TOWARDS SCALABLE SDN: ENHANCEMENT IN DATA AND CONTROL PLANES

Ilora Maity

TOWARDS SCALABLE SDN: ENHANCEMENT IN DATA AND CONTROL PLANES

Thesis submitted to the
Indian Institute of Technology Kharagpur
for award of the degree

of

Doctor of Philosophy

by

Ilora Maity

Under the guidance of

Dr. Sudip Misra and Dr. Chittaranjan Mandal

Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721 302, India
August 2020

CERTIFICATE

Date: 25/08/2020

This is to certify that the thesis entitled **Towards Scalable SDN: Enhancement in Data and Control Planes**, submitted by **Ilora Maity** to Indian Institute of Technology Kharagpur, is a record of *bona fide* research work under my supervision and I consider it worthy of consideration for the award of the degree of Doctor of Philosophy of the Institute.

Dr. Sudip Misra
Professor
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721 302, India

Dr. Chittaranjan Mandal
Professor
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Kharagpur - 721 302, India

DECLARATION

I certify that

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

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

c. I have followed the guidelines provided by the Institute in writing the thesis.

d. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

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

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

Ilora Maity

Dedicated to
My parents, Parents-in-law, and Beloved husband

ACKNOWLEDGMENT

My journey as a doctoral student at the Indian Institute of Technology Kharagpur has been a wonderful and memorable experience that I will treasure for the rest of my life. I came across many amazing personalities in the course of this journey. These people have directly or indirectly helped me shape my academic career by providing their support, advice, and encouragement. I would like to convey my sincere thanks and gratitude to all the people instrumental in my doctoral research.

I express my deepest gratitude to my supervisor Dr. Sudip Misra, for his constant support and motivation. From the very first day of my doctoral research, he has encouraged me to aim big and never settle for anything less. I am especially thankful to him for his precious insights on my research goals and constructive feedback to mitigate any research lacuna. I will cherish his valuable guidance on transforming a research objective into a good quality research article. I firmly believe that I have evolved both as a researcher and a better personality because of his guidance.

I would like to sincerely thank my joint supervisor, Dr. Chittaranjan Mandal, for his intellectual guidance and infectious enthusiasm, which helped me strengthen my research base and motivated me to aspire for a higher goal. The stimulating and engaging interactions with him have made me confident regarding my research outcomes.

I want to express my gratitude to Prof. Rajib Mall and Prof. Goutam Das for their valuable suggestions, which have enriched my work to a great extent. I am indebted to Prof. Jhareswar Maiti and Prof. Sudipta Mahapatra for kindly agreeing to serve on my doctoral scrutiny committee. I sincerely thank the office staffs Mithun Da, Pratap Da, Anup Da, Binod Da, and Malay Da for helping me whenever needed.

I am genuinely thankful to my colleague Ayan Mondal for his continuous help and support even in the initial days when I did not have a clear idea about research. His dedication and work ethic motivated me and enabled me to complete this work. I would also like to thank Niloy, Samaresh, Satendra, and Pradyumna, for patiently listening to my work presentations and giving constructive feedback. I am grateful to have friends and colleagues like Kankana, Kumud, Timam, Anudipa, Aishwariya, Rituparna, Sumana, Arijit, Aritra, and Barnali, who made my campus life colorful and memorable.

Words fail me when I think of the unconditional love and support of my mother, father, mother-in-law, and father-in-law. I want to sincerely thank them for having faith in me and supporting my career decisions. Finally, thanks to my husband, Abhishek,

whose endless energy and enthusiasm have enriched my life in every aspect.

Ilora Maity

Abstract

Software-Defined Networking (SDN) architecture involves separate data and control planes. The SDN data plane consists of switches that store forwarding rules in flow-tables. On the other hand, the SDN control plane consists of controllers that formulate the flow-rules and install or update them at the switches. SDN adds flexibility and programmability to network operations. Due to the additional benefits of softwarization, traditional networks are being migrated to SDN. The intermediate step of transforming a conventional backbone network into pure SDN is termed as hybrid SDN.

The limited storage capacity of switches is a key challenge in SDN, as the switches use Ternary Content Addressable Memories (TCAMs) having very low capacity. Low rule storage capacity eventually leads to a high number of Packet-In messages and control plane overloading. On the other hand, the number and locations of SDN controllers determine the Quality of Service (QoS) parameters, such as network throughput and flow-processing delays. In particular, the placement of controllers is more challenging in hybrid SDN because of additional aspects such as SDN switch placement and incremental upgrades. These challenges increase processing latency and decrease the overall scalability of SDN. Additionally, scalable network operations should ensure optimal energy consumption. However, the lack of centralized control over the power states of legacy switches impedes energy-aware traffic engineering in hybrid SDN. On the other hand, there exists a trade-off between energy-aware routing and programmable traffic as traffic rerouting may transform programmable traffic to a non-programmable one, if not rerouted carefully.

Motivated by these challenges, in this thesis, we propose multiple schemes to enhance the scalability of SDN data and control planes. We propose an approach for consistent update with redundancy reduction that reduces TCAM usage during update. Additionally, we propose a load reduction strategy that prioritizes traffic flows based on QoS demands and aims to avoid link congestion and rule-space overflow during flow migration. Moreover, we apply the concept of tensor decomposition to aggregate flow-rules and increase the available rule-space. On the other hand, we implement a master controller assignment scheme based on IoT devices' mobility and traffic characteristics to prevent controller overload and distribute traffic optimally across the controllers. In addition, we propose a priority-based SDN switch placement approach and a game theory-based

controller placement approach for hybrid SDN. In the final scheme, we focus on reducing energy consumption while maximizing the programmable traffic as it is the primary purpose of transforming a legacy network to an SDN.

Keywords: SDN, Network Update, Flow Migration, Coalition Game, Rule-Space Management, Caching, Markov Predictor, IoT, Hybrid SDN, Controller Placement, Programmable Traffic, Energy Management

Contents

List of Figures

List of Tables

List of Algorithms

List of Symbols and Abbreviations

List of Symbols

S	Set of SDN switches
$\mathcal{R}$	Set of legacy switches
C	Set of controllers
E	Set of links
$\mathcal{Q}^j$	Device queue of $s_j \in S$
$\mathcal{P}_{old}$	Set of old packets
$\mathcal{P}_{new}$	Set of new packets
$\mathcal{P}^j$	Set of incoming packets at s_j
S^{low}	Set of low priority SDN switches
S^{medium}	Set of medium priority SDN switches
S^{high}	Set of high priority SDN switches
b_{ij}	Bandwidth usage of link e_{ij}
w_{ij}	Capacity of link e_{ij}
F	Set of traffic flows
$\mathcal{D}_m$	Completion time of stage m of flow migration
$\mathcal{D}_m^R$	Migration duration of a flow in the m^{th} stage
$RF(t)$	Reduction factor at time t

H	Device mobility history
η^1	Channel overhead of the wireless channel for IoT device to AP communication
η^1	Channel overhead of the wireless channel for AP to switch communication
LB_v	Lower bound of the v^{th} subproblem
LB_v^0	Initial value of the lower bound of the v^{th} subproblem
$Th^{eff}(t)$	Effective SDN throughput at time-slot t
$\mathbb{B}$	Network upgrade budget
$\mathbb{B}^s$	Switch placement budget
$\mathbb{B}^c$	Controller placement budget
$\mathbb{B}^{s(t)}$	Switch placement budget for time-slot t
$\mathbb{B}^{c(t)}$	Controller placement budget for time-slot t
$LW(e_{ij})$	Weight of link e_{ij}
SU_j	Utilization of j^{th} legacy switch
$PR(r_j)$	Priority of legacy switch r_j
$NS(r_j)$	Number of non-SDN links of legacy switch r_j
$Vol(r_j)$	Average traffic that traverses legacy switch r_j
$\overline{W}(r_j)$	Average weight of the links of legacy switch r_j
$\mathbb{P}_{ij}$	Baseline power usage of link e_{ij}
$\mathbb{P}_i^{act}$	Power consumption of the i^{th} switch in active state
$\mathbb{P}_i^{inact}$	Power consumption of the i^{th} switch in inactive state
$\mathbb{P}_{ij}^E$	Power consumption of link e_{ij}
$\mathbb{P}_i^N$	Power consumption of i^{th} switch
$\mathbb{U}_{ij}^E$	Link utility of link e_{ij}
$\mathbb{U}_a$	Route utility of flow $f_a \in F$

List of Abbreviations

IoT	Internet of Things
SDN	Software-Defined Networking
TCAM	Ternary Content Addressable Memory
IP	Internet Protocol
CPP	Controller Placement Problem
QoS	Quality of Service
ILP	Integer Linear Program
AD	Access Device
ACL	Access Control List
SDIoT	Software-Defined Internet of Things
SNMP	Simple Network Management Protocol
AP	Access Point
API	Application Programming Interface
FCFS	First-Come-First-Serve
OvA	One-Vs-All
LSI	Latency-Sensitivity Index
SVD	Singular Value Decomposition
RF	Reduction Factor
LRU	Least Recently Used
LB	Lower Bound
NTU	Non-Transferable Utility
OSPF	Open Shortest Path First
SA	Simulated Annealing

Chapter 1

Introduction

With the recent advancements of the Internet of Things (IoT) technology, a wide variety of smart devices have become an integral part of our daily life. These devices perform several data-intensive operations such as online gaming, video streaming, smart traffic control, extended reality-based services, smart healthcare, and industrial automation. It is predicted that the number of IoT devices will be more than 50 billion by the end of 2025 [1]. Moreover, the next decade is expected to witness a substantial development of next-generation networking platforms such as 5G, beyond 5G, and 6G. These future networks demand ultra-low latency and high bandwidth [2]. Therefore, flexibility and scalability are the need of the hour to address the QoS demands of the evolving networks.

SDN is a recent networking paradigm [3] that separates the control plane from the data plane. As shown in Figure 1.1, SDN controllers placed in the control plane manage SDN switches in the data plane [4]. Several useful features of SDN [5] (e.g., global view of the network, separation of the data and control planes, and ability to program the network functions) adds flexibility to the network management [6] and makes SDN an attractive choice for network service provisioning.

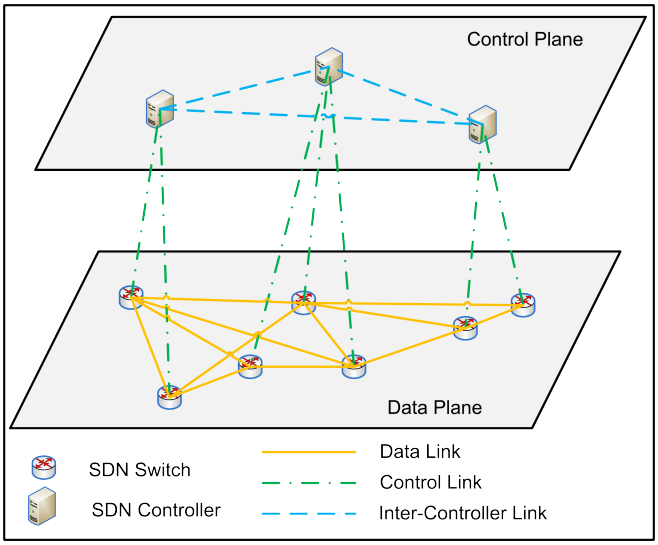


Figure 1.1: SDN: Data and Control Planes

1.1 Scalability Challenges in SDN

Despite offering flexibility in network management, SDN faces scalability issues in data and control planes. The capacity of the switches limits the scalability of the data plane. This limited storage capacity is a problem for network updates that involve changing configuration of each switch separately. Existing update policies require the storage of old configuration rules until the whole update process is complete. Hence, in the worst-case, half of the rule-space needs to be free before the initiation of a network update. Consequently, the cost of storing redundant rules reduces the network scalability. Additionally, traffic flow migration is an essential aspect of network updates in SDN. New traffic flows are generated frequently, and the existing flows are required to migrate paths to accommodate these new flows. However, more than 80% of the total flows in IoT networks are latency-sensitive. Therefore, completing traffic flow migration in minimal time is essential, and the migration process should be consistent.

On the other hand, the control plane's scalability depends mostly on the placement of the controllers. The Controller Placement Problem (CPP) addresses two aspects — (1) the number of controllers required and (2) the position of controllers. Hybrid

SDN is an intermediate step of transforming a traditional backbone network into pure SDN. Therefore, incremental controller placement is an additional aspect of CPP for hybrid SDN. Moreover, the dynamic distribution of the controllers' workload is required to enhance the network performance for large-scale data handling data. Finally, hybrid SDN needs energy-aware traffic engineering to minimize the carbon footprint and ensure a satisfactory amount of programmable traffic.

1.2 Scope of the Work

Rule-space capacity in SDN switches is limited. However, existing SDN update policies ensure consistent packet processing by storing old configuration rules until the whole update process completes. These approaches require maintaining additional rule-space. So, minimizing the trade-off between rule-space usage and packet consistency is an issue which is needed to be addressed. Traffic flow migration is an important aspect of the SDN update. Existing solution approaches for traffic flow migration do not consider the diverse traffic characteristics of traffic flows. An unplanned schedule disrupts the operations of latency-sensitive applications and increases data plane load by link congestion and rule-space overload. Therefore, there is a need for a delay-aware traffic flow migration schedule that aims to reduce the data plane load. On the other hand, in the existing literature, researchers have proposed several works for handling the rule-space capacity constraint. However, the majority of these works do not consider dynamic traffic. Therefore, there is a need for dynamic rule-space aggregation to improve network operations.

On the other hand, legacy Internet Protocol (IP) networks are converted to SDN to utilize flexibility and programmability offered by SDN. However, SDN-based futuristic networks require a distributed control plane instead of a centralized one to address the massive volume of data traffic with low latency. CPP is an essential aspect of realizing a distributed control plane. The majority of the existing studies on CPP

consider pure SDN, which does not require a switch upgrade. However, CPP in hybrid SDN, an intermediate stage of migration from a legacy network to SDN, is incremental and involves newly added SDN switches in each round. Managing these upgraded SDN switches is essential to maintain the QoS requirement of the network. On the other hand, there exists a lacuna in the research literature addressing the problem of control plane load management for large-scale SDN, including SDIoT, where both mobile and static devices are present. However, existing solution approaches do not consider device mobility and heterogeneity while dealing with the dynamic workload. Therefore, there is a need for a controller assignment scheme that considers heterogeneous mobile and static IoT devices to reduce the control plane load.

Traffic engineering is another aspect of network scalability that ensures low-latency processing of traffic flows and limits energy consumption. In the existing literature, researchers proposed different approaches and architecture for green SDNs, viz., [7–9], which achieves energy efficiency by activating the minimum number of links. However, energy management in hybrid SDN should not be at the cost of programmable traffic. Hence, there is a need for a dynamic energy management strategy that optimizes programmable traffic and reduces the overall energy consumption of the hybrid network.

1.3 Problem Statement and Objectives

The objective of this thesis is to study SDN scalability in the context of — *data plane* and *control plane*. The problem statement of this thesis is:

Rule-space capacity constraint and centralized control plane limit the processing capability of SDN. These scalability constraints of SDN data and control planes restrict the adoption of SDN despite offering flexible and programmable network operations.

The *objectives* of the thesis are as follows:

1. Design of an optimized scheduling scheme for consistent SDN update without

storage of redundant rules.

2. Design of a traffic-aware data plane load reduction scheme for traffic flow migration in SDN.

3. Design of a rule-space management scheme for reducing flow-table miss in SDN.

4. Design of a controller-switch assignment scheme for control plane load reduction in SDIoT.

5. Design of a cost-efficient QoS-aware switch and controller placement strategy for hybrid SDN.

6. Design of an energy-efficient traffic engineering approach for hybrid SDN.

The objectives of this thesis are depicted in Figure 8.2 .

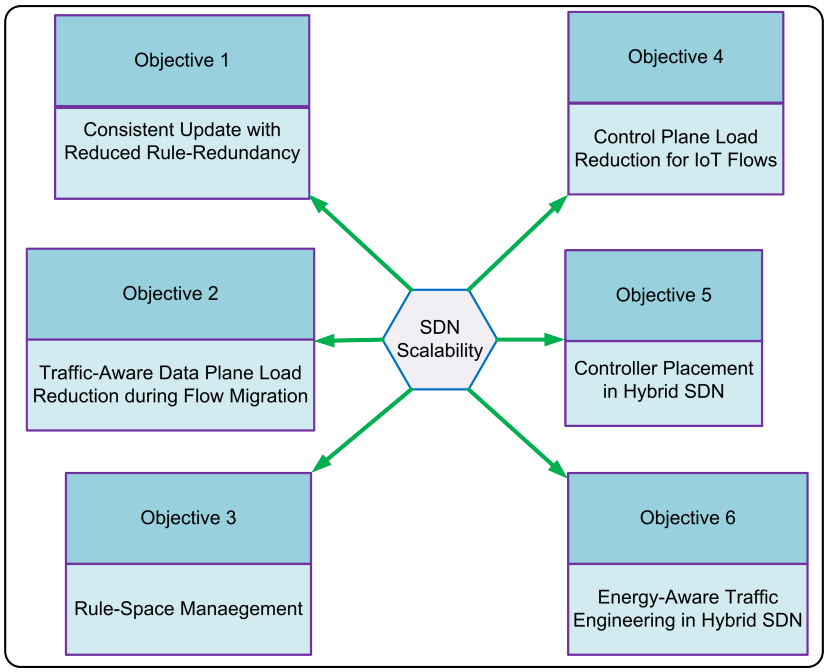


Figure 1.2: Thesis Objectives

1.4 Contributions

The primary contributions of this work are as follows:

- We design a priority-based algorithm for scheduling updates to SDN switches. We propose a packet queueing mechanism for maintaining the consistency of incoming packets during an update. Further, we design a packet processing algorithm that processes the queued packets consistently. We compare our approach with the existing solutions to highlight the benefits of the proposed scheme.

- We formulate an Integer Linear Program (ILP) to minimize the data link bandwidth usage during flow migration. We formulate a coalition graph game to determine the set of flows that must be migrated together. Based on the initial migration schedule, we propose an algorithm to transform the initial migration schedule to a feasible schedule, which reduces the data plane load. Additionally, we analyze the rule-space usage in the switches and propose an algorithm that ensures the rule-space required for the migration process.

- We propose a scheme that is capable of aggregating heterogeneous flow-rules. We envision a tensor-based algorithm to compress rules in each switch. We perform an extensive simulation to analyze the performance of the proposed scheme in terms of rule-space usage.

- We propose a proactive master controller assignment scheme for control plane load reduction in Software-Defined Internet of Things (SDIoT). The proposed scheme uses the Markov Predictor to predict device-switch associations. Based on the prediction results, we propose a device-aware rule-caching approach to reduce the load of the controllers considering device-specific parameters such as QoS demand and flow generation rate. Additionally, we design a master controller assignment algorithm that identifies optimal controller-switch assignments in advance to minimize

the control plane load.

- We present a rank-based approach for the placement of SDN switches in hybrid SDN. In this approach, we consider switch-specific parameters such as the number of links, traffic volume, OSPF weights of the links, and the residual lifetime of legacy switches. We formulate a coalition game to determine the locations of SDN controllers to maximize the network throughput.

- We define a metric named route utility to estimate the cost of the routing paths of the traffic flows. The parameters considered for the cost estimation are power usage of the links, and programmability of the flows. We formulate an ILP for the problem of energy-aware traffic engineering in hybrid SDN. We propose two heuristic algorithms to dynamically generate an optimal network state to achieve the desired objective formulated in the ILP.

1.5 Organization of the Thesis

The rest of the thesis is organized as follows:

- **Chapter 1 – Introduction**: This chapter contains the background, motivation, and objectives of the work.

- **Chapter 2 – Literature Survey**: The related works on scalability in SDN data and control planes are surveyed in this chapter. We identify research gaps in the existing works. Additionally, we study state-of-the-art works related to energy-aware traffic engineering.

