The action taken by the controller on the blacklisted IP-address is based on the (ACP_1). Otherwise, the IP requests are considered under $W[Q]$ and are served with an idle ACP.

The second case exists if $(f[request] \notin \mu(DB))$, i.e., which means it is a new IP request and is inserted in a queue of new requests. For every new request, the controller assigns the flow policy as ACP_2, and the counter $n(\gamma_k)$ is increased by 1. Now, the traffic behavior is observed again by identifying whether the new flow request counter $n(\gamma_k)$ is less than the lower threshold limit of L_{Th}. If yes, then it is a mice flow and it shows the sign of suspicious behavior, otherwise, it is considered under normal flow. The action performed on a suspicious activity involves the transmission of flow statistic message to an OF-Switch and check the request flow activity of a specific IP address. If $\overline{X}$ is equal to or less than L, then the request from that IP address is inserted into a blacklist queue $B[Q]$. The action taken by the core switch on that specific IP address is based on (ACP_1), otherwise, the requests from the IP addresses are considered under $W[Q]$ and are served with the idle ACP. Finally, the value returned is the list of whitelist $W[Q]$ and blacklisted IP addresses $B[Q]$ on which different types of ACPs as shown in Table 3.2 are applied.

4.2) Access Control Policies (ACP) based on Timeout: The objective of designing the ACP is to satisfy the $C6$ constraint so that the time consumed for anomaly mitigation μ_{ADM}, which must be less than a prescribed threshold limit of μ_{THR}. The master controller is the overall in-charge of making routing decisions for each flow behavior. The policies are built on the basis of a timeout to inspect when a specific flow entry gets expired from the flow table. Generally, two types of timeouts are attached to every flow entry.

Algorithm 2 Algorithm for IP-flow anomaly mitigation.

Input: $\mu(DB), \overline{X}, U_{Th}, L_{Th}, l(\gamma_k), n(\gamma_k)$.
Output: $B[Q], W[Q]$.

procedure ACCESS CONTROL POLICIES
  for $(f[request] = 1; f[request] \leq n; f[request]++)$ **do**
    if $(f[request] \in \mu(DB))$ **then**
      INSERT: $trusted[Q] \leftarrow f[request]$;
      update $l(t) \leftarrow l(\gamma_k)$++;
      if $(l(\gamma_k) > U_{Th})$ **then**
        check the flows request from OF-Switch;
        if $(\overline{X} \leq L)$ **then**
          INSERT: $B[Q] \leftarrow f[request]$;
          apply ACP_1;
        else if (INSERT: $W[Q] \leftarrow f[request]$) **then**
          apply ACP_idle;
        end if
      else if (INSERT: $W[Q] \leftarrow f[request]$) **then**
        apply ACP_idle;
      end if
    else if (INSERT: $new[Q] \leftarrow f[request]$) **then**
      apply ACP_2;
      update $n(t) \leftarrow n(\gamma_k)$++;
      if $(l(\gamma_k) \leq L_{Th})$ **then**
        check the flows request from OF-Switch;
        if $(\overline{X} < L)$ **then**
          INSERT: $B[Q] \leftarrow f[request]$;
          apply ACP_1;
        else if (INSERT: $W[Q] \leftarrow f[request]$) **then**
          apply ACP_idle;
        end if
      else if (INSERT: $W[Q] \leftarrow f[request]$) **then**
        (normal flow);
        apply ACP_idle;
      end if
    end if
  end for
  return $\langle W[Q], B[Q] \rangle$;
end procedure

The first timeout is the idle I_TO in which the flow gets expired from an OF-Switch if in case, none of the flows hitting the flow table for a particular time interval. The flow entry is deleted after I_TO is expired and the range of I_TO is between 11-65535 seconds (sec). The second timeout is the hard H_TO called an absolute timeout value after which the flow gets expired from an OF-Switch irrespective of flows hitting the flow table for a particular time interval. The range of H_TO is between 0 to 65535 seconds (sec). All three types of ACPs are installed in a pro-active mode on the controller. The benefits of a pro-active mode are that the flow rule policies are pre-installed and this decreases the overburden generated on the network controller. The controller forwards the different types of ACP to the core switches as shown in Table 3.2. The core switches dispatch a message to the distribution OF-Switches to apply these policies for each request type.

Table 3.2: Types of ACP.

Request Type	**Entry Type**	**I_TO**	**H_TO**	**ACP Type**	**Action**
Adversary	Long	0 sec	3600 sec	ACP_1	delete
New	Short	7 sec	60 sec	ACP_2	fwd, drop
Legitimate	Normal	15 sec	60 sec	ACP_idle	fwd

- *Case 1 (Adversary Request)*: This request type is the most severe malicious request for which a ACP_1 is applied. After the confirmation of an adversary, the controller dispatches a message to the OP-Switch to immediately delete all the flow entries by setting the H_TO to 3600 seconds. In addition, the IP address of the request is removed automatically from the database $\mu(DB)$. Therefore, the flow entries are expired after 3600 seconds and the user request is treated as a new request type.

- *Case 2 (New Request)*: It is a new request for which a ACP_2 is applied. For a new user, not in the database $\mu(DB)$, set I_TO to 7 seconds and H_TO to 60 seconds. I_TO is a very short flow entry type in which if the user activity is found legitimate, then the request table set is changed from $n(t)$ to $l(t)$, otherwise no change is required. The action to forward the flow is taken till I_TO is expired after that the flows are dropped.

- *Case 3 (Normal Request)*: This request comes from a legitimate user for which a ACP_idle is applied. For this, the I_TO is set to 15 seconds and the H_TO remains the same as 60 seconds. The action taken is to forward the flow to the ultimate destination address.

In case, several begin flows and one malicious flow have the same IP address as a result of Network Address Translation (NAT) then the issue of IP spoofing can be resolved with the help of the IP traceback method. The ADS examines both the incoming and outgoing ports on OpenFlow switch which initiates from the victim system till the origin of a packet is found. The IP traceback method traces the DDoS attack and identifies the origin of the offending packets during the IP spoofing attack. Finally, the identified forged IP address is automatically blacklisted

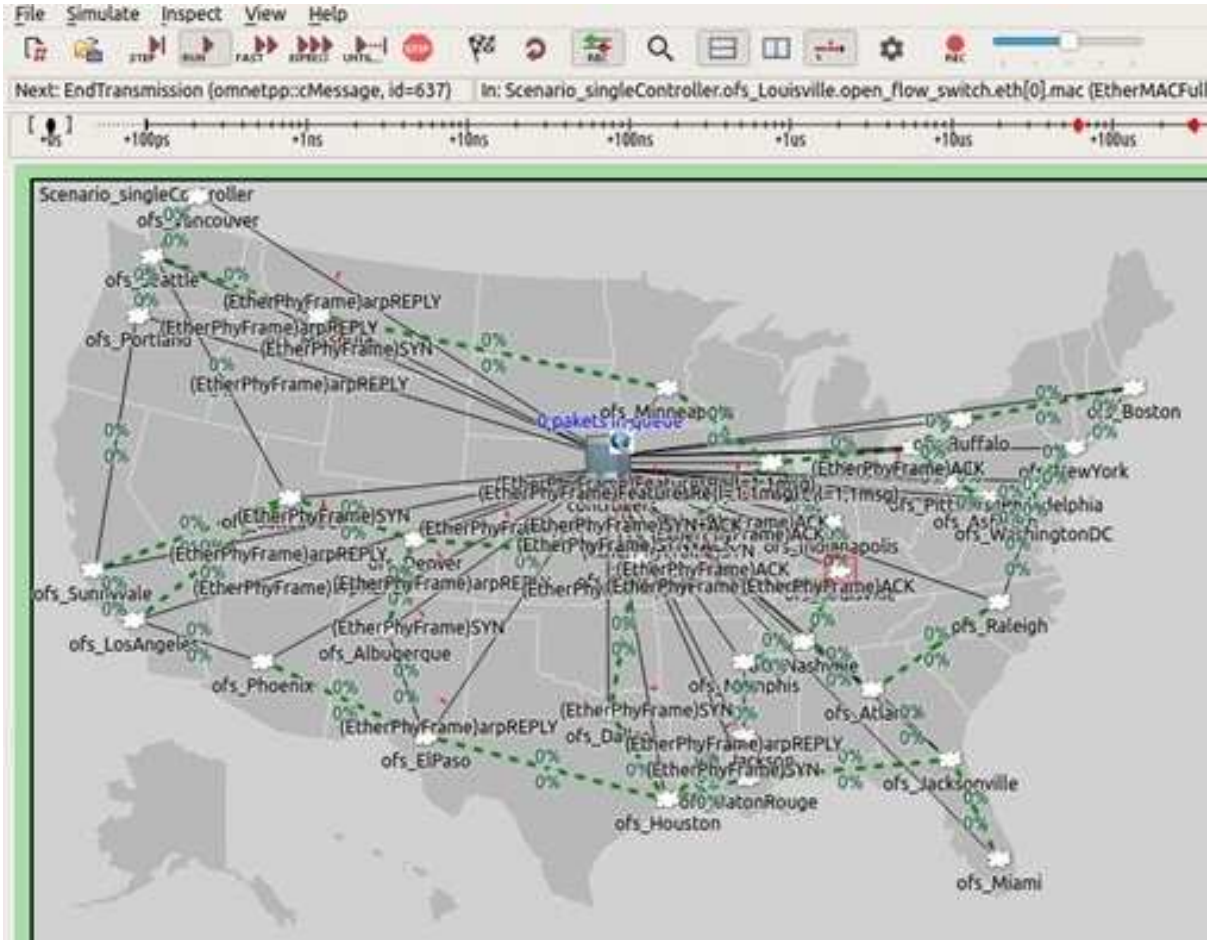

Figure 3.5: GUI representation of BTNorthAmerica dataset of Internet Topology Zoo [220], [221].

by applying ACP_1, where the action is taken is to delete the flow tables from the OpenFlow switch.

Complexity Analysis: The time and space complexity of Algorithm 2 is computed as below.

Time Complexity: The *for* loop iterates the process till n times which takes $O(n)$ time in worst case. Next, the conditional operators take $O(1)$ time. Therefore, the overall time complexity is computed as follows:

$T(C) = O(n) + O(1) \Longrightarrow O(n)$.

Space Complexity: The input variables of the anomaly mitigation algorithm is fixed and cannot exceeds n hence it takes $O(n)$ space in the worst case. After this, the loop takes $O(n)$ space and finally, the arithmetic operations take $O(1)$ space. Therefore, the overall space complexity is computed as follows:

$SC = O(n) + O(n) + O(1) \Longrightarrow O(n)$.

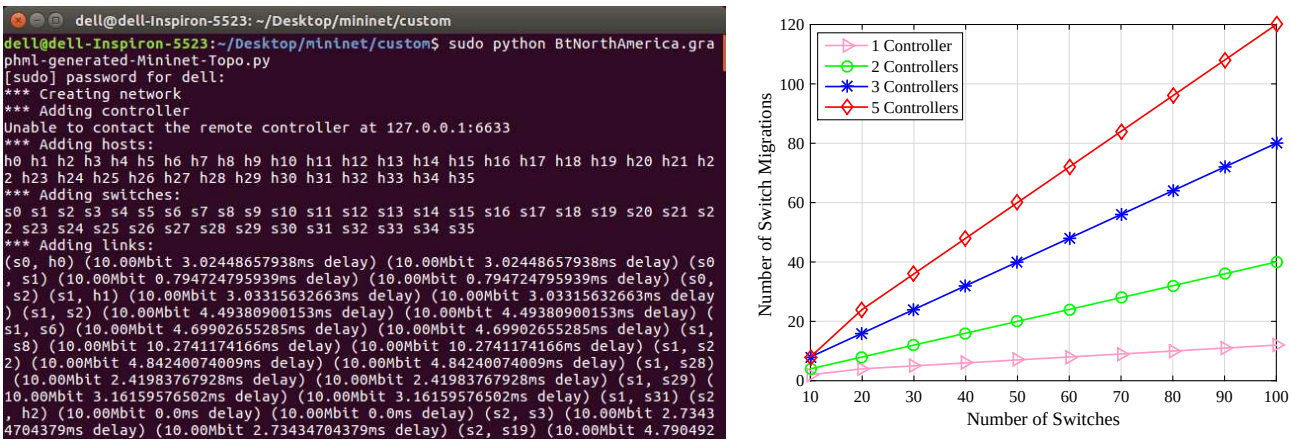

Figure 3.6: (a) Internet Topology Zoo. (b) Number of switches migrated by Controllers.

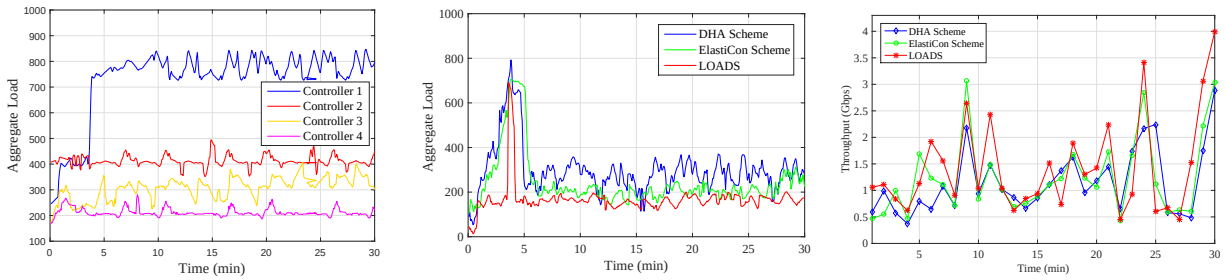

Figure 3.7: (a) Load distribution measurement on multiple controllers under static switch-controller assignment. (b) Comparison of the aggregate load vs time. (c) Comparison of the throughput with time.

3.3 Performance Evaluation

3.3.1 Numerical Settings

The performance of the proposed scheme is tested using the simulation performed on the Mininet 2.0.0 emulator platform [222]. The experiments are performed on a physical server having system configuration Intel Core i7 with 3.20 GHz clock speed, 64-bit processor, and 8 GB of DDR3 RAM with Ubuntu 16.04 LTS OS using POX Controller [223] and the algorithms are implemented using Python. The logical instances of OpenFlow switch named as Open Virtual Switch (Open vSwitch version 2.5.5) [224] that support multiple controllers are created which run on different physical machines. Fig. 3.5 shows the GUI representation of the real-time datasets for BTNorthAmerica zone [220] of Internet Topology Zoo [221], wherein the number of controllers varies from 1-5 which are responsible to control the entire network. Each controller is connected with 36 Open vSwitches deployed in different cities of North America. These Open vSwitches are connected through 72 links with respect to switch-to-switch and switch-to-controller connections [224]. Moreover, 36 hosts are connected with each Open vSwitch via a linear topology through 36 host-to-switch links. Fig. 3.6(a) shows the Internet Topology Zoo using CLI and python using the following command:

$ sudo python BtNorthAmerica.graphml-generated-Mininet-Topo.py

Fig. 3.6(b) demonstrates a scenario of topologies with fat-tree(10), fat-tree(20), fat-tree(30), fat-tree(40), fat-tree(50), fat-tree(60), fat-tree(70), fat-tree(80), fat-tree(90), and fat-tree(100) with 10-100 Open vSwitch. Here, the convergence rate of the number of switch migrations using 1, 2, 3, and 5 controllers are analyzed. The *LOADS* scheme achieves scalability as the number of switch migrations grows linearly with the increase in the size of Open vSwitch in the network.

Fig. 3.7(a) measures the load distribution on multiple controllers under static switch-controller assignment with respect to time. The simulation is performed till 30 minutes and it is observed that the controller 1 is overloaded after 270 seconds with the aggregate load of 700 MB while the load on controllers 3 and 4 are under-utilized. The switch migration scheme balances the uneven distribution of load on the controllers. Thus, after 295 seconds, it is observed that the load on controller 1 is reduced to some extent as a load of some OpenFlow switches is migrated to controllers 3 and 4 while the load on controller 2 is already balanced. Fig. 3.7(b) demonstrates that the aggregated load on the Controller with respect to time. The iteration is performed till 30 minutes and Controller 1 is balanced after 315 seconds on execution of *DHA*, *ElastiCon*, and *LOADS* schemes. The performance analysis shows that the *LOADS* scheme outperforms the existing schemes in terms of aggregated load on controllers. Fig. 3.7(c) shows the comparison of the throughput in Gbps using a TCP connection with respect to time. The experimental results are performed for 30 minutes and the average throughput changes frequently as the time interval of 1 minute is considered. The experimental result demonstrates that the average throughput of the proposed scheme is 1.86 Gbps while the *DHA* and *ElastiCon* schemes are 1.34 Gbps and 1.62 Gbps respectively. The performance analysis shows that the *LOADS* scheme outperforms the existing schemes as the throughput is increased to 38.8% and 14.81% respectively.

3.3.2 Results and Discussion

The performance of *LOADS* is evaluated in comparison to the existing schemes– *ElastiCon* [151] and *DHA* [152] using six performance evaluation metrics (i) response time (ii) execution time (iii) migration cost (iv) average number of flows (v) CPU utilization (vi) accuracy level.

3.3.2.1 Impact on the response time

We measure the impact on the response time after migration of switches between two controllers I and J. Initially, I is connected with 10 switches and J with 7 switches. Each Open vSwitch generates an average of 1800 flows per second with an average traffic rate of 100 Mbps. Here, we consider the flow setup rate of 42,000 flows per second that allows 2,00,000 flow rules using Open vSwitch as the threshold value of the POX controller to handle the flow setup requests.

Fig. 3.8(a) depicts the Cumulative Distribution Function (CDF) plotted with red, green, blue, and black colors. In CDF, we consider that the switch migration probability lies in the range [0–1]. Initially, the controller *I* is overloaded with high response time as shown in black color with the load of 10 switches, whereas the controller *J* is under-utilized with low response time as shown in blue color with the load of 6 switches. Then, we measure the performance of each controller after the migration of switches (one by one) from an overloaded to an under-utilized controller. Next, after migrating two Open vSwitches from controller *I* to *J*, controller *I* handles the load of 8 switches as shown in green color instead of 10 switches and *J* with 8 switches as shown in red color instead of 6 switches. We again plot the CDF and analyzed that the response time of *I* decreases and both *I* and *J* are equally balanced. Thus, the switch migration scheme shows an improvement in the response time of each controller.

In Fig. 3.9(b), we compared the average response time of *LOADS* with the existing schemes. The average response time of our proposed scheme is 32.6 milliseconds while the response time of the *DHA* and *ElastiCon* schemes are 140 milliseconds and 110 milliseconds respectively. The reason why our scheme achieves less response time is that the *ElastiCon* scheme performs switch migration based on the closest neighbor controller without measuring the load on that controller which again increases the overall response time. The *DHA* scheme performs load balancing based on a random selection of controller and broadcasts the action to all the neighbor controller. Our *LOADS* scheme performs switch migration based on the two conditions used in the migration probability (Eq. (3.28)).

3.3.2.2 Impact on the execution time

The execution time of *LOADS* is computed in Fig. 3.8(b) and compared with the existing schemes. On X-axis, we vary the number of migrated switches from 4 to 128. Our *LOADS* and *ElastiCon* scheme are moderate in average execution time with 14.31 seconds and 17.66 seconds while the *DHA* scheme is 26 seconds respectively. The reason for the increase in the execution time of the *DHA* scheme is that the random selection for switch migration needs to synchronize the state of the controllers after migration for the global network view. In addition, the DDoS attacks also affect the execution time in switch migration as shown in Fig. 3.8(c). The DDoS attack floods numerous requests from different IP addresses and increases the execution time to 18 seconds. The traffic is monitored with the average traffic rate of 100 Mbps and a sampling rate of 1/64. The flow statistics of the traffic generation is analyzed by using the Wireshark tool in which the complete information such as– packet size, the time duration of the flow in the flow table, source and destination IP address, number of packets, port number, the protocol used, requests per connections, etc. are monitored on a real-time basis. The flow statistics are collected first and then features such as– the number of requests, the time duration of the flow, and the number of requests send per connection are examined for each flow request in order to categorize the request type and the corresponding access control

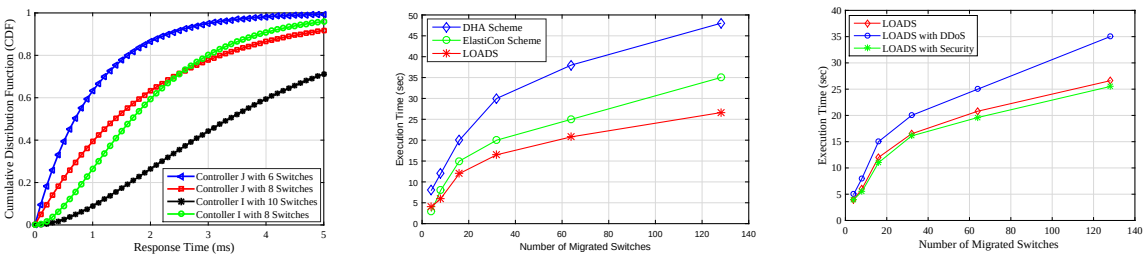

Figure 3.8: (a) Effect on response time after switch migration from overloaded to underload Controller. (b) Number of migrated switches vs execution time. (c) Effect on execution time vs number of migrated switches on *LOADS* scheme with DDoS attack and after mitigation from DDoS Attacks.

policies enforcement are applied. The proposed scheme injected flows emulating DDoS at the attack rate of 250-300 flows per second. The performance of *LOADS* is improved to 13.61 seconds after mitigating from DDoS attack.

3.3.2.3 Impact on the migration cost

The migration cost is used to measure the switch migration efficiency of *LOADS*, *ElastiCon*, and *DHA* scheme as shown in Fig. 3.9(a). The migration costs of the *ElastiCon* and *DHA* scheme are found to be 110 milliseconds and 140 milliseconds. The proposed scheme witness the migration cost of 52.2 milliseconds. The migration cost of *DHA* scheme is highest while the response time is lower than the *ElastiCon* scheme. The reason behind this is that the *DHA* scheme migrated switches several times and uses random switch selection. The *ElastiCon* has lower migration cost than *DHA* scheme but higher response time. The reason being that the *ElastiCon* scheme chooses the nearest neighbor node for switch migration, thus reducing the information sharing cost between the controllers. The proposed scheme only migrates a few switches based on the closest distance and least resource consumption from the out-migrating to the in-migrating controller. The DDoS attack again increases the migration cost of *LOADS* from 52.2 milliseconds to 71.6 milliseconds as shown in Fig. 3.9(c). After blacklisting these IP addresses, the POX controller builds different ACP policies to block the malicious IP addresses, thus the migration cost is reduced to 55.1 milliseconds. Hence, the performance of the *LOADS* scheme is better in reducing migration cost and response time as compared to the existing variants.

3.3.2.4 Impact on the number of flows on OpenFlow switches

Fig. 3.10(a) and 3.10(b) demonstrates the performance of the *LOADS* scheme with DDoS security which shows an impact on the number of flow entries in the flow tables on OpenFlow switches and the CPU usage of controllers with respect to time in seconds. The proposed *LOADS* scheme with DDoS security is compared with the existing baseline schemes, i.e., SVM-i modules, SVM-i with Defense, and OpenFlowSIA. The network traffic is gener-

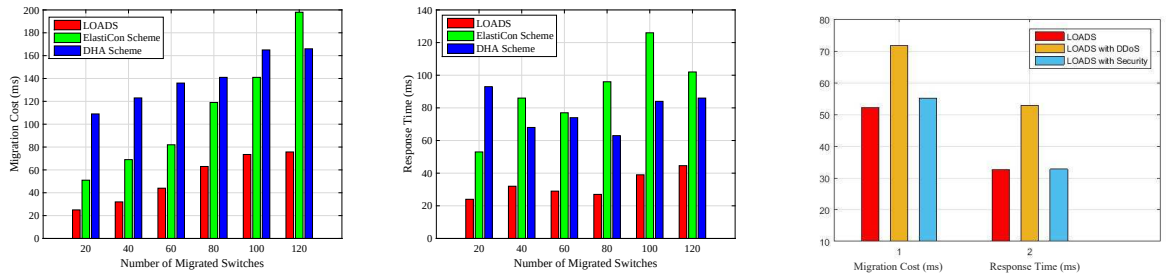

Figure 3.9: (a) Number of migrated switches vs switch migration cost. (b) Number of migrated switches vs average response time. (c) Effect on migration cost and response time on *LOADS* scheme with DDoS attack and after mitigation from DDoS Attacks.

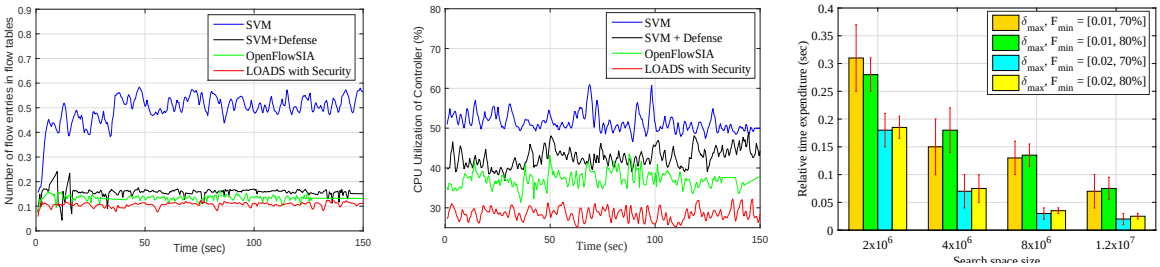

Figure 3.10: (a) Number of flow entries in flow tables of OpenFlow switches with respect to time. (b) Comparison of CPU usage of the controller with respect to time. (c) Accuracy level based on relative time expenditure vs size of the search space.

ated from the *Center for Applied Internet Data Analysis (CAIDA)* datasets [225] of 21 May 2015 consisting of both the normal and attack traffic. Moreover, a *BoNeSi* [226] tool is used in *CAIDA* datasets to generate real-time flooding attacks from different types of traffic. The type of traffic considered is ICMP flooding, TCP (HTTP) flooding attacks from multiple IP addresses. The reason pf choosing *BoNeSi* tool over other IP spoofing tools is that the tool simulates HTTP-GET flooding which is not possible through the Internet from bot networks. For e.g., a 10K bot network means that 10,000 IP addresses are generated randomly in the large networks for spoofing, similarly, *urllist.txt* specifies different URLs are generated for use. A *BoNeSi* is a powerful tool to filter out the identifiable patterns having the capability to spool IP addresses when the requests are generated from different protocols. The flooding attacks (generally targeted in SDN architecture) in three possible cases (i) overload a specified IP address by sending multiple requests from a single source (ii) targets victim destination from multiple spoofed IP addresses (iii) requests sends from a single source IP addresses to target multiple destinations. Fig. 3.10(a) demonstrates the number of flow entries in flow tables in OpenFlow switches with respect to time in seconds. The average number of flows per second in the OpenFlow switches in *SVM-i* modules ranges from 3.8×10^4 to 5.8×10^4. However, after applying rule enforcement on *SVM-i modules with defense* the flow entries are restricted to 1.6×10^4 flows on an average per second while the *OpenFlowSIA* returned 1.5×10^4 number of flows per second on an average. The *LOADS* scheme with security returns the number of flow entries as

1.3×10^4 flows per second. The reason why the *LOADS with security* computes the least number of flow entries is that the non-identified IP addresses and adversary requests are blacklisted by deleting the flows from the flow tables. The idle timeout is very short and set to 5 seconds which gives a short span to the attacker for flooding attacks and thus the small number of flows exists.

3.3.2.5 Impact on the CPU utilization on controllers

Fig. 3.10(b) shows the comparison of the average CPU utilization of controllers. The results demonstrate that the average CPU usage of the existing schemes, i.e., SVM-i module, SVM-i with defense, and OpenFlowSIA are 51.5%, 41.2%, and 37% respectively. The proposed scheme outperforms the existing schemes as the CPU usage is only 32% which saves 5% CPU utilization of controllers. The reason behind saving the CPU usage is that less number of flows and messages are assigned to the POX controller which generates lesser flow statistics compared to the existing schemes.

3.3.2.6 Impact on the accuracy level

The accuracy level of the proposed scheme is computed based on the relative time expenditure with respect to the search space as shown in Fig. 3.10(c). The search space $\sum_{k=1}^{V} \binom{V}{k}$ computes all feasible solutions to assign the nearest controller to each switch. On the x-scale, the search space size grows exponentially from 2×10^6 to 1.2×10^7 grouped into four boxes, while the y-scale represents the relative time expenditure. The relative time expenditure is computed as follows: $t_{rel} = t_{LOADS}/t_{DHA}$ and $t_{rel} = t_{LOADS}/t_{ElastiCon}$. Here, the accuracy level demand is computed on [0.01, 0.02] with the maximum threshold $\delta_{max.}$ of the distance from a switch to its reference controller $\delta_{H,I}$. In addition, F_{min} denotes the minimum fraction in which $\delta_{H,I} < \delta_{max.}$. The height of the bar chart represents the mean value of each box and the error bar indicates the confidence intervals of 95%. Here, the accuracy demand of the $\delta_{max.}$ values [0.01, 0.02] are computed on F_{min} for 70% and 80% respectively. The different colors of the bar indicate distinct levels of accuracy. The performance analysis shows that for the accuracy demand $\delta_{max.} = 0.01$ requires more t_{rel} as compared to accuracy demand on 0.02. The accuracy level concludes that the relative time expenditure t_{rel} decreases as the size of the search space increases but if the search space size is small, the relative time expenditure shows a high degree of variation.

3.4 Summary

In this chapter, a LOADS scheme is proposed to address the load imbalance and malicious flow issues in the SDN. To handle the issue of load imbalance, an optimal switch migration scheme is proposed which minimizes the overall migration cost, execution time, and response

time of controllers. Moreover, an IP flow-based anomaly detection scheme is proposed to identify each user's behavior, and based on the behavior, specific policies are deployed to prevent anomalies. The proposed scheme is evaluated on the Mininet emulator using the POX controller with datasets of Internet Topology Zoo from the BTNorthAmerica zone. The performance analysis reveals that LOADS minimizes the average execution time by 6.74% and 20.64% as compared to the existing competitive schemes, Distributed Hopping Algorithm (DHA), and Elastic Distributed Controller (ElastiCon). Also, it helps in improving the overall migration cost and response time of each controller. The proposed LOADS scheme has the migration cost of 55.1 milliseconds as compared to the ElastiCon and DHA schemes alongwith the migration cost of 110 milliseconds and 140 milliseconds respectively. In addition to the migration cost, the response time of the proposed scheme is 32.8 milliseconds as compared to DHA and ElastiCon which takes almost 90 milliseconds and 78 milliseconds respectively.

Chapter 4

EnFlow: An Energy-Efficient Fast Flow Forwarding Scheme for Software-Defined Networks

Based on the above discussions, the following contributions are presented in this research work.