- **Chapter 3 – Consistent Update with Redundancy Reduction**: This chapter considers the effects of rule-space capacity constraint during SDN update. We evaluate the performance of the proposed solution considering relevant performance metrics.

- **Chapter 4 – Data Plane Load Reduction for Flow Migration**: In this chapter, we study the scenario of flow migration involving latency-sensitive traffic flows and its effects on the data plane load. We compare the proposed solution with suitable benchmarks to evaluate its performance.

- **Chapter 5 – Rule-Space Management**: This chapter presents an approach to enhance rule-space scalability. We present relevant results to prove the effectiveness of the proposed scheme.

- **Chapter 6 – Control Plane Load Reduction**: In this chapter, we explore load balancing among the controllers in the presence of IoT traffic. Subsequently, relevant results of performance evaluation are presented.

- **Chapter 7 – QoS-Aware Switch and Controller Placement**: In this chapter, we present an approach for the placement of SDN switches and controllers in hybrid SDN to satisfy QoS demands. We analyze the performance of the proposed approach considering appropriate performance metrics.

- **Chapter 8 – Energy-Aware Traffic Engineering**: In this chapter, we study the trade-off between energy-aware traffic routing and the amount of programmable traffic in the context of hybrid SDN where both legacy and SDN switches are present. We present relevant results to show the effectiveness of the proposed solution in the presence of high traffic volume.

- **Chapter 9 – Conclusion**: This chapter summarizes the contributions of this thesis. Additionally, we mention some limitations of this work. Finally, we cite a few research directions for the future extension of this work.

Chapter 2

Related Work

In this chapter, we survey the related literature on data plane scalability, control plane scalability, and energy-aware traffic engineering in SDN. In the existing literature, several works focus on addressing the rule-space capacity constraint in SDN. Moreover, some recent studies are devoted to control plane optimization for enhancing network scalability. Additionally, many prior works investigate the problem of energy management, which is necessary for handling large-scale data traffic.

The rest of the chapter is organized as follows. Section 2.1 presents works related to data plane scalability. In Section 2.2, we review the related literature on control plane scalability. Section 2.3 discusses the existing works on energy-aware traffic engineering in SDN. Finally, Section 2.4 concludes the chapter.

2.1 Data Plane Scalability

We divide the exiting literature on data plane scalability into two sections – *capacity-aware consistent update* and *rule-space capacity management*. Table 2.1 shows a summary of different works on data plane scalability.

Table 2.1: Summary of different works on data plane scalability

Studies	Solution Approaches	Shortcomings
Francois *et al.* [10]	Ordered update	Modifies network protocols and switches
Reitblatt *et al.* [11]	Two-phase update	Overhead due to the addition of version tags to incoming packets and storage of old rules
Mizrahi *et al.* [12]	Timed update	Synchronizing updates to all the switches encounters computational complexity and depends on specific switch properties
McGeer *et al.* [13]	Buffered update	Overloads the control plane; additional overhead is incurred due to the installation of the intermediate flow-rules
Meiners *et al.* [14]	Flow-table aggregation	High computation time for the larger partition size
Kanizo *et al.* [15]	Flow-rule partitioning	Increases the total number of rules, because two separate rules are generated for each do not care (*) pivot bits
Kosugiyama *et al.* [16]	Traffic flow aggregation	Considers latency-sensitive flows only

2.1.1 Capacity-Aware Consistent Update

Existing works related to capacity-aware consistent update are categorized in four parts including *ordered*, *incremental*, *timed*, and *buffered updates*.

In case of ordered update, the controller partitions the total update procedure into multiple stages [10], [17], [18]. It waits for the completion of each stage before starting the next stage. The last stage is garbage collection, where older rules are deleted. Francois

et al. [10] proposed an ordered update scheme that ensures packet-level consistency by preventing the formation of loops. However, this approach requires a modification of network protocols as well as of the forwarding devices. Bera *et al.* [17] proposed a prediction-based mobility-aware update mechanism for SDIoT, which inserts new rule at the next Access Device (AD), and performs garbage collection at the current AD. Clad *et al.* [18] generated an optimized sequence for updating the weights of links. The ordered update policy encounters service latency as the completion of the previous phase restricts each phase.

In the incremental update approach, the network is updated in multiple phases, where each phase updates a portion of flow-rules or a subset of switches. Reitblatt *et al.* [11] proposed a two-phase update approach where the internal and ingress switches are updated in phase 1 and phase 2, respectively. Updated ingress switches attach new version tags to the incoming packets. The incoming packets are processed by either old or new rules (not both) based on the version tag. Older rules are deleted after all packets with old version tags are processed. This method increases the load on the ingress switches, as they have to modify the incoming packets. Moreover, memory overhead is incurred for storing old rules. In another work, Canini *et al.* [19] discussed an incremental update approach, which is similar to database transactions, where either all switches are updated, or none are. Therefore, the ordered and incremental update approaches require extra flow-table space for accommodating duplicate rules. Moreover, the controller is involved until all switches complete update.

To reduce this overhead, Mizrahi *et al.* [12] proposed an extension of OpenFlow protocol by scheduling the update phases at particular time instants for both ordered and incremental updates. This approach preserves packet-level consistency by avoiding conflicts in updates. This technique reduces the duration required to store older rules in SDN switches. However, synchronizing updates to all the switches encounters computational complexity and depends on particular forwarding devices' characteristics.

Buffered update approach [13] identifies the incoming packets, whose routes are going to be affected by the upcoming update, and redirects the packets to the controller by installing intermediate flow-rules at all switches. These packets are buffered in the control plane until the switches are updated. After the completion of the update, the packets are processed according to the new rules. The major limitation of this approach is that it overloads the controller and increases service latency. Further, additional overhead is incurred due to the installation of the intermediate flow-rules.

2.1.2 Rule-Space Capacity Management

Prior works related to rule-space capacity management are categorized in three parts — *flow-table aggregation*, *flow-rule partitioning,* and *traffic flow aggregation*.

Earlier, table aggregation approaches considered only prefix entries, where do not care (∗)s do not appear at the beginning of the ternary strings. Applegate *et al.* [20] proposed a prefix-based minimization technique for Access Control Lists (ACLs), which have entries similar to TCAMs. Meiners *et al.* [14] proposed bit weaving, which partitions the total rule-set and permutes the bit positions for each of the partitions to transform all non-prefix entries into prefix entries. Finally, these transformed partitions are merged, after which each entry is re-permuted to their original bit order. However, one of the significant limitations of bit weaving is high computation time for the larger partition size. This is even worse in networks where data frequently changes because bit weaving recomputes the whole rule-set for each rule update.

Other related works concern the approach of partitioning the flow-rules. Kanizo *et al.* [15] presented a decomposition technique, which partitions a flow-table into sub-tables and distributes the sub-tables across the network. They proposed two methods — Pivot Bit Decomposition (PBD) and Cut-Based Decomposition (CBD). PBD decomposes the table into sub-tables by selecting a pivot bit/column. However, PBD increases the total number of rules, because two separate rules are generated for each do not care (*) pivot

bits. On the other hand, CBD represents the set of rules by a dependency graph. Moshref *et al.* [21] proposed a virtual Cloud Rule Information Base (vCRIB), which partitions the overlapping rules by splitting them. Consequently, the overall number of rules increases.

Traffic flow aggregation approaches minimize the total number of flows to reduce the number of flow-rules. Kosugiyama *et al.* [16] proposed an approach that considers end-to-end delay as a parameter of flow aggregation. However, the authors considered latency-sensitive flows only.

2.2 Control Plane Scalability

We divide the prior works on control plane scalability into two categories — *control plane load management* and *controller placement*. Table 2.2 shows a summary of different works on control plane scalability.

Table 2.2: Summary of different works on control plane scalability

Studies	Solution Approaches	Shortcomings
Bari *et al.* [22]	Dynamic controller provisioning	Does not consider queueing delay at the controller
Sahoo *et al.* [23]	Switch migration-based load balancing	Does not consider the effects of uneven traffic distribution.
Heller *et al.* [24]	Latency-aware controller placement	Brute-force approach
Müller *et al.* [25]	Load-aware controller placement	Ignores the switch-to-controller latency
Huque *et al.* [26]	Load and latency-aware controller placement	Frequent activation and deactivation of controllers increases control plane overhead in terms of messages

2.2.1 Control Plane Load Management

Existing approaches in this field are categorized into two parts — *controller placement-based* and *switch migration-based*.

Controller placement-based schemes select the number and locations of the controllers to manage the load. Hock *et al.* [27] considered the maximum control link latency and the number of switches attached to a controller in order to place the controllers and stabilize the load. However, the authors assume static traffic between switches and controllers. Therefore, this approach is not preferable for large-scale networks, including IoT networks. Ksentini *et al.* [28] proposed a controller placement technique based on Nash bargaining game. The authors considered control link latency and equal load distribution to the controllers as the major objectives. However, this approach does not consider the master and slave roles of an SDN controller. Huque *et al.* [26] proposed LiDy+, which places the controller modules based on data plane traffic prediction. The load is distributed evenly among the controllers in each module. However, frequent activation and deactivation increases control plane overhead in terms of messages.

Switch migration-based schemes migrate switches from a highly-loaded controller's domain to the domain of a lightly-loaded controller. Dixit *et al.* [29] proposed a switch-migration scheme that migrates switches from an overloaded controller to a controller with less load. The proposed approach includes the addition and removal of controllers, as required, in the presence of dynamic traffic. However, this approach ignores switch-to-controller latency. Bari *et al.* [22] proposed Dynamic Controller Provisioning with Simulated Annealing (DCP-SA), which dynamically activates and deactivates controllers to reduce the flow setup cost and the overhead for communication. In this work, the authors used two heuristics based on greedy knapsack and Simulated Annealing (SA). The proposed heuristics periodically reassign switches to controllers for addressing load imbalance at the control plane. The proposed approach does not consider queueing delay at the controller. Sahoo *et al.* [23] proposed an Efficient Switch Migration technique for

Load Balancing (ESMLB) scheme to balance the control plane load in SDIoT. The proposed approach identifies the overloaded controllers and the switches which send the maximum Packet-In requests to each overloaded controller. Each selected switch is migrated to a lightly-loaded target controller, which is selected based on multiple criteria such as hop count, memory usage, and bandwidth. However, the proposed approach does not consider the effects of uneven traffic distribution.

2.2.2 Controller Placement

The prior research works related to controller placement can be categorized broadly into three groups depending, on the parameters considered for placing the controllers.

The first category of work considers only the latency between switches and controllers. Heller *et al.* [24] formed an optimization problem to determine the number and location of controllers for given network topology. This work considered average-case (worst-case) latency bound as metrics for the optimization problem formulated as a k-median (k-center) problem. This approach is a brute-force, in which all possible solutions were evaluated to reach the optimal solution. Lange *et al.* [4] proposed a heuristic algorithm to address the CPP. The principal metrics considered in this solution are controller-switch latency, inter-controller latency, and network resiliency. However, this approach does not consider network traffic or controller capacity as a parameter for the solution.

The second category of works considers only the traffic load on the controllers. Müller *et al.* [25] proposed a controller placement scheme based on three major parameters — (1) path diversity between controllers and switches, (2) controller capacity, and (3) ordering of backup controllers. The authors suggested a heuristic method for listing the backup controllers based on proximity or residual capacity. However, the authors ignored the latency between switches and controllers.

Another category of work considers both the control plane load and the latency between the network elements. Ksentini *et al.* [28] suggested a game-theoretic approach

considering controller load, switch-controller latency, and inter-controller latency. However, this work assumes static network traffic. Tanha *et al.* [30] proposed a resilient solution considering deployment cost and propagation latency. The authors recommended backup controllers at multiple resiliency levels to address controller failure. Sallahi *et al.* [31] developed a mathematical model to estimate the number and location of the SDN controller(s). The authors considered different types of controllers and links for the solution. However, this solution does not apply to large-scale networks. Huque *et al.* [26] proposed a controller placement technique for large-scale networks. The authors also made a provision for *open search* that removes the restriction of selecting controller locations from a set of fixed choices only. However, this work did not consider any QoS parameters.

2.3 Energy-Aware Traffic Engineering in SDN

Several existing works investigates the problem of energy management in SDN/hybrid SDN [7, 9, 32]. Giroire *et al.* [7] considers the rule-space constraint of SDN switches and minimizes energy consumption by deactivating data links. The authors formulated an ILP and proposed a heuristic algorithm. Fernández-Fernández *et al.* [9] considers in-band control traffic as the basic criteria for deactivating links and proposed an heuristic algorithm. Huin *et al.* [33] proposed a traffic-aware energy management scheme, named SENAtoR, for hybrid SDN. This work includes traffic rerouting, link deactivation, and traffic monitoring to avoid packet loss. SENAtoR reactivates the deactivated SDN switches in case of a sudden increase in traffic load or link failure. Assefa and Özkasap [32] proposed a new metric named Ratio for Energy Saving in SDN (RESDN), which quantifies the amount of link usage. The authors proposed a heuristic algorithm to assign route, having the maximum RESDN, to each flow.

2.4 Concluding Remarks

In this chapter, we present the state-of-the-art related to SDN scalability. Existing SDN update approaches store old rules and new rules until all switches are updated to maintain packet consistency. Hence, for the worst-case scenario, 50% of the storage space needs to be empty before starting the network update. Therefore, the cost of storing redundant rules decreases the scalability of the overall network. This problem motivates us to design a scheme for SDN update without storing old rules, once the new rules are installed. Moreover, existing solution approaches do not consider the diverse traffic characteristics of traffic flows during an update. This is problematic for the migration of latency-sensitive flows, which is a frequent event during the SDN update. Motivated by this lacuna, we design a traffic-aware flow migration scheme. Additionally, we infer that there exist a few works for handling the capacity constraint of flow-tables. However, most of these works do not consider dynamic network traffic, which is usual for IoT applications. This lacuna motivates us to design a flow-rule aggregation scheme that considers heterogeneous traffic and reduces control messages.

From the detailed study of the existing literature, we infer that there exists a lacuna in the research literature addressing the problem of control plane load management for large-scale SDN, including SDIoT, where the heterogeneous attributes of IoT devices have a major impact on the control plane load. However, existing solution approaches do not consider device heterogeneity while dealing with the dynamic workload. Moreover, existing solution approaches ignore the bursty nature of IoT traffic due to different activation models of IoT devices. Motivated by this problem, we propose a master controller assignment scheme for control plane load reduction in SDIoT while considering heterogeneous attributes of IoT devices such as mobility, activation model, and QoS demand. On the other hand, most of the existing literature on CPP seeks to find the optimum placement of controllers depending on the traffic load and latency between switches and controllers. In addition, most of the existing studies consider pure SDN,

which does not require a switch upgrade. However, CPP in hybrid SDN is incremental and involves newly added SDN switches in each round. Managing these upgraded SDN switches is essential to maintain the QoS requirement of the network. This lacuna motivates us to design a controller placement scheme in hybrid SDN so that each controller is able to deliver guaranteed service in terms of throughput and delay.

From the exhaustive study of existing literature, it is evident that there exists a need for an energy management scheme in hybrid SDN, which optimizes the programmable traffic. Existing solution approaches do not address the trade-off between programmable traffic and energy-aware routing. Motivated by this research gap, we design an energy-aware traffic engineering in hybrid SDN considering programmable traffic as a metric.

Chapter 3

Consistent Update with Redundancy Reduction

In this chapter, we present a scheme for *Consistent Update with Redundancy Reduction (CURE)* in SDN. Existing SDN update approaches store old rules along with new rules until all switches are updated. Therefore, for the worst-case scenario, 50% of the storage space needs to be empty before starting the network update. CURE ensures consistent flow-rule update without storing old flow-rules. Consequently, the maximum number of flow-rules present in the network during the update is reduced.

This chapter consists of four sections. The system model of CURE is presented in Section 3.1. Section 3.2 describes the proposed scheme. Section 3.3 depicts the experimental results. Finally, Section 3.4 concludes the proposed work and discusses directions for future work.

3.1 System Model

We model the network as a graph $\mathcal{G} = (\mathcal{N}, E)$, where $\mathcal{N}$ is the set of nodes, and E is the set of links between the nodes. The set $\mathcal{N}$ is expressed mathematically as:

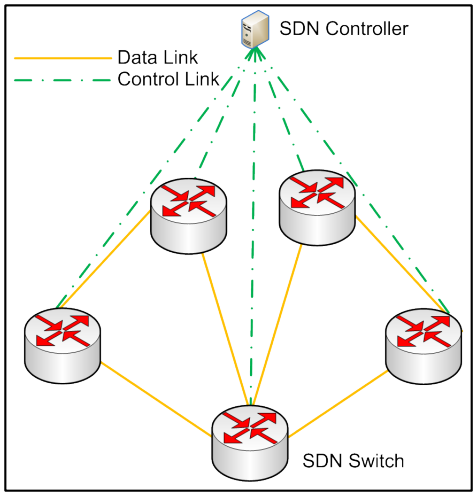


Figure 3.1: CURE: SDN Architecture

$$\mathcal{N} = C \cup S, \tag{3.1}$$

where C is the set of controllers, and S is the set of OpenFlow switches. Figure 3.1 shows the network model. The upper bound of the number of flow-rules which can be stored in an OpenFlow switch s_i is denoted as U_i. Each switch s_j has an associated device queue denoted as $\mathcal{Q}^j$. The set of links E is defined as:

$$E = E_{cc} \cup E_{cs} \cup E_{ss}, \tag{3.2}$$

where E_{cc} is the set of links between the controllers, E_{cs} is the set of control links between the controllers and the OpenFlow switches, and E_{ss} is the set of data links between the OpenFlow switches for packet forwarding.

For simplicity, we assume a centralized control plane containing a single controller c. Hence, $S = \{s_1, s_2, ..., s_{|\mathcal{N}|-1}\}$, $E_{cc} = \phi$ and the number of links in E_{cs} is $|S| = |\mathcal{N}| - 1$.

Each switch stores the flow-rules in one or multiple flow-tables [34]. A flow-rule R_i^j in s_j is a ternary string denoted by a tuple $< Pr_i^j, M_i^j, A_i^j >$, where Pr_i^j denotes rule priority, M_i^j denotes the set of match fields, and A_i^j denotes the set of action values.

Each flow-rule also contains a set of counters for storing the rule statistics, timeout value, cookie, and flags [34]. If an incoming packet matches multiple rules, then the rule with the highest priority value is selected, and the corresponding action is taken.

Definition 1 (State of a Switch)**.** *The state of s_j at time t is defined by:*

$$\Lambda_j(t) = \{R^j(t), E^j_{cs}(t), E^j_{ss}(t), \tau^j(t)\}, \tag{3.3}$$

where $R^j(t)$ is the set of flow-rules of s_j at time t, $E^j_{cs}(t) \in E_{cs}$ is the set of control links involving s_j at time t, $E^j_{ss}(t) \in E_{ss}$ is the set of data links involving s_j at time t, and τ^j is the last update time of s_j at time t.

Definition 2 (Network Configuration)**.** *Network configuration at time t is defined by:*

$$\Gamma(t) = \bigcup_{j=1}^{|S|} \Lambda_j(t) \tag{3.4}$$

Definition 3 (Network Update)**.** *Network update in SDN is migration from one network configuration Γ to another configuration $\Gamma^{'}$ such that,*

$$\Gamma(t_i) \neq \Gamma^{'}(t_j), \text{ where } t_i \neq t_j \tag{3.5}$$

Major objective for this work is to minimize the maximum TCAM usage during update without congesting the links and to maintain packet-level consistency. For a network update from $\Gamma(t_i)$ to $\Gamma^{'}(t_j)$, the optimization problem is formulated as follows:

$$\min \max \sum_{t=t_i}^{t_j} \sum_{j=1}^{|S|} |R^j(t)| \tag{3.6}$$

Equation (3.6) minimizes the maximum number of rules in the whole network, subject to the following constraints:

$$|R^j(t)| \leqslant U_j, \forall s_j \in S \tag{3.7}$$

Equation (3.7) expresses the switch capacity constraint for storing flow-rules.

$$\begin{aligned} &M_r^j = M_s^j \text{ and } A_r^j = A_s^j, \forall \Lambda_j(t_i) = \{R^j(t_i), E_{cs}^j(t_i), E_{ss}^j(t_i), \\ &\tau^j(t_i)\}, \forall \Lambda_j(t_k) = \{R^j(t_k), E_{cs}^j(t_k), E_{ss}^j(t_k), \tau^j(t_k)\}, t_i < t_k, \\ &R_r^j \in R^j(t_i), R_s^j \in R^j(t_k), \text{Duration}(R_r^j) < \text{Duration}(R_s^j), \end{aligned} \tag{3.8}$$

where Duration(R_i^j) is a counter [34], which denotes the elapsed time after installation of the flow-rule R_i^j. Equation (3.8) prohibits the storage of older and newer versions of a rule in a switch simultaneously.

3.2 CURE: The Proposed Scheme

In this section, we describe the proposed scheme, CURE, for the SDN update. Based on workload, we first classify the to-be-updated switches into three priority regions, namely high, medium, and low. Thereafter, we design an algorithm for scheduling updates among the switches of different priority regions. Next, we propose a packet queueing mechanism to maintain packet-level consistency during the update. Finally, we propose an algorithm for processing the queued packets.

3.2.1 Switch Classification

Each OpenFlow switch flow-table maintains a counter field, which records the details of the matching packets. Based on the counter value, we build a training data set. Therefore, we employ the existing One-Vs-All (OvA) multiclass classification algorithm [35] to classify the to-be-updated switches into three priority zones — low, medium, and high. This classification depends on the network topology, packet arrival rate, and existing flows in the network. If the traffic load in all switches are approximately equal, CURE uses the number of active entries in each flow-table as a metric for the classification. The number of active entries in each flow-table is also stored as a counter field [34].

3.2.2 Rule Update

Algorithm 3.1 schedules the update based on the priority zones. Before starting the update, c sends *UPDATE* signal at time T_0 to mark the set of switches that are to be updated. Therefore, the network configuration before update is $\Gamma(T_0)$. c waits for δ time interval before sending the first update packet. Heavily loaded switches are updated first at time $T_{high} > T_0$. Next, medium priority switches are updated at time $T_{medium} > T_{high}$. Finally, low priority switches are updated at time $T_{low} > T_{medium}$. During the update procedure at a switch, the set of new rules is installed first, and the older rules are deleted thereafter. In other words, garbage collection at each switch is performed right after the completion of the update at the switch. Therefore, this algorithm complies with the constraints stated in Equations (3.7) and (3.8). When every switch is updated, the network reaches a configuration $\Gamma(T_{complete})$ at time $T_{complete} > T_{low}$.

Algorithm 3.1: CURE: Rule Update Algorithm

Inputs : S^{low}, S^{medium}, S^{high}
Output: $S^{''}$: Set of updated switches

```
1  UPDATESWITCHES(S^Reg)
2  forall s_j ∈ S^Reg do
3  |   Process P_old
4  |   Insert R^j'
5  |   Remove R^j
6  |   S'' ← S'' ∪ {s_j}
7  end
8  S'' ← ∅
9  forall s_j ∈ S^low ∪ S^medium ∪ S^high do
10 |   SIGNAL(s_j, UPDATE)
11 |   WAIT(δ ms)
12 end
13 UPDATESWITCHES(S^high)
14 UPDATESWITCHES(S^medium)
15 UPDATESWITCHES(S^low)
16 return S''
```

Definition 4 (Old Packet)**.** *After T_0, a packet is marked old, if it is processed by a*

switch, which is yet to be updated.

Definition 5 (New Packet)**.** *After T_0, a packet is marked new, if it is processed by a updated switch.*

Let $\mathcal{P}_{old}$ and $\mathcal{P}_{new}$ denote the sets of *old* and *new* packets, respectively. When c selects a priority region for the update, all $p \in \mathcal{P}_{old}$ in that region are processed before starting the installation of new rules. This ensures that a packet, which is already processed by an old rule, is only processed by old rules. If an *old* packet reaches an updated switch, the packet is sent to c for further decision. Similarly, if a *new* packet reaches a to-be-updated switch, which is not in the current update region, the packet is sent to c for further decision.

Definition 6 (Update Duration)**.** *Update duration is the time interval between the dispatch of the first update message by c and the update completion of the last switch, including garbage collection.*

Definition 7 (Inconsistent Packet)**.** *A packet $p \in \mathcal{P}_{old}$ is termed inconsistent, if it reaches an updated switch. A packet $p \in \mathcal{P}_{new}$ is termed inconsistent if it reaches a switch, which is not updated and is not in the current update region.*

3.2.3 Packet Queueing

Algorithm 3.2 depicts a queueing mechanism for the consistent processing of incoming packets during an ongoing update procedure. The packet queueing algorithm (PQA) is triggered for each to-be-updated switch $s_j \in S$ in the present update region after c starts update in that region. If s_j has received an *UPDATE* signal recently, PQA checks statistics at c to verify whether the switch is already updated. PQA stores the packet if the update process is incomplete in the corresponding switch.

Packets are stored in $\mathcal{Q}^j$ until it is full. Thereafter, the packets are redirected to the least priority switch $s_{neighbor}$, which belongs to a lower priority region and has free buffer

Algorithm 3.2: CURE: Packet Queueing Algorithm

Inputs : $S^{''}$,s_j, P^j
Output: P_{count}: Number of packets buffered outside of $\mathcal{Q}^j$

```
1  storePACKET(p, s_j, j)
2  if Q^j is not full then
3  |  Store p in Q^j
4  else
5  |  if s_neighbor ≠ NULL then
6  |  |  Store p in Q^neighbor
7  |  |  P_count ← P_count + 1
8  |  else
9  |  |  Buffer p at c
10 |  |  P_count ← P_count + 1
11 |  end
12 end
13 forall p ∈ P^j do
14 |  if s_j ∈ S'' then
15 |  |  Process p
16 |  else
17 |  |  storePACKET(p, s_j, j)
18 |  end
19 end
20 return P_count
```

space within one-hop neighbors of s_j. In this scenario, a switch-identifier flag is added to the packet header specifying the switch id where the packet arrived initially. The packets are buffered at c when no such neighbor exists. For each switch, we maintain a counter P_{count} that counts the number of packets stored outside of the switch's buffer.

3.2.4 Packet Processing

After the completion of the update, each switch s_u triggers c by informing that it is ready for processing packets. Algorithm 3.3 describes the procedure of processing the waiting-packets. If $\mathcal{Q}^u$ is full and the buffer size is K, the packet processing algorithm processes the first K packets waiting at $\mathcal{Q}^u$. Then a portion of $\mathcal{Q}^u$ is reserved for storing the waiting packets with matching switch-identifier flag in the one-hop neighbor. We

name this buffer space as *secondary buffer*. The size of *secondary buffer* is determined from the available counter value. Packets waiting in $\mathcal{Q}^{neighbor}$ and/or c are shifted to the *secondary buffer*. After processing these packets, the *secondary buffer* space is merged with the switch's original buffer before processing the new ones.

Algorithm 3.3: CURE: Packet Processing Algorithm

Input : s_u: Switch that triggered packet processing
Output: P'': Set of packets in *secondary buffer*

```
1  if |Q^u| == K then
2      P'' ← ∅
3      Process first K packets in Q^u
4      forall p ∈ P^j stored at s_neighbor do
5          Copy p to secondary buffer of Q^u
6          P'' ← P'' ∪ {p}
7      end
8      forall p ∈ P^j buffered at the c do
9          Copy p to secondary buffer of Q^u
10         P'' ← P'' ∪ {p}
11     end
12     Process packets in secondary buffer
13     Merge secondary buffer with Q^u
14 end
15 Process packets in Q^u
16 return P''
```

3.2.5 Queueing Model

The queue of each switch s_j is modeled as a $M/M/1/K/\alpha$ queueing system [36,37] where the incoming packets follow Poisson's distribution and those packets are processed by s_j with an exponentially distributed service time. Let, $\frac{1}{\mu_j}$ and $\frac{1}{\lambda_j}$ denote the mean service time and mean inter-arrival time at s_j, respectively. We also consider that each switch has a finite queue length of K. Figure 3.2 depicts the queueing model for SDN.

Figure 3.3 shows the state-transition-rate diagram of our proposed queueing model for a single switch. The average packet arrival rate and average service rate for the switch be λ and μ, respectively. Therefore, the traffic intensity is $\rho = \frac{\lambda}{\mu}$. The switch is

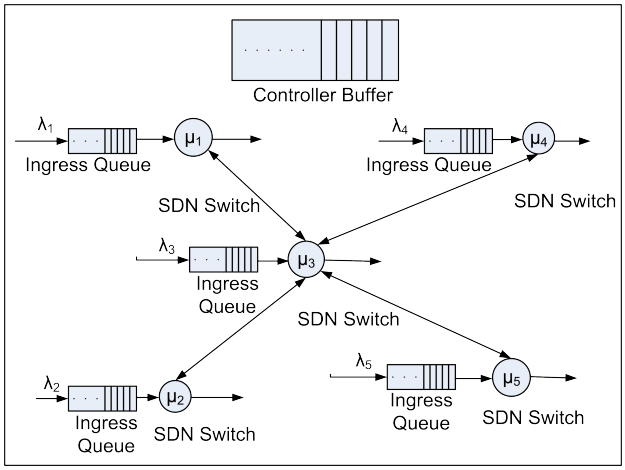


Figure 3.2: SDN Queueing Model

in region $r \in \{high \cup medium \cup low\}$. Initially, c sends an update signal to the switch. As depicted in Figure 3.3, we consider that the update procedure of an OpenFlow switch consists of three stages. In the first stage, the switch receives an update signal, and region r has not started the update. The second stage begins when r starts the update. The final stage begins when the switch completes the update. The switch continues processing until the second stage begins. During the second stage, the switch queues the received packets, unless it completes an update. Therefore, the service rate for this stage is $\mu = 0$. If the switch queue is full, the packets are buffered at the neighbor queue or the controller buffer, according to Algorithm 2. Hence, the increased traffic intensity of a neighbor switch s_a for buffering packets of the current switch is given by:

$$\rho_a^{over} = \left(\frac{\lambda + \lambda_a}{\mu_a} \right) \tag{3.9}$$

During the final stage, the switch processes the packets from the neighbor buffer and its buffer, as mentioned in Algorithm 3.3. Therefore, the new packet arrival rate is $\lambda^{new} = \lambda + \lambda^{neighbor}$, where $\lambda^{neighbor}$ is the rate at which the packets arrive at the current switch from the buffer of the neighbor switch. The traffic intensity in this scenario is

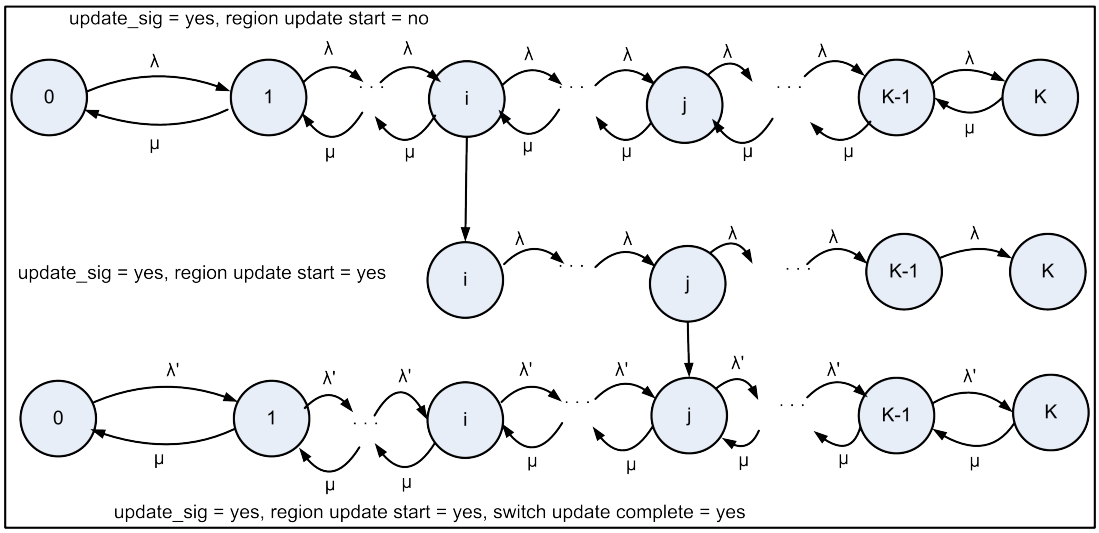


Figure 3.3: State-Transition-Rate Diagram of CURE for a Switch

$\rho^{new} = \frac{\lambda^{new}}{\mu}$. After the switch processes all the packets stored in the neighbor queue, we set $\lambda^{neighbor} = 0$ and $\lambda^{new} = \lambda$.

The probabilities that the switch has a packets in the three stages are denoted by P_a^1, P_a^2, and P_a^3, respectively. However, as per our assumption, the processing of packets at a switch is a Poisson process. Therefore, according to queueing theory, the steady state probability that the switch has i packets in the first stage is given by:

$$P_i^1 = \rho^i P_0^1 \tag{3.10}$$

We consider the scenario that region r starts update when the switch has i packets queued and completes update when it has j packets queued. We know, $P_i^2 = P_i^1$. During the second stage, packets are added to the queue at the rate of λ and no processing is performed. Hence, we get:

$$P_i^2 = P_{i+1}^2 = \cdots = P_K^2 = P_i^1 \tag{3.11}$$

Similarly, from Equation (3.11), we get:

$$P_j^3 = P_j^2 = P_i^1 \tag{3.12}$$

The probability P_j^3 is also expressed as:

$$P_j^3 = (\rho^{new})^j P_0^3 \tag{3.13}$$

From Equations (3.10), (3.12), and (3.13) we have:

$$P_0^3 = \frac{\rho^i}{(\rho^{new})^j} P_0^1 \tag{3.14}$$

According to queueing theory for finite queue length, at steady state:

$$P_0^1 = \frac{1-\rho}{1-\rho^{K+1}},\ P_0^3 = \frac{1-\rho^{new}}{1-(\rho^{new})^{K+1}} \tag{3.15}$$

Hence, from Equations (3.14) and (3.15), the probability P_0^1 is defined as:

$$P_0^1 = \frac{(\rho^{new})^j(1-\rho^{new})}{\rho^i(1-(\rho^{new})^{K+1})} \tag{3.16}$$

Let L and L^{new} be the expected number of packets in the switch before starting update and after the completion of update, respectively. Mathematically,

$$L = \frac{\rho(1 + K\rho^{K+1} - (K+1)\rho^K)}{(1-\rho)(1-\rho^{K+1})} \tag{3.17}$$

$$L^{new} = \frac{\rho^{new}(1 + K(\rho^{new})^{K+1} - (K+1)(\rho^{new})^K)}{(1-\rho^{new})(1-(\rho^{new})^{K+1})} \tag{3.18}$$

Let W and W^{new} be the mean waiting time at the switch before starting update and after the completion of update, respectively. Therefore, the increase in mean waiting time at the OpenFlow switch due to update is given by:

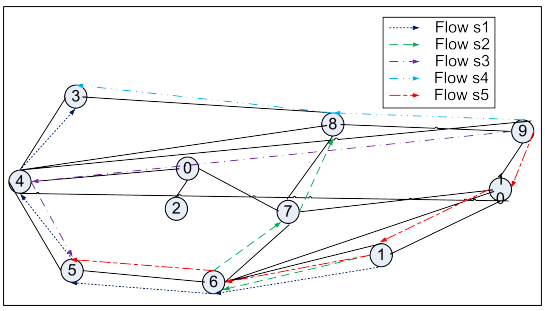
$$\begin{aligned} W^{new} - W = \left(\frac{L^{new}}{\lambda^{new}} - \frac{L}{\lambda}\right) = \frac{1}{\mu}\left(-\frac{1+K\rho^{K+1}-(K+1)\rho^K}{(1-\rho)(1-\rho^{K+1})}\right. \\ \left. +\frac{1+K(\rho^{new})^{K+1}-(K+1)(\rho^{new})^K}{(1-\rho^{new})(1-(\rho^{new})^{K+1})}\right) \end{aligned} \tag{3.19}$$

The value $W^{new} - W$ provides an estimate of the latency incurred due to rule update. After the switch completes processing the packets stored in the neighbor queue, $W^{new} - W$ becomes zero, eventually.

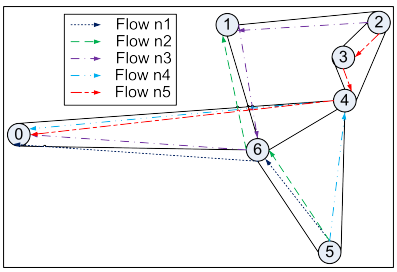
3.3 Performance Evaluation

In this section, we evaluate the performance of CURE in terms of the following metrics: (a) update duration, (b) average rule-space utilization, (c) average packet waiting time, and (d) inconsistent packet count. To evaluate the performance, we performed two experiments. In the first experiment, we measured the update duration and the average rule-space utilization, while varying the number of switches in a leaf-spine topology with $\frac{2N}{3}$ leaf (ingress) switches and $\frac{N}{3}$ spine switches (e.g., [12]). In the second experiment, we simulated three network topologies available at the Internet Topology Zoo [38], namely Sprint, NetRail, and Compuserve. As shown in Figure 3.4, we run five test flows in each

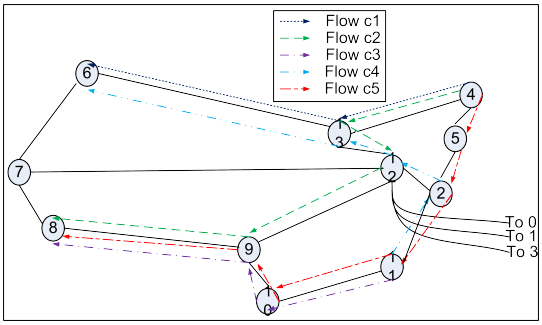
of these topologies to compute the performance concerning the average packet waiting time and inconsistent packet count. Table 3.1 depicts the simulation parameters.



(a) Sprint Topology



(b) NetRail Topology



(c) Compuserve Topology

Figure 3.4: Test Flows in Sprint, NetRail, and Compuserve Topology

Table 3.1: CURE: Simulation Parameters

Parameter	Value
Number of switches in the leaf-spine topology	$6 - 48$
Rule-space size in a switch	8000 flow entries [39]
Upper bound on controller-to-switch delay	4.865 ms [12]
Upper bound on end-to-end network delay	0.262 ms [12]
Upper bound on time interval between dispatch of two consecutive update messages	5.240 ms [12]
Average packet arrival rate per switch	$0.005 - 0.025$ mpps
Average packet service rate per switch	0.030 mpps [40]
Average queue size per switch	0.073 million packets
Flow-table lookup time	33.333 μsec [40]

3.3.1 Result and Discussion

3.3.1.1 Update Duration

The update duration is the time interval between the dispatch of the first update message by the controller and the update completion of the last switch. Garbage collection, i.e., the removal of old rules is included in the update duration, as defined in Definition 6.

Figure 3.5 depicts the update duration for two-phase update [11], timed two-phase update [12], Buffered Update [13], and CURE in a leaf-spine topology. The two-phase update approach (both untimed and timed) updates the spine switches in phase 1, the leaf switches in phase 2, and performs garbage collection after completion of phase 2. From Figure 3.5, we can see that the update duration for the timed two-phase update is 27.919% less than that of the two-phase update. The update duration for CURE is 37.563% less than that of the two-phase update. The update duration is almost similar for timed two-phase update and CURE. Duration for the buffered update is high due to the overhead for the installation of intermediate rules. From Figure 3.5, we yield that the update duration for CURE is short as it does not have a separate garbage collection phase.