- The EAR problem is formulated by using an MINLP for which we proposed FIFO-PO and FIFO-POP scheduling schemes to minimize the average waiting time.
- For energy-efficiency, an efficient flow re-routing and link utilization scheme are proposed.
- Then, to build the shortest path between source and destination, an ant colony routing algorithm is proposed.
- Finally, the proposed scheme is validated using the datasets of NorthAmerica on the OMNET++ platform.

The rest of the chapter is structured as follows. Section II discusses the problem statement. Section III presents the proposed EnFlow scheme. Section IV discusses the performance evaluation, and finally, the paper is concluded in Section V.

4.1 Problem Formulation

4.1.1 System Model

Consider a connected graph $G \in (V,E)$ comprising of nodes and links as shown in Fig. 4.1. Here, $V = (S \cup A)$ is a set of switches with $S = \{s_1, s_2, .., s_n\}$ and APs as $A = \{a_1, a_2, .., a_n\}$. Also, $E = (W \cup W_S)$ is a set of wired (W) and wireless links (W_S). Fig. 4.1(a) an SDN architecture with multiple switches having three interfaces. The first interface I_A is wireless

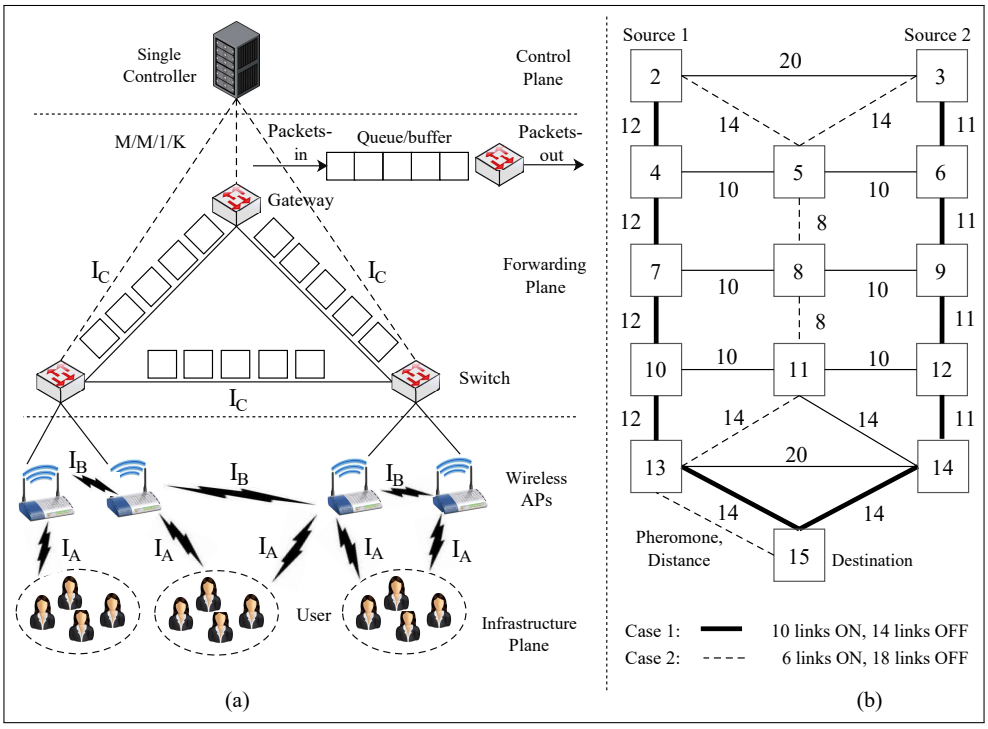

Figure 4.1: System model of SDN architecture.

communication between users and APs, the second interface I_B is wireless communication between APs, and the third interface I_C is a wired interface of APs directly connected to switch or between the two switches. Suppose, $N_C(W_S)$ is the number of channels in wireless links W_S between two APs with each l^{th} channel having capacity defined by C_l. Therefore, for wired links, the number of channel $N_C(W) = 1$ with the same capacity C_l and $N_C(E)$ is the number of channels of both wireless and wired links. Suppose, C_p defines the capacity of each p^{th} AP to serve its connected users, and C_q is the capacity of each q^{th} gateway router that connects the switches to the Internet.

We have considered a priority-based flow scheduling scheme to handle the overloading issues on switches before the incoming packets in the flow are routed to their destination. We compared both the priority and non-priority-based schemes with the minimum average waiting time in the queue buffers. Fig. 4.1(b) shows an energy-efficient routing using which switches do redundancy elimination and identify the links which should be turned off from active to sleep mode for power savings. For example, there are two demands (Fig. 4.1(b)), one from node 2 to 15, i.e., $D_{2,15}$ and another from node 3 to 15, i.e., $D_{3,15}$ based on the minimum path selection.

So, the first solution is to choose the path shown by the bold links for both traffic demands. In this solution, 10 active links and 14 sleep links are used for routing. The second solution is to choose the path using the dotted links with only 6 active links in the usage and rest 18 links are in the sleep mode. So, the energy-efficient solution is the second one as it uses two switches (switch 5 and switch 11) by eliminating the redundancy and turn-off more number of active links to sleep mode to improve the energy consumption.

4.1.1.1 Power Utilization by Switches/APs

The power utilization by switches/APs is based upon three interfaces I_A, I_B, I_C defined above and has the following six different modes (P_m) shown in Fig. 4.2 are defined as follows.

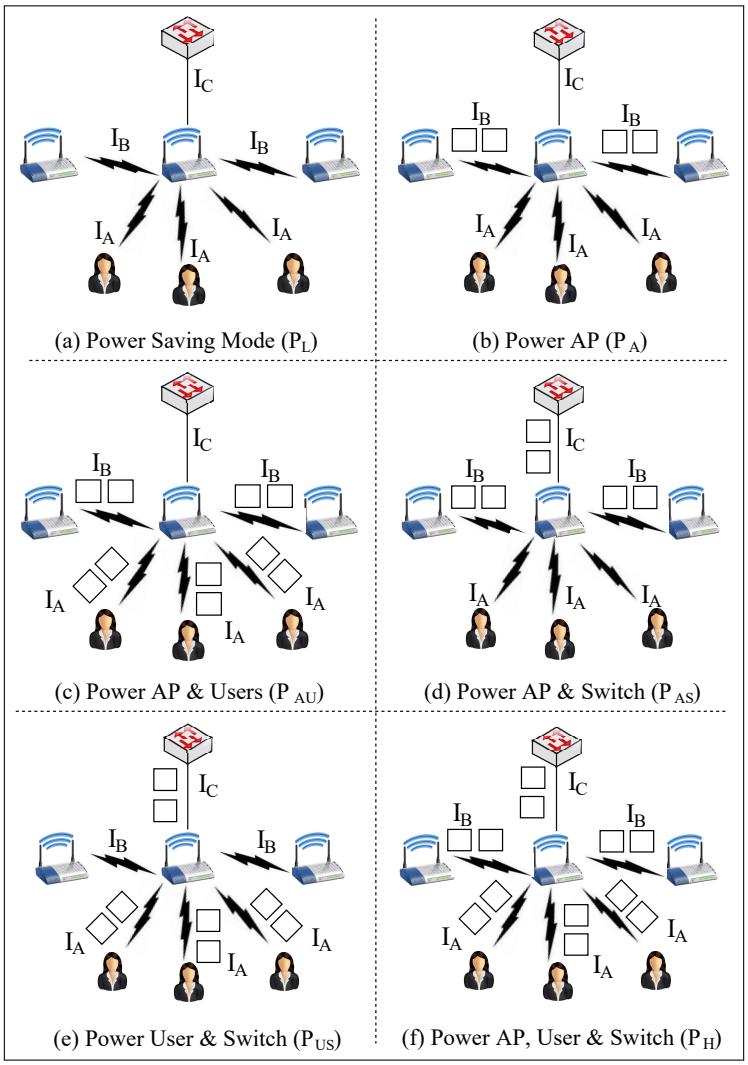

Figure 4.2: Different modes of power utilization on switches/APs.

(i) *Power Saving Mode* (P_L)- It is the lowest power utilization mode as there is no active host and no traffic flow in any interface.

(ii) *Power AP* (P_A)- In this mode, the power is utilized only on the APs as the traffic flow is among the APs through wireless interface I_B.

(iii) *Power AP & Users* (P_{AU})- The power is utilized by the wireless interfaces, i.e., I_A and I_B as the traffic flow is among the APs and also between active users and corresponding APs.

(iv) *Power AP & Switches* (P_{AS})- The power utilization in this mode is on interface I_B and I_C, i.e., among APs and between APs and switches.

(v) *Power User and Switch* (P_{US})- In this mode, the power utilization is from both the interfaces I_A and I_C as there is a flow between active users and APs, and in addition between APs and switches through the wired interface.

(vi) *Power AP, User & Switch* (P_H)- The last power utilization mode is the high power consumption mode as all the three interfaces have active flow because there are active users, traffic flow among APs through wireless links and from APs to switches and controller through wired links.

If an AP has no flow from users and wireless network, then it can be shutdown and thus said to be in low power mode. If an AP has an active host or flows in the links, then it is said to be an active link. An AP that has some traffic load uses an extra one-fourth of the power utilization. Therefore, the power consumption in different modes P_m is defined as follows.

$$P_m = \begin{cases} P_L^{'}(0.75 + 0.25 \times \frac{L_p}{C_p}); & \text{if } (\mu_{i,j,k} = 1), \\ P_L; & \text{Otherwise}, (\mu_{i,j,k} = 0) \end{cases} \tag{4.1}$$

where $\mu_{i,j,k}$ is a binary variable which denotes that the traffic flows of the i^{th} user on the j^{th} switch passing through the k^{th} AP is idle or not, P_L is the power saving mode with no traffic on the links, $P_L^{'}$ is the power mode other than P_L, the load on the AP is denoted by L_p, and C_p is the total AP capacity.

4.1.1.2 Power Utilization Modes of Switches

Energy utilization by a switch depends on factors such as the link rate, the number of active interfaces and switch ports having variable link rate $l_r \in \{l_{r1}, l_{r2}, ..., l_{r_m}\}$. Hence, the power consumption by the switches is denoted by P_S and is defined by using the Eq. (4.2) as follows.

$$P_S = \rho_c + \eta_n \rho_n + \sum \eta_r l_r \rho_r, \tag{4.2}$$

where ρ_c is the power consumption of the switch chassis, η_n is the number of network interface card (NIC), ρ_n is the power consumption of each NIC, η_r is the number of ports, l_r is the link rate of the switch port, and ρ_r is the power consumption of each switch port having a link rate l_r.

The number of active switch ports are: (i) port (transceiver) running at a wired l_r (ii) port to gateway router for interconnection (iii) switch line card (iv) AP connection (v) ingress port (vi) output port. The services running on these switch ports depend on the six different power consumption modes which decide the active number of links and nodes running at a time. It determines the number of physical and virtual interfaces for interconnection. It also determines the link rate of wired connection running at the rate (l_r), and the number of network interface card (NIC).

4.1.2 Problem Statement

Let us consider t is the time interval for any traffic flow, and x represents the state of any variable at time t. Similarly, $(t-1)$ is the time interval before t and the state of any variable rep-

Table 4.1: Decision variables used in problem statement.

Variables	***Description***
U_{fp}	1, if (p^{th} AP is allocated to incoming flow f from user); 0, otherwise.
$F_{e,l,f}$	1, if (l^{th} channel is used to route flow f through link e); 0, otherwise.
x_p	0, if ($\sum_{f\in L}\sum_{e\in E}\sum_{l=1}^{N_c(W_S)} F_{e,l,f} + \sum_{f\in L} U_{fp} = 0$); 1, otherwise. Here, x_p find if an AP is in active mode or idle mode.
x_p^+	1, if ($x_p^{'} = 0$ & $x_p = 1$); 0, otherwise. Here, x_p^+ denotes the switching of APs to active mode from sleep mode.
x_p^-	1, if ($x_p^{'} = 1$ & $x_p = 0$); 0, otherwise. Here, x_p^- denotes the switching of APs to sleep mode from active mode.
$rs_p^+,\ rs_p^-$	cost associated with switching of the modes of APs from sleep to active mode or from active to sleep mode.
w_{pf}^+	1, if ($\sum_{e\in E}\ \sum_{l=1}^{N_C(E)} F_{e,l,f}^{'} = 0$ & $\sum_{e\in E}\sum_{l=1}^{N_C(E)}\ F_{e,l,f} \geq 1$); 0, otherwise. It represent if a flow is added or re-routed via a node p.
w_{pf}^-	1, if ($\sum_{e\in E}\ \sum_{l=1}^{N_C(E)} F_{e,l,f}^{'} \geq 1$ & $\sum_{e\in E}\sum_{l=1}^{N_C(E)}\ F_{e,l,f} = 0$); 0, otherwise. It represent if a flow is not routed through node p.
rw_{pf}^+, rw_{pf}^-	the cost associated with w_{pf}^+ or with w_{pf}^-.

resented as $x^{'}$ at this time interval. The power utilization of an AP/switch modes is represented

by six different power consumption modes P_m using Eq. (3) as follows.

$$P_m = \begin{cases} P_A, & if(\sum_{e\in W_S}\sum_{l=1}^{N_C(W_S)} F_{e,l,f} \geq 1 \ \& \ \sum_{l\in L} u_{fp} = 0 \\ & \& \sum_{e\in W}\sum_{l=1}^{N_C(W)} F_{e,l,f} = 0) \quad \forall p \in \{s(e), d(e)\}, \\ P_{AU}, & if(\sum_{e\in W_S}\sum_{l=1}^{N_C(W_S)} F_{e,l,f} \geq 1 \ \& \ \sum_{l\in L} u_{fp} \geq 1 \\ & \& \sum_{e\in W}\sum_{l=1}^{N_C(W)} F_{e,l,f} = 0) \quad \forall p \in \{s(e), d(e)\}, \\ P_{US}, & if(\sum_{e\in W_S}\sum_{l=1}^{N_C(W_S)} F_{e,l,f} = 0 \ \& \ \sum_{l\in L} u_{fp} \geq 1 \\ & \& \sum_{e\in W}\sum_{l=1}^{N_C(W)} F_{e,l,f} \geq 1) \quad \forall p \in \{s(e), d(e)\}, \\ P_{AS}, & if(\sum_{e\in W_S}\sum_{l=1}^{N_C(W_S)} F_{e,l,f} \geq 1 \ \& \ \sum_{l\in L} u_{fp} = 0 \\ & \& \sum_{e\in W}\sum_{l=1}^{N_C(W)} F_{e,l,f} \geq 1) \quad \forall p \in \{s(e), d(e)\}, \\ P_H, & if(\sum_{e\in W_S}\sum_{l=1}^{N_C(W_S)} F_{e,l,f} \geq 1 \ \& \ \sum_{l\in L} u_{fp} \geq 1 \\ & \& \sum_{e\in W}\sum_{l=1}^{N_C(W)} F_{e,l,f} \geq 1) \quad \forall p \in \{s(e), d(e)\}, \\ P_L, & if(\sum_{e\in W_S}\sum_{l=1}^{N_C(W_S)} F_{e,l,f} = 0 \ \& \ \sum_{l\in L} u_{fp} = 0 \\ & \& \sum_{e\in W}\sum_{l=1}^{N_C(W)} F_{e,l,f} = 0) \quad \forall p \in \{s(e), d(e)\}. \end{cases} \tag{4.3}$$

Thus, the power utilization by switches depends on the P_m. Here, in case, any of the interfaces is OFF or in a sleep mode, then we set the value as 0, otherwise 1 for the active interface. Also, the energy consumption by multiple SDN switches depends on the average waiting time of incoming packets in the switches queue buffer.

The objective function of the *EnFlow* scheme is to minimize the energy consumption of the network switches at the data plane. Mathematically, it can be represented as follows.

$$min. \begin{cases} \omega_E \sum_{p\in(V\cup S)} P_m + \\ \omega_S \sum_{p\in(V\cup S)} [(x_p^+ . rs_p^+) + (x_p^- . rs_p^-)] + \\ \omega_R \sum_{p\in(V\cup S)} [(w_{pf}^+ . rw_{pf}^+) + (w_{pf}^- . rw_{pf}^-)] + \ \tau, \end{cases} \tag{4.4}$$

where $(\omega_E, \omega_S, \omega_R)$ denote the weights associated with balancing the three states– (1) power utilization on different modes when traffic flows on active links, (2) power consumption in switching the nodes or link states (switches or APs), and (3) power consumption in re-routing of the traffic flow respectively. Finally, τ denotes the average waiting time of incoming packets in switches queue buffer.

s.t.

$$\begin{aligned}
C1 \quad &: \quad \sum_{f\in L} U_{fp} \times B_f \leq C_p, \ \ \forall p \in A, \\
C2 \quad &: \quad \sum_{p\in A} U_{fp} = 1, \ \ \forall f \in L, \\
C3 \quad &: \quad \sum_{f\in L} \sum_{e\in E} \sum_{l=1}^{N_C(E)} F_{e,l,f} \times B_f + \sum_{f\in L} U_{fp} \times B_f \leq C_q, \\
C4 \quad &: \quad \sum_{f\in L} F_{e,l,f} \times B_f \leq C_l, \ \ \forall e \in E, l \in \{1,..,N_C(E)\}, \\
C5 \quad &: \quad \sum_{l\in E, s(e)=p} \sum_{l=1}^{N_C(E)} F_{e,l,f} \leq 1, \\
C6 \quad &: \quad \sum_{l\in E, d(e)=p} \sum_{l=1}^{N_C(E)} F_{e,l,f} \leq 1, \\
C7 \quad &: \quad l_{r_e} = min(l_{r_q})_{q\in\{1,..,m\}} \ \geq \sum_{f\in L} F_{e,l,f} \times B_f, \forall e \in \omega, \\
C8 \quad &: \quad \sum_{f\in L} \frac{F_{e,l,f} \times B_l}{C_l} + \sum_{f'\in L} \sum_{e'\in W_S} \frac{F_{e',l,f'} \times B_{f'}}{C_l} \leq 1, \\
C9 \quad &: \quad \sum_{f\in L} \sum_{e\in W, d(e)=p} \sum_{l=1}^{N_C(E)} F_{e,l,f} \times B_l + \sum_{f\in L} U_{fp} \times B_f \\
&\leq \quad C_q + \sum_{f\in L} \sum_{e\in W, s(e)=p} \sum_{l=1}^{N_C(E)} F_{e,l,f} \times B_f, \ \ \forall q \in S, \\
C10 \quad &: \quad \sum_{f\in L} \sum_{e\in E, s(e)=p} \sum_{l=1}^{N_C(E)} F_{e,l,f} \times B_f \ = \sum_{f\in L} \sum_{e\in E d(e)=p} \\
&\qquad \sum_{l=1}^{N_C(E)} F_{e,l,f} \times B_f + \sum_{f\in L} U_{fp} \times B_f, \\
C11 \quad &: \quad F_{e,l,f} \in \{0,1\}, U_{fp} \in \{0,1\}, \forall p \in A, e \in E, f \in L.
\end{aligned}$$

The explanation of each constraint is provided as follows.

- $C1$ states that the traffic flows U_{fp} allocated to the p^{th} AP such that the bandwidth demand B_f should not exceed the capacity of AP defined by C_p.
- $C2$ states that each host in a network is mapped to exactly single AP at a given time interval.
- $C3$ states that the flow on the switch with a particular bandwidth should not exceed the switch capacity.
- $C4$ constraint denotes that the total capacity of each link and channel for a given demand of bandwidth B_f should not exceed the channel capacity of the link. This applies to all the flows and channels in the wireless or wired links.
- $C5$ and $C6$ state that traffic flows through any communication mode enter (source node) or leave (destination node) from the AP or switch should be at most once.
- $C7$ denotes that in the wired part, l_r should satisfy that the bandwidth used on a particular link with an upper bound as l_r.
- $C8$ represents the constraint that in the wireless link, there cannot be the simultaneous transmission of two links to communicate with each other. The sum of their ratio of link usage should be less than or equal to 1.
- $C9$ denotes that the incoming flows are routed through wired and wireless links until the capacity of the gateway exceeds; otherwise, the traffic moves to the other gateway.
- $C10$ is the flow conservation constraint such that the incoming flow entering a node including the traffic from that node should be equal to the total flow going out from that node.
- $C11$ states that the decision variables are binary values.

Fig. 4.3 presents the proposed scheme named as EnFlow: Energy-Efficiency Fast Flow Forwarding Scheme for Software-defined networks.

4.2 *EnFlow*: Energy-Efficiency Fast Flow Forwarding Scheme

The proposed EnFlow scheme uses the M/M/1/K queuing model that runs on the forwarding plane. At the control plane, a single centralized SDN controller (acts as a single server) is serving multiple switches at the forwarding plane. Each switch is linked with multiple access points at the infrastructure plane through which a group of users are attached. The incoming data from users is of a heterogeneous type such as text, audio, video, images, etc. which is coming from different applications. Here, each switch maintains a queue for flow scheduling

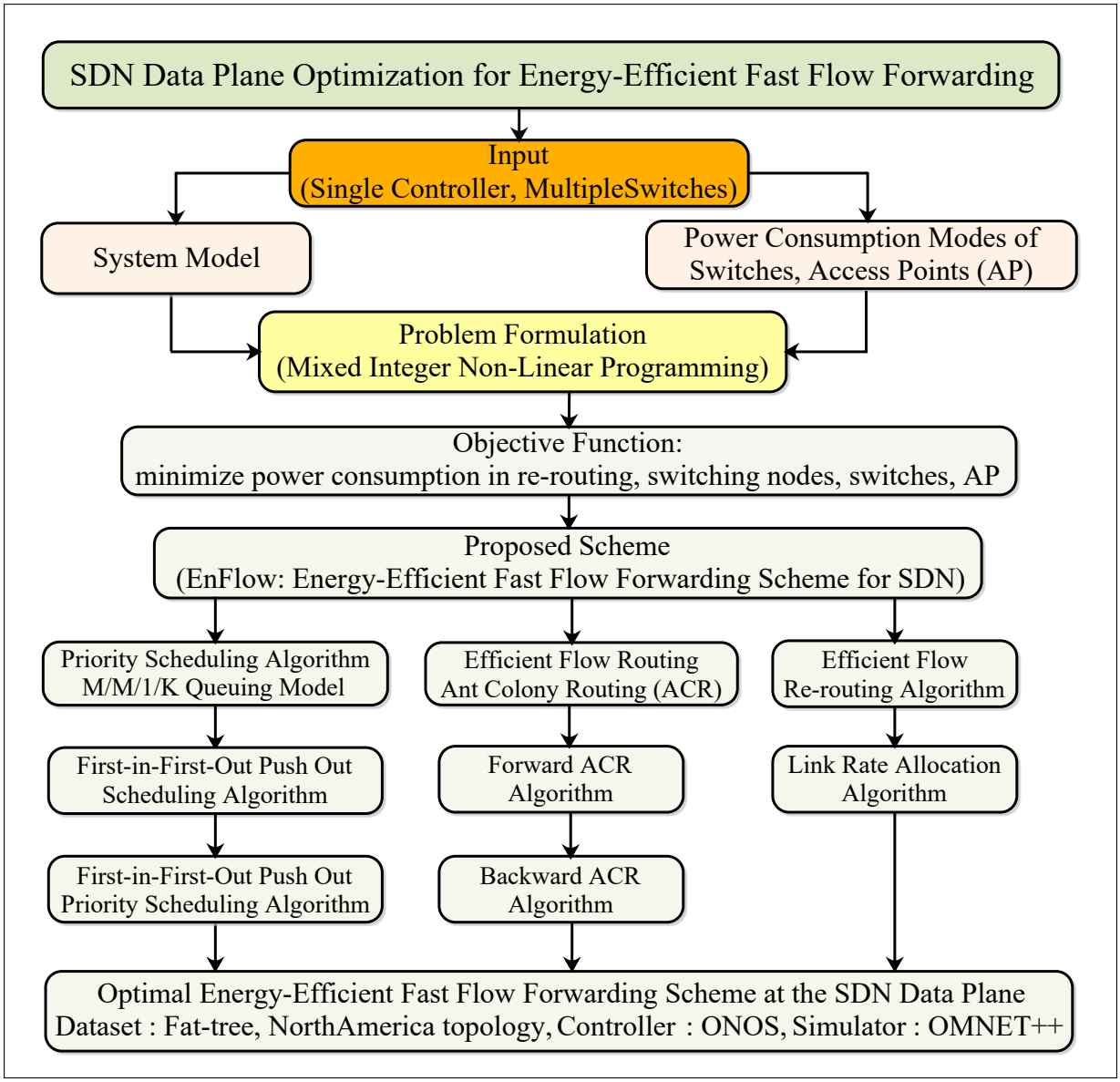

Figure 4.3: Proposed scheme EnFlow: Energy-Efficiency Fast Flow Forwarding Scheme for Software-defined networks.

and the updated record of all the switches is transmitted to the network controller periodically as a global database. The *EnFlow* scheme is divided into three modules. The first module is the FIFO-POP scheduling, which runs on the OpenFlow-enabled switches. Using the Support Vector Machine (SVM) classifier [227], the incoming flows are classified into three different classes as high, medium, and low priority. Then, applying the standard k-means clustering algorithm to create clusters of the three classes. The queue with the highest priority is scheduled with the maximum link rate, and the output is sent to the second module. The second module is the flow re-routing and link rate adaptation algorithm which works during the insertion of any new node or deletion of any existing node to save the routing cost for the maximum link rate. The third module is the flow routing algorithm, which computes the shortest path using Ant Colony Routing (ACR) algorithm for efficient routing in fat-tree topology. These three

modules are illustrated as follows.

4.2.1 Priority Scheduling with Multiple Switches SDN

The data flows in two directions i) from the users to the switches through APs and ii) from multiple switches to a single network controller in the form of packets. We aim to reduce the switches overload by scheduling using two schemes (i) First-In-First-Out Push Out (FIFO-PO), and (ii) First-In-First-Out Push Out Priority (FIFO-POP) [38]. We computed the average waiting time of both the scheduling schemes to find the near-optimal solution among these schemes. These schemes are described as follows.

(i) Scheme 1: FIFO-PO Scheduling Policy

In the FIFO-PO scheme, we assume that the arrival of data packets follows a Poisson distribution with rate parameter as $(\lambda > 0)$ with identical and independent inter-arrival which follows an exponential distribution. The service time is also independent and follows the Poisson rate $(1/r)$. Let there are at most K packets in the system and the buffer size is $(K-1)$.

Also, let C_i is the steady-state probability of the $M/M/1$ queue and P_i is the steady-state probability of the $M/M/1/K$ queue for $1 \leq i \leq (K-1)$. So, from the fundamentals of Queuing theory, following holds.

$$P_i = \frac{C_i}{\sum_{i=0}^{K-1} C_i}, \quad 1 \leq i \leq (K-1). \tag{4.5}$$

Suppose, $P_x(1 \leq x \leq K)$ denotes the steady state probability of $'x'$ packets in the system with an arrival of a new packet.

Then, following holds using the Poisson arrival rate.

$$P_x = \frac{P_x}{(P_0+\rho)}, \quad x = 0,1,2,..,(K-1), \tag{4.6}$$

$$P_K = 1 - \frac{1}{(P_0+\rho)}, \quad x = K, where\ \rho = \frac{\lambda}{r}. \tag{4.7}$$

Let $C_S(x,y)$ be the probability in steady state (x,y) of the packet which only gets the service after a packet exits from the system. If $x = 1$, then the packet is currently in service whereas, for $2 \leq x \leq K$, and $0 \leq y \leq (K-x)$, the packets in state (x,y) wait to get served till the buffer is full and pushes out the packet currently in service. Here, (x,y) is the position of a packet where, x packets are already there in the system and y are waiting after x queue ends.

Suppose, when a packet is being served then there are $'n'$ incoming packets. So, for packets in state (x,y) to get served $n \leq (K-y-2)$, there is no packet to push out if $n \leq (K-x-y)$, i.e., the buffer is not full. Hence, the updated position of x is $(x-1)$ and of y is $(y+n)$.

The buffer is full when $(K-x-y)+1 \leq n \leq (K-y-2)$ so the packet at position 2 is pushed out by the arriving packet. So, the packet position changes from x to $(K-n-y-1)$ and y to $(y+n)$. Now, we have the state transition equation of $C_s(x,y)$ formulated as follows.

$$C_s(x,y) = \prod \alpha_n.(n|(x,y)) + \sum_{n=K-x-y+1}^{K-y-2} \alpha_n C_s(K-n-y-1, y+n), \tag{4.8}$$

where α_n denotes departure in an inter-arrival time interval computed by choosing a set of Markov points [103], where the service is completed using limiting probabilities to analyse the average waiting time. Let $I_k(k \geq 1)$ be the number of packets arriving while k packets are in service and the remaining time of the current packet in service is $J(t)$. Then, the Laplace Stieltjes Transform (LST) of i_x is as follows.

$$i_x(\theta) = \left[\int_0^{\infty} \frac{(\lambda.b)^x}{x!} e^{-(\lambda+\theta)b} d(J(t)) \mid \frac{P_k+\rho-1}{P_k+\rho}\right] \frac{P_k+\rho-1}{P_k+\rho}, \tag{4.9}$$

where θ is a Laplace transform variable, x is the packet waiting to get served, and b is a constant.

Let us define $Z(x,y)$ as the average waiting time for $C_s(x,y)$ till it gets served with its waiting time less than $'t'$. Then, the probability of $Z(x,y,\theta)$ is defined as follows.

$$Z(x,y,\theta) = \int_0^{\infty} e^{-\theta t} d(Z(x,y,\theta)). \tag{4.10}$$

So, $Z(x,y)$ can be obtained as follows.

$$Z(x,y) = -lim_{\theta \to 0} \; z(x,y,\theta). \tag{4.11}$$

The conditional average waiting time τ for packets in state (x,y) is as follows.

$$\begin{aligned}\tau = \rho \sum_{x=1}^{K-1} \sum_{y=0}^{K-x-1} P_x \left[i_x Z(x,y) + C_s(x,y) \frac{(y+1)}{\lambda} i_{y+1} \right] \\ + \rho \sum_{x=1}^{K-1} \sum_{y=K-x}^{K-2} P_x \Big[i_x Z(K-y-1,y) + C_s(K-y-1,y) \\ \frac{(y+1)}{\lambda} i_{y+1} \Big] + \rho \sum_{y=0}^{K-2} P_K \Big[i_x Z(K-y-1,y) \\ + C_s(K-y-1,y) \frac{(y+1)}{\lambda} i_{y+1} \Big].\end{aligned} \tag{4.12}$$

(ii) Scheme 2: FIFO-POP Scheduling Policy

Keeping the assumptions in scheme 1, we consider that the newly arrived packet has a higher priority in comparison to the packets present in the queue. Thus, the newly arrived packets occupy the first position to be served first.