3.3.1.2 Average Rule-Space Utilization

We calculate the average rule-space utilization as the percentage of rule-space used during different stages of the update by N switches in the leaf-spine topology.

Figure 3.6 shows the rule-space utilization percentage for two-phase update [11], timed two-phase update [12], Buffered Update [13], and CURE. CURE and the buffered update utilize a similar amount of rule-space, as they both do not store redundant rules. Whereas, rule-space utilization is almost similar for the two-phase update and the timed two-phase update, as they both require to store both old and new rules until the start of the garbage collection phase. The average rule-space requirement for CURE is 29.954% and 30.348% less than that of the two-phase update and timed two-phase

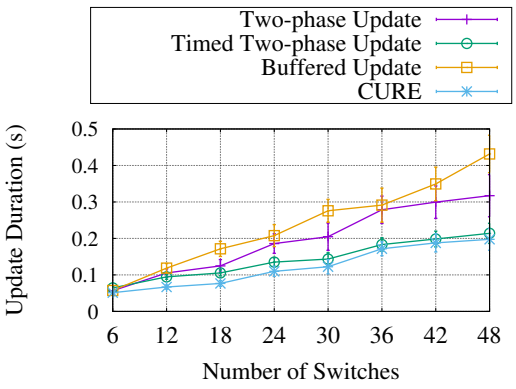


Figure 3.5: CURE: Update Duration

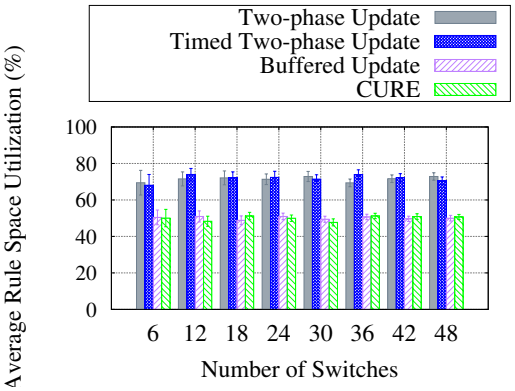


Figure 3.6: CURE: Average Rule-Space Utilization

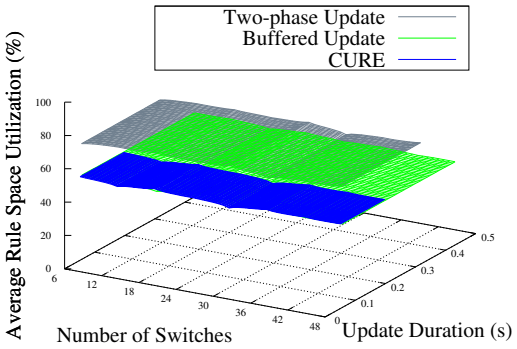
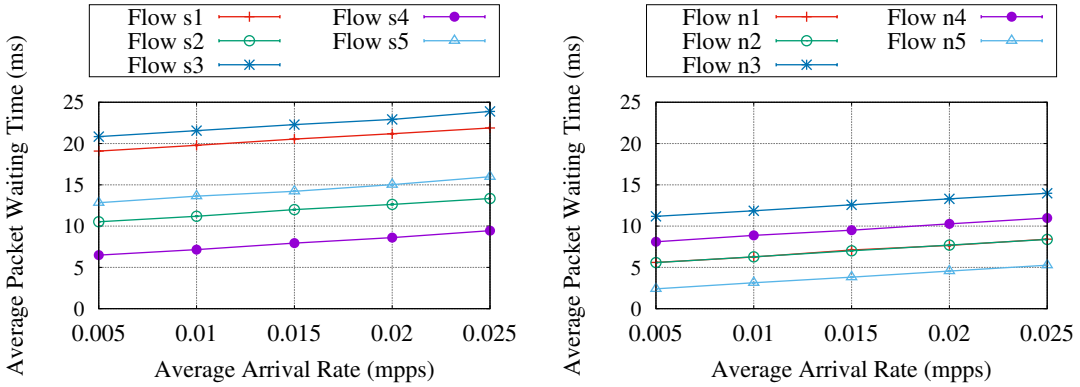


Figure 3.7: CURE: Update Duration and Average Rule-Space Utilization

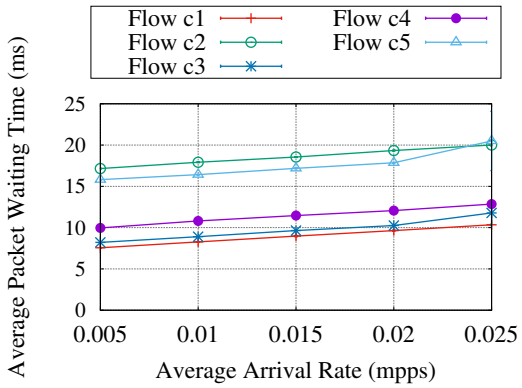
update, respectively. As shown in Figure 3.6, we synthesize that the average rule-space utilization is short in CURE, as the storage of both versions of rules, simultaneously, is not required.

Figure 3.7 portrays the relationship between the number of switches, average rule-space utilization, and update duration for the two-phase update, buffered update, and CURE. We see that CURE outperforms the others, considering both performance metrics — average rule-space utilization and update duration.



(a) Sprint Topology

(b) NetRail Topology

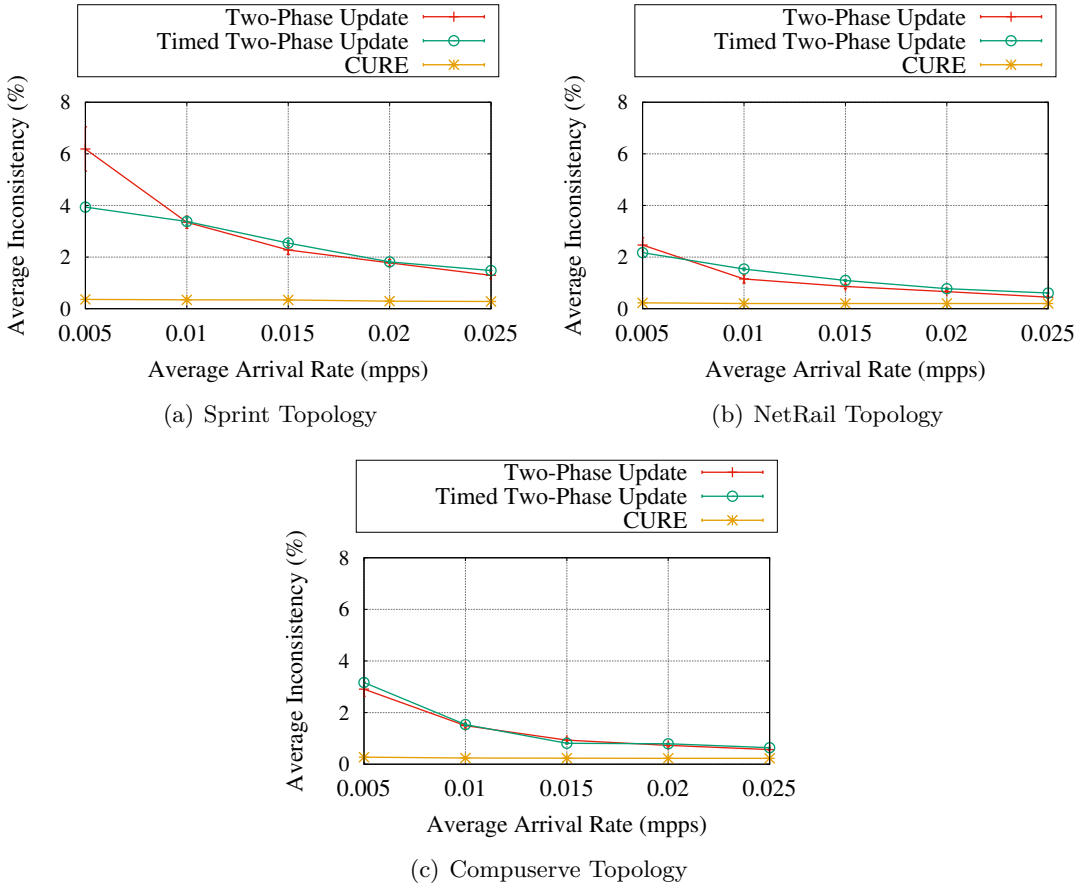


(c) Compuserve Topology

Figure 3.8: CURE: Average Packet Waiting Time

3.3.1.3 Average Packet Waiting Time

For each of the three topologies — Sprint, NetRail, and Compuserve, we simulate five test flows, and calculate the average waiting time for the incoming packets that are either waiting in the switch queues or are in process. Figure 3.4 depicts the topologies, and the test flows. We estimate the delay of each link based on the distance between the corresponding nodes. Similar to Ref. [12], we assume 5 microsecond delay per kilometer.



(a) Sprint Topology

(b) NetRail Topology

(c) Compuserve Topology

Figure 3.9: CURE: Average Packet Inconsistency

3.3.1.4 Inconsistent Packet Count

Figure 3.8 depicts the average packet waiting time for different packet arrival rate for each of the test flows in each of the topologies. The average packet queue size is 0.073 million packets. The average packet waiting time increases with increasing packet arrival rate.

We measure inconsistency as a percentage of inconsistent packets in the system. Inconsistent packets are identified based on Definition 7.

Figure 3.9 compares inconsistency count in CURE with two-phase update and timed two-phase update [12] for different average packet arrival rates. We simulate test flows

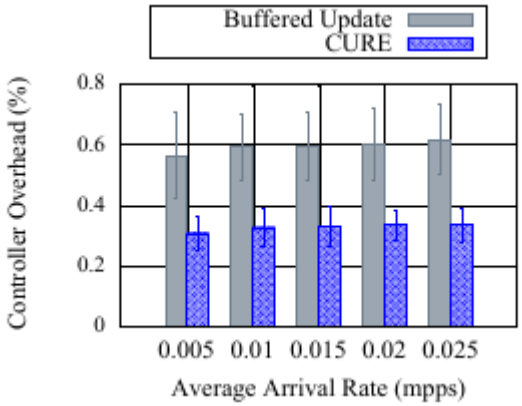


Figure 3.10: CURE: Controller Overhead in Sprint Topology

$s1$, $n1$, and $c1$ in topologies Sprint, NetRail, and Compuserve, respectively. The average queue size per switch is 0.073 million packets. In the two-phase update approaches (both untimed and timed), inconsistency count decreases with increasing packet arrival rate. In two-phase update, the average inconsistency counts for Sprint, NetRail, and Compuserve are 2.976%, 1.118%, and 1.327%, respectively. In the timed two-phase update, the average inconsistency counts for Sprint, NetRail, and Compuserve are 2.629%, 1.237%, and 1.389%, respectively. However, the average inconsistency count for CURE is similar for different packet arrival rates. The average inconsistency count for Sprint, NetRail, and Compuserve is 0.322%, 0.205%, and 0.240%, respectively. Therefore, we yield that in CURE, an initial percentage of incoming packets become inconsistent due to the ongoing network update, and inconsistency count reduces as time elapses after completion of the update.

3.3.1.5 Controller Overhead

Controller overhead is calculated as the percentage of packets sent to the controller during an ongoing update. In Sprint topology, CURE incurs 0.31% controller overhead for packet arrival rate 0.005 mpps. Figure 3.10 depicts that the controller overhead in the buffered update is 82.209% higher than that in CURE. This is because CURE redirects

packets to the controller only in the absence of a neighbor switch having lower priority and free buffer space. Whereas, buffered update keeps redirecting all the affected packets to the controller until the update completes.

3.4 Concluding Remarks

In this chapter, we present a scheme, named CURE, that emphasizes reduction of TCAM usage during SDN update to increase scalability required for handling large-scale data. CURE modifies the update scheme of OpenFlow-enabled SDN and proposes a multilevel queue-based policy for ensuring packet-level consistency. Simulation results show that CURE significantly reduces the update duration and the average rule-space requirement by approximately 38% and 30%, respectively, during the SDN update.

Chapter 4

Data Plane Load Reduction for Flow Migration

In this chapter, we present a scheme for *Data Plane Load Reduction for Traffic Flow Migration (DART)* in SDN. SDN update involves rerouting of multiple traffic flows to accommodate new flows. An unplanned flow migration schedule overloads the data plane by burdening the data links and flooding the rule-space of capacity-constrained SDN switches. The overload of data links and switches blocks the update process, and the network fails to address the QoS demands of the traffic flows especially latency-sensitive flows. Prior approaches migrate flows without considering load reduction of the data plane and QoS demands of the flows. DART prioritizes traffic flows based on QoS demands and aims to avoid link congestion and rule-space overflow during flow migration.

This chapter consists of four sections. The system model of DART is presented in Section 4.1. Section 4.2 describes the proposed scheme. Section 4.3 depicts the experimental results. Finally, Section 4.4 concludes the proposed work and discusses directions for future work.

4.1 System Model

In this section, we discuss the system model and the problem statement for data plane load reduction during flow migration in SDN. The system consists of a set of network elements and a set of traffic flows.

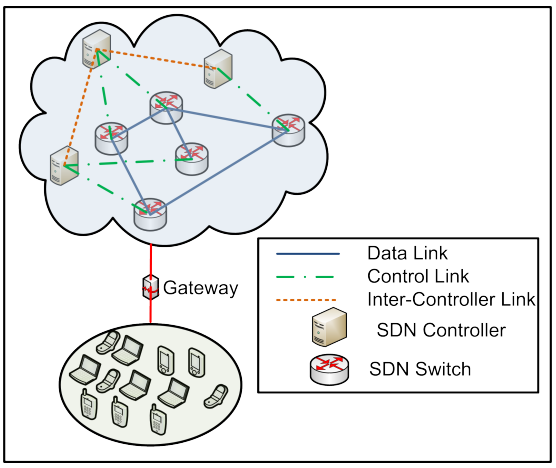


Figure 4.1: DART: SDN Architecture

As shown in Figure 4.1, SDN involves heterogeneous devices that transmit flows to SDN switches via gateways. The rule-space of each switch is managed by a controller. Let C and S denote the set of controllers and the set of switches, respectively. The rule-space usage for switch $s_a \in S$ is represented as R_a. Let R^{max} be the rule-space capacity of a switch. At time t, the bandwidth usage and capacity of the data link between s_a and $s_{a'}$ is denoted by $b_{aa'}(t)$ and $w_{aa'}$, respectively.

4.1.1 Traffic Flow Model

F denotes the set of existing traffic flows in the network. A traffic flow $f_j \in F$ is denoted by a tuple $<src(f_j), dest(f_j), bw(f_j), P(f_j), \alpha(f_j)>$, where $src(f_j)$ denotes the source, $dest(f_j)$ is the destination, $bw(f_j)$ is the bandwidth of f_j, $P(f_j)$ represents the ordered set of switches along the path of f_j, and $0 \leq \alpha(f_j) \leq 1$ signifies the latency-sensitivity index (LSI) for f_j. A high $\alpha(f_j)$ indicates that f_j is highly latency sensitive.

Let $F^{'} \subset F$ denote the set of to-be-migrated traffic flows. A traffic flow f_j is a member of $F^{'}$ if $P(f_j) \neq P^{'}(f_j)$, where $P^{'}(f_j)$ is the new path of f_j after migration. In this work, we assume that the source and destination of a traffic flow $f_j \in F^{'}$ do not change after migration.

Let us consider that the network update procedure for traffic flow migration starts at time t_0. After t_0, a packet is termed *old* if it is handled by a to-be-updated switch. Otherwise, the packet is termed *new*. Therefore, the migration of a traffic flow $f_j \in F$ is consistent when each old packet follows the old path only, and each new packet follows the new path only. We express consistent traffic flow migration as:

$$\Psi(f_j) = \begin{cases} 1 & \text{if the migration of } f_j \text{ is consistent,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.1}$$

Initially, each switch in the new path receives an UPDATE signal from its master controller. Therefore, the set of to-be-updated switches are represented as:

$$\gamma(s_a) = \begin{cases} 1 & \text{if } s_a \in S \text{ received UPDATE signal,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.2}$$

Therefore, the set of to-be-updated switches for a flow $f_j \in F^{'}$ is expressed as:

$$S^{'}(f_j) = \bigcup_{a=1}^{|S|} s_a, \text{ where } s_a \in P^{'}(f_j) \text{ and } \gamma(s_a) = 1 \tag{4.3}$$

The migration of a flow f_i involves the update of each switch $s_a \in S^{'}(f_j)$. For the migration of a flow $f_j \in F^{'}$, the controller sends update packets to all switches in the set $S^{'}(f_j)$. Therefore, the total rule update time required f_j is given by:

$$T_{f_j} = \left(\sum_{k=1}^{|S^{'}(f_j)|} \gamma(s_k) - 1 \right) \Delta + \delta_{sc}, \tag{4.4}$$

where Δ is the maximum time interval between dispatch of two successive update messages from the controller and δ_{sc} is the maximum controller-to-switch delay [12].

Let the flow migration process be divided into multiple update stages, and in each stage, single or multiple flows are migrated, based on the flow migration schedule. Let M be the total number of update stages. To express the flow migration schedule, we define a binary variable as:

$$\chi(f_j, m) = \begin{cases} 1 & \text{if } f_j \in F^{'} \text{ is migrated in stage } m, \\ 0 & \text{otherwise.} \end{cases} \tag{4.5}$$

Definition 8 (Correlated Flow)**.** *Two flows f_i and f_j are correlated if at least one common link exists between the old (new) path of f_i and the new (old) path of f_j.*

Definition 9 (Stage Completion Time)**.** *The completion time of a stage m is defined as:*

$$\mathcal{D}_m = \sum_{f_j \in F^{'}} \chi(f_j, m) T_{f_j} \tag{4.6}$$

Definition 10 (Flow Migration Duration)**.** *The migration duration of each flow which is migrated in stage m is defined as:*

$$\mathcal{D}_m^R = (m^0 - t_0) + \mathcal{D}_m, \tag{4.7}$$

where m^0 is the time when stage m starts.

4.1.2 Problem Formulation

The objective of this work is to minimize the maximum data link bandwidth usage during flow migration. Therefore, we formulate the load-aware flow migration problem (LFMP) as:

$$\underset{\chi}{\text{Minimize}} \quad \sum_{s_a, s_{a'} \in S} b_{aa'}(t), t \in [m^0, m^0 + \mathcal{D}_m], m \leq M \tag{4.8}$$

subject to

$$\Psi(f_j) = 1, \forall f_j \in F^{'}, \tag{4.9}$$

$$R_a \leq R^{max}, \quad \forall s_a \in S^{'}(f_j), \forall f_j \in F^{'} \tag{4.10}$$

$$\mathcal{D}_m^R \leq T_j^{max}, \chi(f_j, m) = 1, m \leq M, \forall f_j \in F^{'}, \tag{4.11}$$

$$\sum_{m=1}^{M} \chi(f_j, m) = 1, \forall f_j \in F^{'}, \tag{4.12}$$

$$b_{aa'} \leq w_{aa'}, \forall s_a, s_{a'} \in S, a \neq a^{'} \tag{4.13}$$

Equation (4.9) expresses the consistency constraint for all traffic flows in the network during update. Equation (4.10) represents the rule-space capacity constraint of switches. Equation (4.11) ensures that the flow migration duration for each traffic flow f_j does not exceed the maximum allowable delay T_j^{max} of the flow. Equation (4.12) ensures that each flow is migrated only once. Equation (4.13) denotes the link capacity constraint.

Theorem 1. *The load-aware flow migration problem (LFMP) is NP-hard.*

Proof. To prove the NP-hardness of LFMP, we reduce the well-known $0-1$ knapsack problem [41] to LFMP. The $0-1$ knapsack problem, which is an NP-hard problem, involves a set of items so that each item has a weight and a value. Given a knapsack with a fixed capacity, the goal is to maximize the total value of items included in the knapsack. Moreover, the decision for including an item in a knapsack is binary, i.e., an item can be added to the knapsack as a whole or not added at all.

We construct an instance I of the LFMP for an update stage m. We reduce an instance $I^{'}$ of the $0-1$ knapsack problem to I. In this case, each item in $I^{'}$ refers to the flows $f_j \in F^{'}$. The weight and value of each item correspond to bandwidth $bw(f_j)$ and LSI $\alpha(f_j)$, respectively. The capacity of the knapsack is mapped to the link capacity

$b_{ab}, \forall s_a, s_{a'} \in S$. In I, the value of the decision variable $\chi(f_j, m)$ is restricted to 0 or 1, depending on whether f_j is migrated in stage m or not. The goal of LFMP is to find a feasible solution that includes the maximum number of flows with high LSI in each update stage without violating the link capacity constraint for any flow. Therefore, the optimal solution to the instance of the $0-1$ knapsack problem $I^{'}$ is also the optimal solution of the instance of LFMP I. Hence, the LFMP is also NP-hard. □

As the optimization problem in Equation (4.8) is NP-hard, we propose a heuristic approach for solving the problem.

4.2 DART: The proposed scheme

In this section, we discuss the proposed scheme, DART, which has three modules — 1) generation of QoS-aware migration schedule, 2) generation of feasible migration schedule, and 3) rule-space management. The QoS-aware migration scheduling module analyzes the QoS demand of each migrating flow and generates an initial flow migration schedule as defined in Equation 4.5. The feasible migration scheduling module evaluates whether the initial flow migration schedule is feasible or not and updates the schedule to avoid link congestion. The rule-space management module checks the available rule-space in each to-be-updated switch and frees up rule-space as per the requirement.

4.2.1 Generation of QoS-Aware Migration Schedule

We formulate a coalition graph game to form groups of flows so that each group is migrated in each update stage. In this game, $F^{'}$ is the set of players. Each coalition $A_k \in F^{'}$ denotes the set of traffic flows $\{f_1, f_2, \ldots, f_{|A_k|}\}$ which are migrated simultaneously. Within a coalition, the traffic flow having the highest LSI is termed as the coalition-head. Therefore, a coalition-head has $|A_k| - 1$ children nodes, which are termed as coalition members. To form the coalitions, the proposed game constructs a coalition

graph $G = (F^{'}, E)$, where E is the set of edges representing the correlation between flows as defined in Definition 8. Therefore, there exists an edge between $f_i \in F^{'}$ and $f_j \in F^{'}$ if f_i and f_j are correlated flows.

4.2.1.1 Suitability of the Coalition Graph Game for QoS-Aware Migration Scheduling

In SDN, multiple flows, which share the data links, are migrated together. Therefore, for migration, each flow behaves cooperatively and decides its optimum strategy to satisfy QoS demand and achieve Pareto optimal distribution of link capacity. Moreover, the correlation between flows serves as a critical aspect for forming the groups as the update of one flow may cause link congestion in the flow-path of a correlated flow. Hence, a coalition graph game approach is the most appropriate approach for the formation of a QoS-aware migration schedule, where migrating flows form cooperative groups, which are migrated simultaneously for optimal utilization of the available link capacity.

Definition 11 (Coalition Structure)**.** *A coalition structure is a set of coalitions defined as:*

$$V_A = \{A_1, A_2, \ldots, A_Q\}, where \bigcup_{k=1}^{Q} A_k = F^{'}, A_k \cap A_l = \phi, \forall k \neq l \tag{4.14}$$

4.2.1.2 Utility Function of a Coalition

The controllers try to maximize the cumulative payoff obtained from the utility functions of the coalitions. Let $U(A_k, V_A)$ denote the utility value of a coalition $A_k \in V_A$ and $u_j(.)$ denote the utility value of a player $f_j \in A_k$. The marginal utility of each traffic flow f_j increases with decrease in the rule update time of the flow. Mathematically,

$$\frac{\partial u_j(.)}{\partial T_{f_j}} < 0 \tag{4.15}$$

The utility function $u_j(.)$ varies linearly with the LSI, and the number of correlated flows in the coalition (N_j) so that a high number of flows are migrated at a time depending on their traffic characteristics. Therefore, we get:

$$\frac{\partial u_j(.)}{\partial \alpha(f_j)} > 0, \text{ and } \frac{\partial u_j(.)}{\partial N_j} > 0 \tag{4.16}$$

Therefore, we define the utility function of a flow f_j as:

$$u_j(.) = N_j \left(\alpha(f_j) - \frac{T_{f_j}}{T_j^{max}} \right) \tag{4.17}$$

Hence, the utility function $U(A_k, V_A)$ is formulated as:

$$U(A_k, V_A) = \begin{cases} \sum\limits_{f_j \in A_k} u_j(.) & \text{if } |A_k| > 1, \\ 0 & \text{otherwise.} \end{cases} \tag{4.18}$$

The total utility of all the coalitions in a coalition structure V_A is given by:

$$U(V_A) = \sum_{k=1}^{M} U(A_k, V_A) \tag{4.19}$$

4.2.1.3 Coalition Graph Formation

The to-be-updated traffic flows, which are the players of the coalition graph game, form the coalition graph based on the utility function defined in Equation (4.19). We consider that the proposed coalition graph game is hedonic, which implies that a player has a preference for the choice of the coalition. The preference relation is defined as:

Definition 12 (Preference Relation)**.** *The relation $V_A \succ_{F''} V_B$ denotes that the way V_A partitions F'' is preferred to the way V_B partitions F'', where $F'' \subseteq F'$ is a set of players.*

In this work, we consider Pareto order [42] as the basis for the preference relation $\succ$.

According to Pareto order, a coalition structure V_A is preferred over another coalition structure V_B if the change of coalition structure from V_B to V_A improves utility for at least one player without decreasing the utility of any other player. Let $u_j(A)$ denote the utility of player f_j in a coalition $A_k \in V_A$. Mathematically,

$$V_A \succ_{F''} V_B \Leftrightarrow \{u_j(A) \geq u_j(B)\}, \forall f_j \in F^{''}, F^{''} = \bigcup_{k=1}^{|V_A \cup V_B|} A_k, \forall A_k \in V_A \cup V_B, \quad (4.20)$$

with at least one player f_x having the strict inequality $u_x(A) > u_x(B)$.

The coalitions are updated periodically based on merge and split rules as follows:

Definition 13 (Merge Rule)**.** *Merge any set of coalitions* $\{A_1, A_2, \ldots, A_k\}$ *where* $\{\bigcup_{l=1}^{k} A_l\} \succ_{F''} \{A_1, A_2, \ldots, A_k\}$, $F^{''} = \bigcup_{l=1}^{k} A_i$. *Therefore,* $\{A_1, A_2, \ldots, A_k\} \rightarrow \bigcup_{l=1}^{k} A_l$.

Definition 14 (Split Rule)**.** *Split any set of coalitions* $\bigcup_{i=1}^{k} A_l$ *where* $\{A_1, A_2, \ldots, A_k\} \succ_{F''} \{\bigcup_{l=1}^{k} A_l\}$, $F^{''} = \bigcup_{l=1}^{k} A_l$. *Therefore,* $\bigcup_{l=1}^{k} A_l \rightarrow \{A_1, A_2, \ldots, A_k\}$.

Therefore, multiple coalitions merge into a large coalition if merging is preferable to the set of players according to Equation (4.20). Similarly, one large coalition splits into multiple coalitions if splitting is beneficial to the set of players. To restrict the search space for the merge operation, we consider a greedy approach to decide the potential candidates for the attempt of merging. In this approach, a coalition A_l attempts to merge with coalition A_k only if there exists at least one edge $e_{ij} \in E$ between $f_i \in A_l$ and $f_j \in A_k$. This constraint ensures that the merged utility is always positive.

Definition 15 (Stable Coalition)**.** *A coalition* $A_k \in V_A$ *is stable if*

1. *no player* f_j *can improve its utility by leaving its coalition* A_k *and acting individually.*

2. *no other coalition* $A_l \in V_A$ *can improve its utility by joining* A_k.

Definition 16 (Stable Coalition Structure)**.** *A coalition structure V_A is stable if $A_k \in V_A, \forall k \in [1, Q]$ is stable.*

Algorithm 4.1 describes the generation of the initial migration schedule. Initially, each traffic flow forms an individual coalition. The Initial Migration Scheduling Algorithm (IMSA) sorts the coalitions in descending order based on the LSI of the coalition-heads. In each iteration, each coalition A_k forms a potential candidate list L_k. The list L_k is sorted based on the LSI of the coalition-heads. A_k attempts to merge with the first coalition in L_k. If the merge attempt is successful, both coalitions are merged. Otherwise, A_k attempts to merge with the next coalition in the list. This merge process can be performed distributively, where each coalition makes a greedy attempt to merge with the coalitions in its potential candidate list. After completing greedy merge attempts for all coalitions, the split operation is performed, if any split is possible. The merge and split process is repeated until V_A is stable. The initial migration schedule $\chi^{'}$ is formed by scheduling the flows of each coalition from v_A in each update stage.

The time complexity of IMSA depends on the number of merge and split attempts. For $|F^{'}|$ flows, the maximum number of possible coalitions is $|F^{'}|$. In the worst case, each coalition attempts to merge with all the others. In this case, the fist coalition makes $|F^{'}| - 1$ merge attempts, the second coalition requires $|F^{'}| - 2$ merge attempts, and so on. Therefore, the maximum number of merge attempts is $\frac{|F^{'}|(|F^{'}|-1)}{2}$. However, in a practical scenario, the number of merge attempts is significantly less as each coalition attempts to merge only with coalitions in the potential candidate list. In the worst case, the split operation of a coalition involves finding all partitions of the coalition. The total number of partitions is given by the Bell number [43], which grows exponentially with the number of players in the coalition. However, in a practical scenario, once a coalition splits based on the Pareto order as stated in Equation (4.20), no further split is attempted. Therefore, the total number of split attempts is significantly less in practice.

Theorem 2. *IMSA converges to a stable coalition structure.*

Algorithm 4.1: DART: Initial Migration Scheduling Algorithm

Input : $F^{'}$: Set of migrating flows
Output: $\chi^{'}$: Initial migration schedule

Set $E \leftarrow E \cup \{e_{ij}\}$ if $f_i \in F^{'}$ and $f_j \in F^{'}$ are correlated flows
$A_k \leftarrow A_k \cup \{f_k\}, V_A \leftarrow V_A \cup \{A_k\}, \forall f_k \in F^{'}$
while V_A *is not stable* **do**
    Sort V_A in descending order of the LSI of the coalition-heads
    forall $A_k \in V_A$ **do**
        Form potential candidate list L_k for merge attempt using E
        Sort the coalitions in L_k in descending order of the LSI of the coalition-heads
        if *merge attempt successful for* $A_l \in L_k$ **then**
            Merge A_k and A_l using Definition 28
            Update V_A
        end
        Attempt merge with $A_{l+1} \in L_k$
    end
    Split coalitions in V_A using Definition 29
    Update V_A
end
Set $\chi^{'}(f_j, k) = 1, \forall f_j \in A_k, \forall A_k \in V_A$
return $\chi^{'}$

Proof. Initially, each player forms an individual coalition having zero utility. Therefore, a player has the lowest utility value when it acts individually. In subsequent iterations, each player tries to increase its utility via the merge and split operations. This process continues if at least one player is capable of improving its utility by joining another coalition. Hence, the termination of the merge and split process implies that no coalition can improve its utility by joining another coalition. Therefore, IMSA generates a stable coalition structure, V_A. □

4.2.2 Generation of Feasible Migration Schedule

The coalitions from the stable coalition structure V_A are selected one-by-one for consistent flow migration, and only one coalition is migrated in each update stage. However, the migration of a flow may trigger congestion in one or multiple links. This is because

those links have to-be-migrated flows which are scheduled to be migrated in a later stage. Therefore, prospective link congestion makes a flow migration schedule infeasible. Therefore, we propose a greedy heuristic algorithm to analyze the feasibility of the initial migration schedule and prepare the final migration schedule that reduces the data link load. Algorithm 4.2 shows the steps for the generation of a feasible flow migration schedule.

Algorithm 4.2: DART: Feasible Migration Scheduling Algorithm

```
   Inputs : χ', V_A
   Output: χ
 1 while m ≠ |V_A| do
 2 |   forall f_j ∈ F' do
 3 |   |   if χ'(f_j, m) = 1 and migration of f_j violates link capacity constraint
   |   |    then
 4 |   |   |   Set χ(f_j, m + 1) = 1 and update V_A
 5 |   |   end
 6 |   |
 7 |   |   Set χ(f_j, m) = 1
 8 |   end
 9 end
10 return χ
```

Each iteration of the Feasible Migration Scheduling Algorithm (FMSA) checks the initial migration schedule and determines whether the migration of the flows in a stage is feasible in terms of the link capacity constraint. If any flow violates the link capacity constraint, FMSA moves the infeasible flow to the next update stage. As we migrate the flows is each update stage together, the possibility of link congestion reduces for some links, and some infeasible flows become feasible. Therefore, FMSA takes a greedy approach to allocate the infeasible flows to the nearest update stage. FMSA runs in $O(|F^{'}|)$ time as each flow in an update stage checks for link capacity violation based on the bandwidth usage data of the links, which is available to the controller.

4.2.3 Rule-Space Management

FMSA generates the final migration schedule, which reduces link congestion during flow migration. However, another part of the data plane load is rule-space usage. SDN switches have limited rule-space, and the overflow of rule-space makes the migration process inconsistent and incomplete. However, in each stage, we update the switches based on the approach proposed in our earlier work, CURE [44]. This approach deletes old rules immediately after installing new flow-rules. Therefore, the switches, which are part of both old and new paths of a flow, require no additional rule-space. However, the switches, which only belong to the new path, require the installation of additional flow-rules to define the new path. So, we propose a heuristic algorithm to ensure that these switches have enough capacity to address the additional rule-space requirement.

The proposed rule-space management process requires the deletion of unimportant flow-rules from the switches, which have low free rule-space. To select the rules that are no longer required, we estimate the popularity of the flow-rules store in the rule-space of a switch. We sort the flow-rules of the corresponding switch in descending order of the received packet count. For a switch s_a, the rule popularity is denoted by $\Theta = \{\theta_1, \theta_2, \theta_3, \ldots, \theta_{R_a}\}$, where θ_k is the probability that a flow matches with the k^{th} rule. We estimate the rule popularity based on Zipf distribution [36], which is expressed as:

$$\theta_k = \frac{\frac{1}{k^{\epsilon}}}{\sum_{i=1}^{R_a} \frac{1}{i^{\epsilon}}}, \tag{4.21}$$

where ϵ is the skewness of the rule popularity. The value $\epsilon = 0$ denotes uniform popularity distribution and a larger ϵ signifies more uneven rule popularity.

Algorithm 4.3 shows the steps of the rule-space management process based on the feasible flow migration schedule.

The Rule-space Management Algorithm (RSMA) identifies the set of switches S'',

Algorithm 4.3: DART: Rule-Space Management Algorithm

Inputs : χ, λ
Output: $S^{''}$

while $m \neq |V_A|$ **do**
  forall $f_j \in F^{'}$ **do**
    if $\chi(f_j, m) = 1$ **then**
      $S^{''} \leftarrow S^{''} \cup \left(P^{'}(f_j) \setminus P(f_j)\right)$
      $addRules(s_a) \leftarrow addRules(s_a) + 1, \forall s_a \in \left(P^{'}(f_j) \setminus P(f_j)\right)$
    end
  end
end
forall $s_a \in S^{''}$ **do**
  if $R_a \geq \lambda$ **then**
    Delete $R_a - addRules(s_a)$ least popular rules using Equation 4.21
  end
end
return $S^{''}$

which require the installation of additional flow-rules. Additionally, RSMA estimates the rule-space requirement for each switch in $S^{''}$. To identify the overloaded switches, RSMA checks if the rule-space usage for any switch in $S^{''}$ exceeds a predefined rule-space threshold λ. Finally, RSMA frees the required rule-space in the overloaded switches by deleting the required number of rules starting with the least popular rule. The time complexity of RSMA is composed of two parts — the time complexity for the formation of $S^{''}$ and the time complexity for the reduction of rule-space usage in the overloaded switches. Each flow is visited to identify the set of switches for inclusion in $S^{''}$. This operation is completed in $O(|F^{'}|)$ time. The rule-space reduction process takes $O(|S|)$ time because, in the worst case, the reduction must be performed for all switches. Therefore, RSMA run in $O(|F^{'}| + |S|)$ time.

4.2.4 Consistent Flow Migration

The set of to-be-updated switches for update stage m, is expressed as:

$$S_m = \bigcup_{j=1}^{|F^{'}|} S^{'}(f_j), \text{ where } \chi(f_j, m) = 1 \tag{4.22}$$

For consistent flow migration, in each update stage m, DART processes the old packets and starts queuing the new packets for the switches in S_m. This step ensures packet-level consistency. After processing all the old packets, new rules are installed, and old rules are deleted. This step addresses the rule-space constraint of the switches as only a single version of a flow-rule is stored at a time. After the modification of all the required rules, DART processes the queued packets [44].

Theorem 3. *Flow migration in DART is blackhole free.*

Proof. Let $f_j \in F^{'}$ be a flow that is scheduled to be migrated in stage m. In stage m, new flow-rules are installed in all switches in $S^{'}(f_j)$. However, the old packets are processed by the old flow-rules before updating the first switch in stage m. As the update of the first switch in stage m starts, the new packets are queued until all switches in stage m complete update. Once stage m completes update, the queued packets are processed by the new flow-rules. Therefore, all packets that enter a switch belonging to the old path $P(f_j)$ or to the new path $P^{'}(f_j)$ is equal to the packets that leave the switch. Since, no packet of a flow f_j is dropped, the flow migration process in DART is blackhole free.

□

Theorem 4. *Flow migration in DART is loop free.*

Proof. All the old packets of a flow $f_j \in F^{'}$ are processed by old flow-rules entirely. New flow-rules are installed to all switches in $S^{'}(f_j)$ before processing the new packets. Therefore, each packet in f_j either follows the old path $P(f_j)$ or the new path $P^{'}(f_j)$. Since, no packet is processed by incorrect flow-rules, the flow migration in DART is loop free.

□

4.3 Performance Evaluation

4.3.1 Simulation Settings

We evaluate DART's performance by performing simulations on the Abilene topology, which has 12 switches and 30 directed links [38]. We use the Abilene topology because it is a small-scale topology, where the number of correlated flows for each flow is high. For simulation, we use the Abilene dataset [45], which provides the OSPF weights and the maximum capacity of each link. Based on the parameters available in the Abilene dataset, we randomly generate traffic flows to perform the simulations for different traffic volumes. Table 4.1 represents the simulation parameters. For the simulation, we consider that 80% flows are latency-sensitive with LSI between 0.9 to 1.

Table 4.1: DART: Simulation Parameters

Parameter	Value
Topology	Abilene [38]
Number of traffic flows	$100 - 400$
Bandwidth of a traffic flow	$0.0001 - 0.39$ Gbps [45]
Maximum link capacity	9.92 Gbps [45]
Number of switches	12 [45]
Number of links	30 [45]
Maximum controller-to-switch delay (δ_{sc})	4.87 ms [44]
Maximum time interval between dispatch of two successive update messages from the controller (Δ)	5.24 ms [44]
Maximum allowable delay	$1 - 1000$ ms [46]
Rule popularity skewness (ϵ)	0.56
Rule-space capacity (R^{max})	500 flow-rules
Rule-space threshold (λ)	250 flow-rules

4.3.2 Benchmark schemes

We compare the performance of DART with three benchmark schemes — two-phase update [11], flow migration scheme proposed by Basta *et al.* [47], and the Greedy approach. The two-phase update is not incremental and schedules all traffic flows together for mi-

gration. The two-phase update scheme updates the ingress switches after updating the internal switches. Basta *et al.* [47] update switches according to the shortest common supersequence formed from the ordered sets of switches denoting the new paths of the migrating flows. In the Greedy approach, flows are migrated in descending order of the LSI value, and the correlated flows are migrated together. On the other hand, DART considers flow-specific QoS requirements while preparing the migration schedule and migrates the flows consistently. We select the two-phase update as one of the benchmark schemes to show the effectiveness of incremental flow migration. We select the flow migration scheme proposed by Basta *et al.* [47] to show the importance of considering individual flow paths as compared to an integrated flow path in order to reduce data link congestion. In addition, we select the Greedy approach as a benchmark scheme to show that the formation of groups for flow migration should not depend only on the LSI value.