Let $C_s(x)$ be the steady-state probability of the packet placed at position $'x'$. So, the packet at position x, where $[2 \leq x \leq K]$, waits in the queue. Let there be $'n'$ incoming packets while a packet is currently in service. Then, for the packet to be served $n \leq (K-x)$, the new packet position is now shifted. The new position of x is $(x+n-1)$ after the service is completed. So, state transition equation for $C_s(x)$ is given as follows.

$$C_s(x) = \left[\prod \alpha_n(n|(x,y)) + \sum_{n=0}^{K-x} \alpha_n C_s(x+n-1) \right], \tag{4.13}$$

where $x = (2,3,..,K)$ and α_n can be computed as in the previous case. We now analyse the average waiting time using τ. Let the probability of packet at position x being served with its

service time less than t be $H(x,b)$. Then, we formulate the Laplace Transform of $H(x,b)$ as follows.

$$z(x,\theta) = \int_0^{\infty} e^{-\theta b} dH(x,b). \tag{4.14}$$

So, similar to the state-transition Eq. (13), we can obtain the recursive equation of $z(x,\theta)$ as follows.

$$z(x,\theta) = \sum_{n=0}^{K-x} \alpha_n(\theta) z(x+n-1,\theta), \;\; x = 2,3,..,K. \tag{4.15}$$

Let $Z(x)$ be average waiting time for a packet at position x till it gets served. So, as computed in Eqs. (10) and (11), the average waiting time becomes.

$$Z(x) = -lim_{\theta \to 0} \; z(x,\theta). \tag{4.16}$$

Finally, the average waiting time τ for a packet at position x is computed as follows.

$$\tau = \rho \sum_{x=0}^{K-2} \left[i_x Z(x+1) + C_s(x+1) \frac{(x+1)}{\lambda} i_{x+1} \right]. \tag{4.17}$$

Algorithm 3 Priority scheduling with multiple switches.

Input: total packets (K), flow arrival rate (λ), **Output**: average waiting time (τ).

1: **procedure** SCHEDULING
2: set λ and $1/r$ as the flow arrival rate and service rate;
3: **for** $(i = 0; i < K; i{+}{+})$ **do**
4: **if** (packet$[i]$ == highest) **then**
5: add i into Q_A;
6: **else if** (packet$[i]$ == middle) **then**
7: add i into Q_B;
8: **else if** (packet$[i]$ == lowest) **then**
9: add i into Q_C;
10: **else if** $(Q[front]$ && $Q[rear] == Null)$ **then**
11: set $Q[front] = 1$;
12: add $Q[rear] \leftarrow packet[i]$;
13: **end if**
14: update $Q[rear]{+}{+}$;
15: $C_S(x,y)$ & $C_S(x)$ be the steady state probability;
16: Calculate τ in Eq. (12) and Eq. (17) for $C_S(x)$ & $C_S(x,y)$;
17: **if** $(\tau_{C_S(x)} \leq \tau_{C_S(x,y)})$ **then**
18: set $\tau_{C_S(x)}$ is the minimum avg. waiting time;
19: **else** (set $\tau_{C_S(x,y)}$ is the minimum avg. waiting time);
20: **end if**
21: (return overflow);
22: **end for**
23: return τ;
24: **end procedure**

Algorithm 3 computes the priority scheduling using both the FIFO-PO and FIFO-POP schemes. The inputs chosen are K, λ and the output returned is τ. Initially, for each incoming

packet, it is checked whether the queue Q is empty or not. If yes, then set the $Q[front]$ at the first position and insert the incoming packets at the rear end of the queue, so the $Q[rear]$ is increment by 1. However, in case, if the incoming packet belongs to the highest priority, then these are served according to the FIFO-POP scheme which adds the packet into *Queue_A*. The next case occurs, if the packet belongs to middle priority then add into *Queue_B*, otherwise, for add the packet into *Queue_C* for lowest priority. However, if the queue is empty then add the packet on the rear end of the Queue and update $Q[rear]$ to place the priority packet in the first position otherwise, the packets are served in FIFO-PO order. So, the average waiting time is denoted by τ of both the schemes is computed by choosing the steady-state probability as $C_S(x)$ and $C_S(x,y)$. Finally, the optimal solution is computed by choosing the minimum value of τ among both the schemes. In case, the queue is already full, and then the overflow is returned. Thus, the output returned is the average waiting time τ with the minimum value.

The time complexity of the Algorithm 3 is computed as follows: The for loop in Steps (3-22) iterates K times, where K is the total number of packets in the system. Also, the conditional operators take $O(1)$ time. Thus, the total computation time is in the order of $O(K)+O(1)=O(K)$ in the worst case.

4.2.2 Efficient Flow Re-routing & Link Rate Allocation

The minimum energy consumption is achieved if re-routing is performed via a newly inserted node in the network but without changing the existing routing path. The second case of re-routing occurs if the flow is not routed through the new node. Algorithm 4 shows an efficient flow re-routing scheme and the allocation of the switch port link rates. The inputs are nodes and links in $G \in (V,E)$, where f represents the new flow and (C_p, C_l) denotes the residual capacities of the p^{th} AP and channel link l. The output returned is the minimum distance from the new source node to the gateway router with the minimum score in power and the link rate l_r allocation of the wired link. Initially, a new node V_f is inserted in (V,E) and add virtual links between V_f and APs such that it is in range of all the APs and can serve the maximum number of users. So, a priority queue of Q that covers all the nodes is created such that the initial power score of each node (*AP, switch*) from the new node is infinity, and the list of previous nodes is null. Hence, if the new node v is not the source node then insert it in Q and consider this new node as the source node by setting the distance to zero. Therefore, we perform re-routing for flow f by inserting the new node. Then, the minimum distance from the source node to each node is computed until Q is empty. Thus, for all the unvisited neighbouring nodes v of u if the pair of nodes (u,v) satisfies the bandwidth requirement of links then compute the minimum distance from u to each neighbor node v as $min\_dst[v]$ using $add\_pow(u,v)$. Here, $add\_pow(u,v)$ is the additional power score if the new node v_f is inserted in Q through path u for flow f. Then, using the Dijkstra's algorithm $min\_dst[v]$ is computed which returns the output as the shortest path after re-routing through host considering the minimum power score

for flow f.

The computational complexity of the ALgorithm 4 is in the order of $O(|A| + |S|^2 + |E|)$, where $|A|$ is the total number of access points, $|S|$ is the total number of switches, and $|W|$ is the total number of wired links in a graph $G(V, E)$.

However, once the re-routing computes the minimum distance for the new node then the different switch ports are allocated with the link rates. The power consumption of each switch port is different which depends on P_m. The function *Link_Rate* in Algorithm 4 maximizes the link rate utilization by setting an upper limit that satisfies the requirement of all the edges $e \in E$. We consider the following link rates: $l_{r_e} = \{0.01, 0.1, 1, 10\}$ Gbps for the wired link allocated as per the requirements of different modes of power utilization on APs. In case of low traffic flow, where the usage of l_{r_e} is not more than 0.01 Gbps, assign the new link rate to 0.01 Gbps which satisfies the maximum link utilization of the edges. Similarly, if the usage of l_{r_e} is more than 0.01 Gbps but less than 0.1 Gbps then assign new l_{r_e} to 0.1 Gbps. The next case checks, if l_{r_e} is more than 0.1 Gbps but less than 1 Gbps then assign l_{r_e} to 1 Gbps. The last case is the power-hungry mode where the link rate l_{r_e} is assigned 10 Gbps.

4.2.3 Efficient Flow Routing using Ant Colony Routing

The third module uses Ant Colony Routing (ACR) algorithm for adaptive multi-path routing in OpenFlow switches. ACR algorithm is a meta-heuristic approach to compute the shortest-path routing which is best suited for dynamic flow management [102]. The ACR algorithm is inspired by the biological behavior of ants having the capability to perform a local search for food in the best possible shortest path in G. The proposed approach is based on the probabilistic technique as a stochastic decision is used at each intermediate node to forward the data packets towards the destination. Here, the artificial ants play the role of the data packets in G which behaves like a software mobile agents. These data packets switch from one node to the neighbor node to compute an optimal solution between the source and the destination.

The ants explore the solution based on the pheromone which is a liquid dispersed by ants on each node along the travel path. In a centralized network controller architecture, the routing policy depends on the information maintained on each forwarding node such as– pheromone tables, data routing tables, link queues, statistical delay parameters (sample mean, variance, minimum traveling time) and so on [228]. The routing policy is built on the controller in both the directions, i.e., forward and backward which is executed on the OpenFlow switches. The forwarding ants find the shortest path from source to destination switch while the backward ants retrace the flow path for sending the acknowledgment or any updates notification back to the controller [229].

1) Forward ACR Model

The forward ACR model is denoted by $F^p_{s \to d}$, where each node consists of data routing and pheromone table to take the stochastic decision of the link and switch selection based on probability for selection of the shortest path. The selection of link and neighbor switch is

Algorithm 4 Computation of efficient flow re-routing.

Input: $G = \{V, E\}, f, (C_p, C_l)$, **Output**: $min\_dist[v]$, l_r

procedure RE-ROUTING(V, E)
    insert V_f as a new node in G;
    add virtual links between V_f and APs such that V_f is in the range of the positions of users that can be linked to AP;
    create Q as a priority queue of vertex set;
    for all $v \in G$ **do**
        $pow[v] \leftarrow \infty$;
        $visited[v] \leftarrow false$;
        $prev[v] \leftarrow NULL$;
        if ($v! = source$) **then**
            insert v to Q;
            $dist[V_f] = 0$;
        end if
    end for
    $Q.add(f)$ and calculate the shortest route;
    while ($Q \neq \emptyset$) **do**
        $u \leftarrow$ extract shortest distance $dst()$;
        remove u from Q;
        $u[visited] = true$;
        for each unvisited neighbour v of u **do**
            if ($(u, v) \leq C_l$) **then**
                $min\_dst[v] \leftarrow pow[u] + add\_pow(u, v)$;
                if ($min\_dst[v] < pow[v]$) **then**
                    $pow[v] \leftarrow min\_dst[v]$;
                    $prev[v] \leftarrow u$;
                    if ($visited[v] = false$) **then**
                        $Q.add(v)$;
                    else (set $min\_dst[v] \leftarrow pow[v]$);
                    end if
                end if
            end if
        end for
    end while
    return ($min\_dst[v]$);
end procedure
procedure LINK_RATE(V, E)
    for all $e \in E$ in W **do**
        if ($l_{r_e} \leq l_{0.01}$) **then**
            assign $l_{r_e} \leftarrow l_{0.01(Gbps)}$;
        else if ($l_{r_{0.01}} \leq l_{r_e} \leq l_{0.1(Gbps)}$) **then**
            assign $l_{r_e} \leftarrow l_{0.1(Gbps)}$;
        else if ($l_{r_{0.1}} \leq l_{r_e} \leq l_{1(Gbps)}$) **then**
            assign $l_{r_e} \leftarrow l_{1(Gbps)}$;
        else
            assign $l_{r_e} \leftarrow l_{10(Gbps)}$;
        end if
        return (l_{r_e});
    end for
end procedure

done based on two parameters (i) distance from the current switch to adjacent switch (shortest distance has the maximum probability), and (ii) the density of pheromone which is selected using the distance between the switches. The probability of switch selection by a data packet p to move forward from current switch u to neighbor switch v by using the Eq. (18) is as follows.

$$P_{u,v}^{p} = \frac{[I_{u,v}]^{\alpha}.[D_{u,v}]^{\beta}}{\sum_{v \in unvisited}[I_{u,w}]^{\alpha}.[D_{u,w}]^{\beta}}, (v \in allowed_{p}), \tag{4.18}$$

where $I_{u,v}$ is the pheromone intensity collected for transmission from u to v switch, $D_{u,v}$ is the distance between switch u and v, $I_{u,w}$ is the pheromone intensity for other possible switches w, $D_{u,w}$ is the distance from switch u to other possible switches w, the parameter to regulate the intensity of pheromone is α and the parameter to control the distance is β. The value of α is between 0 and 1 such that $0 \leq \alpha \leq 1$, where 0 refers to low intensity and 1 refers to high intensity while the value of $\beta \geq 1$.

Thus, the updated entries of the pheromone table through the pheromone matrix are used to compute the probability of choosing the outgoing link towards destination switch. The pheromone matrix lies in the interval $[0,1]$ and the sum of all the columns values is equal to 1. The pheromone has a unique property of evaporation with respect to time t. Therefore, the current updated pheromone level at a time $(t+1)$ with an evaporation rate is calculated by using the Eq. (19) as follows.

$$I_{u,v}^{p}(t+1) = (1-\varepsilon)I_{u,v}^{p}(t) + \sum_{p=1}^{K} \Delta I_{u,v}^{p}. \tag{4.19}$$

Here, K is the total number of packets, the term $\varepsilon \in [0,1]$ which denotes the evaporation rate. Suppose, the pheromone level reduces by 50% after every 4 seconds, so till 4 seconds the pheromone is 100% but after 4 seconds it is 50% then after 8 seconds is 25% and finally after 12 seconds, the pheromone is completely evaporated to 0%.

The change in pheromone is denoted by $\Delta I_{u,v}^{p}$ and is computed by using the following equation.

$$\Delta(I_{u,v}^{p}) = \begin{cases} C/Total\_D_{p}, & if(p\ chooses\ path\ u \rightarrow v), \\ 0, & \text{Otherwise}, \end{cases} \tag{4.20}$$

where C is a constant, and $Total\_D_{p}$ is the total distance traveled in G if the packet p chooses the shortest route from u to v. The forward data packets maintain a local memory M while traveling along its path to keep the record of visited switches $S_{s_1 \rightarrow s_2} = \{s_1, s_2, .., s_n\}$ and the migration time $T_{s_1 \rightarrow s_n} = \{T_{s_1 \rightarrow s_2}, ..., T_{s_{n-1} \rightarrow s_n}\}$ while moving from one hop to next hop. Also, the network load is sensed at regular intervals Δt to find the selection probability of the destination node d by the following equation.

$$p_{d} = \frac{K'_{s,d}}{\sum_{d'=1}^{S} K'_{s,d'}}, \tag{4.21}$$

where S is the number of switches in G, and K'_{sd} is the number of data packets bounded for destination switch d. Moreover, sample statistical parameter S^{p} consists of $\langle \overline{X}_{d}, Var(\overline{X}_{d}), T_{s \rightarrow d}^{min} \rangle$

to monitor the global traffic of all the network nodes. Suppose, if there are η number of effective samples comprising of $X_1, X_2, ..., X_n$ used for the exponential averages. Then the sample mean of the estimated travel time to reach switch d is computed as follows.

$$\overline{X}_d = \eta[m_{u\to d} - \overline{X}_d] + \overline{X}_d, \tag{4.22}$$

here $m_{u\to d}$ is the new measurement of the reported travel time from current switch u to d. The sample variance of the estimated travel time is calculated as follows.

$$Var(\overline{X}_d) = \eta\{[m_{u\to d} - \overline{X}_d]^2 - Var(\overline{X}_d)^2\} + Var(\overline{X}_d)^2. \tag{4.23}$$

The third parameter $T_{s\to d}^{min}$ is the minimum travel time to reach d from the present switch u.

Now, the stochastic decision policy applied at the controller to select the next-hop among v neighbors excluding the switches which are already visited is based on probability $p_{v,d}$ defined as follows.

$$p_{v,d} = \begin{cases} \frac{1}{|V_u|-1}, & \forall v \in V_u, \\ 0, & \text{Otherwise}, \end{cases} \tag{4.24}$$

where V_u is the list of all the neighbor switches of u. The probability $p_{v,d}$ can also be modified based on the load status of the network.

$$p_{v,d} = \begin{cases} \frac{I_{v,d}+\alpha.T_W}{1+\alpha.[|V_u|-1]}, & \forall v \in V_u, \\ 0, & \text{Otherwise}, \end{cases} \tag{4.25}$$

where $T_W = 1 - (D_{u,v}/\sum_{v'=1}^{V_u} D_{u,v})$ is the estimated waiting time for a newly arrived data packets. Finally, set the maximum time-to-live (TTL) value to 20 seconds to avoid cycles from the already visited switches in the memory.

2) Backward ACR Model

The backward ACR model is denoted by $B_{d\to s}^{p}$ which retraces the path used by the forward data packet but in the reverse direction. The backward model uses a priority queue rather than link queues used by the forward data packets. The statistical parameters S^p updates the mean and variance of the traveling time as the packet moves from destination towards source switch. Next, the reinforcement signal r_s is evaluated to update the data routing and pheromone table in G. Suppose, $r_s > 0$ selects a specific switch i, i.e., the neighbor of current switch u towards d based on pheromone level while j and k are the other neighbors of u then, the probability of choosing switch i is increased while the probabilities of other adjacent switches are decreased as follows.

$$I_{i,d} = I_{i,d} + r_s(1 - I_{i,d}). \tag{4.26}$$

The updated pheromone table value is increment by 1 due to the high reinforcement signal received for a neighbor switch i.

$$I_{j,d} = I_{j,d} - r_s(I_{j,d}), \quad \forall v \in V_u, \tag{4.27}$$

$$I_{k,d} = I_{k,d} - r_s(I_{k,d}), \quad \forall v \in V_u. \tag{4.28}$$

The updated pheromone table value for the other neighbor switch j or k is decrement by 1 due to low reinforcement signal. Hence, the data routing table $R^p_{j,d} = (I_{j,d})^{\varepsilon}$ is also updated after the pheromone table gets updated. Here, ε is the exponential transformation function of the pheromone table to avoid the chances of data forwarding in the wrong directions. The updated data routing table is updated as given below.

$$R^p_{j,d} = \frac{(I_{j,d})^{\varepsilon}}{\sum_{v \in V_u} (I_{v,d})^{\varepsilon}}. \tag{4.29}$$

Here, the stochastic data routing table $R^p_{j,d}$ helps to perform efficient load balancing to disperse the data packets over multi-path routing. Finally, all the intermediate links consisting of the forward path are updated. In case, if there are total s_v switches on the link then the total of $s_v(s_v-1)/2$ number of updates on switches are required.

Algorithm 5 shows the step-by-step working of forwarding and backward ACR. In the forward routing, initially, we assume a node u as a source node and set the initial travel time to 0. Then, we repeat the procedure until the time reaches the maximum iterations for all the incoming packets or until the current node reaches the destination node. The path followed hop-by-hop from source to destination is maintained in a queue Q. We computed the intensity of pheromone and probability to move from u to neighbor node v towards destination d. The travel time taken by the packet during migration from the current switch position to the next switch is recorded and the next chosen switch becomes the current switch. Then, it is checked that if the hop cycle is detected so that it is avoided to reduce the time wastage and resources. Finally, the pheromone is updated after each iteration.

The backward ACR works in the reverse direction and maintains a priority queue to communicate to the switches for the route followed by the forward data packet model which is stored in *bwd_node*. The hop count of the backward switch is automatically decremented by 1 which returns the switch ordered list maintained by the forward model. Here, the update is done for all the intermediate switches and the traveling time is increased for every forward hop count. It is used until the travel time is less than or equal to the estimated travel time to reach the destination switch. Finally, the pheromone and data routing tables are updated according to the status of the reinforcement signal. The constraints $C5, C6$, and $C9$ are satisfied by Algorithm 5 to perform an efficient flow routing in the network model. The computation time of Algorithm 5 is approximately 380 milliseconds.

4.3 Performance Evaluation

The performance evaluation of the *EnFlow* scheme is analyzed using extensive simulations. The numerical settings alongwith results and discussion are explained as follows.

4.3.1 Numerical Settings

The experiments are performed by using the simulation parameters as shown in Table 4.2. Fig. 4.4 shows the real-time traffic traces from the Survivable Network Design library

Algorithm 5 Computation of forward and backward ACR.

Input: $G = (V,E), \{s,d\}, previous\ flow$, **Output**: $F^{min}_{s\to d},\ B^{p}_{d\to s}$.

procedure FORWARD_ACR(S, D)
  $t \leftarrow 0,\ u \leftarrow$ source;
  $F^{min}_{s\to d}[fwd\_node] = u,\ T_{u\to v}[fwd\_node] = 0;$
  while $(t! = max\_iterations)$ **do**
    for $(p = 1; p \le K; p++)$ **do**
      while $(u! = d)$ **do**
        $t_{entry} \leftarrow$ current time;
        $v \leftarrow$ next hop(s,d,Q);
        calculate $I^{p}_{u,v}(t), (P^{p}_{u,v}, P_d, P_{v,d})$;
        Q[waiting time] $\leftarrow T_W$;
        $T_{u\to v}$ = current time$-t_{entry}$;
        $u \leftarrow v$;
        if $(p \in S)$ **then**
          hop cycle $\leftarrow$ check switch cycle(p,v);
          $fwd\_node = fwd\_node-$ hop cycle;
        else
          $fwd\_node = fwd\_node + 1$;
          $F^{min}_{s\to d} \leftarrow u$;
          $T_{u\to v}[fwd\_node] \leftarrow T_{u,v}$;
        end if
        update pheromone $I^{p}_{u,v}(t+1)$ and S^{p};
      end while
    end for
  end while
  return $(F^{min}_{s\to d})$ as the shortest path;
end procedure
procedure BACKWARD_ACR(D, S)
  $t \leftarrow 0, u \leftarrow$ destination;
  $B^{min}_{d\to s}[t] = u, bwd\_node \leftarrow fwd\_node$;
  while $(t! = max\_iterations)$ **do**
    for $(p = 1; p \le K; p++)$ **do**
      while $(u! = s)$ **do**
        $temp = bwd\_node - 1$;
        $S_i = S_{u\to v}[bwd\_node]$;
        Q[priority] $\leftarrow (u, S_i)$;
        $u \leftarrow S_i$;
        for $(l = temp + 1; l \le fwd\_node; l++)$ **do**
          $v = S[l]$;
          $T_{u\to v} = t + t[l]$;
          if $(T_{u\to v} \le d)$ **then**
            update table $(r_s, I_{i,d}, R^{p}_{j,d}, S^{p\prime})$;
          else
            $B^{p}_{d\to s} = u$;
          end if
        end for
      end while
    end for
  end while
  return $(B^{p}_{d\to s})$;
end procedure

Table 4.2: Simulation Parameters.

Parameter	***Value***
Simulator	OMNET++ 5.1
OS	Ubuntu 16.04 LTS
System configuration	Intel i7 with 3.20 GHz, 8 GB RAM
Topology and dataset	Fat-tree topology (4-ary), NorthAmerica
Network Controller	ONOS Controller peacock version 2.0
Programming language	C++
Switch	OpenFlow vSwitch 1.3
Number of switches & APs	35 & 10
Number of users	10, 20, 30, 40, 50
Flow arrival rate on switches	300-2000 flows/second
M/M/1 queue frame capacity	30K packets/second
Controller capacity	100K packets/second
Packet size	1000 bytes
Transition time of each port	5 seconds
Switch flow timeout	500 seconds
Link bandwidth capacity	100 Mbps
Delay rate	1 μs
Radio-propagation model	TwoRay Ground
Wireless antenna model	Omni-directional Antenna
Transmission range	250 m
Initial energy on switch	100 J
ρ_c	80 Watt
η_n	2 Watt
Power saving mode (P_L)	3 Watt
$P_{l_{0.01}}, P_{l_{0.1}}, P_{l_1}, P_{l_{10}}$	4, 8, 10, 15 Watt
Evaporation rate (ε)	0.2
Simulation time	2000 seconds

(SNDlib) [230] for performing the experiments. The fat-tree topology is used which contains the data of 35 cities of NorthAmerica zone simulated in Omnet++ 5.1. In this scenario, a single ONOS 2.0 controller [231] is placed in the network, i.e., $singleControllerofs\_Vancouver$. The nodes depicted with the white cloud symbol represents OF-enabled switches (ofs) located at different cities (Eg; $ofs\_vancouver, ofs\_Boston, ofs\_NewYork$). The connection between switch and controller is through a linear topology with a total of 135 links. The messages on links are depicted by $packetin\_queue$ and 0% on links means that there is no packet loss. We consider the time interval of 600 seconds for the ONOS controller to monitor the messages $\langle packetin\_queue, packetout\_queue \rangle$ periodically and maintains a datastore of all the activities. We run the Wireshark tool for monitoring the real-time statistics and the amount of energy consumption by each application based on CPU and memory utilization, e.g., $ovsdb-server/etc/openvswitch/conf.db$ command initially consumes 1.4% of CPU utilization which later becomes stable to 0.7% after few seconds with the memory consumption of 0.1%.

Fig. 4.5 shows the flow functions model to depict the amount of energy consumption on 10 clients connected via a switch port when multiple events occurred. Here, a single controller is

deployed with an IP address 10.0.0.2/30 which is directly connected with a ofs having an IP address 10.0.0.1/30 and each port of the switch is linked with 10 clients. The C4 constraints states that the flow between a pair of nodes (i, j) should not exceed the capacity of the link, i.e; $f_e(i, j) \leq C_l(i, j)$. We compute the residual capacity C_R of each link with respect to flow f which is calculated by taking the difference of link capacity and the current flow ($C_R = C_l(i, j) - f_e(i, j)$). The value of C_R and energy consumption on different clients $(C\_0 - C\_4)$ connected via a switch ports are computed as follows: $\{(0.0235, 47\%), (0.05, 99\%), (0.0385, 77\%), (0.049, 98\%), (0.0025, 5\%)\}$. The FIFO-POP scheduling serves the packet in the ofs queue buffer having the highest priority using the maximum link bandwidth.

Fig. 4.6(a) shows the insertion of a flow table entry on an OpenFlow-enabled switch. The flow table consists of 8×18 matrix, where 8 rows depict the flow rule entry and 18 columns represent the fields. The list of parameters in the fields are as follows: $\langle$*policy, priority*$\rangle$.

Fig. 4.6(b) shows the fast flow ACR forwarding scheme using EAR in the underlying fat-tree network. Here, in this undirected graph, the nodes depicted with big circle are routers (R), rectangle as switches (S), dark grey edges are linked between switches, light grey edges are inactive link with no traffic, small circle are users, and dot with a number ($\bullet 2$) represent the connected host count. The routing aim is to reach to destination (D) node from two source nodes, i.e., R1 and R2. Following are the solutions for this scenario.
Solution 1: $(R1 \rightarrow S1 \rightarrow S3 \rightarrow S5 \rightarrow D), (R2 \rightarrow S2 \rightarrow S4 \rightarrow S6 \rightarrow D)$ = 68% energy consumption on links.
Solution 2: $(R1 \rightarrow S1 \rightarrow S4 \rightarrow S5 \rightarrow D), (R2 \rightarrow S2 \rightarrow S4 \rightarrow S5 \rightarrow D)$ = 39% energy consumption on links.
Solution 3: $(R1 \rightarrow S1 \rightarrow S3 \rightarrow S6 \rightarrow D), (R2 \rightarrow S1 \rightarrow S3 \rightarrow S6 \rightarrow D)$ = 30% energy consumption on links.

Solution 3 is the most efficient EAR solution as the condition is to select the common links with maximum bandwidth so that the routing schemes finish the task fast and turned off the links and ports to the sleep mode.

Fig. 4.7(a), 4.7(b) shows the performance of *EnFlow* scheme using 5 users by considering the network energy consumption in switches and wireless APs with time. It shows that the energy consumption on switch is initially increased but after a time period, it becomes stable due to flow consolidation particularly, the network energy consumption in APs is lesser as compared to switch but there is small variation in energy consumption. In both the cases, we consider the time interval of 100 seconds and the simulation time as 2000 seconds.

4.3.2 Results and discussion

The results of the proposed EnFlow are compared with the existing baseline routing schemes– *RE-FPR* [88], *ILP-EAR* [99], *FFHA* [97], and *EXR* algorithms [26].

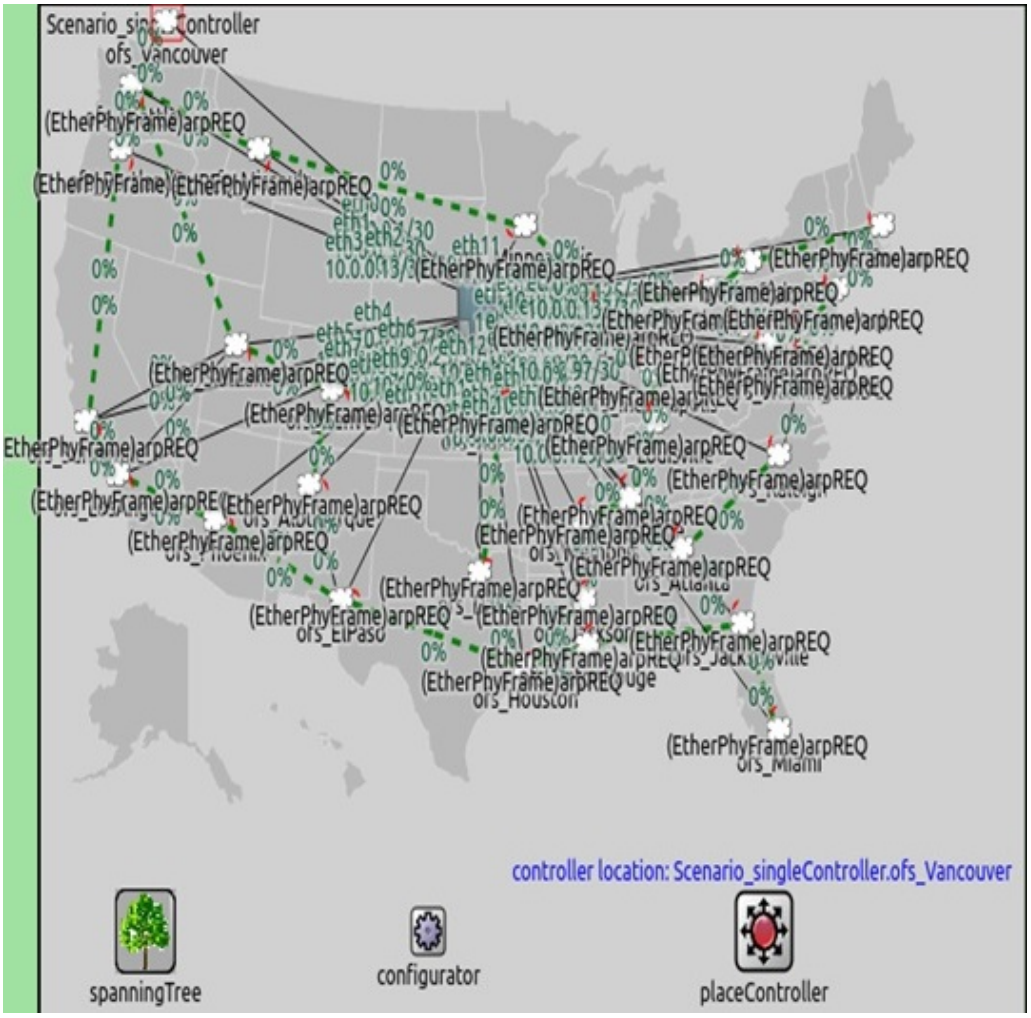

Figure 4.4: Fat-tree topology using NorthAmerica datasets for OMNET++.

4.3.2.1 Impact of network size

We analyzed the impact of network size on the energy consumption with respect to two parameters (i) energy consumption, and (ii) remaining energy. We consider two baseline routing schemes for comparison, *RE-FPR* and *ILP-EAR* as shown in Fig. 4.8(a), 4.8(b). Fig. 4.8(a) shows that the proposed scheme outperforms the *RE-FPR* and *ILP-EAR* in terms of energy consumption with an increase in the network size. The *EnFlow* scheme achieves the average energy savings upto 88.15% while the *RE-FPR* and *ILP-EAR* schemes achieve upto 63.6% and 17% respectively. The performance analysis shows that the proposed scheme is 24.55% and 71.15% more energy-efficiency as compared to the *RE-FPR* and *ILP-EAR* schemes. The reason for such behavior is that we consider the common links in routing and uses the shortest-path forwarding based on the pheromone level. The *EnFlow* scheme performs fast flow forwarding which helps to complete the flow transmission fast resulting to turn off the active network devices and links to sleep mode. The *RE-FPR* performs energy-efficiency by eliminating the redundancy and by switching off the maximum number of active devices or links as the size of the network increases along with the traffic demands. The reason for the performance degra-

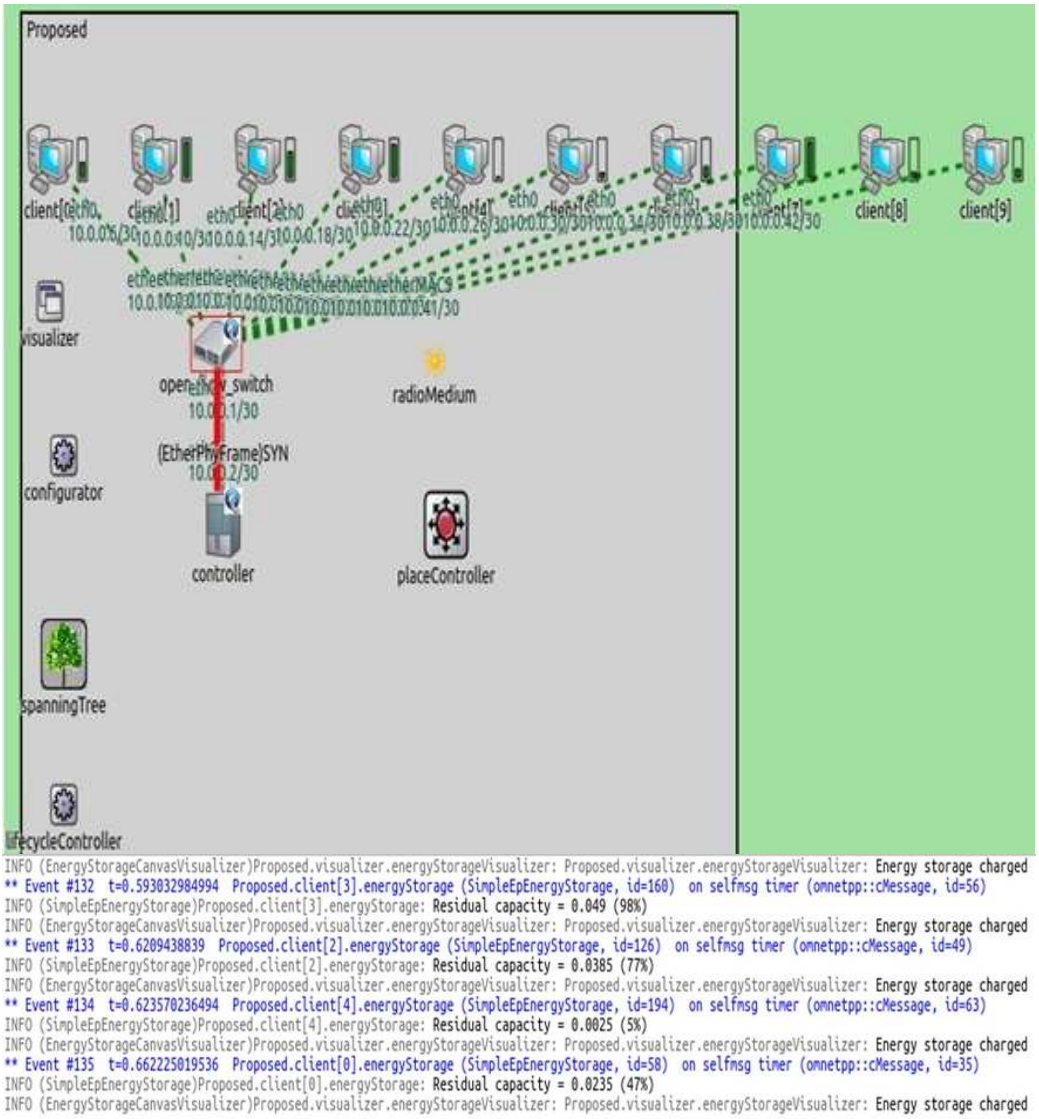

Figure 4.5: Energy consumption of each switch port connected via a OpenFlow Switch.

dation of the *RE-FPR* scheme is due to the fact that there is a frequent transition state of the active network devices that creates network oscillation during peak traffic load. The *ILP-EAR* scheme has lower energy saving because it does not have an efficient flow routing scheme to avoid congestion for the large network size in a fat-tree network topology.

Fig. 4.8(b) computes the residual capacity of links (C_R) by considering the network size into account. The maximum residual capacity $max.(C_R)$ calculates the widest-path routing of any link by choosing the maximum path available capacity through the shortest path between nodes. Here, C_R is computed by subtracting the current usage and trunk reservation from the link capacity. The performance analysis demonstrates that C_R decreases as the network size increases. The simulation results reveal that the average C_R of the *EnFlow* scheme is 83% while the *RE-FPR* and *ILP-EAR* schemes have the average remaining capacities as 36.40% and 12.06%.

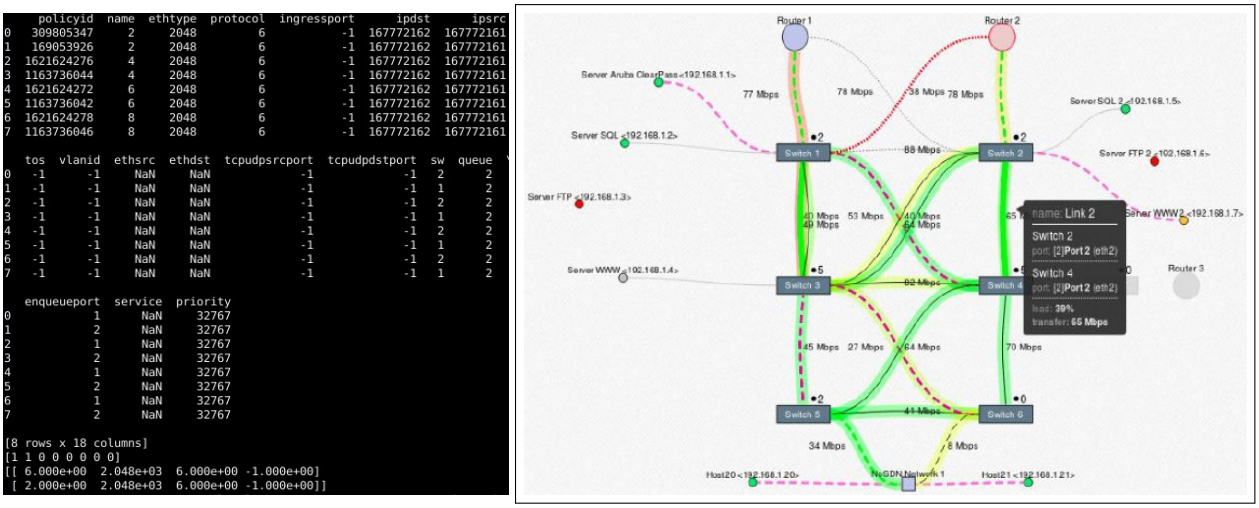

Figure 4.6: (a) Insertion of flow table entry on an OpenFlow-enabled switch. (b) Visualization of fast flow forwarding in the underlying fat-tree network.

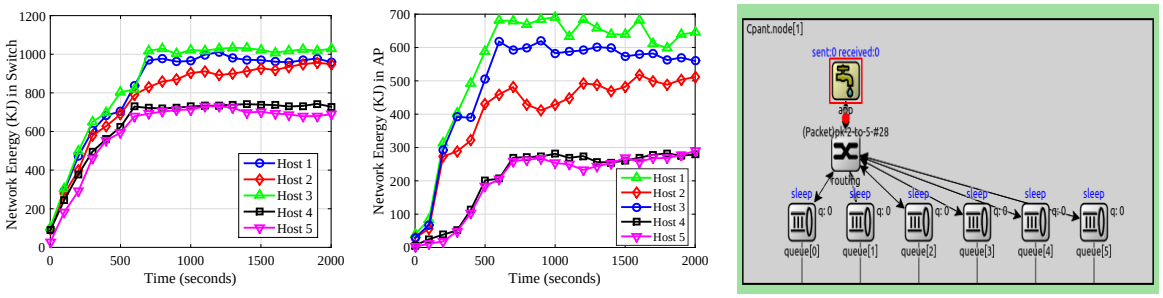

Figure 4.7: (a) Network energy consumption of switch. (b) Network energy consumption of Wireless AP. (c) Transition state of active links to sleep mode.

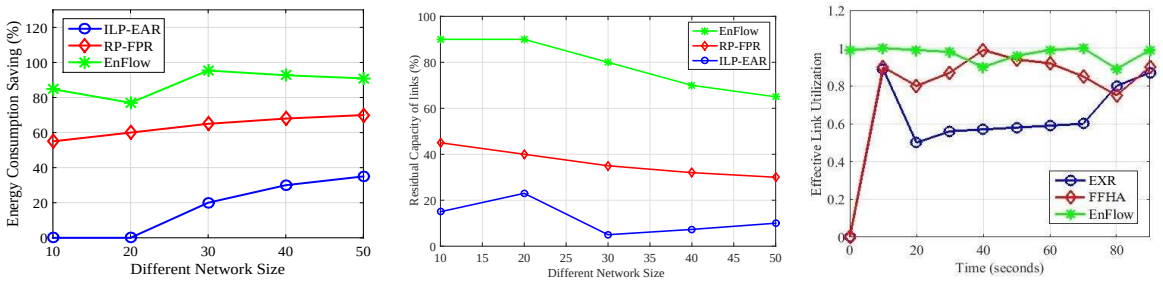

Figure 4.8: (a) Energy consumption savings vs network size. (b) Remaining energy vs network size. (c) Effective utilization ratio of links with time.

4.3.2.2 Impact of re-configuration time

The impact of re-configuration time on network links utilization is analyzed. We estimated the link utilization as the ratio of the volume of flow transmission concerning the link capacity. The energy-efficiency on the forwarding devices (switches/ports) can be achieved by maximum utilization ratio of switch links as every flow can be utilized for optimal utilization. Fig. 4.8(c) shows the comparison of the proposed scheme with both *FFHA* and *EXR* schemes.

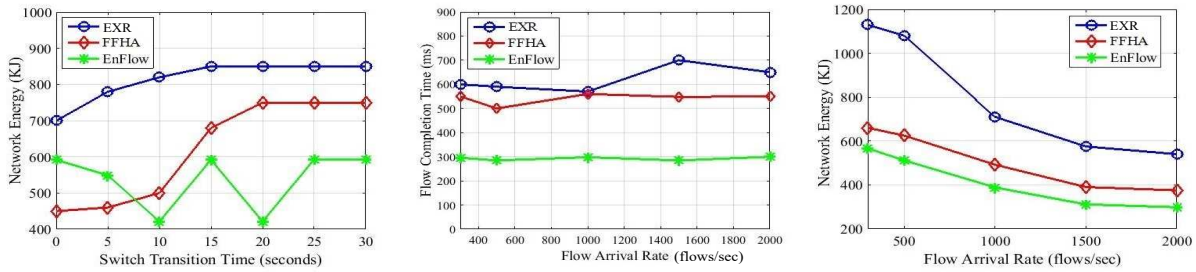

Figure 4.9: (a) Network energy vs transition time. (b) Flow completion time vs flow arrival rate in fat-tree networks. (c) Network Energy vs flow arrival rate.

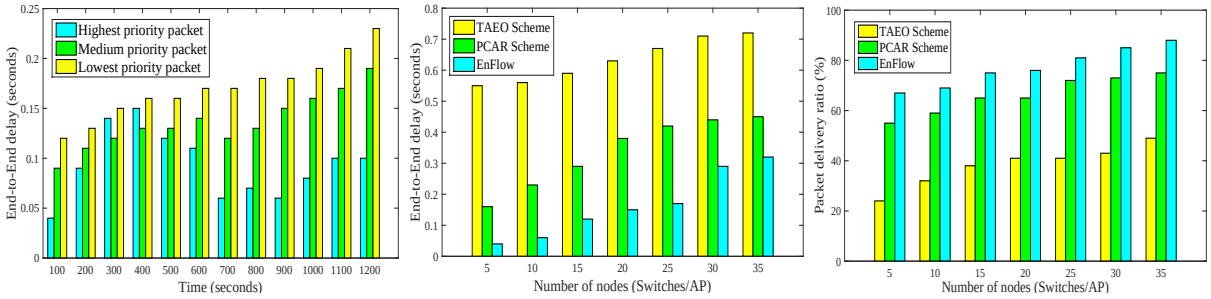

Figure 4.10: (a) End-to-End delay vs time. (b) End-to-End delay vs number of the nodes. (c) Packet delivery ratio (PDR) vs number of nodes.

The performance analysis shows that the average effective bandwidth link utilization of the proposed scheme reaches to 96.90% among all flows while the link utilization of *FFHA* and *EXR* schemes reaches to 79.2% and 59.6%. The re-configuration time of the proposed scheme is less as it achieves the utilization of active switch links because there is no flow competition in links. The reason why *EXR* achieves less link utilization in comparison to the *FFHA* scheme is that in *EXR* every flow is having the competition of network link bandwidth which results in some links over-utilized while the others are under-utilized. The proposed scheme sufficiently utilizes the link bandwidth also up to 17.7% and 37.3% more as compared to *FFHA* and *EXR* schemes. Thus, the *EnFlow* scheme outperforms the existing schemes in terms of the traffic load.

4.3.2.3 Impact of switch/AP power state transition time

Fig. 4.7(c) shows the power saving mode on a switch based on the power state transition. The symbol tap represents that the application sends any request/reply messages on an OpenFlow switch. In this scenario, a switch consists of multiple queues $q[0]-q[5]$ for scheduling the packets with no traffic flow on the links. The proposed *EnFlow* scheme achieves more power savings during no traffic flows on the links by turn off the idle links from active to sleep mode. We set the switch transition time of 5 seconds for idleness while, if the active link is idle till 5 seconds then it automatically transits to the sleep mode and again switched to active mode if

the traffic flow starts. Therefore, the proposed *EnFlow* scheme works effectively to minimize power consumption at the time of low traffic.

Fig. 4.9(a) analyzed the impact of switch transition time by varying the state transition time from 0-30 seconds corresponding to the network energy. The performance analysis shows that if the state transition time is short, then the network energy consumption is less while for longer transition time, the network energy consumption is high. The reason for this behavior is that it takes more time in switching the power modes and turn off the switch port or links from active to sleep mode and vice-versa. However, in case, if the state transition time is more than the maximum threshold limit, then it results in constant energy usage as it prevents idle links of switches to turn off to sleep mode. The results show that the average network energy usage of *FFHA* and *EXR* routing schemes is 620 KJ and 814.28 KJ respectively, while the *EnFlow* scheme consumes only 537.23 KJ on average network energy.

4.3.2.4 Impact of flow arrival rate

We consider the flow arrival rate (flows/second) denoted by λ, which follows the Poisson distribution process. Fig. 4.9(b) shows the impact of average flow completion time in milliseconds (ms) with a variation of λ in a fat-tree network. The experiment results show that when the value of λ is high in both the *EXR* and *FFHA* routing schemes, then the average flow completion time increases. The reason for such behavior is that in both the routing schemes, the large flows are competing for the bandwidth utilization of links, but the EnFlow scheme shows stable performance in the flow completion time. The proposed scheme uses FIFO-POP scheduling and constructs the flow rules according to the priority. The *EnFlow* scheme shows a variation in the link bandwidth of switches and uses clustering to serve the high priority flow with the maximum link rate. We have varied λ from 300 flows/second to 2000 flows/second and test the performance of all the three schemes. The average flow completion time of *EXR* and *FFHA* schemes is 622 ms and 541.6 ms, respectively, while, the proposed EnFlow scheme has 292.58 ms. Thus, compared to *EXR* and *FFHA* schemes, *EnFlow* schemes is 2.1 times faster than EXR and 1.8 times faster in comparison to the *FFHA* schemes.

Fig. 4.9(c) depicts the impact of network energy usage (KJ) with variation in λ. The performance analysis shows that network energy usage decreases with an increase in the values of λ. The reason for a decrease in the network energy is that the scheduling algorithm performs load balancing among multiple switches as the flow arrival rate increases to utilize the network efficiently, so it results in a better link utilization ratio. The simulation results show that *EnFlow* scheme consumes 9.72% and 40.83% lower network energy in comparison to the *FFHA* and *EXR* schemes.

The performance analysis of *EnFlow* scheme shows that if the flow arrival rate is low ($\lambda < 80$), then the energy saving on switches/APs is almost negligible. The reason for lower power savings in small flows is due to the usage of fat-tree topology which is of arbitrary size, and

users are dispersed at different geographical locations hence, it requires a long duration to turn off more active network devices and links. For large flows ($\lambda \geq 200$), the power saving on switches/APs is improved a lot, and the performance is found to be stable for the longer duration. The reason for this behavior is that flow consolidation is possible for a dispersed network to perform the transition from active to sleep mode.

4.3.2.5 Impact of end-to-end delay

In this subsection, the impact of the number of nodes (switches/AP) on average end-to-end (E2E) delay is analyzed. The average E2E delay is defined as the average time duration between packet receiving and transmission time. Fig. 4.10(a) demonstrates the impact of the E2E delay versus the simulation time for the highest, medium, and lowest priority packets. The simulation time is considered till 1200 seconds. The proposed scheme incurs longer delay for highest priority packets at the time of starting transmission but significantly the delay decreases as the simulation time increases. However, in the case of medium and lowest priority packets, the proposed packet scheduling is consistent. The experimental results show that the average E2E delay for the highest priority packets is 1.15 seconds, for the medium priority packets is 1.73 seconds, and for the lowest priority packets is 2.16 seconds. The reason for such behavior is that the priority is given for scheduling the emergency data traffic which reduces the overall E2E delay. However, the proposed scheme limits the network size in case more number of highest priority packets are generated.

Fig. 4.10(b) shows the performance of the E2E delay considering the network size. The average E2E delay of the EnFlow scheme is 1.15 seconds while the existing PCAR and TAEO schemes are 2.37 and 4.43 seconds. It is observed that the proposed scheme outperforms the existing schemes and incurs 17.43% and 46.86% lower delay in comparison to the PCAR and TAEO schemes respectively. The reason for such behavior is that the forwarding ACR finds the shortest path from source to destination switch based on pheromone table to take the stochastic decision of the link and switch selection. Moreover, the backward ants retrace the flow path for sending the updated notification back to the controller.

4.3.2.6 Impact of packet delivery ratio (PDR)

The packet delivery ratio (PDR) is defined as the ratio of the aggregate number of packets received by a receiver to the aggregate number of packets transmitted by the sender. Fig. 4.10(c) computes the PDR corresponding to the number of the nodes. The PDR of the EnFlow scheme minimizes energy consumption through effective PDR due to an optimized routing scheme. The simulation results show that the PDR of the EnFlow scheme is higher than the existing schemes. The proposed scheme achieves 11% and 39% higher PDR than that of the existing PCAR and TAEO schemes, respectively.

4.4 Summary

In this chapter, we propose an energy-efficient fast flow forwarding scheme for SDN. We modeled the EAR problem and proposed heuristic solutions comprising of three algorithms, namely–priority scheduling, flow re-routing with link adaptation, and ACR to build an energy-efficient and an optimal routing path. For evaluation of the proposed scheme, we consider a fat-tree 4-ary network topology simulated on OMNET++ using the ONOS controller. The experimental results demonstrated that the proposed *EnFlow* scheme achieves an average energy consumption savings up to 88.15% while, the *RE-FPR* and *ILP-EAR* schemes achieve energy-efficiency up to 63.6% and 17%. The simulation results show that the proposed scheme utilizes the link bandwidth up to 17.7% and 37.3% more as compared to *FFHA* and *EXR* schemes. Thus, the proposed *EnFlow* scheme improves the energy-efficiency of the network in the heavy traffic load.

Chapter 5

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

The major research contributions of this chapter are summarized below.

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

The rest of the chapter is structured as follows. Section I discusses the problem formulation.

Section II presents the proposed PARC scheme. The performance evaluation is discussed in Section III. Section IV concludes the paper and discuss future work.

5.1 Problem Formulation

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

5.1.1 SDN Architecture

The network model of a horizontal view of the distributed SDN controllers is presented in Fig. 5.1 which comprises three planes, namely, data, control, and application discussed as follows.

Data Plane: It consists of OpenFlow-enabled switches. The requests are sent as *packet-in messages* by the hosts as they are directly connected to switches such that the control plane installs the forwarding path in the form of *flow-mod messages* in the switches. The incoming *packet-in messages* arise due to new flow arrivals, link failure, or logical changes in the network topology. The edge switches follow Top-of-Rack (ToR) switch architecture, i.e., the switches are installed in racks such that every switch is directly connected to the topmost switch of each rack via a link [232]. Such ToR switches are also directly connected to the aggregate switch which in turn carries its aggregate flow. The benefits of ToR switch design are the minimum cost, reduces cabling complexity, and high data rate (the network is upgraded to 10-40 Gigabit Ethernet). Suppose, q be the total ports available at each ToR switch which helps to connect with other switches, then each ToR switch uses only p out of the available q ports to interconnect multiple switches such that $0 < p < q$. The aggregate switches are supervised by the core switches, and finally, the core switches are connected in a Peer-to-Peer (P2P) manner with the distributed controllers.

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

tal model follows the leaderless approach, unlike the master-slave approach of the hierarchical model. The benefits of the flat model are fault-tolerance (easy to handle fail-over mechanism by avoiding a single point of failure), replication (multiple copies of the global database), and reduced inter-controller latency.

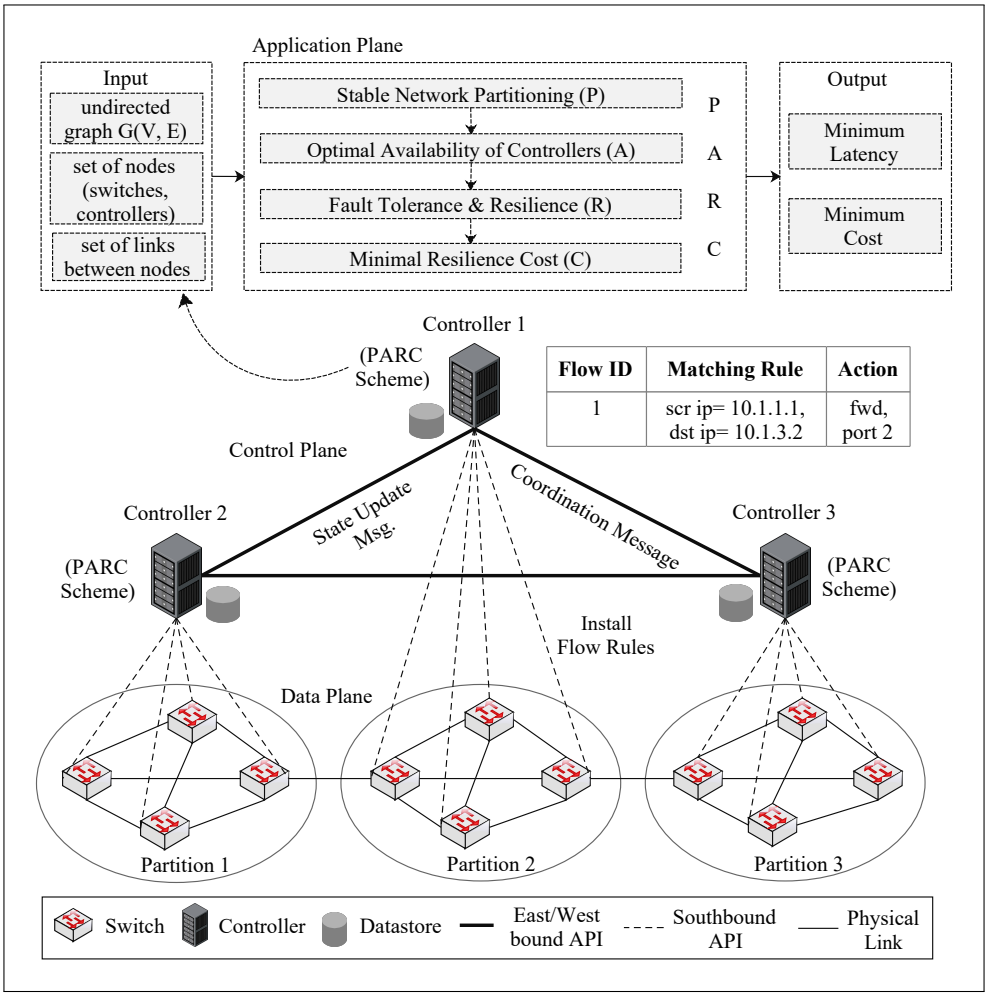

Figure 5.1: Horizontal view of the distributed SDN controllers.

Application Plane: The SDN architecture is network programmable, i.e., the programmers run multiple applications and are deployed on the control plane. A software programmer resides at the remote location to provide services to the end-users at the data plane via the control plane. Moreover, the logic of the multiple SDN applications (fault tolerance, network virtualization, and load balancers) runs remotely at the application plane whose instructions are executed on the controller through northbound interfaces. An instance migration is also an application of SDN-enabled datacenter [233]. The application of network virtualization splits a physical network into sub-networks in order to deploy atleast one distributed controller in each virtual network. Finally, the application plane ensures that the network is fully resilient with minimum network cost and delay.

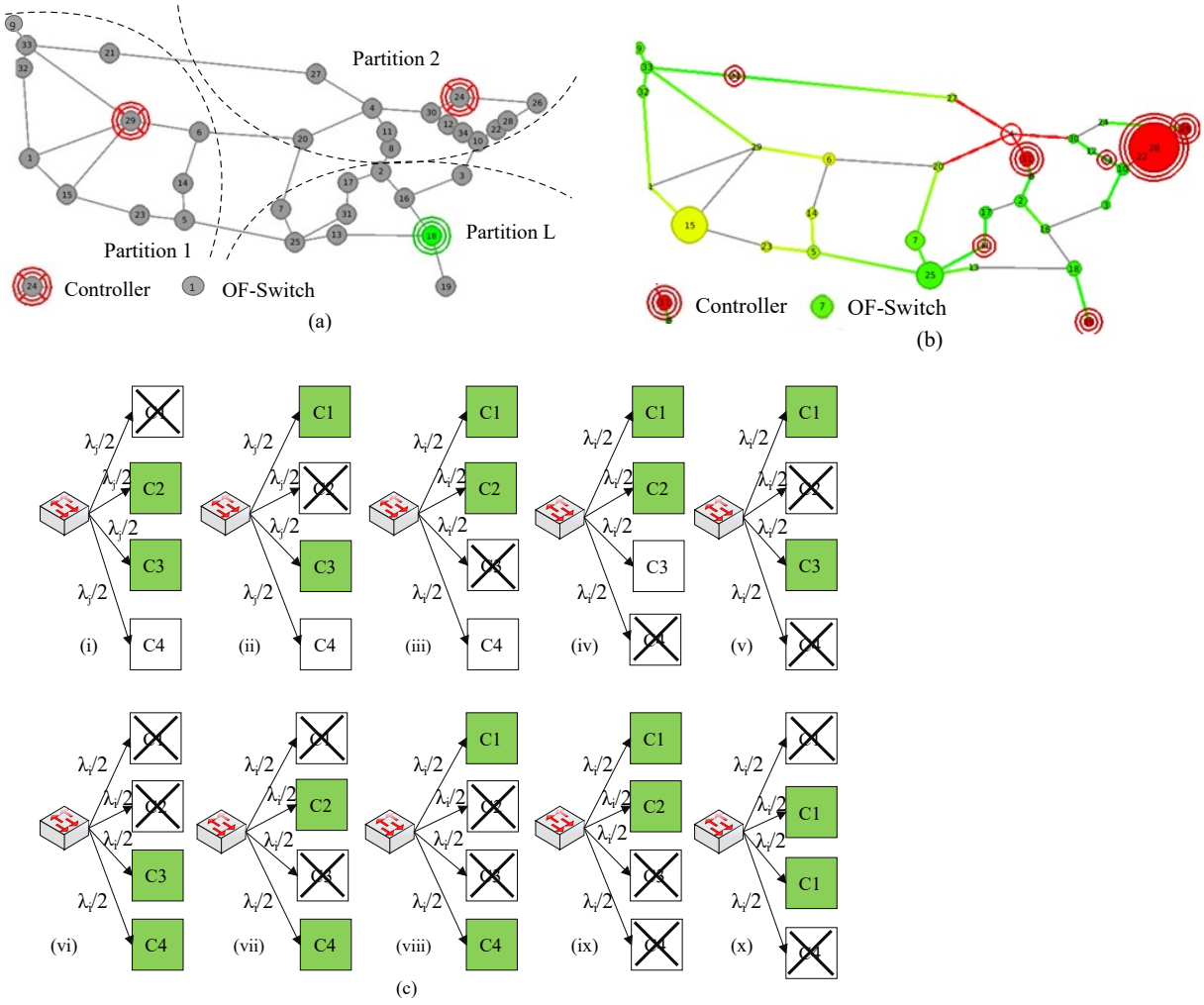

Figure 5.2: (a) Network partitioning in Internet2 OS3E. (b) Optimal controller placement. (c) Resilience in case of controller-less nodes.

5.1.2 Network Model

As discussed in the related work, the issues of multiple CPP are resolved by dividing the problem into four sub-problems. Hence, let us consider a physical network topology $G(V,E)$, where, V represents a set of nodes, and E represents a set of edges between nodes. Let $X = \{x_1, x_2, .., x_N\}$ be the set of switches, and $Y = \{y_1, y_2, .., y_M\}$ be the set of controllers in the network. Assume that, $N > M$ and $V = (X \cup Y)$ for all the nodes in G. Let A_{ij} be a matrix of shortest path distance for each pair of switches x_i and x_j. Let us assume that each distributed controller has the same processing capacity and is denoted by θ_i. Also, the flow arrival rate on switch j at time interval t is denoted by $\lambda_j(t)$. Table 5.1 shows the nomenclature of the list of symbols for mathematical description.