4.3.3 Performance Metrics

We consider the following metrics to analyze the performance of DART:

- **Flow migration duration**: The migration duration of a traffic flow is defined in Definition (10). This metric quantifies the time required for a flow to change its path from old to new.
- **Peak Load of the data links** : This metric shows the highest load of the data links during flow migration. A high data link load signifies that the possibility of link congestion is high.
- **Rule-space usage for flow migration**: This metric measures the rule-space required for the flow migration process. This metric is important because of the rule-space capacity limitation of SDN switches.
- **QoS violated flows**: QoS violated flows are traffic flows that have a migration

duration greater than the maximum allowable duration. This metric shows the QoS-awareness of DART.

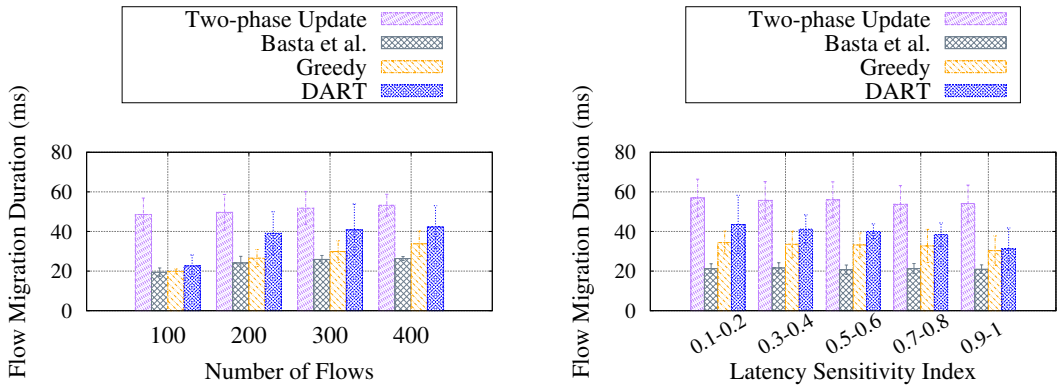
4.3.4 Result and Discussion

4.3.4.1 Flow Migration Duration

We estimate the average migration duration by varying the number of flows. From Figure 4.2(a), we observe that the average flow migration duration for DART is 28.82% less than that of the two-phase update. This is because DART migrates the flows incrementally, resulting in reduced waiting time for each traffic flow. However, the average flow migration duration for DART is higher than the approach proposed by Basta *et al.* [47] and the Greedy approach because DART migrates flows in multiple stages, where controller-switch communications are initiated for each switch in each update stage. Figure 4.2(b) depicts the effects of LSI on the average flow migration duration for 400 flows. For this experiment, we form 5 groups, each having 80 flows. The LSI of the flows in the group 1, group 2, group 3, group 4, and group 5 are $[0.1, 0.2]$, $[0.3, 0.4]$, $[0.5, 0.6]$, $[0.7, 0.8]$, and $[0.9, 1]$, respectively. We observe that the average flow migration duration for both DART and Greedy decreases as LSI of migrating traffic flows increases. In particular, for DART, the average migration duration of the flows in the group 5 is 28.22% less than the flows in the group 1. However, the change of LSI does not affect the migration duration of benchmark schemes. Therefore, it is evident that DART prioritizes latency-sensitive flows and schedules their migration earlier to satisfy the QoS demands.

4.3.4.2 Peak Load of the Data Links

We analyze the peak load of data links as it is the primary contributor to the data plane load. Figure 4.3 sketches the peak data link load with varying number of flows. From the simulation result, we observe that the peak data link load is 13.75%, 13.78%, and



(a) Effect of Traffic Load

(b) Effect of LSI

Figure 4.2: DART: Flow Migration Duration

9.19% less than the two-phase update, the approach proposed by Basta *et al.* [47], and Greedy approach, respectively. Moreover, we observe that the performance of DART improves as the traffic load increases. This is due to the fact that high traffic load increases the possibility of link congestion, and using FMSA DART reduces the data link load. Therefore, for high traffic load, DART proves to be a reliable scheme that reduces data loss caused by link congestion.

4.3.4.3 Rule-Space Usage for Flow Migration

For DART and the benchmark schemes, we estimate the additional rule-space requirement because of the flow migration process. Figure 4.4 shows the average rule-space usage with varying traffic load. From the simulation result, we observe that DART uses 67.93%, 57.74%, and 63.07% less rule-space as compared to the two-phase update, the approach proposed by Basta *et al.* [47], and Greedy approach, respectively. This is because RSMA deletes less popular rules to accommodate new flow-rules, and DART performs consistent flow migration where old flow-rules are not stored redundantly.

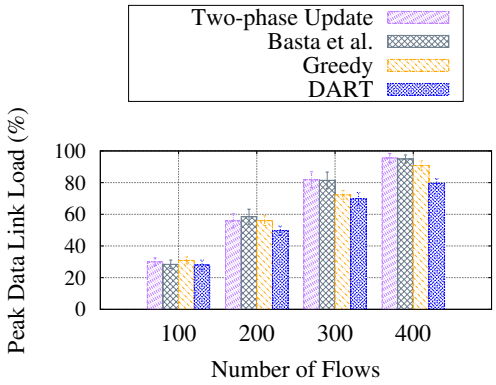


Figure 4.3: DART: Peak Data Link Load

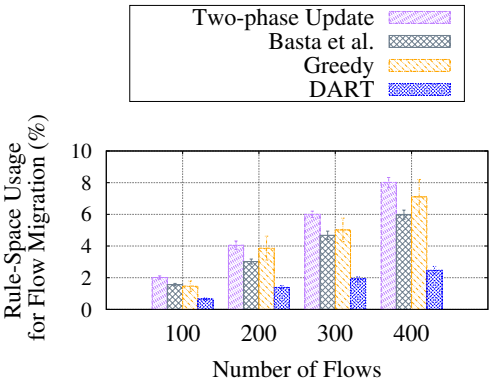


Figure 4.4: DART: Rule-Space Usage for Flow Migration

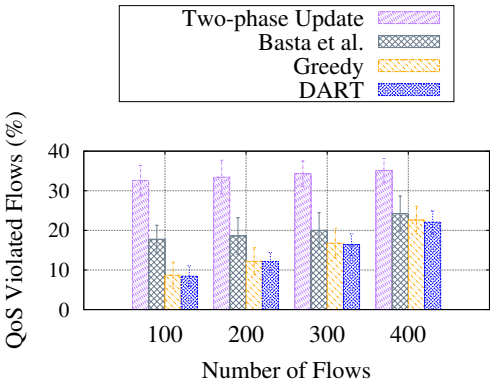
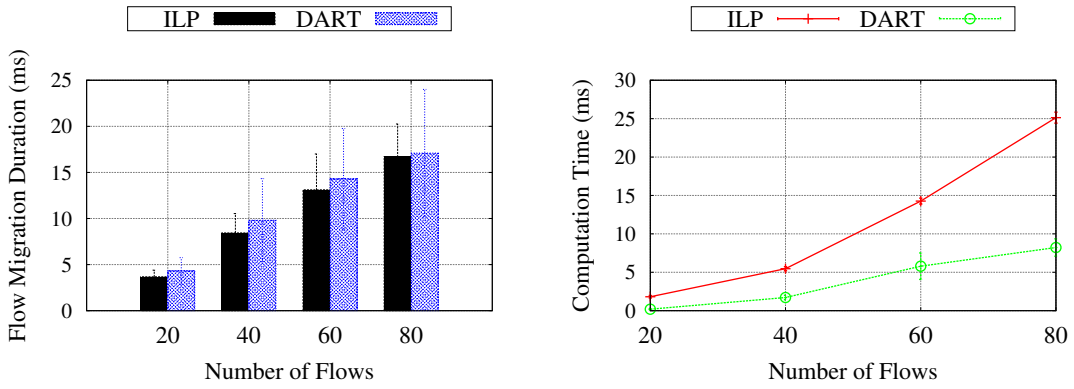


Figure 4.5: DART: QoS Violated Flows

4.3.4.4 QoS Violated Flows

We analyze the amount of QoS violation considering heterogeneous traffic where each flow f_j has different QoS requirement in terms of the maximum allowable delay T_j^{max}. From Figure 4.5, we observe that the number of QoS violated flows in DART is 56.36% less than the same using the two-phase update, 26.65% less than the approach proposed by Basta *et al.* [47], and 1.92% less than the same using Greedy approach. This is due to the fact that DART migrates the traffic flows in order of the LSI values so that each flow fulfils the specific QoS demand. Additionally, DART considers the link capacity constraint and schedules feasible flows together.

From the above analysis, it is evident that the proposed scheme, DART, significantly



(a) Flow Migration Duration

(b) Computation Time

Figure 4.6: DART: Comparison between ILP Solution and DART

reduces the peak load of the data links and additional rule-space usage for flow migration with acceptable flow migration duration. Additionally, it is noteworthy to observe that DART achieves remarkable performance in terms of addressing QoS demands of heterogeneous flows considering heterogeneous traffic as an essential parameter of realistic networks.

We solve the ILP formulated in Equation (4.8) using Gurobi Optimizer [48]. Figure 4.6 shows the comparison between the ILP solution and the proposed heuristic approach, DART. We observe that DART achieves performance similar to the ILP solution while having low computation time.

4.4 Concluding Remarks

In this chapter, we present a scheme, named DART, that migrates traffic flows in different update stages. Each update stage is formed based on the QoS demand of the flows, and bandwidth usage of the links. DART also addresses the rule-space capacity constraint so that no switch reaches its rule-space capacity limit due to flow migration. Simulation results show that DART reduces the additional rule-space usage by 67.93%, and the number of QoS violated flows by 56.36% compared to the two-phase update.

Chapter 5

Rule-Space Management

In this chapter, we present a scheme for *Tensor-Based Rule-Space Management (TERM)* in SDN. The limited storage capacity of switches is a crucial challenge in SDN as the switches use TCAMs having deficient capacity. Low rule storage capacity eventually leads to a high number of Packet-In messages and control plane overloading. TERM addresses rule-space capacity constraint by aggregating flow-rules using tensor decomposition.

This chapter consists of four sections. The system model of TERM is presented in Section 5.1. Section 5.2 describes the proposed scheme. Section 5.3 depicts the experimental results. Finally, Section 5.4 concludes the proposed work and discusses directions for future work.

5.1 System Model

Figure 5.1 depicts the network architecture considered for TERM. The set of switches in the data plane is denoted as $S = \{s_1, s_2, \ldots, s_D\}$. In the control plane, there exist multiple sub-controllers connected with a controller c. The set of sub-controllers is denoted as $C^{sub} = \{c_1^{sub}, c_2^{sub}, \ldots, c_M^{sub}\}$. The sub-controllers are placed using existing controller placement algorithms [26]. All the sub-controllers are connected to c. Each

switch s_j is connected to a sub-controller. Therefore, the assignment between switches and sub-controllers is defined as a $D \times M$ binary matrix L. Each element of L is expressed as:

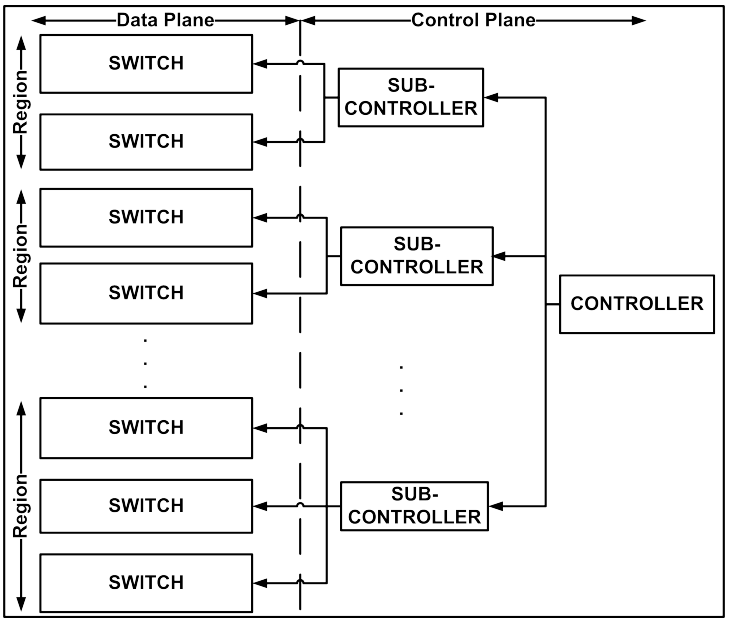


Figure 5.1: TERM: Network Architecture

$$l_{ij} = \begin{cases} 1, & \text{if } s_i \text{ is connected to } c_j^{sub}. \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

Definition 17 (Region)**.** *A region r_j is defined as:*

$$r_j = \bigcup_i s_i, \forall l_{ij} = 1 \tag{5.2}$$

The set of rules in switch s_i at time t is denoted as:

$$R^i(t) = R_c^i(t) \cup R_a^i(t) \cup R_u^i(t), \tag{5.3}$$

where $R_c^i(t)$ is the set of cached rules, $R_a^i(t)$ is the set of aggregated rules, and $R_u^i(t)$

denotes the set of uncompressed rules in switch s_i at time t.

A switch s_i generates $p_i(t)$ number of Packet-In messages at time t. Packet-In messages are generated whenever incoming packets fail to match with any of the flow-rules in $R^i(t)$.

The objective of this work is to minimize the number of Packet-In messages by maximizing the total number of rules stored in each switch. Mathematically,

$$min \sum_{i=1}^{|S|} p_i(t) \tag{5.4}$$

subject to

$$|R_c^i(t)| < N_i, \forall s_i \in S \tag{5.5}$$

$$|R_u^i(t)| < N_i, \forall s_i \in S, \tag{5.6}$$

where N_i denotes that the TCAM in a SDN switch s_i is capable of storing N_i entries. Equations (5.5) and (5.6) express that the number of cached rules and the number of uncompressed rules are less than the storage capacity of the TCAM.

5.2 TERM: The Proposed Scheme

In this section, we describe the proposed scheme, TERM, which includes three modules — rule aggregation, rule reconstruction, and rule caching. Rule aggregation and rule reconstruction procedures of a region r_j are performed by c_j^{sub}. The rule aggregation module compresses the flow-rules in each switch with a tensor-based approach to increase the available capacity of the flow-tables. The rule reconstruction module reconstructs the compressed rules in a switch, whenever an incoming packet fails to match the uncompressed rules. Additionally, each switch has a rule caching module which caches the

most frequently used rules. This avoids the reconstruction of rules every time a packet reaches a switch.

Therefore, for an incoming packet, a switch first checks for a rule match in the cached rules, and then the uncompressed flow-rules in the flow-tables. If no match is found, it informs the sub-controller that the reconstruction of compressed rules is required. The sub-controller reconstructs the compressed rules and checks for a possible rule match. If a match is found in multiple rules, the higher priority rule is selected. If no match is found even after checking the compressed rules, the packet is redirected to c, which generates the new rule as per the existing OpenFlow policy [34].

5.2.1 Rule Aggregation

The rule aggregation module includes three sub-modules — rule restructuring, tensorization, and reduction.

5.2.1.1 Rule Restructuring

Rule restructuring converts the ternary string of each rule into an integer value. We consider a 4-bit ternary value for each match field and 4-bit binary value for the action field. Each ternary string of length 2 is transformed into an integer digit based on the transformation rules presented in Table 5.1.

Table 5.1: Integer representation of ternary strings

Ternary String	***Integer Representation***
**	1
*0	2
*1	3
0*	4
1*	5
00	6
01	7
10	8
11	9

Example 1. *Consider a ternary string of two match fields and one action value* $\{*11*, 10*0, 1101\}$. *Therefore, after transformation, the resulting integer string will be* $\{35, 82, 97\}$.

5.2.1.2 Tensorization

In this work, we use tensor to formalize the flow-tables in SDN switches. We transform each modified rule-set into a three-order tensor, as shown below:

$$T \in R^{1 \times N_f \times N_t}, \tag{5.7}$$

where N_f denotes the number of fields in a TCAM entry, which includes the priority value, match fields, and action value. N_f depends on the OpenFlow protocol version. N_t denotes the total number of uncompressed rules in the switch.

5.2.1.3 Reduction

Algorithm 5.1 transforms T to a compressed tensor $\mathbb{C} \in R^{1 \times N_f \times N_r}$, where $N_r < N_t$. N_r is termed as the *reduction factor* (RF). The value of RF at time t is selected as:

$$RF(t) = N_r = N_f + \Big\lfloor \frac{(Q_{max} - Q_{current})}{Q_{max}} \times 100 \Big\rfloor, \tag{5.8}$$

where Q_{max} and $Q_{current}$ denote the queue length and the number of packets queued at the switch, respectively. If a switch has a high number of queued packets, a low N_r enables the switch to store more number of uncompressed rules.

Therefore, the reduction coefficient is expressed as:

$$q = \frac{N_t - N_r}{N_t} \times 100\% \tag{5.9}$$

Algorithm 5.1 reduces the dimensions of the initial rule tensor T and transforms it to the reduced tensor C. In Theorem 5, we discuss that this reduction permits a switch to store more rules, than in the case of a traditional SDN architecture. As we aim to

Algorithm 5.1: TERM: Rule Aggregation Algorithm

Input : $T \in R^{1\times N_f \times N_t}$: Initial rule tensor
Output: $key = \{\mathbb{C} \in R^{1\times N_f \times N_r}, U_k\}$: Core data set

1 Compute $T_{(3)}$ from tensor T
2 $[USV] \leftarrow SVD(T_{(3)})$
3 Truncate $U_k \in R^{N_t \times N_r}$ from U
4 $\mathbb{C} \leftarrow T \times_3 U_k^T$
5 $key \leftarrow \{\mathbb{C}, U_k\}$
6 **return** key

reduce the rule count, we consider the mode-3 unfolded matrix to perform the tensor decomposition method. Mode-3 matrix of tensor T is computed in Line 1 using the procedure of *Tensor Unfolding* or *Matricization* [49]. Figure 5.2 shows three unfolded matrices of an initial rule tensor $T \in R^{1\times 4\times 8}$, which represents eight flow-rules each with one priority value and two match fields and action value. The corresponding unfolded matrices are $T_{(1)} \in R^{1\times 32}, T_{(2)} \in R^{4\times 8}$, and $T_{(3)} \in R^{8\times 4}$.