Table 5.1: NOMENCLATURE

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

5.1.3 Problem Statement

Fig. 5.1 represents an overview of the SDN architecture with data, control, and application planes. The distributed controllers are P2P linked to each other through an eastbound and westbound interface. The application plane takes the input as $G(V,E)$ to solve the four sub-problems by designing a Pareto near-optimal solution for the control and data planes which returns the output with minimum latency and minimum cost. Fig. 5.2(a) represents an insight into the first step of the application plane i.e., network partitioning. The physical network

is split into virtual sub-networks such that atleast one controller is mapped to supervise all the switches in each sub-network. Fig. 5.2(b) depicts the issue of optimal controller placement such that the minimum latency is incurred from controller-to-the-switch and inter-controller latency. The output returned is the stable network partitions. Fig. 5.2(c) presents the third and the fourth issue of the application plane that takes care of the fault-tolerance in case of controller or link failure. The objective is to minimize the network cost by deploying the minimum number of primary and backup controllers. The detailed explanation of each sub-problem is as follows.

5.1.3.1 Network Partitioning in SDDC

The first issue is a stable network partitioning in SDDC so that multiple controllers can be deployed in the entire network [234]. Fig. 5.2(a) depicts the splitting of the large-scale network into different sub-networks in Internet2 OS3E topology. The problem of network partitioning is formulated as a cooperative game theory using a coalition formation game. The final output is to minimize switch-to-controller and inter-controller latency.

Definition 1. *Cooperative Game– A cooperative game is denoted as a pair $g:(N,U)$, where, N denotes the set of players in the game or switches in the network and U: $2^N \rightarrow \mathbb{R}$ is a utility function from the total possible number of coalitions to the overall payoff that satisfies $U(\emptyset) = 0$. A cooperative game is defined in which all the players cooperate to work together to form a coalition so that the total payoff obtained is equally distributed among all the players [235]. The coalition formation game is defined as a set of partition denoted as $P = \{P_1, P_2, .., P_L\}$ with L partitions of N finite players (switches) such that it satisfies the following conditions:*
(i) $\forall l \in \{1,2,..,L\}, P_l \neq \emptyset$, i.e., the coalitions are not empty.
(ii) $\bigcup_{l=1}^{L} P_l = N$, i.e., the union of all sub-networks should be equal to the entire network.
(iii) $P_i \cap P_j = \emptyset$, where, $i \neq j, (i,j) \in \{1,2,..,L\}$, i.e., every partition has distinct players. Here, i and j represent the switches.

Definition 2. *Core of the Game (Player's Payoff)– Payoff is the profit or loss received by any player $i \in X$ in terms of network cost and latency from the outcome of the game generated by dividing the obtained $U(P)$. The core of the game (N,U) is defined as a set of stable payoff allocation to players such that each player has equal gain so as to stay in the coalition [236]. Let $F = \{f_1, f_2, .., f_n\}$ be the set of payoff allocation of a individual switch. The main criteria in a convex game are how to divide a payoff among the N players such that the stability of the game is achieved. The payoff allocation f is said to be optimal if $\sum_{i=1}^{N} f_i = U(N)$. The stable state of a player is the optimal reaction which maximizes their payoff to other player's stable strategies. The payoff condition is satisfied if any player in a group, disjoints the coalition in order to join another coalition then the payoff achieved is not greater than the sum of the payoff originally received from the coalition. The payoff allocation is said to be stable with respect to the coalition and is defined as $f: \sum_{i \in P} f_i \geq U(P)$. The core satisfies the two properties of payoff allocation defined as below.*

(i) Group Rational: $\{f : \sum_{i=1}^{N} f_i = U(N)\}$.
(ii) Coalitional Rational: $\{f : \sum_{i \in P} f_i \geq U(P), \forall P \subseteq N\}$.

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

$$U(P_1 \cup \{i\}) - U(P_1) \geq U(P_2 \cup \{i\}) - U(P_2), \forall P_2 \subseteq P_1 \subseteq N, i \in N. \tag{5.1}$$

Definition 4. *Shapley Value of a convex Game– It is defined as a solution in which the total value of the gain is split fairly among the players as per their individual contributions in the game [237]. The Shapley value of a player i denoted as* $\psi_i(N,U)$ *is obtained by using the formula defined as below.*

$$\psi_i(N,U) = \sum_{P \subseteq N \setminus \{i\}} [U(P \cup \{i\}) - U(P)] \frac{|P|!(n-|P|-1)!}{n!}, \tag{5.2}$$

where $U(P \cup \{i\}) - U(P)$ denotes the marginal contribution of a player i, $|P|!$ is the number of possibilities of positioning the players in the beginning of position, and $(n-|P|-1)!$ is the possible positioning of players other than player i at the termination of positioning.

In a convex game for a player i, the Shapley value is obtained as below.

$$\psi_i(N,U) = \frac{1}{|N|!} \sum_{\delta \in \zeta} (f_i^{\delta}) = \frac{1}{|N|!} \sum_{\delta \in \zeta} U(S_{\delta,\delta(i)}) - U(S_{\delta,\delta(i)-1}), \tag{5.3}$$

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

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

5.1.3.2 Optimal Availability of Distributed Controllers

The cooperative game theory discussed above considers the graph $G(V,E)$ that takes the set $X = \{x_1, x_2, .., x_N\}$ of switches as the input and returns the output as the initial number of controllers to be assigned in each sub-network. Fig. 5.2(b) illustrates the issue of the controller positioning as the switches which are comparatively far from the controller in each sub-network increase the overall latency. Initially, it is considered that each sub-network has been assigned with atleast one controller. The set of availability is used to decide the most preferred locations of each controller such that minimum latency is achieved. The optimal placement of controllers is decided on the basis of their shortest distance to the switches such that in every partition, each switch is controlled by its nearest controller. Hence, assuming the shortest distance between

controller-to-controller and switch-to-controller for the set of vertices i and j be denoted by a_{ij}. A reliability factor is defined to optimize the shortest distance by using the formula defined as follows.

$$R_{i,j} = \prod_{e \in E} \omega_e . d_{e,a_{i,j}} . \prod_{v \in V} \omega_v . d_{v,a_{i,j}}, \tag{5.4}$$

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

5.1.3.3 Resilience in Controllerless Nodes

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

Fig. 5.2(c) represents an example of resilience in case of controller-less nodes by considering $\alpha = 4$ as the number of controllers associated to a switch and $\beta = 2$ as the allowed number of backup controllers assigned to a switch to handle full resilience. Fig. 5.2(c) (i-iv) represents all the possible options of a single controller failure and Fig. 5.2(c)(v-x) represents the scenario of two controllers failure. Clearly, the system has the first $(\alpha - \beta)$ as the master controllers depicted by the green colour while the rest β is the backup controller depicted by the white colour. The inflow requests are served by the master controllers and for the flow arrival rate, λ_j of any switch j, each of associated α controllers stores a capacity of $\lambda_j/(\alpha - \beta)$ to meet the demands of the switch j. Thus, the starting $(\alpha - \beta)$ controllers are considered as the first working controllers serving the packet requests of a switch in case of no failure in the network (as mentioned in Fig. 5.2(c)).

Initially, the average traffic flows on each primary active controller are computed then the stored capacity is analyzed for the extra backup controllers. Thus, the load on each k^{th} controller is computed as follows.

$$C_k = \sum_{j=1}^{N} \lambda_j(t) R_{i,j,k}, \tag{5.5}$$

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

Table 5.2: Decision variables used in problem statement.

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

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

Mathematically, it can be represented as follows.

$$Problem\ 1 : \min \sum_{i=1}^{N} (C_i), \tag{5.6}$$

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

The explanation of each constraint is provided as follows.

- $\mu 1$ ensures that the traffic flow or number of packet-in messages processed by a core switch directly linked to a controller deployed at position i does not exceed the processing capacity of that controller θ_i. Here, the processing capacity of each controller determines the total number of switches a controller can control i.e., the maximum number of packet-in messages processed per second by each controller.

- $\mu 2$ indicates that core switch i mapped to its corresponding active controllers are all α nearest controllers. The nearest distance between the core switch and controller depends on the propagation latency a_{ij}.

- $\mu 3$ ensures that switch i mapped to all the associated α controllers are at distinct position j.

- $\mu 4$ constraint denotes that each switch is mapped to a distinct k^{th} associated controller.

- $\mu 5$ ensures that the latency between any core switch i linked with normal switch j cannot exceed the maximum allowed propagation latency $A_{\max}$ between any pair of switches (i, j).

- $\mu 6$ ensures that every switch is supervised by at least one controller.

- $\mu 7$ states that the value of a decision variable H_{ij} is binary either 0 or $\frac{1}{(\alpha-\beta)}$.

- $\mu 8$ states if the controller is one of α linked controllers at position j may avail $\frac{1}{(\alpha-\beta)}$ as a primary controllers of switch i packet-in messages.
- $\mu 9 - \mu 11$ states that the decision variables are binary.

5.1.3.4 Minimal Resilience Cost

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

Problem 2: The objective function of the problem 2 is to minimize the cost of extra backup controllers deployed in the network in such a way that every switch can be mapped with atleast two controllers in the network topology. In addition, the reliability factor $R_{i,j}$ ensures the reliability of nodes i and j using the shortest path in every partition such that each switch is controlled by its nearest controller.

Mathematically, it can be expressed as follows.

$$
\begin{aligned}
Problem\ 2 \quad &: \quad \min\ (\overline{C}) = \sum_{i \in N} (C_i - W_{ij}), \\
s.t.\ \ \mu 12 \quad &: \quad C[x]_{ij} \geq 2, \ \ \forall i \in N, \\
\mu 13 \quad &: \quad C_{total} = \sum_{i=1}^{|N|} C_i, \\
\mu 14 \quad &: \quad \alpha \in \{\beta + 2, \beta + 3, ..., C_{total}\}, \\
\mu 15 \quad &: \quad \alpha = \sum_{i=1}^{|N|} W_{i,j}, \ \ \forall i \in N,\ W_{i,j} \in \{0, 1\}
\end{aligned}
\tag{5.7}
$$

where $\mu 12 - \mu 15$ constraints are applicable in addition to $\mu 1 - \mu 11$ defined for the objective function in Eq. (5.7). The explanation of the additional constraint is provided as follows.

- $\mu 12$ ensures that every switch is supervised by atleast two controllers.

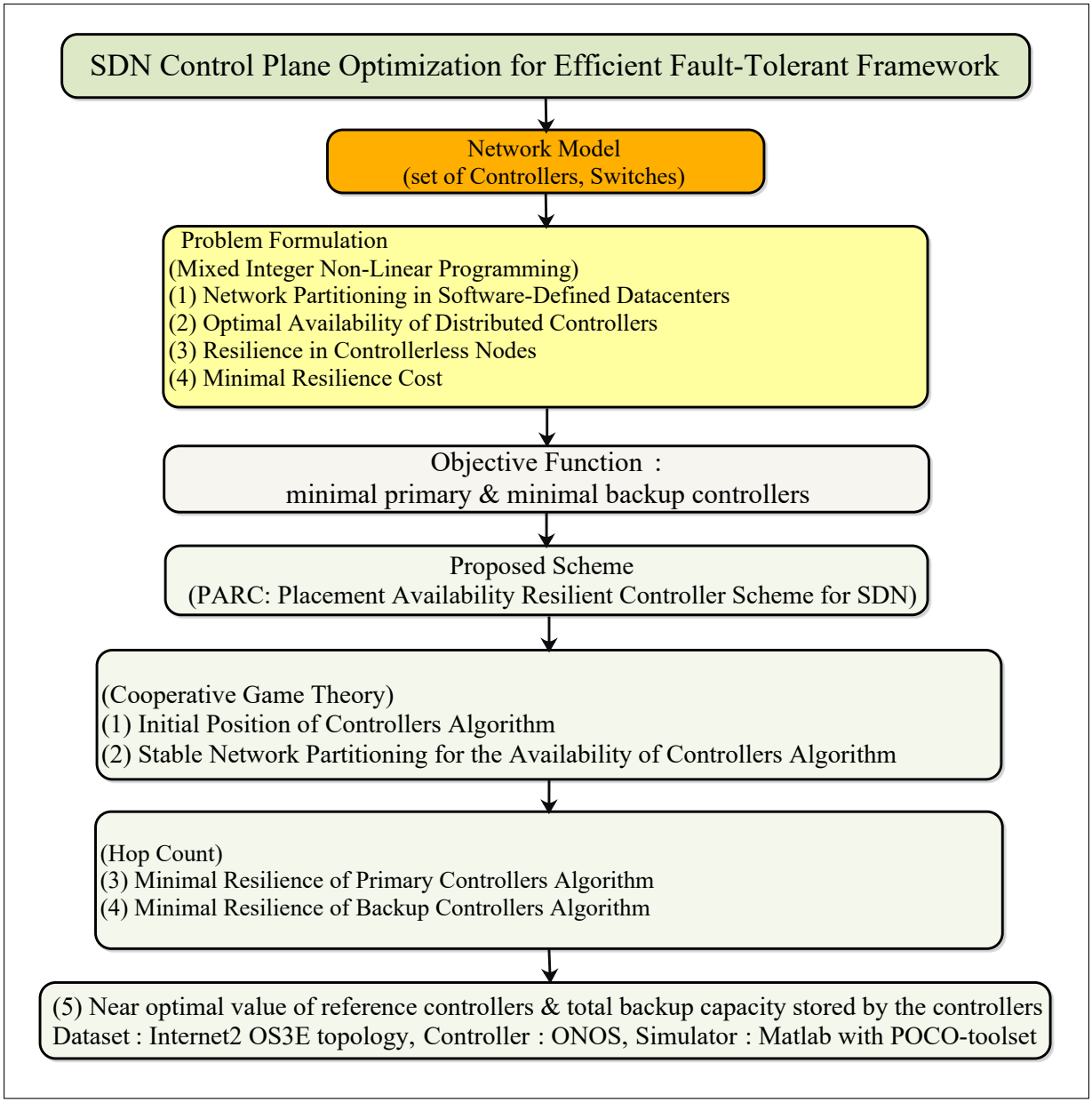

Figure 5.3: Proposed scheme PARC: Placement Availability Resilient Controller Scheme for Software-defined networks.

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

Fig. 5.3 presents the proposed scheme named as PARC: Placement Availability Resilient Controller Scheme for Software-defined networks. In the next section, the PARC scheme is discussed in details.

5.2 PARC: Placement Availability Resilient Controller Scheme

The PARC scheme is designed to addresses the aforementioned four issues of CPP in SDN. The four sub-problems are as follows (1) network partitioning based on cooperative game theory to split a physical network into virtual sub-networks (2) optimal position for the controller placement based on reliability factor $R_{i,j}$ that ensures the reliability of nodes i and j using the shortest path in every partition such that each switch is controlled by its nearest controller (3) the minimum number of primary controllers in order to minimize the network cost, and (4) the minimum number of backup controllers deployed in the network to achieve full resilience, in case, the primary controller fails in each sub-network. All these four problems are inter-related to each other, hence cannot be solved parallely. For example, the output of the Algorithm 6 is $Y_{initial}$ which is the deciding factor for stable network partitioning and is considered as the input of Algorithm 7. The initial positions of controllers are computed by selecting a switch position having maximum Shapley value. The output of the Algorithm 7 is Y_{final}, Z which is considered as the input of Algorithm 8. The final stable positions of the controllers are computed based on the euclidean distance between switch pairs assigned to the same controller, and from the switch to the controller. Finally, the output of Algorithm 8 is C which is the input of Algorithm 9. The minimum number of primary controllers/core switches are computed based on the adjacent switches having only one hop distance. Next, the output of the Algorithm 9 is $\overline{C}$ which is the set of backup controllers which are directly linked to the core switches in a single hop distance. Finally, Algorithm 10 computes the near optimal values of α, and C_{total}. Therefore, the sub-problems are solved one by one as the output of the first algorithm is the input of the next algorithm. The Pareto near optimal solution of the four optimization problems is described as follows.

5.2.1 Network Partitioning based on cooperative game theory

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

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

$$a_{ij} = \sqrt{(i_1 - j_1)^2 + ... + (i_n - j_n)^2} = \sqrt{\sum_{s=1}^{n} (i_s - j_s)^2}, \quad \forall i, j \in N. \tag{5.8}$$

Therefore, the similarity function is defined as below.

$$\gamma[a_{ij}] = 1 - \frac{a_{ij}}{(1+A_{\max})}, \tag{5.9}$$

where $A_{\max}$ is the highest value of propagation latency between any pair of switches. Thus, the utility function for the set of partitions $P = \{P_1, P_2, .., P_L\}$ is given as below.

$$U(P') = \frac{1}{2} \sum_{\substack{x_i, x_j \in P' \\ x_i \neq x_j}} \gamma[a_i j],\ where, P' \subseteq X, P' \neq \emptyset. \tag{5.10}$$

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

$$\begin{aligned} U(P_1 \cup \{x_t\}) - U(P_2 \cup \{x_t\}) &= \frac{1}{2} \sum_{\substack{x_i, x_j \in P_1 \\ x_i \neq x_j}} \gamma[a_{ij}] \\ &+ \sum_{x_i \in P_1} \gamma[a_{it}] - \frac{1}{2} \sum_{\substack{x_i, x_j \in P_2 \\ x_i \neq x_j}} \gamma[a_{ij}] + \sum_{x_i \in P_2} \gamma[a_{it}], \\ &= U(P_1) - U(P_2) + \sum_{x_i \in P_1 \backslash P_2} \gamma[a_{ij}], \\ U(P_1 \cup \{x_t\}) - U(P_2 \cup \{x_t\}) &\geq U(P_1) - U(P_2), \end{aligned} \tag{5.11}$$

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

$$\begin{aligned} \psi_i(X,U) &= \frac{1}{|N|!} \sum_{\delta \in \zeta} (f_i^{\delta}) = \frac{1}{|N|!} \sum_{\delta \in \zeta} U(S_{\delta,\delta(i)}) - U(S_{\delta,\delta(i)-1}), \\ &= \frac{1}{|N|!} \sum_{\delta \in \zeta} \left[\frac{1}{2} \sum_{\substack{x_c, x_d \in S_{\delta,\delta(i)} \\ x_c \neq x_d}} \gamma[a_{cd}] - \frac{1}{2} \sum_{\substack{x_c, x_d \in S_{\delta,\delta(i)-1} \\ x_c \neq x_d}} \gamma[a_{cd}] \right], \\ &= \frac{1}{|N|!} \sum_{\delta \in \zeta} \left[\sum_{\substack{x_c, x_d \in S_{\delta,\delta(i)} \\ x_c \neq x_d}} \gamma[a_{cd}] - \sum_{\substack{x_c, x_d \in S_{\delta,\delta(i)-1} \\ x_c \neq x_d}} \gamma[a_{cd}] \right], \\ &= \frac{1}{|N|!} \sum_{\delta \in \zeta} \left[\sum_{\delta(j) < \delta(i)} \gamma[a_{ij}] \right] = \frac{1}{|N|!} \sum_{\substack{\delta \in \zeta, \delta(i)=1 \\ \delta(j) < \delta(i)}} \gamma[a_{ij}], \\ &+ \frac{1}{|N|!} \sum_{\substack{\delta \in \zeta, \delta(i)=2 \\ \delta(j) < \delta(i)}} \gamma[a_{ij}] + .. + \frac{1}{|N|!} \sum_{\substack{\delta \in \zeta, \delta(i)=n \\ \delta(j) < \delta(i)}} \gamma[a_{ij}], \end{aligned}$$

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

$$\psi_i(X,U) = \frac{1}{|N|!}\frac{(N-1)!}{(N-1)N!}[1+2+..+(N-1)] \sum_{x_j \in X, i \neq j} \gamma[a_{ij}]. \tag{5.12}$$

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

Algorithm 6 computes the initial position for deploying distributed controllers which is the deciding factor for splitting the entire network into sub-networks. The input of the algorithm is the cooperative game $g : (X,U)$, the flow arrival rate denoted by $\lambda_j(t)$, the processing capacity of each controller denoted by θ_i, the shortest distance latency matrix between any two pairs of switches denoted by A_{ij}. In addition, ε is the similarity threshold parameter, where $\varepsilon \in \{0,1\}$ that decides the number of sub-networks. The output is the set of initial position of controllers denoted by $Y_{initial}$.

Algorithm 6 Computation of initial position of controllers.

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

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

Step 2 is initialized with a set of the initial position of controllers $Y_{initial}$ as empty and assumed that $X^\star$ as a set of the initial position of static switches in step 3. Initially, we consider that $X^\star$ be the position of controllers. Steps 5-7 execute the *for* loop to compute the Shapley value $\psi_i(X,U)$ for all the switches. Then in steps 8-17, the *while* loop works iteratively until

the $Y_{potential}$ is not empty to check if the switch demand is less than the controller processing capacity. If yes, then computes the initial position of controllers $Y_{initial}$ by selecting a switch position having maximum Shapley value in each sub-network; otherwise, the controller rejects the switch demands. Then we check, the similarity between switch pairs based on the Euclidean distance (a_{ij}) and assign those switches set in the same partition, i.e., at least ε similar to $temp$. Finally, the updated set of initial controller positions $Y_{initial}$ is returned as an output.

5.2.2 Availability of Controllers based on Stable Partitioning

The second phase computes the stable network partitioning for the final placement of controllers Y_{final}, and the mapping pair between controllers and switches denoted by Z that determines which switch is assigned to a specific controller. Algorithm 7 takes input as the initial position of controllers and returns the output as (Y_{final}, Z). Initially, we consider the set (Y_{final}, Z) as empty. Then, the *while* loop repeats the process iteratively from steps (3-21) until $(Y_{initial} \neq Y_{final})$. The first *for* loop iteratively repeats the process in steps (4-10) until the mapping of switches to a particular controller is done on the basis of shortest distance from the switch to the controller in each partition. Hence, the closest controller is assigned to each switch in every sub-network, and the updated switch-controller mapping pairs are finally stored in a set Z. The second *for* loop in steps (11-17) iteratively repeats the process in order to find the shortest distance between every switch pairs $(a_{x_i x_r})$ controlled by the same controller and is computed in $temp$. The updated partition is finally stored in Y_{final}. However, still in case $(Y_{initial} \neq Y_{final})$ then the updated Y_{final} position is the current position of controllers and returns as an output.

Time Complexity of Algorithm 7: Steps (4-11) repeats the first *for* loop with respect to the number of switches and takes $O(n)$ times in the worst case. Also, in steps (12-19) in the second *for* loop iterates with respect to the number of controllers and takes $O(n)$ times in the worst case. In addition, steps (20-22) are conditional operations which take $O(1)$ times. Finally, the time complexity is computed as below.

$\Longrightarrow T(n) = [O(n) + O(n) + O(1)] = O(n)$. The overall time computation of Algorithm 7 is 56 seconds that iteratively executes the *for* and *while* loops.

5.2.3 Minimal Resilience Computation of Primary Controllers

In this phase, the PARC scheme solves the third issue of resilience with minimal number of primary controllers. Algorithm 8 computes the minimum number of controllers required with maximum coverage while reducing the overall cost of the network. As core switches are linked directly to the controllers, so the least number of core switches required in the network is computed. The algorithm takes the input as $(G(V,E), \lambda_j(t), \theta_i, h, Z)$, where h is the threshold on the hop count, and Z is the record of the switches deployed to a single controller. The output set C is the set of switches directly connected to a controller, i.e., the set of core switches.

Initially, C and $x_i[adj]$ are considered as empty sets, where $x_i[adj]$ stores the record of the

Algorithm 7 Computation of stable network partitioning.

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

procedure PARTITION
  initialize $(Y_{final}, Z) = NULL$;
  while $(Y_{initial} \neq Y_{final})$ **do**
    for $(i = 1; i \leq N; i{+}{+})$ **do**
      if $(x_i.\lambda_j(t) \leq \theta_i)$ **then**
        Calculate the euclidean distance between switch and controller:
        $temp = arg\{\min(a_{x_i.y_j})\},\ \forall x_i \in X, y_j \in Y$;
        $Z \leftarrow Z_{temp} \cup \{i\}$;
      else (controller rejects the switch demands);
      end if
    end for
    for $(j = 1; j \leq M; j{+}{+})$ **do**
      if $(x_i.\lambda_j(t) \leq \theta_i)$ **then**
        Calculate the euclidean distance between switch pairs assigned to the
        same controller: $temp = arg\{\min(a_{x_i.x_r})\}\ \forall x_i, x_r \in Z$;
        $Y_{final} \leftarrow Y_{final} \cup \{temp\}$;
      else (controller rejects the switch demands)
      end if
    end for
    if $(Y_{initial} \neq Y_{final})$ **then**
      $Y_{initial} \leftarrow Y_{final}$;
    end if
  end while
  return (Y_{final}, Z);
end procedure

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

Next, the function coverage is executed that covers the list of uncovered switches in multiple hops by the core switches based on a depth-first search in $G(V,E)$. Initially, it is checked if the hop distance is less than the allowed threshold h of hop distance and also the controller capacity is enough to serve the switch demands then cover the uncovered switches. The *for* loop in steps (22-28) continuously repeats the process that checks if any switch x_j is not supervised by any controller then switch j is supervised by the core switch and the controller residual capacity decreases by one and the value of hop count increases by one. Finally, the updated output returns are in Z to store the number of switches assigned to the controllers.

Time Complexity of Algorithm 8: The iterations in steps (6-15) in the first *while* loop computes the least number of controllers required in the network and takes $O(n)$ times in the worst case. Also, in steps (22-28) in the second *for* loop in the *Coverage* function which iterates with respect to the utmost number of core switches supervised by the distributed controllers and takes $O(n)$ times in the worst case. Finally, the time complexity is computed as below.

$\Longrightarrow T(n) = [O(n) + O(n)] = O(n)$. The overall time computation of Algorithm 8 is 67 seconds that iteratively executes the *for* and *while* loops.

5.2.4 Minimal Resilience Cost of Backup Controllers

In the final phase, the Algorithm 9 computes the minimal cost of resilience to minimize the number of backup controllers required in the case, some of the switches are left as controller-less nodes. The concept of backup controllers states that each switch is supervised by two controllers in $G(V,E)$. In addition, the algorithm performs synchronization of events among controllers at the control plane by assigning a timestamp for the proper ordering of updated events. The input of the algorithm are $(G(V,E), \lambda_j(t), \theta_i, C, Z^\star)$, where, $Z^\star$ maintains the record in which switches are assigned two controllers. The output returns are the set of backup controllers denoted by $\overline{C}$ which are directly linked to the core switches. Initially, the record is built by the number of adjacent controllers of x_i and $Z^\star$. Then for all the switches check, whether the controller in set C is fully utilized. Next, in steps (11-22), the *while* loop repeats the process to

Algorithm 8 Minimal resilience of primary controllers.

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

procedure CORE
  initialize $(x_i[adj], C) = ;$
  initially set $sum = 0, hop = 1, core[x_i] = 0;$
  build set of adjacent switches to x_i in $G(V,E) : \overline{x} \leftarrow x_i[adj];$
  while (for switches not in Z) **do**
    for $(i = 1; i \leq N; i{+}{+})$ **do**
      if (x_i is an unsupervised switch) **then**
        $total \leftarrow count[\overline{x}];$
        if $(sum \leq total)$ **then**
          $sum \leftarrow total;$
          $core[x_i] \leftarrow i;$
        end if
        $C \leftarrow C \cup \{core[x_i]\};$
      end if
    end for
  end while
  $Z \leftarrow$ COVERAGE$(Z, \lambda_j(t), \theta_i, hop, h, core[x_i], total, sum);$
  return $(C);$
end procedure
procedure COVERAGE
  if $((hop \leq h)$ && $(x_i.\lambda_j(t) \leq \theta_i))$ **then**
    for $(j = 1; j \leq \overline{x}; j{+}{+})$ **do**
      if (x_j is an unsupervised switch) **then**
        switch j is covered by $core[x_i];$
        $\theta \leftarrow (\theta_i - 1);$
        $hop{+}{+};$
      end if
    end for
  end if
  COVERAGE$(Z, \lambda_j(t), \theta, hop, h, x_i);$
  return $(Z);$
end procedure

find the number of extra core switches until all the switches are not assigned to two controllers. Then for all the switches check, if it is a normal switch then it again repeats a similar process to compute the extra number of core switches and returns the output in set $\overline{C}$.

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

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

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

The overall time computation of Algorithm 9 is 73 seconds that iteratively executes the *for* and *while* loops.

5.2.5 Analysis of the proposed scheme

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

5.2.6 Cost analysis of the proposed scheme

Fig. 5.2(c) illustrates the mapping of a switch to all its reference controllers. Here, $\alpha = 4$ is the total number of controllers associated to a switch with $\beta = 2$ as the allowed number of backup controllers and each reference controller holds a capacity of $\frac{\lambda_j}{(\alpha-\beta)}$ times the switch demand. In this scenario, our goal is to minimize the network cost with resilience in case of previously known β controllers. In order to reduce the backup capacity and the network cost, switches linked to two different primary controllers working without any failure can rely on the same backup capacity stored on another controller. In case, the primary controller C_j^* fails then the incoming traffic on the switch is transmitted to the backup controller C_j which is calculated

Algorithm 9 Minimal resilience cost of backup controllers.

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

procedure COST
    initialize $(x_i[adj],\ \overline{C}) = \emptyset$;
    initially set $sum = 0, core[x_i] = 0$;
    build set of adjacent switches to x_i in $G(V,E) : \bar{x} \leftarrow x_i[adj]$;
    build $Z^\star$ from set C;
    for $(i = 1; i \leq N; i{+}{+})$ **do**
        if $(x_i.\lambda_j(t) < \theta_i)$ **then**
            Controller is capable of serving more switches;
        end if
    end for
    while (for switches not in $Z^\star$) **do**
        for $(i = 1; i \leq N; i{+}{+})$ **do**
            if (x_i is a normal switch) **then**
                $total \leftarrow count[\bar{x}]$;
                if $(sum \leq total)$ **then**
                    $sum \leftarrow total$;
                    $core[x_i] \leftarrow i$;
                end if
                $\overline{C} \leftarrow \overline{C} \cup \{core[x_i]\}$;
            end if
        end for
    end while
    $Event_T \leftarrow$ TIMESTAMP$(Event_T, \lambda_j(t), \theta_i, Event_{no.}, term_{no.}, x_{ID})$;
    return $(\overline{C})$;
end procedure
procedure TIMESTAMP
    initialize $(term_{no.},\ Event_{no.}) = \emptyset$;
    x_{ID} be a unique switch Identifier;
    if $(x_i.\lambda_j(t) \leq \theta_i)$ **then**
        for $(j = 1; j \leq M; j{+}{+})$ **do**
            if $(x_{ijk} == 1)$ **then**
                $term_{no.}{+}{+}$;
            end if
            $Event_{no.}{+}{+}$;
            $Event_T = (x_{ID}||term_{no.}||Event_{no.})$;
            $C_{|M|} \leftarrow C_j(Event_T)$;
            update $Event_T$;
        end for
    end if
    TIMESTAMP$(Event_T, \lambda_j(t), \theta_i, Event_{no.}, term_{no.}, x_{ID})$;
    return $(Event_T)$;
end procedure

by using the formula:

$$Q_{j^*,j} = \sum_{i=1}^{|N|} \Bigg[\sum_{\substack{k^*=1 \\ \forall j \in N, j^* \in N, j \neq j^*}}^{\alpha-\beta} R_{i,j^*,k^*} \sum_{k^*=(\alpha-\beta+1)}^{\alpha} R_{i,j,k} \Bigg] \frac{\lambda_j}{(\alpha-\beta)}, \tag{5.13}$$

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

$$Q_j = \max_{\beta} \Big(\beta, \ Q_{j^*,j} \Big), \ \forall j \in N, j^* \in N, j \neq j^*, \tag{5.14}$$

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

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

5.2.7 Backup controller capacity analysis

The additional backup capacity stored at a pre-specified β number of controllers is inversely proportional to α (reference controllers) and is defined as

$$\Big(\sum_{j=1}^{|N|} \beta \frac{\lambda_j}{(\alpha-\beta)} \Big). \tag{5.15}$$

In order to minimize the value of backup capacity at controllers, the aim is to maximize the number of reference controllers α. It can be formulated as: $(\max \ \alpha)$.

s.t. constraints $\mu 13, \mu 14, \mu 15$ defined in Eq. (5.7).

Algorithm 10 Computing optimal value of α, C_{total}.

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

procedure CAPACITY
    initialize $\alpha = \beta + 2$;
    $\alpha^* = \infty$;
    C = call_Algorithm 8$(G(V,E), \lambda_j(t), \theta_i, \beta, a_{ij})$;
    $C_{total} = C$;
    while $(\alpha != \alpha^*)$ **do**
        $C = C_{total}$;
        $\overline{C}$ = call_Algorithm 9$(G(V,E), \lambda_j(t), \theta_i, \beta, a_{ij})$;
        $C = \overline{C}$;
        C_{total} = call_Algorithm 8$(G(V,E), \lambda_j(t), \theta_i, \beta, a_{ij})$;
    end while
    $\overline{\alpha} = \alpha^*$;
    $\overline{C}_{total} = C_{total}$;
end procedure

Algorithm 10 computes the near optimal values of α and C_{total} by taking input parameters as $G(V,E), \lambda_j(t), \theta_i, \beta, a_{ij}$. The values of α and C_{total} are dependent on each other and initially, it is considered that $\alpha = \beta + 2$. The solution of minimum cost (Problem 1) is used returns the value of C_{total}. Then, the solution of minimum backup (Problem 2) takes C_{total} as input to return the updated value of α as α^*. Similarly, α^* is used as input in Problem 1 to evaluate the updated value of C_{total} as $\overline{C}_{total}$. These two steps are iteratively performed till the value of α does not changed any further resulting in near optimal values as $\overline{\alpha}, \overline{C}_{total}$.

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

$$\left(\sum_{j=1}^{|N|} \beta \frac{\lambda_j}{(\overline{\alpha} - \beta)} \right). \tag{5.16}$$

5.2.8 Resilience analysis

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

$$\min \left(100,\ \max \left[\frac{(\alpha - n_F)100}{\alpha - \beta},\ \ 0 \right] \right), \tag{5.17}$$

where n_F denotes the number of controller failures.

Now in this case, two situations arises (i) $n_F \leq \beta < \alpha$, where the proposed model achieves

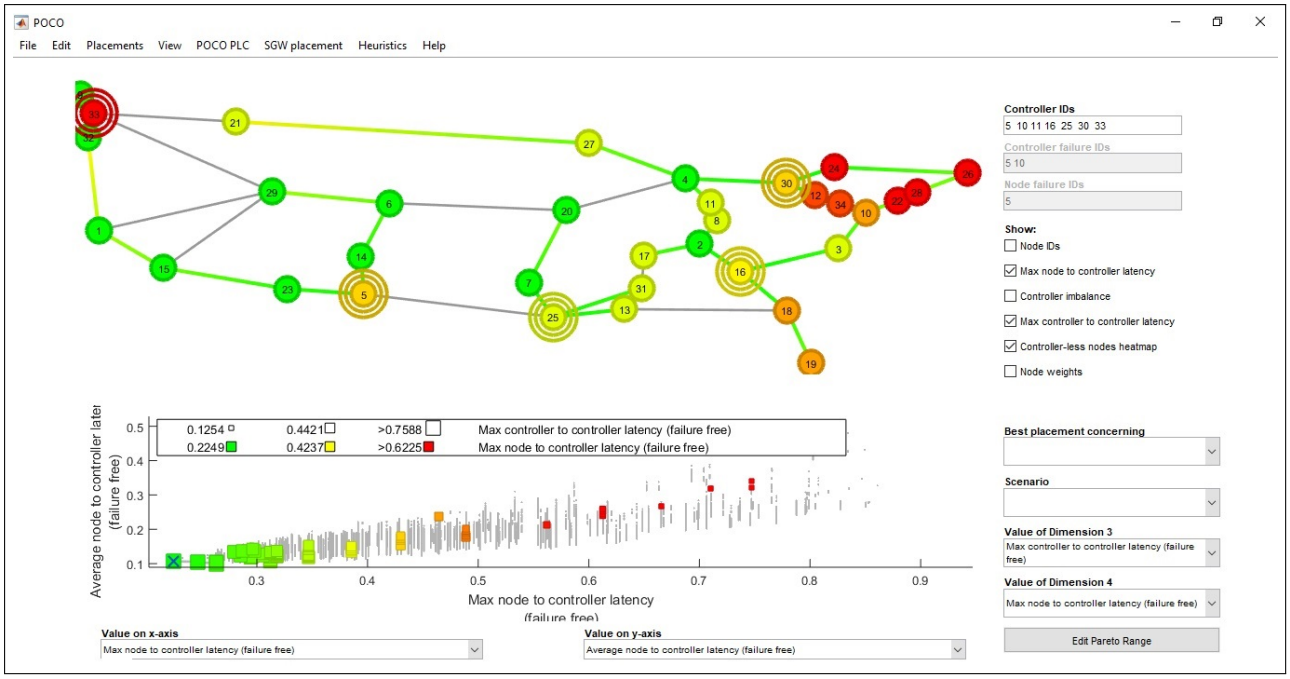

Figure 5.4: Optimal Controller placement with k= 5 on Internet2 OS3E topology by considering node-to-controller and controller-to-controller latency.

complete resilience with 100% traffic flow is unaffected (ii) $\beta < n_F < \alpha$, where the model achieves only partial resilience against n_F controller failures.

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

5.3 Performance Evaluation

5.3.1 Numerical Settings

The list of parameters used in numerical settings to perform the experiments are listed in Table 5.3.

Simulation results are tested on Internet2 OS3E, Planet V2, and Jellyfish topology by considering the latency between node-to-controller and inter-controller as shown in Fig. 5.4 and Fig. 5.5(a) and Fig. 5.5(b). In this setup, the topology consists of 34 nodes with $k = 5$ as the number of controllers placed in the network which are denoted by double-circles nodes while the rest 29 nodes are OpenFlow virtual switches.

Both Algorithm 6 and Algorithm 7 uses the concept of cooperative game theory with k-medoids, which results in a Pareto near-optimal solution for the placement of controllers, as shown in Fig. 5.5(a). The cooperative game with k-medoids partition the network into four and five sub-networks resulting in the latency of 8.96 milliseconds and 6.51 milliseconds in the worst case. Fig. 5.5(b) shows the Graphical User Interface (GUI) representation of the best placement for Planet V2 topology in case of controller imbalance and controller-less nodes.

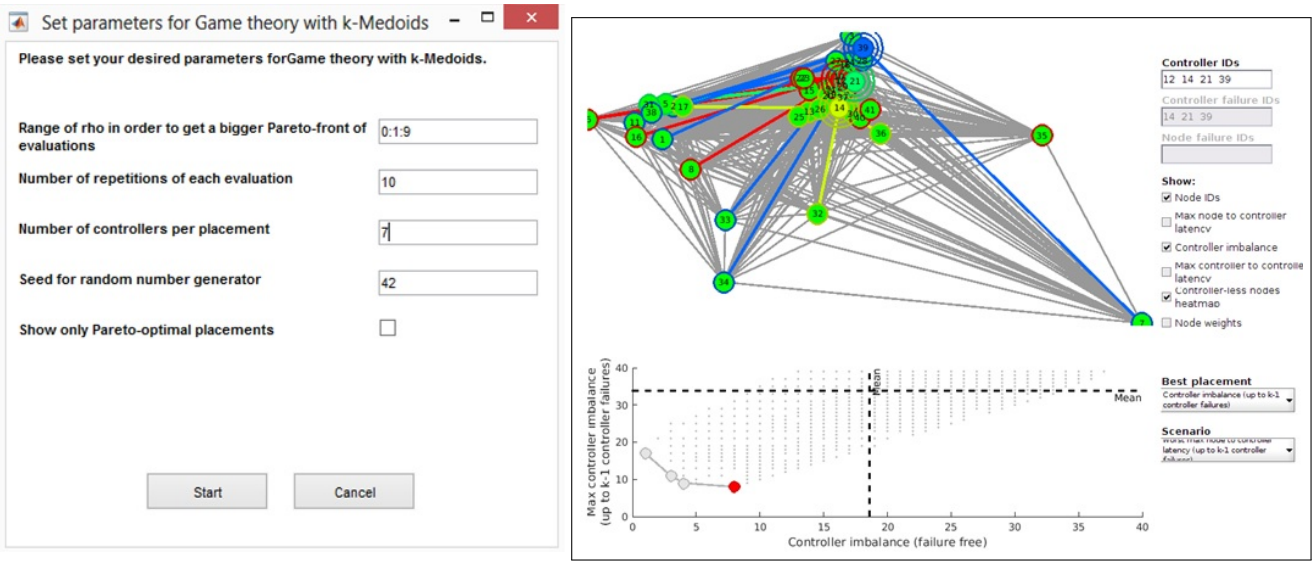

Figure 5.5: (a) Pareto optimal placement using cooperative game with k-medoids. (b) Controller imbalance and controller-less nodes on Planet V2 topology.

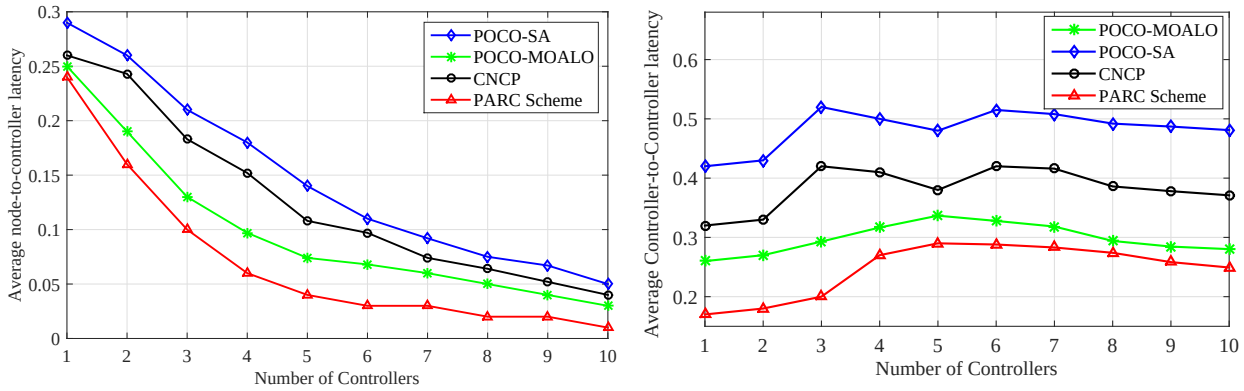

Figure 5.6: (a) Average node-to-controller latency vs number of controllers. (b) Average controller-to-controller latency vs number of controllers.

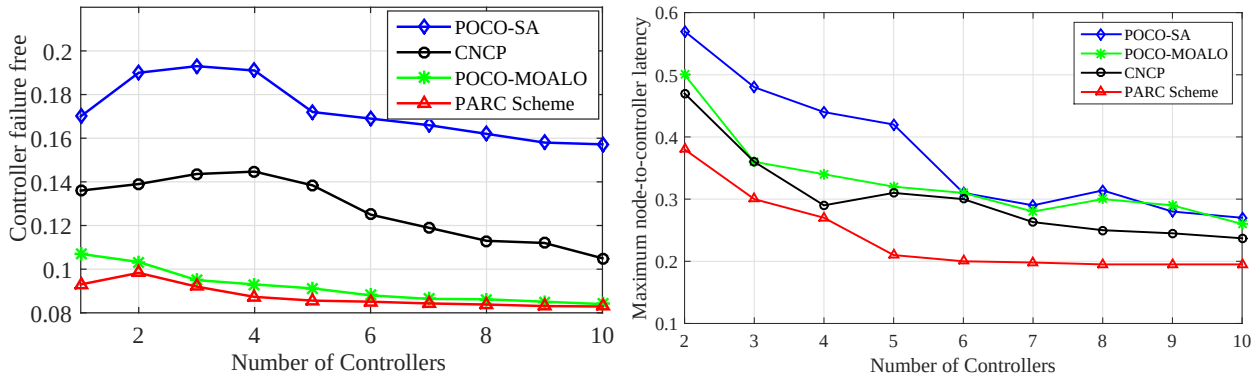

Figure 5.7: (a) Controller failure free vs number of controllers. (b) Maximum node-to-controller latency vs number of controllers.

Here, the controller-less nodes are those switches which are not linked to any controller due to controller or link failures. The graph is shown in Fig. 5.5(b) calculates the mean value for

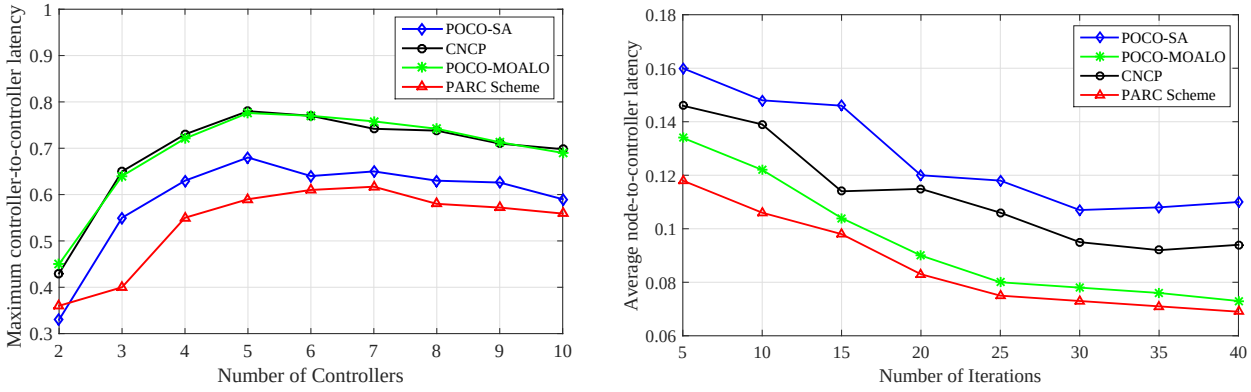

Figure 5.8: (a) Maximum controller-to-controller latency vs number of controllers. (b) Average node-to-controller latency vs the number of iterations.

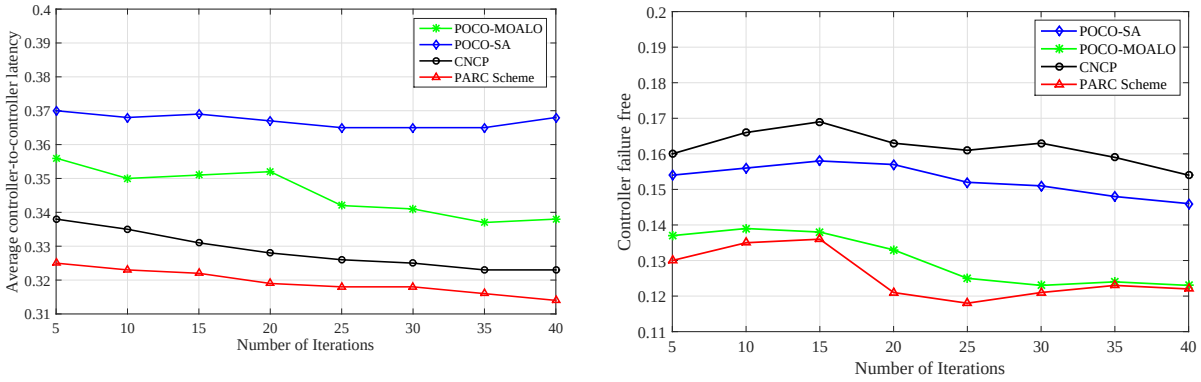

Figure 5.9: (a) Average controller-to-controller latency vs number of iterations. (b) Controller failure-free vs number of iterations.

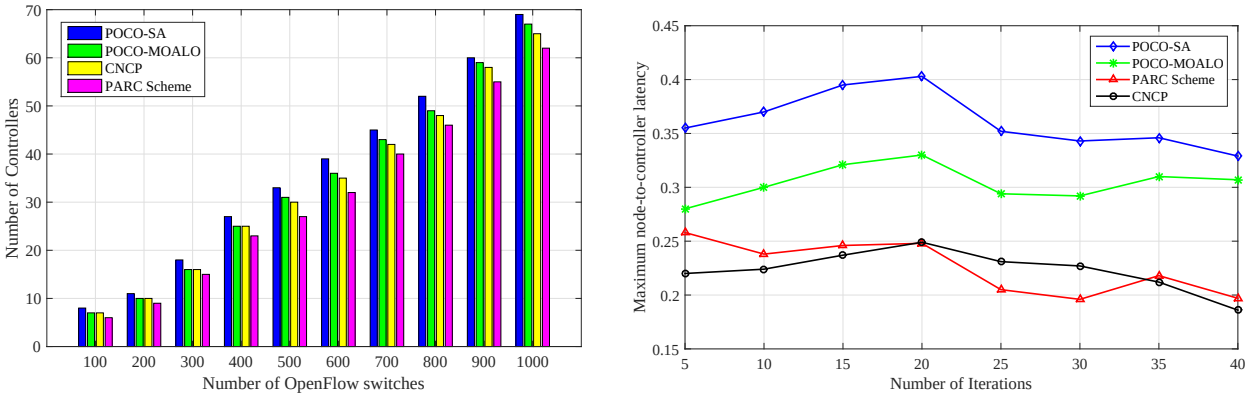

Figure 5.10: (a) Number of controllers vs the number of switches. (b) Maximum node-to-controller latency vs number of iterations.

controller failure-free versus maximum controller imbalance, i.e., upto $k-1$ controller failures. The size of the datacenters varies from small-sized to large-sized networks ranging from 34-1000 switches. In the next subsection, the PARC scheme performance in terms of the inter-controller and node-to-controller latency is discussed.

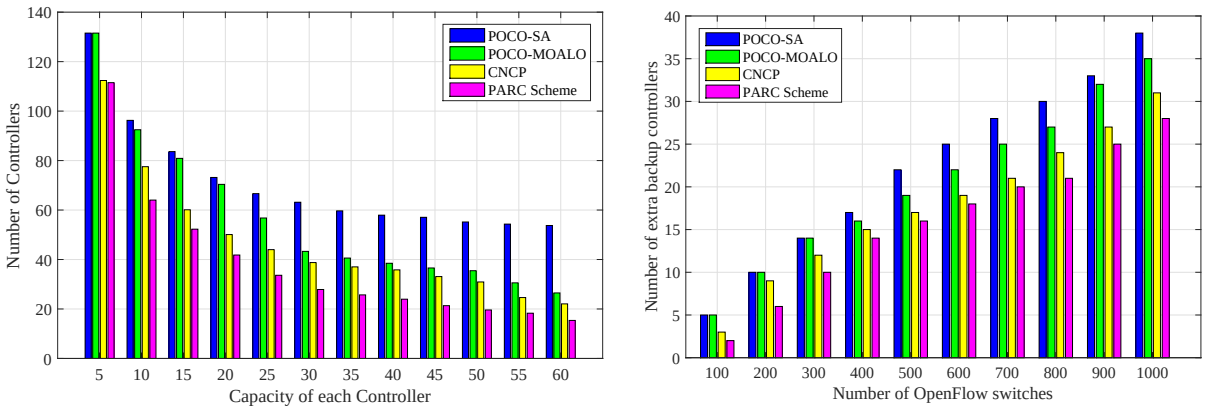

Figure 5.11: (a) Number of controllers vs controllers capacity. (b) Number of extra backup controllers with the number of OpenFlow switches.

Table 5.3: Numerical Settings.

Parameter	***Description***
System Configuration	Intel core i7 4770 with 3.20 GHz, 8 GB RAM
Simulation Platform	Matlab 2015a with POCO toolset [238]
Datasets (Topology)	Internet Topology Zoo (Internet2 OS3E [239], Planet V2 [240], Jellyfish topology [241])
Switch type	OpenFlow vSwitch version 2.5.5
Comparison Schemes	POCO-SA [113], POCO-MOALO [114], CNCP [117]
Each Switch ports	12 ports for switch interconnection
Number of nodes (V)	34 nodes, 110 nodes, 1000 nodes
Demands on switches (λ_j)	1072 flows/sec (average flow arrival rate)
Each Controller capacity (θ_i)	30 Kilo flows/sec $(0.03 \times 10^6$ flows/sec)
Packet size	160 Bytes
Maximum bandwidth	2 Gbps
Number of iterations	40
Max. Resilience level	$\{1, 2\}$ Controller failure

5.3.2 Results and Discussion

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

$$\min \frac{1}{|N|}\left(\sum_{i=1}^{|N|}\sum_{j=1}^{|N|}\sum_{k=1}^{\alpha-\beta}\frac{1}{(\overline{\alpha}-\beta)}a_{ij}R_{i,j,k}\right), \tag{5.18}$$

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

$$\min\left(\max_{i\in N}\left(a_{ij}R_{i,j,\alpha-\beta}\right)\right), \tag{5.19}$$

subject to the following constraints $\mu 1, \mu 2, \mu 3, \mu 4, \mu 7, \mu 8, \mu 10, \mu 15$. The detailed explanation is as follows.

5.3.2.1 Impact on Node-to-Controller Latency

The node-to-controller latency is the latency incurred as the number of demands increases from the multiple switches to be served by its assigned controller. Fig. 5.6(a) shows the average node-to-controller latency with respect to the number of controllers. There is an increase in latency if the number of demands between the switch-to-controller increases. The average node-to-controller latency in Fig. 5.6(a) is monotonically decreasing as the number of controllers increases. The experimental results show that the average latency between the switch-to-controller by using the PARC scheme is reduced to 51.83%, 28.21%, and 44.22% as compared to the POCO-SA, POCO-MOALO, and CNCP schemes respectively. Likewise, Fig. 5.7(b) shows the maximum node-to-controller latency with respect to the number of controllers. Initially, there is a regular decrease in the latency between the node-to-controller, but with an increase in the size of controllers, there is a very slight decrease in the latency. The latency of the PARC scheme is reduced to 36.48%, 27.60%, and 21.35% in comparison to the POCO-SA, POCO-MOALO, and CNCP schemes, respectively.

Similarly, in Figs. 5.8(b) and 5.10(b), the average and maximum latencies are computed between the switch-to-controller within a sub-network with respect to the number of iterations. It has been observed from Fig. 5.8(b) that as the number of iterations increases, there is a gradual decrease in the latency of the *PARC* scheme. Here, the number of iterations to perform the experiments are 40. The node-to-controller latency computes the Euclidean distance from the switch-to-controller in each partition. The experimental results show that the average latency between the node-to-controller of the proposed scheme is reduced to 31.85%, 8.45%, and 23.07% in comparison to the POCO-SA, POCO-MOALO, and CNCP schemes respectively. Moreover, in Fig. 5.7(b), the maximum latency between the switch-to-controller of the proposed scheme is 38.26%, 26.62%, and 1.2% which is comparatively lesser in comparison to the existing POCO-SA, POCO-MOALO, and CNCP schemes respectively.

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

5.3.2.2 Impact on Inter-Controller Latency

The inter-controller latency occurs during the communication of messages among the distributed controllers which are located in different sub-networks. Fig. 5.6(b) demonstrates the average controller-to-controller latency with respect to the number of controllers. Initially, the inter-controller latency increases as the size of the controller increases but as the number of controllers reach five; the inter-controller latency shows an improvement in the proposed scheme. The average inter-controller latency using the PARC scheme reduces to 49%, 35.73%, and 17.41% in comparison to the POCO-SA, POCO-MOALO, and CNCP schemes respectively. The reason why the PARC scheme outperforms the existing schemes is due to the variation in the controller position and assignment decisions.

Fig. 5.8(a) demonstrates that as the controller's size increases, there is an increase in latency between the pair of controllers. The results show that the maximum inter-controller latency of the PARC scheme is reduced to 10.08%, 22.71%, and 22.56% as compared to the POCO-SA, POCO-MOALO, and CNCP schemes, respectively. Also, in Fig. 5.9(a), it has been observed that the average inter-controller latency does not increase gradually with an increase in the number of iterations. The average latency of the proposed scheme is compared with the existing schemes and is reduced to 7.66%, 13%, and 2% respectively. The reason for such improvements is that the PARC scheme gives more priority to place the controllers close together as it influences the inter-controller latency. The proposed scheme implements the timestamp strategy to maintain the global synchronization between two controllers while the existing schemes incur more latency as they lack in synchronization of messages.

5.3.2.3 Impact on the Network Disruption

The network disruption occurs due to controller or link failures, which may lead to controller-less nodes in the worst case. Figs. 5.7(a) and 5.9(b) demonstrate the scenarios of failure-free controllers in which the backup controller is assigned to the switches to avoid additional delay during failure. Fig. 5.7(a) shows that as the number of controllers increases, the network is resilient and incurs minimum latency. The results show that the controller failure rate is improved to 49.34%, 4.76%, and 31.37% in comparison to the POCO-SA, POCO-MOALO, and CNCP schemes, respectively. Similarly, the performance of the failure-free controllers is analyzed with the number of iterations. Fig. 5.9(b) shows the improvement in the existing solutions in comparison to the PARC scheme by 14.72%, 3.45%, and 22.31% respectively. The reason why the PARC scheme outperforms the existing schemes is due to the mapping of switches with exactly two β backup closest controllers. The existing schemes, i.e., POCO-SA, POCO-MOALO, and CNCP schemes consider the mapping of each switch with exactly a single primary controller based on the nearest position, therefore, the issue of network disruption and delay occurs in case the single controller fails.

5.3.2.4 Impact on the least number of Controllers

Fig. 5.10(a) computes the required number of controllers to supervise all the OpenFlow switches in the network. The number of required controllers is the ratio of the processing capacity of an individual controller to the average flow arrival rate for switches. Thus, the total number of controllers required to supervise the number of OpenFlow switches is computed as follows = (30,000/1072) = 27 in the entire network. However, the required number of controllers varies as the size of the datacenter varies from 100-1000 switches, and each OpenFlow switch reserves 12 out of 24 ports for switches interconnection. Initially, the lower bound is calculated by dividing the total number of switches with the processing capacity of the controller. The results show that the cost of deploying the number of controllers in the proposed scheme is 12.98%, 8.16%, and 6.25% which is less in comparison to the POCO-SA, POCO-MOALO, and CNCP schemes respectively. Hence, the PARC scheme deployed less number of controllers irrespective of the switch size.

5.3.2.5 Impact on Controller's Capacity

Fig. 5.11(a) shows the controllers capacity vs the number of controllers deployed in the network. The processing capacity of each controller to control the number of switches in each sub-network differs from 10–60 switches according to the proposed Algorithm 8. Here, as the capacity of each controller increases, it leads to a decrease in the number of required controllers, but after exceeding a threshold limit of 50 switches, the number of required controllers does not decrease. The average capacity of the total number of controllers to control the switches of the PARC scheme is 51.5% while the POCO-SA, POCO-MOALO, and CNCP are 73.8%, 60.6%, 53.7% respectively. The reason for this behavior is that the coverage range of multiple controllers overlaps and the propagation delay mainly depends upon the coverage range of a controller, i.e., the number of hops or physical distance in the topology. The PARC scheme deploys less number of controllers with a maximum coverage range of switches as compared to the existing schemes. The maximum range is defined as the distance from the controller to the farthest switch, which is two hops.

5.3.2.6 Impact on an extra number of backup Controllers

Algorithm 9 illustrates that the number of minimal extra backup controllers required for resilience in case any primary active controller fails. Fig. 5.11(b) shows that the number of extra backup controllers increases as the number of OpenFlow switches increases. The proposed scheme reduces the number of additional controllers up to 27.92%, 21.95%, and 10.11% as compared to the POCO-SA, POCO-MOALO, and CNCP schemes respectively. It has been observed that the PARC scheme redeploys less number of controllers as a backup in comparison to the existing schemes. The reason for the better performance of the proposed scheme is that the existing schemes randomly redeploy the number of extra controllers while

the proposed scheme considers two-hop as the maximum coverage range which redeploys less number of controllers.

5.4 Summary

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

Chapter 6

Conclusion and Future Scope

Software-Defined Networking (SDN) emerges as one of the leading technologies to address the aforementioned issues using the programmable switches and controllers. The decoupling of control functionality from the forwarding devices to the control plane in SDN provides a unique platform to design a reconfigurable network. In the aforementioned challenges, the research work focused on three problems: (i) load balancing at the control plane, (ii) energy efficiency and fast flow forwarding for the data plane, and (iii) scalability and resilience at the control plane. This task has been accomplished in this research work with three different approaches.

In the first approach, a LOADS scheme is proposed to address the load imbalance and malicious flow issues in the SDN. To handle the issue of load imbalance, an optimal switch migration scheme is proposed which minimizes the overall migration cost, execution time, and response time of controllers. Moreover, an IP flow-based anomaly detection scheme is proposed to identify each user's behavior, and based on the behavior, specific policies are deployed to prevent anomalies. The proposed scheme is evaluated on the Mininet emulator using the POX controller with datasets of Internet Topology Zoo from the BTNorthAmerica zone. The performance analysis reveals that LOADS minimizes the average execution time by 6.74% and 20.64% as compared to the existing competitive schemes, Distributed Hopping Algorithm (DHA), and Elastic Distributed Controller (ElastiCon). Also, it helps in improving the overall migration cost and response time of each controller. The proposed LOADS scheme has the migration cost of 55.1 milliseconds as compared to the ElastiCon and DHA schemes alongwith the migration cost of 110 milliseconds and 140 milliseconds respectively. In addition to the migration cost, the response time of the proposed scheme is 32.8 milliseconds as compared to DHA and ElastiCon which takes almost 90 milliseconds and 78 milliseconds respectively.

However, failure of any actively distributed controllers is one of the limitations of the LOADS scheme as it may again lead to the situation of load imbalance. Hence in the next approach, we explore the issues of resilience and data consistency amongst multiple controllers at the control plane with the intent to improve performance. Resilience is one of the biggest challenges of the centralized architecture as backup controllers are required so that the for-

warding devices continue to operate smoothly in case of the failure of the primary controller. In addition, we intend to explore the Controller Placement Problem (CPP) which includes the issues like determining the optimal position for the placement of distributed controllers, identifying the number of multiple controllers and identifying the number of extra sub-ordinate controllers required as a backup in case any primary active controller fails in a large-scale network.