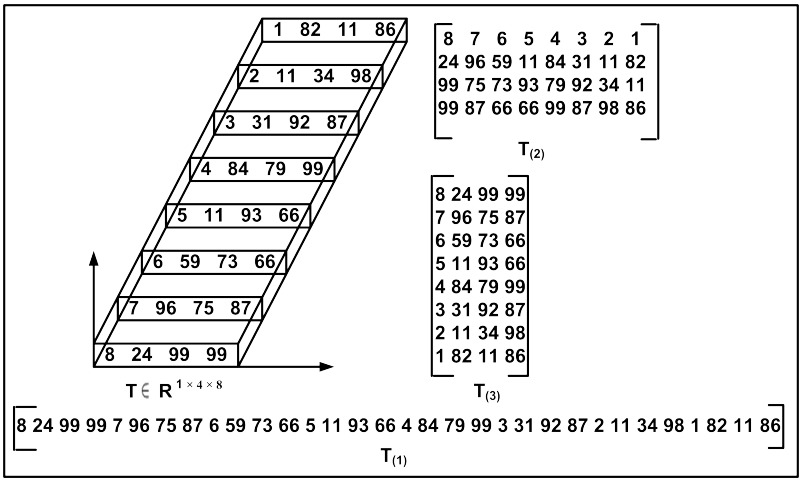


Figure 5.2: Matricization of initial rule tensor

A tensor element $T(a_1, a_2, ..., a_N)$ for a tensor $T \in R^{I_1\times I_2\times ...\times I_N}$ corresponds to the matrix element $T_{(p)}(a_p, b)$, where

$$b = 1 + \sum_{k=1,k\neq p}^{N} \left((a_k - 1) \prod_{m=1,m\neq p}^{k-1} I_m \right) \tag{5.10}$$

In Line 2, the unfolded matrix $T_{(3)}$ is decomposed using the singular value decomposition (SVD) technique. SVD factorizes matrix $T_{(3)}$ into the form:

$$T_{(3)} = USV^T, \tag{5.11}$$

where U and V are the left and right unitary orthogonal matrices, respectively; S is a diagonal matrix, whose elements are singular values of $T_{(3)}$ [50]. Singular values of matrix $T_{(3)}$ are the square roots of the common eigen values of $T_{(3)}T_{(3)}^T$ and $T_{(3)}^T T_{(3)}$. The matrices U and V consist of column vectors, which are transposed eigen vectors of matrices $T_{(3)}T_{(3)}^T$ and $T_{(3)}^T T_{(3)}$, respectively.

The left singular matrix U is truncated in Line 3, which is given by:

$$U_k \in R^{N_t \times N_r} \tag{5.12}$$

The matrix U_k is needed to be stored for rule reconstruction. We store this matrix U_k in parts in the sub-controllers based on their available memory.

Line 4 generates the compressed tensor $\mathbb{C}$ by computing the mode-3 product of tensor T with transpose of matrix U_k, which can be expressed with unfolded matrices as:

$$\mathbb{C} = (T \times_3 U_k^T) \Leftrightarrow C_{(3)} = U_k^T \times T_{(3)} \tag{5.13}$$

Space complexity of Algorithm 5.1 is $O(N_t^2) + O(N_t(N_r + N_f))$, which decomposes to $O(N_t^2)$ as N_t is greater than both N_f and N_r. The time complexity of performing SVD on the unfolded matrix $T_{(3)}$ in Line 2 is $O(min\{N_t^2 N_f, N_t N_f^2\})$ [51]. The time complexity of computing mode-3 product in Line 3 is $O(N_t N_r N_f)$. Therefore, time complexity of Algorithm 5.1 is $O(min\{N_t^2 N_f, N_t N_f^2\}) + O(N_t N_r N_f)$. Figure 5.3 describes the computation of mode-3 product for an order-3 tensor $T \in R^{1\times 4\times 8}$ multiplied by transpose of truncated orthogonal matrix $U_k \in R^{8\times 4}$.

The sub-controller triggers the rule aggregation procedure if free memory of a switch

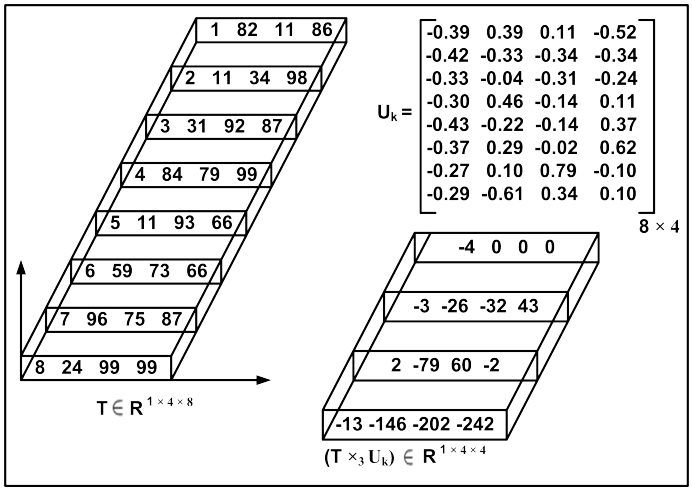


Figure 5.3: Mode-3 product of the initial rule tensor

s_i drops below a specific threshold value th. This limit th is predefined based on the nature of the applications. During an aggregation procedure at time t, all the rules in $R_u^i(t)$ are aggregated to form a new set of aggregated rules.

Theorem 5. *The maximum number of rules stored in the TERM SDN architecture is greater than the maximum number of rules stored in a traditional SDN architecture with D OpenFlow switches, where $D \geqslant 1, N > N_f$; N is the storage capacity of each OpenFlow switch in the traditional SDN architecture in terms of the number of entries, and N_f is the number of fields in a TCAM entry.*

Proof. The maximum number of entries stored in a traditional SDN architecture with D switches, each having a TCAM capable of storing N entries, is given by:

$$Max_t = D \times N \tag{5.14}$$

We denote the maximum number of entries stored in the TERM SDN architecture as:

$$Max_m = D \times \alpha, \tag{5.15}$$

where α is the storage capacity of each switch in terms of the number of entries of the modified architecture. Therefore, we need to prove that,

$$Max_m > Max_t, \text{ where } D \geqslant 1. \tag{5.16}$$

Let $T \in R^{1 \times N_f \times N_t}$ be the tensor representing rules of a switch with N_t uncompressed entries, where each entry has N_f fields, and $0 < N_t < N$. The corresponding switch contains total $(N - N_t)$ entries comprising of cached entries and the aggregated entries generated from the previous aggregation phase.

The p-mode product of a tensor is the basic flow-rule reduction operation for reducing tensor dimensions. To reduce the dimension of the n^{th} order of a tensor T from I_n to I_p $(I_n > I_p)$, we need to compute n-mode product of tensor T by a truncated left singular vector matrix $U \in R^{I_p \times I_n}$.

The aim of our rule aggregation approach is to reduce the 3^{rd} order of tensor $T \in R^{1 \times N_f \times N_t}$ from N_t to N_r, where $N_r < N_t$, to allow storage of larger flow-tables. Therefore, the compressed tensor $\mathbb{C} \in R^{1 \times N_f \times N_r}$ is expressed as:

$$\mathbb{C} = T \times_3 U_3^T, \tag{5.17}$$

where U_3 is obtained by retaining the left N_r unitary orthogonal vectors from the left singular vector matrix generated from singular value decomposition of the mode-3 matrix of tensor T. Figure 5.3 illustrates the operation of computing compressed tensor $\mathbb{C}$ from an initial tensor T. From experimental results, we observe that the minimum value for N_r is N_f for the exact reconstruction of flow-rules. Therefore, the maximum percentage of rule reduction for a switch is $\frac{N-N_f}{N} \times 100$ %. N_r can be chosen dynamically, depending on the application type. If the application is latency-sensitive, then the optimal value of N_r should be chosen, considering the processing time of both rule reduction and reconstruction for approximate rule-set size.

After rule aggregation in each switch, an extra space is available for storing maximum $(N_t - N_f)$ entries. So, L switches can support upto $((N_t - N_f) \times D)$ extra entries. So, Max_m can be expressed as:

$$Max_m = D \times (N + (N_t - N_f)), \tag{5.18}$$

where the storage capacity of each switch in TERM is $\alpha = (N + (N_t - N_f))$. The term $(N + (N_t - N_f)) > 0$, as $0 < N_f \leqslant 45, L > 0, N_t > 0$, and $N > N_f$ [34]. Hence, the statement of Equation (5.16) is true.

□

5.2.2 Rule Reconstruction

The corresponding sub-controller reconstructs the aggregated rules of the switch to verify whether there is a match. Rules in the switch do not change during this process. The reconstructed rules are stored in the sub-controller. After the sub-controller completes the verification process for a possible rule match, it releases the memory used for rule reconstruction.

Change in network policies or topology triggers rule update or the addition of new rules. To handle these changes, the aggregated rules of selected switches are reconstructed and then aggregated again after incorporating the changes.

5.2.2.1 Approximate Rule Tensor Generation

For rule reconstruction, initially, we generate an approximate rule tensor $T_A \in R^{1 \times N_f \times N_t}$, by computing the mode-3 product of compressed tensor $\mathbb{C} \in R^{1 \times N_f \times N_r}$ with truncated unitary orthogonal matrix U_k computed using (Equation (5.12)) and stored. This pro-

cess is expressed as:

$$T_A = \mathbb{C} \times_3 U_k. \tag{5.19}$$

Equation (5.19) can also be expressed as:

$$T_{A_{(3)}} = U_k \times \mathbb{C}_{(3)}, \tag{5.20}$$

where $T_{A_{(3)}}$ and $\mathbb{C}_{(3)}$ are the mode-3 unfolded matrices of approximate rule and compressed rule tensors, respectively [49]. The space complexity of the rule reconstruction procedure is $O(N_t(N_r + N_f))$. The time complexity of the rule reconstruction procedure is $O(N_t N_r N_f)$.

The absolute error of approximation between initial rule tensor T and approximated rule tensor T_A is measured as:

$$\epsilon = ||T - T_A|| = \sqrt{\sum_{i_1=1}^{1} \sum_{i_2=1}^{N_f} \sum_{i_3=1}^{N_t} (t_{i_1 i_2 i_3} - t_{A_{i_1 i_2 i_3}})^2}, \tag{5.21}$$

where $||X||$ denotes the *norm* of a tensor X [49]. This error is introduced due to approximation of the floating-point values in the truncated unitary orthogonal matrix U_k. From experimental results, it is observed that $\epsilon = 0$ for $N_r = [N_f, N_t]$.

The size of the matrix U_k depends on RF which we calculate using Equation (5.8). Hence, the rule reconstruction time is high for a high value of RF due to the computation of mode-3 product in Equation (5.19).

5.2.2.2 Rule Recovery

After approximate rule tensor is generated, exact rule entries are recovered. Each mode-2 *fiber* [49] of tensor T_A corresponds to one row of flow-table. At this stage, the action

value and all the match fields of the flow-table entries are in an integer format. Using transformation logic described in Table 5.1 , we transform each entry of the flow-table back to ternary string. Figure 5.4 shows the process of rule recovery.

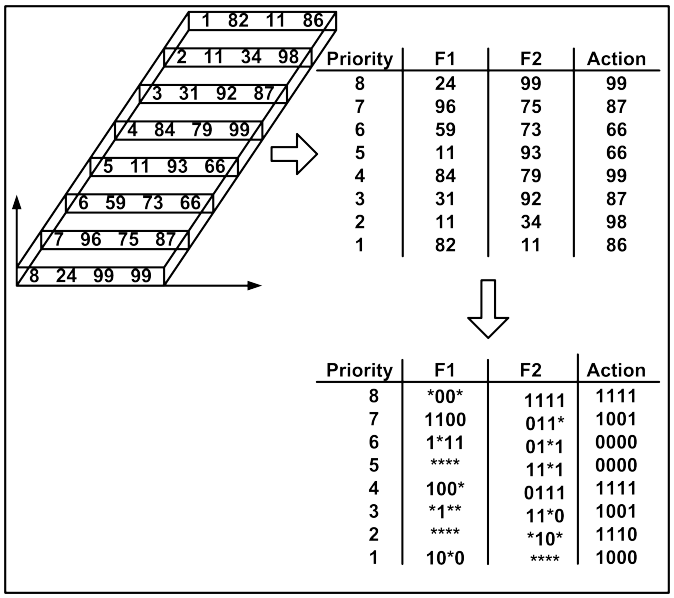


Figure 5.4: Rule recovery process

5.2.3 Rule Caching

Each switch s_i caches the most frequently used rules. Incoming packets that match the cached rules directly follow the actions mentioned in the matched rule. For “cache miss", the packets first search for a match in $R_u^i(t)$. If no match is found, the corresponding sub-controller reconstructs the aggregated rules and checks for a match.

We used the *least recently used* (LRU) cache algorithm. In the OpenFlow protocol version (v1.5.1) [34], each flow-table entry contains a *counters* field, which is updated when incoming packets are matched with the corresponding flow-rule. Investigation of this field allows us to discard the least recently used rules and select the frequently used ones as the potential caching candidates. The discarded rules are added to the $R_u^i(t)$ set. If a rule in the aggregated rule-set $R_a^i(t)$ qualifies as a potential caching candidate, then

that rule is added to set $R_c^i(t)$ with a flag indicating that the rule is also available in the aggregated rule-set $R_a^i(t)$. Therefore, when the rule is no longer needed to be cached, it can be simply deleted from set $R_c^i(t)$ without adding it to set $R_u^i(t)$.

5.3 Performance Evaluation

In this Section, we evaluate the efficiency of TERM, while comparing with traditional OpenFlow-based approach, flow-table partitioning approach — *Pallet* [15], and flow aggregation approach (FAA) [16]. We consider Sprint topology [38] for performance evaluation. We generate random flow-table entries, each with a priority value, a counter value, 45 match fields, and an action value. The performance of TERM is evaluated based on *throughput*, *average packet waiting time*, *free rule-space*, *Packet-In message count*, and *rule aggregation and reconstruction time*. The simulation parameters are depicted in Table 5.2.

Table 5.2: TERM: Simulation Parameters

Parameter	***Value***
Network topology	Sprint [38]
Simulation area	3563.90 km × 1655.20 km [38]
Total number of flows	$[20000, 100000]$
Number of switches	11
Switch capacity	8000 flow-rules [39]
Packet arrival rate per switch	0.02 mpps [44]
Packet processing rate per switch	0.03 mpps [44]
Rule matching time	20 μs [52]
Transmission delay	5 μs per kilometer [12]
Average queue size per switch	0.07 million packets [44]

5.3.1 Result and Discussion

5.3.1.1 Throughput

Throughput is measured as the percentage of packets processed per unit time. Figure 5.5 shows the average throughput when the total number of flows is varied between 20000 and 100000. The average packet arrival rate and packet processing rate per switch are 0.02 mpps and 0.03 mpps, respectively. From the simulation, we observe that the average throughput for TERM is almost similar to Pallet, traditional SDN approach, and FAA.

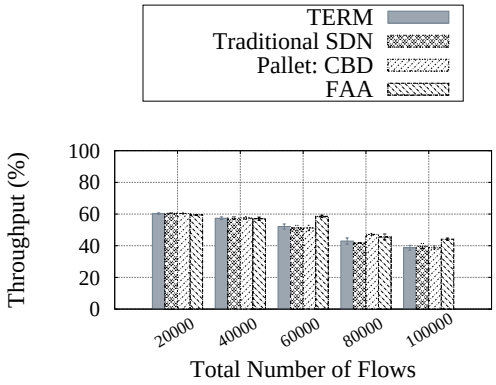


Figure 5.5: TERM: Average Throughput

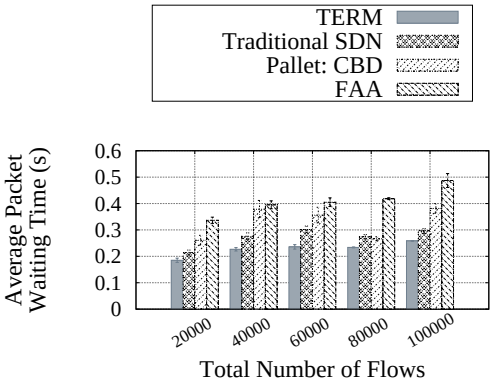


Figure 5.6: TERM: Average Packet Waiting Time

5.3.1.2 Average Packet Waiting Time

Figure 5.6 depicts the average packet waiting time for TERM, traditional SDN, Pallet, and FAA. The average packet waiting time of TERM is 14.81%, 30.30%, and 43.90% less than traditional SDN, Pallet, and FAA, respectively. Therefore, we yield that the average packet waiting time is short in TERM, as the most frequently used rules are cached in each switch.

5.3.1.3 Free Rule-Space

The amount of free rule-space is the percentage of total rule-space available for storing new flow-rules. As shown in Figure 5.7, the average free rule-space is significantly higher

in TERM, as each rule aggregation procedure aggregates the existing rules and releases rule-space.

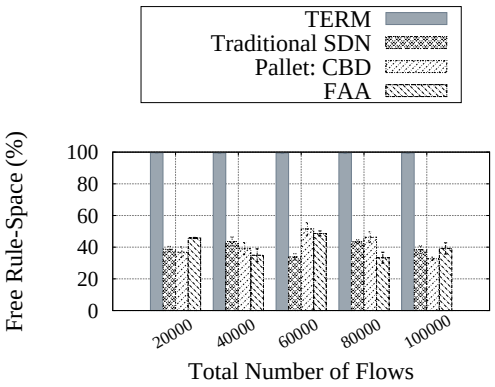


Figure 5.7: TERM: Average Free Rule-Space

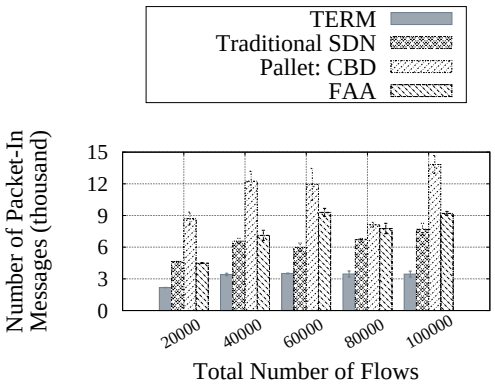


Figure 5.8: TERM: Average Number of Packet-In Messages

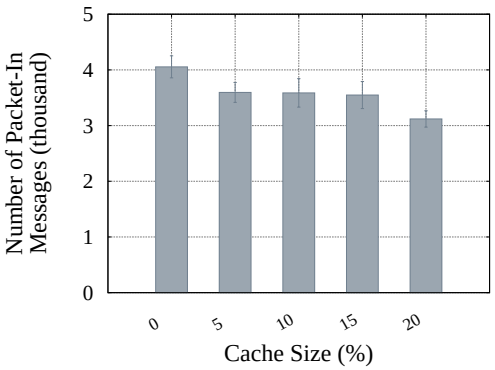


Figure 5.9: TERM: Effect of Cache Size on Packet-In Message Count

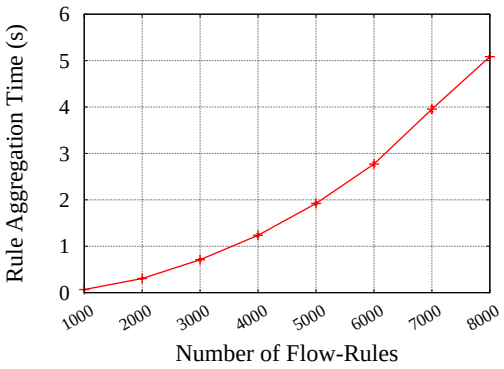


Figure 5.10: TERM: Rule Aggregation Time

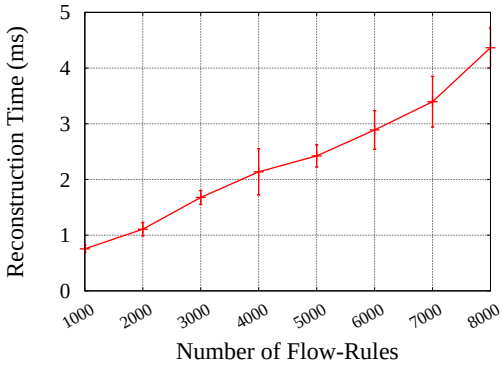


Figure 5.11: TERM: Rule Reconstruction Time

5.3.1.4 Packet-In Message Count

Figure 5.8 shows the average number of Packet-In messages generated from each switch in the network. The cached rule-space size is fixed to 10% of the total rule-space. The average number of Packet-In messages is 49.45%, 70.83%, and 57.78% less than traditional SDN, Pallet, and FAA, respectively.