In the second approach, we propose an energy-efficient fast flow forwarding scheme for SDN. We modeled the EAR problem and proposed heuristic solutions comprising of three algorithms, namely–priority scheduling, flow re-routing with link adaptation, and ACR to build an energy-efficient and an optimal routing path. For evaluation of the proposed scheme, we consider a fat-tree 4-ary network topology simulated on OMNET++ using the ONOS controller. The experimental results demonstrated that the proposed *EnFlow* scheme achieves an average energy consumption savings up to 88.15% while, the *RE-FPR* and *ILP-EAR* schemes achieve energy-efficiency up to 63.6% and 17%. The simulation results show that the proposed scheme utilizes the link bandwidth up to 17.7% and 37.3% more as compared to *FFHA* and *EXR* schemes. Thus, the proposed *EnFlow* scheme improves the energy-efficiency of the network in the heavy traffic load.

The limitations of the EnFlow scheme is that the considered priority scheduling algorithm relies on a single network controller for flow scheduling. In case the network controller fails, the flow scheduling strategies of the EnFlow scheme suffer from inefficient routing and high power consumption issues on the forwarding plane. Besides, for a large-scale DC with 1K-100K servers, the issue of scalability arises. This issue is addressed by using multiple distributed controllers on the control plane but there are other factors like load balancing, fault-tolerance, routing along with energy-efficiency that play a significant role with an increase in the network size. The problem size of scalability depends on the length of the routing path as a large-scale DC has a much longer flow path that increases the size of the search space. Hence in the next approach, we explore the fault-tolerance issue by considering distributed controllers at the control plane. Furthermore, to improve the scalability of routing, we will consider local optimizers on multiple controllers that synchronizes the global knowledge of all the switches periodically. In the third approach, we propose the PARC scheme to address the issues of multiple CPP, with a view that the data plane continues to operate without any failure. The PARC scheme divides the CPP into four sub-problems, i.e., network partitioning, the position of controllers, the minimal number of controllers, and the minimal number of extra backup controllers for resilience in each sub-network. The simulation is performed on the Internet topology zoo on the Matlab platform. The numerical results are tested on Internet2 OS3E topology using POCO-toolset, simulated on the Matlab framework. The experimental results demonstrate that the proposed scheme reduces the controller deployment cost to 12.98%, 8.16%, and 6.25% as compared to the POCO-SA, POCO-MOALO, and CNCP schemes, respectively. In addition, the PARC scheme outperforms the existing state-of-the-art schemes such as- POCO-SA, POCO-MOALO,

and CNCP in terms of inter-controller and switch-to-controller latency.

In the future, we will explore the resilience of the POCO scheme by increasing the resilience level upto three or more controllers' failure to analyze the control plane reliability. Furthermore, we will explore the issues related to security threats and resource allocation using virtualization in distributed SDN controllers for a multi-cloud environment.

Bibliography

[1] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. ur Rasool, and W. Dou, “Complementing IoT Services through Software Defined Networking and Edge Computing: A Comprehensive Survey,” *IEEE Communications Surveys & Tutorials*, May 2020, DOI: 10.1109/COMST.2020.2997475.

[2] Li, Zhong, Cheng Wang, and Chang-Jun Jiang, "User Association for Load Balancing in Vehicular Networks: An Online Reinforcement Learning Approach," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18 , no. 8, pp. 2217-2228, Aug. 2017.

[3] Kreutz, Diego and Ramos, Fernando MV and Verissimo, Paulo and Rothenberg, Christian Esteve and Azodolmolky, Siamak and Uhlig, Steve, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14-76, Jan. 2015.

[4] Aujla, Gagangeet Singh and Chaudhary, Rajat and Kumar, Neeraj and Rodrigues, Joel JPC and Vinel, Alexey, "Data Offloading in 5G-enabled Software-Defined Vehicular Networks: A Stackelberg-Game-based Approach," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 100-108, Aug. 2017.

[5] Liyanage, Madhusanka and Abro, Ahmed Bux and Ylianttila, Mika and Gurtov, Andrei, "Opportunities and Challenges of Software-Defined Mobile Networks in Network Security," *IEEE Security & Privacy*, vol. 14, no. 4, pp. 34-44, Aug. 2016.

[6] Chaudhary, Rajat and Aujla, Gagangeet Singh and Kumar, Neeraj and Rodrigues, Joel JPC, "Optimized Big Data Management across Multi-Cloud Data Centers: Software-Defined-Network-based Analysis," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 118-126, Feb. 2018.

[7] Networking, Cisco Visual, "Cisco Global Cloud Index: Forecast and Methodology, 2016–2021," *White paper. Cisco Public*, San Jose, USA, 2016.

[8] A. Al-Darrab, I. Al-Darrab, and A. Rushdi, “Software-Defined Networking Load Distribution Technique for an Internet Service Provider,” *Journal of Network and Computer Applications*, vol. 155, pp. 102547, Apr. 2020.

[9] Aujla, Gagangeet Singh and Chaudhary, Rajat and Kaur, Kuljeet and Garg, Sahil and Kumar, Neeraj and Ranjan, Rajiv, "SAFE: SDN-assisted Framework for Edge–Cloud Interplay in Secure Healthcare Ecosystem," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 1, pp. 469-480, Jan. 2019.

[10] Cheng, Guozhen and Chen, Hongchang and Hu, Hongchao and Lan, Julong, "Dynamic Switch Migration towards a Scalable SDN Control Plane," *International Journal of Communication Systems*, vol. 29, no. 9, pp. 1482-1499, Feb. 2016.

[11] Z. Latif, K. Sharif, F. Li, M. M. Karim, S. Biswas, and Y. Wang, "A Comprehensive Survey of Interface Protocols for Software Defined Networks," *Journal of Network and Computer Applications*, vol. 156, pp. 102563, Apr. 2020.

[12] Kuo, Kuan-Tsen and Wen, Charles H-P and Suo, Cheng and Tsai, I-Chen, "SWF: Segmented Wildcard Forwarding for Flow Migration in OpenFlow Datacenter Networks," *IEEE International Conference on Communications (ICC)*, London, UK, DOI: 10.1109/ICC.2015.7248340, pp. 313-318, Jun. 2015.

[13] Zhang, Shaojun and Lan, Julong and Sun, Penghao and Jiang, Yiming, "Online Load Balancing for Distributed Control Plane in Software-defined Data Center Network," *IEEE Access*, vol. 6, pp. 18184-18191, Mar. 2018.

[14] Selvi, Hakan and Gür, Gürkan and Alagöz, Fatih, "Cooperative Load Balancing for Hierarchical SDN Controllers," *IEEE 17th International Conference on High Performance Switching and Routing (HPSR)*, Yokohama, Japan, pp. 100-105, Jun. 2016.

[15] Amaral, Alexandre Aguiar and de Souza Mendes, Leonardo and Zarpelão, Bruno Bogaz and Junior, Mario Lemes Proença, "Deep IP Flow Inspection to Detect beyond Network Anomalies," *Computer Communications*, vol. 98, pp. 80-96, Jan. 2017.

[16] "Cisco 2018 Annual Cybersecurity Report," (2018). [Online] Available: https://www.cisco.com/c/dam/m/hu_hu/campaigns/security-hub/pdf/acr-2018.pdf) (Accessed on Nov. 2018).

[17] Tuysuz, Mehmet Fatih and Ankarali, Zekiye Kubra and Gözüpek, Didem, "A Survey on Energy Efficiency in Software Defined Networks," *Computer Networks*, vol. 113, pp. 188-204, Feb. 2017.

[18] J. Singh, A. Gujral, H. Singh, J. U. Singh and N. Auluck, "Energy Aware Scheduling on Heterogeneous Multiprocessors with DVFS and Duplication," *17th International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, Guangzhou, pp. 105-112, 2016, doi: 10.1109/PDCAT.2016.036.

[19] Mangu, A., "Managing Energy Consumption of Data Centers," (2018). [Online] Available: (http://large.stanford.edu/courses/2018/ph240/mangu2/), (Accessed on Jan. 2019).

[20] Witham, J., "Achieving Data Center Energy Efficiency," (2018). [Online] Available: (https://www.datacenterknowledge.com/industryperspectives/achieving-data-center-energy-efficiency), (Accessed on Jan. 2019).

[21] Garg, Sahil and Kaur, Kuljeet and Ahmed, Syed Hassan and Bradai, Abbas and Kaddoum, Georges and Atiquzzaman, Mohammed, "MobQoS: Mobility-Aware and QoS-Driven SDN Framework for Autonomous Vehicles," *IEEE Wireless Communications*, vol. 26, no. 4, pp. 12-20, Aug. 2019.

[22] Ender Ayanoglu, "Energy Efficiency in Data Centers," (2019). [Online] Available: (https://www.comsoc.org/publications/tcn/2019-nov/energy-efficiency-data-centers), (Accessed on Apr. 2020).

[23] Kaur, Kuljeet and Garg, Sahil and Kaddoum, Georges and Ahmed, Syed Hassan and Jayakody, Dushantha Nalin K, "En-OsCo: Energy-aware Osmotic Computing Framework using Hyper-Heuristics," *Proceedings of the ACM MobiHoc Workshop on Pervasive Systems in the IoT Era*, Catania, Italy, [Online] Available: (https://doi.org/10.1145/3331052.3332473), pp. 19-24, Jul. 2019.

[24] Duan, Hancong, Chao Chen, Geyong Min, and Yu Wu. "Energy-aware scheduling of virtual machines in heterogeneous cloud computing systems," *Future Generation Computer Systems*, vol. 74, pp. 142-150, 2017.

[25] Kaur, Kuljeet and Garg, Sahil and Kaddoum, Georges and Ahmed, Syed Hassan and Atiquzzaman, Mohammed, "KEIDS: Kubernetes based Energy and Interference Driven Scheduler for Industrial IoT in Edge-Cloud Ecosystem," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4228-4237, May. 2020.

[26] Li, Dan and Shang, Yunfei and Chen, Congjie, "Software defined Green Data Center Network with Exclusive Routing," *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, Toronto, Ontario, Canada, pp. 1743-1751, Jul. 2014.

[27] Goyal, Nitin, Mayank Dave, and Anil Kumar Verma, "Energy Efficient Architecture for Intra and Inter Cluster Communication for Underwater Wireless Sensor Networks" *Wireless Personal Communications*, vol. 89, no. 2 pp. 687-707, 2016.

[28] Aujla, Gagangeet Singh and Kumar, Neeraj, "MEnSuS: An Efficient Scheme for Energy Management with Sustainability of Cloud Data Centers in Edge–Cloud Environment," *Future Generation Computer Systems*, vol. 86, pp. 1279-1300, Sep. 2018.

[29] Jindal, Anish and Aujla, Gagangeet Singh and Kumar, Neeraj and Chaudhary, Rajat and Obaidat, Mohammad S and You, Ilsun, "SeDaTiVe: SDN-enabled Deep Learning Architecture for Network Traffic Control in Vehicular Cyber-Physical Systems," *IEEE Network*, vol. 32, no. 6, pp. 66-73, Nov. 2018.

[30] Chaudhary, Rajat and Jindal, Anish and Aujla, Gagangeet Singh and Aggarwal, Shubhani and Kumar, Neeraj and Choo, Kim-Kwang Raymond, "BEST: Blockchain-based secure energy trading in SDN-enabled intelligent transportation system," *Computers & Security*, vol. 85, pp. 288-299, Aug. 2019.

[31] Zhu, Chao and Xiao, Yu and Cui, Yong and Yang, Zhenjie and Xiao, ShiHan and Ylä-Jääski, Antti, "Dynamic Flow Consolidation for Energy Savings in Green DCNs," *IEEE 34th International Performance Computing and Communications Conference (IPCCC)*, Nanjing, China, DOI: 10.1109/PCCC.2015.7410277, Feb. 2015.

[32] "Cisco Nexus 9500 Platform Switches," (2018) [Online] Available: (https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/datasheet-c78-729404.pdf), (Accessed on Feb. 2019).

[33] Aujla, Gagangeet Singh and Chaudhary, Rajat and Kumar, Neeraj and Das, Ashok Kumar and Rodrigues, Joel JPC, "SecSVA: Secure Storage, Verification, and Auditing of Big Data in the Cloud Environment," *IEEE Communications Magazine*, vol. 56, no. 1, pp. 78-85, Jan. 2018.

[34] Yu, Rong and Ding, Jiefei and Huang, Xumin and Zhou, Ming-Tuo and Gjessing, Stein and Zhang, Yan, "Optimal Resource Sharing in 5G-Enabled Vehicular Networks: A Matrix Game Approach," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 10, pp. 7844-7856, Oct. 2016.

[35] Wang, Weiyang and Dong, Mianxiong and Ota, Kaoru and Wu, Jun and Li, Jianhua and Li, Gaolei, "CDLB: A Cross-Domain Load Balancing Mechanism for Software Defined Networks in Cloud Data Centre," *International Journal of Computational Science and Engineering*, vol. 18, no. 1, pp. 44-53, Dec. 2019.

[36] Chaudhary, Rajat and Kumar, Neeraj and Zeadally, Sherali, "Network Service Chaining in Fog and Cloud Computing for the 5G Environment: Data Management and Security Challenges," *IEEE Communications Magazine*, vol. 55, no. 11, pp. 114-122, Nov. 2017.

[37] "Software Defined Data Center (SDDC) Market," (2016) [Online] Available: https://www.alliedmarketresearch.com/software-defined-data-center-market) (Accessed on Jun. 2019)

[38] Aujla, Gagangeet Singh and Chaudhary, Rajat and Kumar, Neeraj and Kumar, Ravinder and Rodrigues, Joel JPC, "An Ensembled Scheme for QoS-aware Traffic Flow Management in Software Defined Networks," *IEEE International Conference on Communications (ICC)*, Kansas City, MO, USA, DOI: 10.1109/ICC.2018.8422596, May. 2018.

[39] Li, He and Ota, Kaoru and Dong, Mianxiong and Guo, Minyi, "Mobile Crowdsensing in Software Defined Opportunistic Networks," *IEEE Communications Magazine*, vol. 55, no. 6, pp. 140-145, Jun. 2017.

[40] Zhang, Yuan and Cui, Lin and Wang, Wei and Zhang, Yuxiang, "A Survey on Software Defined Networking with Multiple Controllers," *Journal of Network and Computer Applications*, vol. 103, pp. 101-118, Feb. 2018.

[41] Singh, Rajwinder, and Mayank Dave, "Antecedence Graph Approach to Checkpointing for Fault Tolerance in Mobile Agent Systems" *IEEE Transactions on Computers*, vol. 62, no. 2, pp. 247-258, Feb. 2013.

[42] Heller, Brandon and Sherwood, Rob and McKeown, Nick, "The Controller Placement Problem," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, Aug. 2012.

[43] Chaudhary, Rajat and Aujla, Gagangeet Singh and Garg, Sahil and Kumar, Neeraj and Rodrigues, Joel JPC, "SDN-enabled Multi-Attribute-based Secure Communication for Smart Grid in IIoT Environment," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2629-2640, Jun. 2018.

[44] Oktian, Yustus Eko and Lee, SangGon and Lee, HoonJae and Lam, JunHuy, "Distributed SDN Controller System: A Survey on Design Choice," *Computer Networks*, vol. 121, pp. 100-111, Jul. 2017.

[45] Venkatesan S., Chellappan C. and Dhavachelvan P.(2010), "Performance analysis of mobile agent failure recovery in E-Service applications," *International Journal on Computer Standard and Interfaces*, Elsevier, vol. 32, no. 1-2, pp. 38-43, Jan. 2010.

[46] Kaur, Kuljeet and Garg, Sahil and Aujla, Gagangeet Singh and Kumar, Neeraj and Rodrigues, Joel JPC and Guizani, Mohsen, "Edge Computing in the Industrial Internet of Things Environment: Software-defined-Networks-based Edge-Cloud Interplay," *IEEE Communications Magazine*, vol. 56, no. 2, pp. 44-51, Feb. 2018.

[47] Alam, Kazi Masudul, Mukesh Saini, and Abdulmotaleb El Saddik. "Toward Social Internet of Vehicles: Concept, Architecture, and Applications," *IEEE Access*, vol. 3, pp. 343-357, 2015, doi: 10.1109/ACCESS.2015.2416657.

[48] Ejaz, Waleed, Muhammad Naeem, Adnan Shahid, Alagan Anpalagan, and Minho Jo. "Efficient energy management for the internet of things in smart cities." *IEEE Communications Magazine*, vol. 55, no. 1, pp. 84-91, 2017.

[49] Networking, Cisco Visual "Cisco Global Cloud Index: Forecast and Methodology, 2015-2020. White Paper," (2016) [Online] Available: (http://www.cisco.com/c/dam/en/us/solutions/collateral/ service-provider/global-cloud-index-gci/white-paper-c11-738085.pdf) (Accessed on: March 2017).

[50] Whitney, Josh and Delforge, Pierre, "Data Center Efficiency Assessment–Scaling up Energy Efficiency across the Data Center Industry: Evaluating Key Drivers and Barriers," *NRDC and Anthesis, Rep. IP:14-08-A*, Aug. 2014.

[51] Nunes, Bruno Astuto A and Mendonca, Marc and Nguyen, Xuan-Nam and Obraczka, Katia and Turletti, Thierry, "A Survey of Software-defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1617-1634, Feb. 2014.

[52] F. Hu, Q. Hao, and K. Bao, "A Survey on Software-Defined Network and Openflow: From Concept to Implementation," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 4, pp. 2181–2206, May 2014.

[53] A. Lara, A. Kolasani, and B. Ramamurthy, "Network Innovation using Openflow: A Survey," *IEEE communications surveys & tutorials*, vol. 16, no. 1, pp. 493–512, First Quarter, 2014.

[54] I. F. Akyildiz, A. Lee, P. Wang, M. Luo, and W. Chou, "A Roadmap for Traffic Engineering in SDN-Openflow Networks," *Computer Networks*, vol. 71, pp. 1–30, Oct. 2014.

[55] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[56] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 27–51, Firstquarter 2015.

[57] 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, Firstquarter 2015.

[58] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network Function Virtualization: State-of-the-Art and Research Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 236–262, Firstquarter 2016.

[59] C. Trois, M. D. Del Fabro, L. C. de Bona, and M. Martinello, "A Survey on SDN Programming Languages: Toward a Taxonomy," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 4, pp. 2687–2712, Fourthquarter 2016.

[60] X.-N. Nguyen, D. Saucez, C. Barakat, and T. Turletti, "Rules Placement Problem in Openflow Networks: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 2, pp. 1273–1286, Secondquarter 2016.

[61] Q. Yan, F. R. Yu, Q. Gong, and J. Li, "Software-Defined Networking (SDN) and Distributed Denial of Service (DDoS) Attacks in Cloud Computing Environments: A Survey, Some Research Issues, and Challenges," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 602–622, Firstquarter 2016.

[62] S. Scott-Hayward, S. Natarajan, and S. Sezer, "A Survey of Security in Software Defined Networks," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 623–654, Firstquarter 2016.

[63] R. Alvizu, G. Maier, N. Kukreja, A. Pattavina, R. Morro, A. Capello, and C. Cavazzoni, "Comprehensive Survey on T-SDN: Software-defined Networking for Transport Networks," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2232–2283, Fourthquarter 2017.

[64] T. Huang, F. R. Yu, C. Zhang, J. Liu, J. Zhang, and Y. Liu, "A Survey on Large-Scale Software Defined Networking (SDN) Testbeds: Approaches and Challenges," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 2, pp. 891–917, Secondquarter 2017.

[65] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can Edge Computing Benefit from Software-Defined Networking: A Survey, use Cases, and Future Directions," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2359–2391, Fourthquarter 2017.

[66] D. B. Rawat and S. R. Reddy, "Software Defined Networking Architecture, Security and Energy Efficiency: A Survey," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 325 - 346, Oct. 2016.

[67] S. Khan, A. Gani, A. W. A. Wahab, M. Guizani, and M. K. Khan, "Topology Discovery in Software Defined Networks: Threats, Taxonomy, and State-of-the-Art," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 1, pp. 303–324, Firstquarter 2017.

[68] J. W. Guck, A. Van Bemten, M. Reisslein, and W. Kellerer, "Unicast QoS Routing Algorithms for SDN: A Comprehensive Survey and Performance Evaluation," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 388–415, Firstquarter 2018.

[69] H. Wang, Y. Li, Y. Zhang, and D. Jin, "Virtual Machine Migration Planning in Software Defined Networks," *IEEE Transactions on Cloud Computing*, vol. 7, no. 4, pp. 1168-1182, Oct. 2019.

[70] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. ur Rasool, and W. Dou, "Complementing IoT Services through Software Defined Networking and Edge Computing: A Comprehensive Survey," *IEEE Communications Surveys & Tutorials*, May 2020, DOI: 10.1109/COMST.2020.2997475.

[71] J. M. Smith, D. J. Farber, C. A. Gunter, S. M. Nettles, D. Feldmeier, and W. D. Sincoskie, "Switchware: Accelerating Network Evolution (White Paper)," 1996.

[72] 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, 1998.

[73] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click Modular Router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263–297, 2000.

[74] M. Handley, O. Hodson, and E. Kohler, "Xorp: An open platform for network research," *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 1, pp. 53–57, 2003.

[75] "Quagga routing software suite." [Online]. Available: (http://www. nongnu.org/quagga/) (Accessed on: Nov. 2019).

[76] "The bird internet routing daemon." [Online]. Available: (http: //bird.network.cz/) (Accessed on: Jan. 2020).

[77] J. E. Van der Merwe, S. Rooney, L. Leslie, and S. Crosby, "The Tempesta Practical Framework for Network Programmability," *IEEE Network*, vol. 12, no. 3, pp. 20–28, May. 1998.

[78] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe, "The Case for Separating Routing from Routers," in *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture, ser. FDNA '04*, New York, NY, USA: ACM, 2004, pp. 5–12. [Online]. Available: (http://doi.acm.org/10.1145/1016707.1016709).

[79] J. Rexford, A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, G. Xie, J. Zhan, and H. Zhang, "Network-Wide Decision Making: Toward a Wafer-Thin Control Plane," *Proc. HotNets*, Nov 2004, pp. 59–64.

[80] 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," *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, pp. 41–54, Oct. 2005.

[81] X. Chen, Y. Zhong, and A. Jukan, "Multipath routing in path computation element (pce): Protocol extensions and implementation," *Proceedings of the 2013 18th European Conference on Network and Optical Communications & 2013 8th Conference on Optical Cabling and Infrastructure (NOC-OC&I)*, DOI: 10.1109/NOC-OCI.2013.6582871, Jul. 2013.

[82] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking Control of the Enterprise," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4, 2007, pp. 1–12, Aug. 2007.

[83] H. Wang, A. Srivastava, L. Xu, S. Hong, and G. Gu, "Bring your own Controller: Enabling Tenant-Defined SDN Apps in IaaS Clouds," *IEEE INFOCOM 2017- IEEE Conference on Computer Communications*, Atlanta, GA, USA, DOI: 10.1109/INFOCOM.2017.8057137, May. 2017.

[84] B. Yu, Y. Han, X. Wen, X. Chen, and Z. Xu, "An Energy-aware Algorithm for Optimizing Resource Allocation in Software Defined Network," *IEEE Global Communications Conference (GLOBECOM)*, Washington, DC, USA, DOI: 10.1109/GLOCOM.2016.7841589, Dec. 2016.

[85] D. Zeng, G. Yang, L. Gu, S. Guo, and H. Yao, "Joint optimization on switch activation and flow routing towards energy efficient software defined data center networks," *IEEE International Conference on Communications (ICC)*, Kuala Lumpur, Malaysia, DOI: 10.1109/ICC.2016.7511463, May. 2016.

[86] J. Son, A. V. Dastjerdi, R. N. Calheiros, and R. Buyya, "Sla-aware and Energy-Efficient Dynamic Overbooking in SDN-based Cloud Data Centers," *IEEE Transactions on Sustainable Computing*, vol. 2, no. 2, pp. 76–89, 2017.

[87] A. Amokrane, R. Langar, R. Boutaba, and G. Pujolle, "Flow-based management for energy efficient campus networks," *IEEE Transactions on Network and Service Management*, vol. 12, no. 4, pp. 565–579, Dec. 2015.

[88] L. Zhang, S. Guo, Y. Yang, D. Liu, and R. Liu, "RE-FPR: Flow Preemption Routing Scheme with Redundancy Elimination in Software Defined Data Center Networks," *Sustainable Computing: Informatics and Systems*, vol. 18, pp. 14–24, Jun. 2018.

[89] A. Fernandez-Fernandez, C. Cervello-Pastor, and L. Ochoa-Aday, "Achieving Energy Efficiency: An Energy-Aware Approach in SDN," *IEEE Global Communications Conference (GLOBECOM)*, Washington, DC, USA, DOI: 10.1109/GLOCOM.2016.7841561, Dec. 2016.

[90] F. Francois, N. Wang, K. Moessner, and S. Georgoulas, "Optimizing Link Sleeping Reconfigurations in ISP Networks with Off-Peak Time Failure Protection," *IEEE Transactions on Network and Service Management*, vol. 10, no. 2, Jun. 2013.

[91] B. Heller et al., "Elastictree: Saving energy in data center networks," *Proc. USENIX NSDI*, vol. 10, pp. 249–264, Apr. 2010.

[92] X. Wang, X. Wang, K. Zheng, Y. Yao, and Q. Cao, "Correlation-Aware Traffic Consolidation for Power Optimization of Data Center Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 4, pp. 992-1006, Apr. 2016.

[93] M. Javed, G. Ahmed, D. Mahmood, M. Raza, K. Ali, and M. Ur-Rehman, "TAEO—A thermal aware & energy optimized routing protocol for wireless body area networks," *Sensors*, vol. 19, no. 15, pp. 3275, Jul. 2019.

[94] K. M. Awan et al., "A Priority-based Congestion-Avoidance Routing Protocol using IOT-based Heterogeneous Medical Sensors for Energy Efficiency in Healthcare Wireless Body Area Networks," *International Journal of Distributed Sensor Networks*, vol. 15, no. 6, Jun. 2019.

[95] Bolla, R., Bruschi, R., Davoli, F., & Lombardo, C., "Fine-Grained Energy-Efficient Consolidation in SDN Networks and Devices," *IEEE Transactions on Network and Service Management*, vol. 12, no. 2, pp. 132-145, Jun. 2015.

[96] Lian, Jie, Kshirasagar Naik, and Gordon B. Agnew. "Data Capacity Improvement of Wireless Sensor Networks using Non-Uniform Sensor Distribution." *International Journal of Distributed Sensor Networks*, vol. 2, no. 2, pp. 121-145, 2006.

[97] R. Maaloul, R. Taktak, L. Chaari, and B. Cousin, "Energy-Aware Routing in Carrier-Grade Ethernet Using SDN Approach," *IEEE Transactions on Green Communications and Networking*, vol. 2, no. 3, pp. 844–858, Sept. 2018.

[98] K. Kannan and S. Banerjee, "Compact TCAM: Flow Entry Compaction in TCAM for Power Aware SDN," *Springer International Conference on Distributed Computing and Networking*, Berlin, Germany, pp. 439–444, Jan. 2013.

[99] F. Giroire, J. Moulierac, and T. K. Phan, "Optimizing Rule Placement in Software-Defined Networks for Energy-aware Routing," *IEEE Global Communications Conference (GLOBECOM)*, Austin, TX, USA, pp. 2523–2529, Dec. 2014.

[100] G. Huang and H. Y. Youn, "Proactive Eviction of Flow Entry for SDN based on Hidden Markov Model," *Frontiers of Computer Science*, vol. 14, no. 4, pp. 1–10, Aug. 2020.

[101] F. Dabaghi, Z. Movahedi, and R. Langar, "A Survey on Green Routing Protocols using Sleep-Scheduling in Wired Networks," *Journal of Network and Computer Applications*, vol. 77, pp. 106–122, Jan. 2017.

[102] M. Kurdi, "Ant Colony System with a Novel Non-Daemon Actions Procedure for Multiprocessor Task Scheduling in Multistage Hybrid Flow Shop," *Swarm and Evolutionary Computation*, vol. 44, pp. 987-1002, Feb. 2019.

[103] W. Miao, G. Min, Y. Wu, and H. Wang, "Performance Modelling of Preemption-Based Packet Scheduling for Data Plane in Software Defined Networks," *IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity)*, Chengdu, China, DOI: 10.1109/SmartCity.2015.48, pp. 60–65, Dec. 2015.

[104] K. Kaur, S. Garg, G. Kaddoum, N. Kumar, and F. Gagnon, "SDN-Based Internet of Autonomous Vehicles: An Energy-Efficient Approach for Controller Placement," *IEEE Wireless Communications*, vol. 26, no. 6, pp. 72–79, Dec. 2019.

[105] K. Zheng, X. Wang, and J. Liu, "Distributed Traffic Flow Consolidation for Power Efficiency of Large-scale Data Center Network," *IEEE Transactions on Cloud Computing* (Early Access), DOI: 10.1109/TCC.2020.2970403, Jan. 30, 2020.

[106] K. Kaur, S. Garg, G. Kaddoum, E. Bou-Harb, and K.-K.-R. Choo, "A Big Data-Enabled Consolidated Framework for Energy Efficient Software Defined Data Centers in IoT Setups," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 4, pp. 2687–2697, Apr. 2020.

[107] A. M. Abdelmoniem, B. Bensaou, and A. J. Abu, "Sicc: Sdn-based Incast Congestion Control for Data Centers," *IEEE International Conference on Communications (ICC)*, Paris, France, DOI: 10.1109/ICC.2017.7996826, May. 2017.

[108] J. Liu, J. Liu, and R. Xie, "Reliability-based Controller Placement Algorithm in Software Defined Networking," *Computer Science and Information Systems*, vol. 13, no. 2, pp. 547-560, 2016.

[109] H. Li, P. Li, S. Guo, and A. Nayak, "Byzantine-Resilient Secure Software-Defined Networks with Multiple Controllers in Cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 4, pp. 436–447, Oct. 2014.

[110] S. A. Astaneh and S. S. Heydari, "Optimization of SDN Flow Operations in Multi-Failure Restoration Scenarios," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 421–432, Sept. 2016

[111] J. Xie, D. Guo, X. Zhu, B. Ren, and H. Chen, "Minimal Fault-tolerant Coverage of Controllers in IaaS Datacenters," *IEEE Transactions on Services Computing*, (Early Access), DOI: 10.1109/TSC.2017.2753260, Sep. 2017.

[112] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. TranGia, "Pareto-optimal resilient controller placement in SDN-based core networks," *IEEE Proceedings of the 2013 25th International Teletraffic Congress (ITC)*, Shanghai, China, DOI: 10.1109/ITC.2013.6662939, Sep. 2013.

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

[114] M. Ramasamy and S. Pawar, "Pareto-Optimal Multi-Controller Placement in Software Defined Network," *2018 3rd International Conference for Convergence in Technology (I2CT)*, Pune, India, DOI: 10.1109/I2CT.2018.8529822, Apr. 2018.

[115] A. Ksentini, M. Bagaa, T. Taleb, and I. Balasingham, "On Using Bargaining Game for Optimal Placement of SDN Controllers," *IEEE International Conference on Communications (ICC)*, Kuala Lumpur, Malaysia, DOI: 10.1109/ICC.2016.7511136, May 2016.

[116] 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," *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, Sydney, NSW, Australia, DOI: 10.1109/WoWMoM.2014.6918987, Jun. 2014.

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

[118] M. Tanha, D. Sajjadi, and J. Pan, "Enduring Node Failures through Resilient Controller Placement for Software Defined Networks," *IEEE Global Communications Conference (GLOBECOM)*, Washington, DC, DOI: 10.1109/GLOCOM.2016.7841786, Dec. 2016.

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

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

[121] A. Alshamrani, S. Guha, S. Pisharody, A. Chowdhary, and D. Huang, "Fault Tolerant Controller Placement in Distributed SDN Environments," *IEEE International Conference on Communications (ICC)*, Kansas City, MO, DOI: 10.1109/ICC.2018.8422593, May 2018.

[122] N. E. Petroulakis, G. Spanoudakis, and I. G. Askoxylakis, "Fault Tolerance Using an SDN Pattern Framework," *IEEE Global Communications Conference (GLOBECOM)*, Singapore, DOI: 10.1109/GLOCOM.2017.8254082, Dec. 2017.

[123] P. M. Mohan, T. Truong-Huu, and M. Gurusamy, "Primary-Backup Controller Mapping for Byzantine Fault Tolerance in Software Defined Networks," *IEEE Global Communications Conference (GLOBECOM)*, Singapore, DOI: 10.1109/GLOCOM.2017.8254755, Dec. 2017.

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

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

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

[127] H. Amarasinghe, A. Jarray, and A. Karmouch, "Fault-tolerant IaaS Management for Networked Cloud Infrastructure with SDN," *IEEE International Conference on Communications (ICC)*, Paris, France, DOI: 10.1109/ICC.2017.7996342, May 2017.

[128] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, and R. Buyya, "CloudSimSDN: Modeling and Simulation of Software-Defined Cloud Data Centers," *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, Shenzhen, China, DOI: 10.1109/CCGrid.2015.87, May 2015.

[129] F. Li, J. Cao, X. Wang, and Y. Sun, "A QoS Guaranteed Technique for Cloud Applications Based on Software Defined Networking," IEEE Access, vol. 5, pp. 21229-21241, Sep. 2017.

[130] F. R. De Souza, C. C. Miers, A. Fiorese, and G. P. Koslovski, "QoS-Aware Virtual Infrastructures Allocation on SDN-Based Clouds," *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*, Madrid, Spain, DOI: 10.1109/CCGRID.2017.57, May 2017.

[131] H. Yuan, J. Bi, J. Zhang, W. Tan, and K. Huang, "Workload-Aware Revenue Maximization in SDN-Enabled Data Center," *IEEE 10th International Conference on Cloud Computing (CLOUD)*, Honolulu, CA, USA, DOI: 10.1109/CLOUD.2017.12, Jun. 2017. pp. 18–25.

[132] W. Li, H. Qi, K. Li, I. Stojmenovic, and J. Lan, “Joint Optimization of Bandwidth for Provider and Delay for User in Software Defined Data Centers,” *IEEE Transactions on Cloud Computing*, vol. 5, no. 2, pp. 331–343, Feb. 2017.

[133] M. Amiri, A. Sobhani, H. Al Osman, and S. Shirmohammadi, “SDN-Enabled Game-Aware Routing for Cloud Gaming Datacenter Network,” *IEEE Access*, vol. 5, pp. 18633-18645, Sept. 2017.

[134] C.-H. Chiu, D. K. Singh, Q. Wang, K. Lee, and S.-J. Park, “Minimal Coflow Routing and Scheduling in OpenFlow-Based Cloud Storage Area Networks,” *IEEE 10th International Conference on Cloud Computing (CLOUD)*, Honolulu, CA, USA, DOI: 10.1109/CLOUD.2017.36, Jun. 2017.

[135] M. Caria, A. Jukan, and M. Hoffmann, “SDN Partitioning: A Centralized Control Plane for Distributed Routing Protocols,” *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 381–393, Jun. 2016.

[136] Y. Wu, Z. Zhang, C. Wu, C. Guo, Z. Li, and F. C. Lau, “Orchestrating Bulk Data Transfers across Geo-Distributed Datacenters,” *IEEE Transactions on Cloud Computing*, vol. 5, no. 1, pp. 112–125, Mar. 2017.

[137] D. Li, Y. Shang, W. He, and C. Chen, “EXR: Greening Data Center Network with Software Defined Exclusive Routing,” *IEEE Transactions on Computers*, vol. 64, no. 9, pp. 2534–2544, Sept. 2015.

[138] Q. Yan and F. R. Yu, “Distributed Denial of Service Attacks in Software-Defined Networking with Cloud Computing,” *IEEE Communications Magazine*, vol. 53, no. 4, pp. 52–59, Apr. 2015.

[139] C. Lorenz, D. Hock, J. Scherer, R. Durner, W. Kellerer, S. Gebert, N. Gray, T. Zinner, and P. Tran-Gia, “An SDN/NFV-Enabled Enterprise Network Architecture Offering Fine-Grained Security Policy Enforcement” *IEEE Communications Magazine*, vol. 55, no. 3, pp. 217–223, Mar. 2017.

[140] K. Giotis, C. Argyropoulos, G. Androulidakis, D. Kalogeras, and V. Maglaris, “Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments,” *Computer Networks*, vol. 62, pp. 122–136, Apr. 2014.

[141] B. Subba, S. Biswas and S. Karmakar, "A Neural Network based system for Intrusion Detection and attack classification," *Twenty Second National Conference on Communication (NCC)*, Guwahati, pp. 1-6, 2016, doi: 10.1109/NCC.2016.7561088.

[142] M. Monshizadeh, V. Khatri, and R. Kantola, "Detection as a service: An SDN application," *IEEE International Conference on Advanced Communication Technology (ICACT)*, Bongpyeong, South Korea, DOI: 10.23919/ICACT.2017.7890099, Feb. 2017.

[143] M. Shamseddine, W. Itani, A. Kayssi, and A. Chehab, "Virtualized Network Views for Localizing Misbehaving Sources in SDN Data Planes," *IEEE International Conference on Communications (ICC)*, Paris, France, DOI: 10.1109/ICC.2017.7997296, May 2017.

[144] S. Pisharody, J. Natarajan, A. Chowdhary, A. Alshalan, and D. Huang, "Brew: A Security Policy Analysis Framework for Distributed SDN-Based Cloud Environments," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 6, pp. 1011-1025, Nov. 2019.

[145] A. S. Sendi, Y. Jarraya, M. Pourzandi, and M. Cheriet, "Efficient Provisioning of Security Service Function Chaining Using Network Security Defense Patterns," *IEEE Transactions on Services Computing*, vol. 12, no. 4, pp. 534-549, Jul. 2016.

[146] H. Feng, J. Llorca, A. M. Tulino, D. Raz, and A. F. Molisch, "Approximation algorithms for the NFV service distribution problem," *IEEE INFOCOM 2017- IEEE Conference on Computer Communications*, Atlanta, GA, USA, DOI: 10.1109/INFOCOM.2017.8057039, May 2017.

[147] Z. Zhao, F. Hong, and R. Li, "SDN Based VxLAN Optimization in Cloud Computing Networks," *IEEE Access*, vol. 5, pp. 23312–23319, Oct. 2017.

[148] K. A. Noghani, C. H. Benet, A. Kassler, A. Marotta, P. Jestin, and V. V. Srivastava, "Automating Ethernet VPN deployment in SDN-based Data Centers," *IEEE Fourth International Conference on Software Defined Systems (SDS)*, Valencia, Spain, DOI: 10.1109/SDS.2017.7939142, May 2017.

[149] D. Caixinha, P. Kathiravelu, and L. Veiga, "ViTeNA: An SDN-based Virtual Network Embedding Algorithm for Multi-Tenant Data Centers," *IEEE 15th International Symposium on Network Computing and Applications (NCA)*, Cambridge, MA, USA, DOI: 10.1109/NCA.2016.7778608, Oct. 2016.

[150] R. Cziva, S. Jouet, D. Stapleton, F. P. Tso, and D. P. Pezaros, "SDN-Based Virtual Machine Management for Cloud Data Centers," *IEEE Transactions on Network and Service Management*, vol. 13, no. 2, pp. 212–225, Jun. 2016.

[151] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an Elastic Distributed SDN Controller," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 7–12, Oct. 2013.

[152] G. Cheng, H. Chen, Z. Wang, and S. Chen, "DHA: Distributed decisions on the switch migration toward a scalable SDN control plane," *IEEE IFIP Networking Conference (IFIP Networking)*, Toulouse, France, DOI: 10.1109/IFIPNetworking.2015.7145319, May 2015.

[153] A. A. Amaral, L. de Souza Mendes, B. B. Zarpelão, and M. L. P. Junior, "Deep IP Flow Inspection to Detect beyond Network Anomalies," *Computer Communications*, vol. 98, pp. 80–96, Jan. 2017.

[154] F. Shang, L. Mao, and W. Gong, "Service-aware Adaptive Link Load Balancing Mechanism for Software-Defined Networking," *Future Generation Computer Systems*, vol. 81, pp. 452–464, Apr. 2018.

[155] R. Trestian, K. Katrinis, and G.-M. Muntean, "OFLoad: An OpenFlowbased Dynamic Load Balancing Strategy for Datacenter Networks," *IEEE Transactions on Network Service Management*, vol. 14, no. 4, pp. 792–803, Dec. 2017.

[156] T. V. Phan, T. Van Toan, D. Van Tuyen, T. T. Huong, and N. H. Thanh, "OpenFlowSIA: An Optimized Protection Scheme for Software-Defined Networks from Flooding Attacks," *IEEE Sixth International Conference on Communications and Electronics (ICCE)*, Ha Long, Vietnam, DOI: 10.1109/CCE.2016.7562606, Jul. 2016.

[157] L. F. Carvalho, G. Fernandes, J. J. Rodrigues, L. S. Mendes, and M. L. Proença, "A Novel Anomaly Detection System to Assist Network Management in SDN Environment," *IEEE International Conference on Communications (ICC)*, Paris, France, DOI: 10.1109/ICC.2017.7997214, May 2017.

[158] T. Ha et al., "Suspicious Traffic Sampling for Intrusion Detection in Software-Defined Networks," *Computer Networks*, vol. 109, pp. 172–182, Nov. 2016.

[159] M. A. Al Faruque and K. Vatanparvar, "Energy Management-as-a-Service Over Fog Computing Platform," *IEEE Internet of Things Journal*, vol. 3, no. 2, pp. 161–169, Apr. 2016.

[160] G. S. Aujla, N. Kumar, A. Y. Zomaya, and R. Ranjan, "Optimal Decision Making for Big Data Processing at Edge-Cloud Environment: An SDN Perspective," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 2, pp. 778–789, Feb. 2018.

[161] S. Sarkar, S. Chatterjee, and S. Misra, "Assessment of the Suitability of Fog Computing in the Context of Internet of Things," *IEEE Transactions on Cloud Computing*, vol. 6, no. 1, pp. 46-59, Jan. 2018.

[162] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "ifogsim: A Toolkit for Modeling and Simulation of Resource Management Techniques in the Internet of Things,

Edge and Fog Computing Environments," Software: Practice and Experience, vol. 47, no. 9, pp. 1275–1296, Jun. 2017.

[163] B. Yin, W. Shen, Y. Cheng, L. X. Cai, and Q. Li, "Distributed resource sharing in fog-assisted big data streaming," *IEEE International Conference on Communications (ICC)*, Paris, France, DOI: 10.1109/ICC.2017.7996724, May 2017.

[164] S. O. Aliyu, F. Chen, Y. He, and H. Yang, "A Game-Theoretic Based QoS-Aware Capacity Management for Real-Time EdgeIoT Applications," *IEEE International Conference on Software Quality, Reliability and Security (QRS)*, Prague, Czech Republic, DOI: 10.1109/QRS.2017.48, Jul. 2017.

[165] A. Ruia, C. J. Casey, S. Saha, and A. Sprintson, "Flowcache: A cachebased approach for improving sdn scalability," *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, San Francisco, CA, USA, DOI: 10.1109/INFCOMW.2016.7562149, Apr. 2016.

[166] P. K. Sharma, M.-Y. Chen, and J. H. Park, "A Software Defined Fog Node Based Distributed Blockchain Cloud Architecture for IoT," IEEE Access, vol. 6, pp. 115–124, Sept. 2017.

[167] I. Stojmenovic and S. Wen, "The Fog computing paradigm: Scenarios and security issues," *IEEE Federated Conference on Computer Science and Information Systems*, Warsaw, Poland, DOI: 10.15439/2014F503, Sept. 2014.

[168] Y. Tseng, F. Nait-Abdesselam, and A. Khokhar, "SENAD: Securing Network Application Deployment in Software Defined Networks," *IEEE International Conference on Communications (ICC)*, Kansas City, MO, USA, DOI: 10.1109/ICC.2018.8422405, May 2018. 2018, pp. 1–6.

[169] R. Bruschi, F. Davoli, P. Lago, and J. F. Pajo, "A scalable SDN slicing scheme for multi-domain fog/cloud services," *IEEE Conference on Network Softwarization (NetSoft)*, Bologna, Italy, DOI: 10.1109/NETSOFT.2017.8004244, Jul. 2017.

[170] K. Li and J. Nabrzyski, "Virtual machine placement in cloudlet mesh with network topology reconfigurability," *IEEE 6th International Conference on Cloud Networking (CloudNet)*, Prague, Czech Republic, DOI: 10.1109/CloudNet.2017.8071545, Sept. 2017.

[171] K. Liang, L. Zhao, X. Chu, and H.-H. Chen, "An Integrated Architecture for Software Defined and Virtualized Radio Access Networks with Fog Computing," *IEEE Network*, vol. 31, no. 1, pp. 80–87, Jan. 2017.

[172] C. K. Dominicini, G. L. Vassoler, L. F. Meneses, R. S. Villaca, M. R. Ribeiro, and M. Martinello, "VirtPhy: Fully Programmable NFV Orchestration Architecture for Edge

Data Centers," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 817–830, Dec. 2017.

[173] R. Bruschi, F. Davoli, P. Lago, A. Lombardo, C. Lombardo, C. Rametta, and G. Schembra, "An SDN/NFV Platform for Personal Cloud Services," *IEEE Transactions on Network and Service Management*, vol. 14, no. 4, pp. 1143–1156, Dec. 2017.

[174] L. Hu, J. Wang, E. Song, A. Ksentini, M. A. Hossain, and M. Rawashdeh, "SDN-SPS: Semi-Physical Simulation for Software-Defined Networks," *IEEE Sensors Journal*, vol. 16, no. 20, pp. 7355– 7363, Oct. 2016.

[175] T. Al-Janabi and H. Al-Raweshidy, "Efficient Whale Optimisation Algorithm-based SDN Clustering for IoT focused on Node Density," *IEEE 16th Annual Mediterranean Ad Hoc Networking Workshop (Med-Hoc-Net)*, Budva, Montenegro, DOI: 10.1109/MedHocNet.2017.8001651, Jun. 2017.

[176] R. Wang, Z. Zhang, Z. Zhang, and Z. Jia, "ETMRM: An Energy-efficient Trust Management and Routing Mechanism for SDWSNs," Computer Networks, vol. 139, pp. 119–135, Jul. 2018.

[177] S. Wen, C. Huang, X. Chen, J. Ma, N. Xiong, and Z. Li, "Energy-efficient and delay-aware distributed routing with cooperative transmission for Internet of Things," *Journal of Parallel and Distributed Computing*, vol. 118, pp. 46–56, Aug. 2018.

[178] Y. Zhao, B. Yan, and J. Zhang, "Software defined Passive Optical Networks with Energy-Efficient Control Strategy," *Optik-International Journal for Light and Electron Optics*, vol. 127, no. 23, pp. 11211– 11219, Dec. 2016.

[179] C.-H. Wang, J.-J. Kuo, D.-N. Yang, and W.-T. Chen, "Green Software-Defined Internet of Things for Big Data Processing in Mobile Edge Networks," *IEEE International Conference on Communications (ICC)*, Kansas City, MO, USA, DOI: 10.1109/ICC.2018.8422236, May 2018.

[180] T. Pan, X. Guo, C. Zhang, W. Meng, and B. Liu, "Alfe: A replacement policy to cache elephant flows in the presence of mice flooding," *IEEE International Conference on Communications (ICC)*, Ottawa, ON, Canada, DOI: 10.1109/ICC.2012.6364403, Jun. 2012.

[181] D. Bendouda, A. Rachedi, and H. Haffaf, "An hybrid and proactive architecture based on SDN for Internet of Things," *IEEE 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, Valencia, Spain, DOI: 10.1109/IWCMC.2017.7986414, Jun. 2017.

[182] H.-L. Shi, K. M. Hou, H.-Y. Zhou, and X. Liu, "Energy Efficient and Fault Tolerant Multicore Wireless Sensor Network: E²MWSN," *IEEE 7th International Conference*

on Wireless Communications, Networking and Mobile Computing, Wuhan, China, DOI: 10.1109/wicom.2011.6040317, Sept. 2011.

[183] W. H. F. Aly, "A Novel Fault Tolerance Mechanism for Software Defined Networking," *IEEE European Modelling Symposium (EMS)*, Manchester, UK, DOI: 10.1109/EMS.2017.47, Nov. 2017.

[184] M. S. Munir, S. F. Abedin, M. G. R. Alam, N. H. Tran, and C. S. Hong, "Intelligent service fulfillment for software defined networks in smart city," *IEEE 2018 International Conference on Information Networking (ICOIN)*, Chiang Mai, Thailand, DOI: 10.1109/ICOIN.2018.8343172, Jan. 2018.

[185] N. Bouten, R. Mijumbi, J. Serrat, J. Famaey, S. Latré, and F. De Turck, "Semantically Enhanced Mapping Algorithm for Affinity-Constrained Service Function Chain Requests," *IEEE Transactions on Network and Service Management*, vol. 14, no. 2, pp. 317–331, Mar. 2017.

[186] W. Cerroni, C. Buratti, S. Cerboni, G. Davoli, C. Contoli, F. Foresta, F. Callegati, and R. Verdone, "Intent-based management and orchestration of heterogeneous openflow/IoT SDN domains," *IEEE Conference on Network Softwarization (NetSoft)*, Bologna, Italy, DOI: 10.1109/NETSOFT.2017.8004109, Jul. 2017.

[187] F. Banaie, M. H. Yaghmaee, and S. A. Hosseini, "SDN-based scheduling strategy on load balancing of virtual sensor resources in sensor-cloud," *IEEE 8th International Symposium on Telecommunications (IST)*, Tehran, Iran, DOI: 10.1109/ISTEL.2016.7881905, Sept. 2016.

[188] D. Wu, D. I. Arkhipov, E. Asmare, Z. Qin, and J. A. McCann, "UbiFlow: Mobility management in urban-scale software defined IoT," *IEEE Conference on Computer Communications (INFOCOM)*, Kowloon, Hong Kong, DOI: 10.1109/INFOCOM.2015.7218384, Apr. 2015.

[189] M.-C. Lee and J.-P. Sheu, "An Efficient Routing Algorithm based on Segment Routing in Software-Defined Networking," *Computer Networks*, vol. 103, pp. 44–55, Jul. 2016.

[190] N. Kitsuwan and E. Oki, "Analysis of flows reduction scheme by adopting two MPLS tags in software-defined network," IEEE Conference on Standards for Communications and Networking (CSCN), Berlin, Germany, DOI: 10.1109/CSCN.2016.7784891, Nov. 2016.

[191] R. Challa, Y. Lee, and H. Choo, "Intelligent eviction strategy for efficient flow table management in OpenFlow Switches," IEEE NetSoft Conference and Workshops (NetSoft), Seoul, South Korea, DOI: 10.1109/NETSOFT.2016.7502427, Jun. 2016.

[192] Z. Han, Y. Li, and J. Li, "A novel routing algorithm for iot cloud based on hash offset tree," Future Generation Computer Systems, 2018. C. Kharkongor, T. Chithralekha, and R.

Varghese, "A SDN Controller with Energy Efficient Routing in the Internet of Things (IoT)," *Procedia Computer Science*, vol. 89, pp. 218–227, 2016.

[193] C. Kharkongor, T. Chithralekha, and R. Varghese, "A sdn controller with energy efficient routing in the internet of things (iot)," *Procedia Computer Science*, vol. 89, pp. 218–227, 2016.

[194] C. Lin, K. Wang, and G. Deng, "A QoS-aware routing in SDN hybrid networks," *Procedia Computer Science*, vol. 110, pp. 242–249, 2017.

[195] A. Gharaibeh, M. A. Salahuddin, S. J. Hussini, A. Khreishah, I. Khalil, M. Guizani, and A. Al-Fuqaha, "Smart Cities: A Survey on Data Management, Security, and Enabling Technologies," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2456–2501, Aug. 2017.

[196] M. V. De Assis, A. H. Hamamoto, T. Abrao, and M. L. Proença, "A Game Theoretical Based System Using Holt-Winters and Genetic Algorithm With Fuzzy Logic for DoS/DDoS Mitigation on SDN Networks," *IEEE Access*, vol. 5, pp. 9485–9496, May 2017.

[197] M. Ozcelik, N. Chalabianloo, and G. Gur, "Software-Defined Edge Defense Against IoT-Based DDoS," *IEEE International Conference on Computer and Information Technology (CIT)*, Helsinki, Finland, DOI: 10.1109/CIT.2017.61, Aug. 2017.

[198] T. V. Phan, N. K. Bao, and M. Park, "A Novel Hybrid Flow-Based Handler with DDoS Attacks in Software-Defined Networking," *IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress*, Toulouse, France, DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0069, Jul. 2016.

[199] N.-N. Dao, J. Kim, M. Park, and S. Cho, "Adaptive Suspicious Prevention for Defending DoS Attacks in SDN-Based Convergent Networks," *PLOS ONE*, https://doi.org/10.1371/journal.pone.0160375, vol. 11, no. 8, Aug. 2016.

[200] R. F. Moyano, D. F. Cambronero, and L. B. Triana, "A User-centric SDN Management Architecture for NFV-based Residential Networks," *Computer Standards & Interfaces*, vol. 54, pp. 279–292, Nov. 2017.

[201] S. Arezoumand, H. Bannazadeh, and A. Leon-Garcia, "HyperExchange: A protocol-agnostic exchange fabric enabling peering of Virtual Networks," *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, Lisbon, Portugal, DOI: 10.23919/INM.2017.7987281, May 2017.

[202] D. Drutskoy, E. Keller, and J. Rexford, "Scalable Network Virtualization in Software-Defined Networks," *IEEE Internet Computing*, vol. 17, no. 2, pp. 20–27, Mar. 2013.

[203] Q. Duan, N. Ansari, M. Toy et al., "Software-defined Network Virtualization: An Architectural Framework for Integrating SDN and NFV for Service Provisioning in Future Networks." *IEEE Network*, vol. 30, no. 5, pp. 10–16, Sept. 2016.

[204] A. Blenk, A. Basta, J. Zerwas, M. Reisslein, and W. Kellerer, "Control Plane Latency With SDN Network Hypervisors: The Cost of Virtualization." *IEEE Transactions Network and Service Management*, vol. 13, no. 3, pp. 366– 380, Jul. 2016.

[205] G. S. Aujla, R. Chaudhary, N. Kumar, R. Kumar, and J. J. Rodrigues, "An Ensembled Scheme for QoS-Aware Traffic Flow Management in Software Defined Networks," *IEEE International Conference on Communications (ICC)*, Kansas City, MO, USA, DOI: 10.1109/ICC.2018.8422596, May 2018.

[206] S. Li, D. Hu, W. Fang, and Z. Zhu, "Source routing with protocol-oblivious forwarding (POF) to enable efficient e-Health data transfers," *IEEE International Conference on Communications (ICC)*, Kuala Lumpur, Malaysia, DOI: 10.1109/ICC.2016.7511385, 2016.

[207] K. Lin, F. Xia, W. Wang, D. Tian, and J. Song, "System Design for Big Data Application in Emotion-Aware Healthcare," *IEEE Access*, vol. 4, pp. 6901–6909, Oct. 2016.

[208] M. A. Salahuddin, A. Al-Fuqaha, M. Guizani, K. Shuaib, and F. Sallabi, "Softwarization of Internet of Things Infrastructure for Secure and Smart Healthcare," *IEEE Computer Magazine*, vol 50, issue 7, pp. 74-79, May 2018.

[209] M. Al Shayokh, A. Abeshu, G. Satrya, and M. Nugroho, "Efficient and secure data delivery in software defined WBAN for virtual hospital," *IEEE International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, Bandung, Indonesia, DOI: 10.1109/ICCEREC.2016.7814973, Sept. 2016.

[210] S. Jiang, M. Duan, and L. Wang, "Toward Privacy-Preserving Symptoms Matching in SDN-Based Mobile Healthcare Social Networks," *IEEE Internet of Things Journal*, vol. 5, no. 3, pp. 1379-1388, Jun. 2018.

[211] W. Meng, K.-K. R. Choo, S. Furnell, A. V. Vasilakos, and C. W. Probst, "Towards Bayesian-Based Trust Management for Insider Attacks in Healthcare Software-Defined Networks," *IEEE Transactions on Network and Service Management*, vol. 15, no. 2, pp. 761–773, Jun. 2018.

[212] R. Chaudhary, A. Jindal, G. S. Aujla, N. Kumar, A. K. Das, and N. Saxena, "Lscsh: LSCSH: Lattice-Based Secure Cryptosystem for Smart Healthcare in Smart Cities Environment," *IEEE Communications Magazine*, vol. 56, no. 4, pp. 24–32, Apr. 2018.

[213] V. Varadharajan, U. Tupakula, and K. Karmakar, "Secure Monitoring of Patients With Wandering Behavior in Hospital Environments," *IEEE Access*, vol. 6, pp. 11523–11533, Nov. 2017.

[214] T.-M. Li, C.-C. Liao, H.-H. Cho, W.-C. Chien, C. F. Lai, and H.-C. Chao, "An e-healthcare Sensor Network Load-balancing Scheme using sdn-sfc," *IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom)*, Dalian, China, DOI: 10.1109/HealthCom.2017.8210833, Oct. 2017.

[215] G. Muhammad, M. F. Alhamid, M. Alsulaiman, and B. Gupta, "Edge Computing with Cloud for Voice Disorder Assessment and Treatment," *IEEE Communications Magazine*, vol. 56, no. 4, pp. 60–65, Apr. 2018.

[216] A. Rego, L. Garcia, S. Sendra, and J. Lloret, "Software Defined Network-based control system for an efficient traffic management for emergency situations in smart cities," *Future Generation Computer Systems*, Vol. 88, pp. 243-253, 2018.

[217] W. Junior, B. Silva, and K. Dias, "A systematic mapping study on mobility mechanisms for cloud service provisioning in mobile cloud ecosystems," *Computers & Electrical Engineering*, vol. 69, pp. 256-273, Jul. 2018.

[218] F. Silva, J. Castillo-Lema, A. Neto, F. Silva, and P. Rosa, "Software defined ehealth Networking Towards a Truly Mobile and Reliable System," *IEEE 16th International Conference on e-Health Networking, Applications and Services (Healthcom)*, Natal, Brazil, DOI: 10.1109/HealthCom.2014.7001903, pp. 560–564, Oct. 2014.

[219] N. Chilamkurti, J. H. Park, and N. Kumar, "Concurrent Multipath Transmission with Forward Error Correction Mechanism to Overcome Burst Packet Losses for Delay-sensitive Video Streaming in Wireless Home Networks," *Multimedia Tools and Applications*, vol. 65, no. 2, pp. 201–220, Jul. 2013.

[220] "Btnorthamerica," 2012. [Online]. Available: (http://www.topology-zoo.org/files/BtNorthAmerica.graphml), (Accessed on Jan. 2019).

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

[222] "Mininet," 2014. [Online]. Available: (http://mininet.org/walkthrough/), (Accessed on Nov. 2018).

[223] B. Linkletter, "Using the POX SDN Controller," Retrieved from Open-Source Routing Netw. Simul. Website. 2015. [Online]. Available: (http://www.brianlinkletter.com/using-the-poxsdn-controller) (Accessed on Dec. 2018)

[224] "The OpenFlow Switch Specification," 2014. [Online]. Available: (http://OpenFlowSwitch.org), (Accessed on Jan. 2019).

[225] C. Walsworth, E. Aben, K. Claffy, and D. Andersen, "The CAIDA UCSD anonymized internet traces," 2015, Accessed: May 2019. [Online]. Available: (https://data.caida.org/datasets/passive-2015/) (Accessed on Dec. 2018)

[226] BoNeSi, "The DDoS Botnet Simulator," 2018. [Online]. Available: (https://github.com/Markus-Go/bonesi), (Accessed on Jun. 2019).

[227] A. Santos Da Silva, J. A. Wickboldt, L. Z. Granville, and A. Schaeffer-Filho, "ATLANTIC: A framework for anomaly traffic detection, classification, and mitigation in SDN," *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Istanbul, Turkey, DOI: 10.1109/NOMS.2016.7502793, Apr. 2016.

[228] G. Singh, N. Kumar, and A. K. Verma, "ANTALG: An innovative ACO based routing algorithm for MANETs," *Journal of Network and Computer Applications*, vol. 45, pp. 151–167, Oct. 2014.

[229] R. Skinderowicz, "An improved ant colony system for the sequential ordering problem," *Comput. Oper. Res.*, vol. 86, pp. 1–17, Oct. 2017.

[230] "SNDlib.," (Oct. 2009) [Online]. Available: (http://sndlib.zib.de/) (Accessed on Feb. 2019).

[231] "Basic Onos Tutorial," (2019). [Online]. Available: (https://wiki. onosproject.org) (Accessed on Feb. 2019).

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

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

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

[235] K. E. Avrachenkov, A. Y. Kondratev, and V. V. Mazalov, "Cooperative Game Theory Approaches for Network Partitioning," *International Computing and Combinatorics Conference (COCOON)*, pp. 591–602, Jul. 2017.

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

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

[238] "POCO: Pareto-Optimal Controller Placement." [Online]. Available: (http://www3.informatik.uni-wuerzburg.de/poco) (Accessed on Jun. 2019)

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

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

[241] A. Singla, C. Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking Data Centers Randomly," *ACM SIGCOMM 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI*, San Jose, CA), pp. 225–238, Apr. 2012.