Figure 5.9 depicts the average number of Packet-In messages generated from each switch for different cache sizes. The total number of flows is 10000. As shown in Figure 5.9, the number of Packet-In messages for 20% cache size is 22.96% less than that for no cache. Therefore, we yield that caching reduces the Packet-In message count. In addition, we synthesize that after a specific size of $R_c^i(t)$, the Packet-In message count decreases as most of the packets are matched in $R_c^i(t)$.

5.3.1.5 Rule Aggregation and Reconstruction Time

The rule aggregation time is the time required to compress the flow-rules of a switch into a lesser number of TCAM entries. Similarly, the rule reconstruction time is the time needed to transform the aggregated TCAM entries into actual flow-rules. Figure 5.10 and Figure 5.11 depict the average rule aggregation and reconstruction time of a switch, respectively. From the simulation results, we observe that the rule reconstruction time is significantly less than the rule aggregation time.

5.4 Concluding Remarks

In this chapter, we present a scheme, named TERM, that aims to reduce flow-table miss by increasing the available capacity of switches in SDN. TERM uses the tensor decomposition technique to compress heterogeneous flow-rules. Simulation results indicate enhanced performance in terms of reduced packet waiting time, increased free rule-space, and reduced number of Packet-In messages.

Chapter 6

Control Plane Load Reduction

In this chapter, we propose a scheme for — *Control Plane Load Reduction (CORE)* in SDIoT. The management of control plane load is an essential issue for IoT networks because of the dynamic traffic characteristics. IoT traffic is highly dynamic due to the heterogeneity of IoT devices in terms of mobility, activation model, QoS demand, and flow generation rate. The challenge is to prevent controller overload and distribute traffic optimally, considering heterogeneous IoT devices. CORE estimates the load on each controller based on the mobility and traffic characteristics of IoT devices and performs an optimal master controller assignment to reduce the control plane load.

This chapter consists of four sections. The system model of CORE is presented in Section 6.1. Section 6.2 describes the proposed scheme. Section 6.3 depicts the experimental results. Finally, Section 6.4 concludes the proposed work and discusses directions for future work.

6.1 System Model

The SDIoT architecture considered in CORE is depicted in Figure 6.1. The architecture comprises three layers — application, network, and perception. The application layer consists of IoT applications, which perform services requested by the users based on the

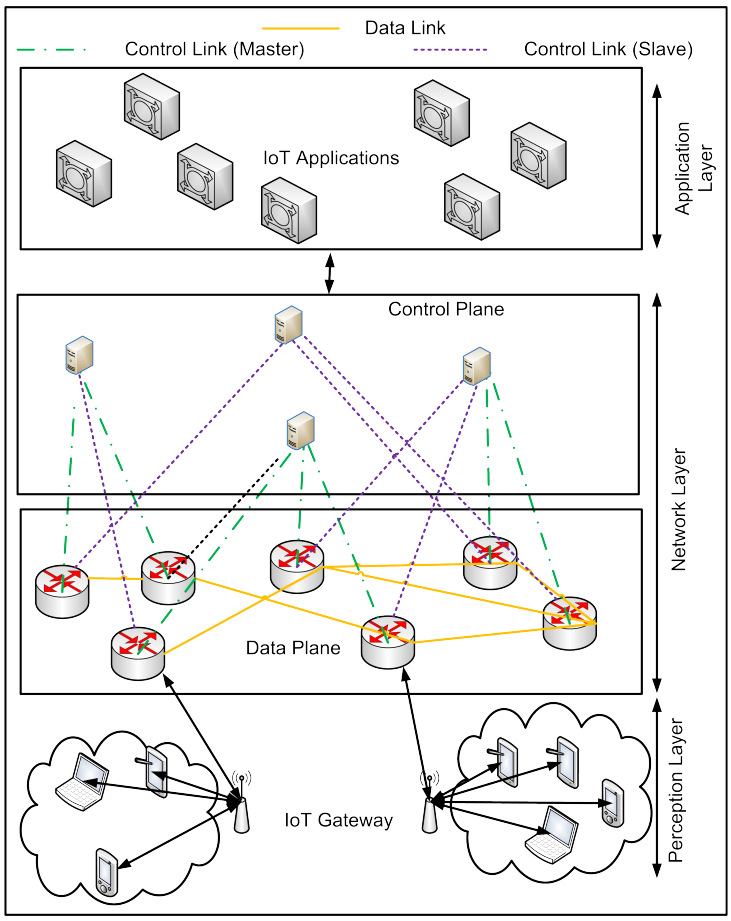


Figure 6.1: CORE: SDIoT Architecture

data collected from the network layer. The network layer consists of data plane and control plane. Let $G = (S, E)$ represent the data plane topology, where S is the set of SDN switches and E is the set of links. We assume that each switch stores up to R_{max} flow-rules. Each flow-rule r_e has a timeout duration T_e seconds. Let C represent the set of controllers. We consider that each switch s_i is attached to single master controller and one or multiple slave (read-only) controllers [34] during a time-slot. Let, ϵ seconds be the duration of each time-slot. The master controller for switch s_i is expressed as:

$$x_{ij}(t) = \begin{cases} 1 & \text{if } c_j \text{ is the master controller of } s_i, \\ 0 & \text{otherwise.} \end{cases} \tag{6.1}$$

The slave controller for switch s_i is expressed as:

$$y_{ij}(t) = \begin{cases} 1 & \text{if } c_j \text{ is the slave controller of } s_i, \\ 0 & \text{otherwise.} \end{cases} \tag{6.2}$$

A controller cannot be both master and slave for the same switch at the same time-slot.

$$x_{ij}(t) + y_{ij}(t) \leq 1, \forall s_i \in S, \forall c_j \in C \tag{6.3}$$

Definition 18 (Controller Capacity)**.** *The capacity of a controller c_j is the maximum number of Packet-In requests the controller handles in a time-slot and is denoted by Ω_j.*

The perception layer contains static and mobile IoT devices that are heterogeneous in terms of QoS requirements. The flows generated by the IoT devices are transmitted over the wireless channel to switches via access points (APs) having different radio access capabilities such as WiFi, WiMax, Bluetooth, 3G, 4G, Zigbee, mmWave, and TV White Space. In this work, we assume that each IoT device is capable of communicating via more than one radio access technique. For a time-slot t, we consider that the number of IoT devices present in the network is $n_r(t)$. The set of IoT devices is denoted by the set $D(t) = \{d_1, d_2, \ldots, d_{n_r(t)}\}$. For simplicity, we assume that all the flow-rules are exact-match flow-rules [34], where the mapping between flow-rule and flow type is one-to-one. Further, we assume that each device generates only single type of flow. Let Q_k be the number of flows generated by d_k per second. We assume that the controllers record device specific parameters such as the flow generation rate, the mapped flow type, and QoS requirement for each time-slot. At time-slot t, the association between an IoT device and an SDN switch is expressed as:

$$z_{ik}(t) = \begin{cases} 1 & \text{if } d_k \text{ is associated with } s_i, \\ 0 & \text{otherwise.} \end{cases} \tag{6.4}$$

We consider that a device is associated with single switch only, as each AP sends data to a specific SDN switch. Therefore, $\sum_{i=1}^{|S|} z_{ik}(t) = 1, \forall d_k \in D(t)$.

Each IoT device d_k activates/deactivates following either — (1) random activation model or (2) periodic activation model [53]. A device d_k following random activation model activates at time $\tau \in [0, T]$ according to the beta distribution with shape parameters β_1, and β_2, which is expressed as:

$$f_k(\tau) = \frac{\tau^{\beta_1 - 1}(T - \tau)^{\beta_2 - 1}}{T^{\beta_1 + \beta_2 - 1} \int_0^1 \tau^{\beta_1 - 1}(1 - \tau)^{\beta_2 - 1} d\tau}, \tag{6.5}$$

where $[0, T]$ is the duration within which the devices are operational. On the other hand, a device d_k following periodic activation model activates repeatedly after a fixed duration τ_k seconds. Therefore, the probability that a device d_k following periodic activation model activates at time $\tau \in [0, T]$ given by:

$$f_k(\tau) = \begin{cases} 1 & \text{if the interval between } \tau \text{ and the last active time of } d_k \text{ is more than } \tau_k, \\ 0 & \text{otherwise.} \end{cases} \tag{6.6}$$

The maximum number of Packet-In messages generated by d_k in time-slot t is $M_k(t) = \int_{t_0}^{t_0 + \epsilon} f_k(\tau) Q_k d\tau$, where t_0 is the start time of time-slot t.

6.1.1 Mobility Model

We consider a network which has a large number of static or mobile IoT devices. Examples of some mobile IoT devices are smart wearables, cameras, and AR/VR glasses [54]. During each time-slot, SDN controllers collect device locations using Simple Network Management Protocol (SNMP) via south bound APIs [55]. We use this collected data as a history data set to predict device-switch associations.

6.1.2 Caching Model

Each flow-entry has a default timeout duration [34]. However, an IoT device usually generates similar flow requests for a particular time duration. The interval of the arrival of such similar flows may be greater than the timeout duration of the corresponding flow-rule. In this case, a Packet-In message is re-generated, and an expired rule is re-installed. Rule-caching is one of the measures to reduce the number of Packet-In requests. However, as the cache size increases, the rule-space required for storing new rules decreases, and the number of Packet-in requests increases. Therefore, CORE considers that each SDN switch caches maximum $R_{cache} < R_{max}$ flow-rules. To express whether a switch s_i caches a flow-rule for d_k during time-slot t we define a binary variable as:

$$w_{ik}(t) = \begin{cases} 1 & \text{if } s_i \text{ caches flow-rule that maps to the flow type of } d_k, \\ 0 & \text{otherwise.} \end{cases} \tag{6.7}$$

6.1.3 Delay Model

Delay of an IoT flow of type f_k has three components — a) device to AP communication delay $\delta_k^1(t)$, b) AP to switch communication delay $\delta_k^2(t)$, and c) flow setup delay $\delta_k^{ex}(t)$ at the switch. Mathematically, $\delta_k^1(t) = \Delta^1(t) + \frac{g_k(t)}{G^1\eta^1}$ and $\delta_k^2(t) = \Delta^2(t) + \frac{g_k(t)}{G^2\eta^2}$, where $\Delta^1(t)$ is the transmission delay from device to AP, $\Delta^2(t)$ is the transmission delay from from AP to switch, $g_k(t)$ represents the number of bytes sent by d_k in time-slot t, G^1 is the bandwidth of the wireless channel from device to AP, G^2 represents the bandwidth of the wireless channel from AP to switch, η^1 and η^2 represent the channel overheads of the corresponding wireless channels. The flow setup delay $\delta_k^{ex}(t)$ is:

$$\delta_k^{ex}(t) = \sum_{i=1}^{|S|} \sum_{j=1}^{|C|} z_{ik}(t) x_{ij}(t) \left(2\delta_{ij}(t) + \delta_j^{que}(t) \right), \tag{6.8}$$

where $\delta_{ij}(t)$ is the transmission delay associated with the control link and $\delta_j^{que}(t)$ is the queueing delay at controller c_j. A controller stores the Packet-In requests in its queue and processes the requests in a First-Come-First-Serve (FCFS) order. We consider each request as an individual and independent Poisson process. Therefore, we model controller queue as a $M/M/1$ queue. The service rate of this queueing model is controller capacity Ω_j. The maximum request arrival rate $\lambda_j(t)$ is given by:

$$\lambda_j(t) = \sum_{i=1}^{|S|} \sum_{k=1}^{|D(t)|} x_{ij}(t) z_{ik}(t)(1 - w_{ik}(t)) M_k(t) \tag{6.9}$$

Here, (6.9) considers the associated devices which have no rules cached and estimates the maximum number of Packet-In requests based on the active duration of the devices for each controller c_j in time-slot t. The queueing delay at c_j is: $\delta_j^{que}(t) = \frac{1}{\Omega_j - \lambda_j(t)}$.

6.1.4 Cost Model

Control plane cost has two components — 1) controller-switch communication cost and 2) inter-controller communication cost due to device mobility. The controller-switch communication cost at c_j is the traffic intensity $\rho_j(t) = \frac{\lambda_j(t)}{\Omega_j}$. Controllers collect global network data by synchronizing with other controllers at regular intervals. We assume that each controller completes this synchronization process at the beginning of a time-slot. Additionally, there exist two cases when a controller synchronizes with another.

- Case 1: Change in Controller-Switch Association

  At time-slot t, each controller c_j records the switches to which c_j served as a slave controller for time-slot $t-1$ before changing its role to a master controller. In this case, c_j needs to synchronize with the former master controller(s) of the switches. Figure 6.2 shows an example where the master controller of switch s_3 changes from c_1 to c_2 at time-slot t. Therefore, for seamless handover, c_2 collects unfinished session data and flow information from c_1. For each controller c_j, the number of

master-slave role changes during a time-slot is $\chi_c^j(t) = \sum_{i=1}^{|S|} |x_{ij}(t) - x_{ij}(t-1)|$.

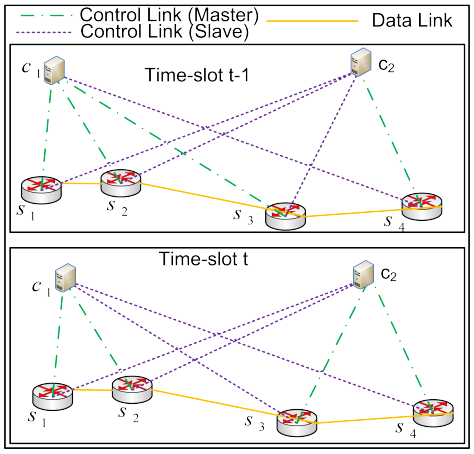


Figure 6.2: Case 1: Change in Controller-Switch Association

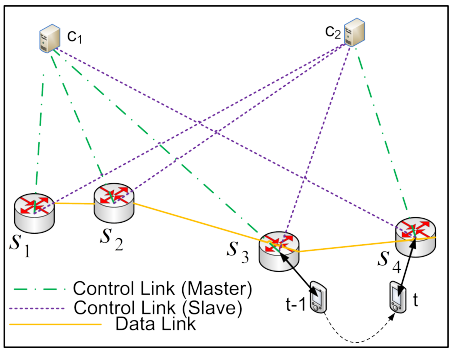


Figure 6.3: Case 2: Change in Device-Switch Association

- Case 2: Change in Device-Switch Association

  At time-slot t, each controller c_j records the mobile IoT devices which are newly associated with the switches assigned to c_j. If the old switches have different master controller(s), c_j needs to synchronize with the master controller(s) of the old switches. Figure 6.3 shows an example in which a mobile device changes the associated switch from s_3 to s_4 at time-slot t. As s_3 and s_4 have different master controllers c_1 and c_2, controller synchronization is required for seamless handover. For each controller c_j, the number of such changes where controller synchronization is required is $\chi_s^j(t) = \sum_{i=1}^{|S|} \sum_{k=1}^{|D(t)|} \xi_k^j(t)$, where $\xi_k(t)$ is expressed as:

$$\xi_k^j(t) = \begin{cases} 1 & \text{if } x_{ij}(t) z_{ik}(t) = x_{i'j'}(t-1) z_{i'k}(t-1) = 1,\ i \neq i',\ j \neq j', \\ 0 & \text{otherwise.} \end{cases} \tag{6.10}$$

Therefore, the total inter-controller communication cost for c_j is $\Gamma_j(t) = \chi_c^j(t) + \chi_s^j(t)$.

6.1.5 Problem Formulation

The objective of CORE is to determine optimal controller-switch assignments to minimize the control plane cost of the network. Therefore, we formulate the cache-enabled minimum cost master controller assignment (CMCA) problem as:

$$\underset{x(t),w(t)}{\text{Minimize}} \quad \alpha \sum_{j=1}^{|C|} \rho_j(t) + (1-\alpha) \sum_{j=1}^{|C|} \Gamma_j(t) \tag{6.11}$$

subject to

$$\lambda_j(t) \leq \Omega_j, \forall c_j \in C, \tag{6.12}$$

$$\sum_{j=1}^{|C|} x_{ij}(t) = 1, \forall s_i \in S, \tag{6.13}$$

$$\sum_{k=1}^{|D(t)|} w_{ik}(t) \leq R_{cache}, \forall s_i \in S, \tag{6.14}$$

$$x_{ij}(t) = x_{ij}(t-1) + y_{ij}(t-1),$$

$$\forall s_i \in S, \forall c_j \in C \tag{6.15}$$

$$\sum_{i=1}^{|S|} w_{ik}(t) \leq 1, \forall d_k \in D(t), \tag{6.16}$$

$$\delta_k \leq \delta_k^{max}, \forall d_k \in D(t) \quad , \tag{6.17}$$

where $\alpha \in [0, 1]$ is a weighting factor to control the relative importance of controller-switch communication cost and inter-controller communication cost. The relation in (6.12) ensures that none of the controllers is overloaded. The truth that each switch belongs to a single master controller is presented in (6.13). Additionally, (6.14) ensures that the number of cached flow-rules in each switch does not exceed the maximum allowable limit R_{cache}. The relation in (6.15) ensures that a controller can be assigned with the master role for a switch in time-slot t if and only if it is the master or slave controller for that switch in the previous time-slot. Moreover, (6.16) ensures that a device can have cached rule only in single switch as each switch has limited rule storage capacity. Finally, (6.17) expresses the delay constraint for each device, where δ_k^{max} is the maximum allowable delay for d_k.

Theorem 6. *CMCA problem is NP-hard.*

Proof. Let us consider a particular instance of the CMCA problem by excluding the rule caching at switches. In this case, we have $|C|$ controllers and $|S|$ switches. Each controller-switch association increases traffic intensity at the corresponding controller. In addition, each controller has a maximum capacity. For example, a switch s_i can be associated with a master controller c_j only if $\lambda_j(t) < \Omega_j$. A feasible solution ensures completeness constraint in (6.13) that each switch is assigned to exactly one master controller. The goal of the problem is to find a feasible solution that minimizes the total control traffic intensity. This is in the form of a generalized assignment problem [56], which has been proved as NP-hard. Hence, the CMCA problem is also NP-hard. □

As the optimization problem in (6.11) is NP-hard, it is difficult to obtain a solution in reasonable time. Therefore, we propose a master controller assignment scheme based on the branch and bound technique [57] to determine near-optimal solutions.

6.2 CORE: The Proposed Scheme

CORE contains three modules for the purpose of — (a) mobility prediction, (b) rule-caching, and (c) master controller assignment. The mobility prediction module analyzes mobility history of IoT devices to predict device-switch association information. Thereafter, the selected flow-rules are cached by the rule-caching module to reduce the control plane load. Finally, the master controller assignment module determines optimal controller-switch associations.

6.2.1 Mobility Prediction

We determine the control plane load based on the number of control messages it handles during a time-slot. However, the number of control messages depends on the devices associated with the switches at a time-slot. Consequently, we predict the device-switch

associations, while considering the movement history of the devices. Subsequently, we use this prediction data to cache device-specific flow-rules in switches and determine optimal controller-switch associations. We use the existing Markov Predictor [58], which is one of the most popular location prediction algorithms to predict the future location of a mobile device based on its mobility history. An $O(m)$ Markov Predictor considers m most recent locations of a mobile device and predicts the next location. Markov Predictor consumes less space and performs better than other popular predictors for low values of m [59]. Therefore, we use order-m ($O(m)$) Markov Predictor to determine each device's future location. For predicting the location of a device d_k, the components of an order-m ($O(m)$) Markov Predictor are:

- *Input*: The input set $H_k(t) = \{\{L_{t,k}, \Im_{t,k}, V_{t,k}\}, P_{t,k}\}$ represents the mobility history of d_k at time-slot t, where $L_{t,k} = \{l_{tk1}, l_{tk2}, \ldots, l_{tkn}\}$ is the set of locations or meaningful places that the device visits, $\Im_{t,k} = \{\tau_{tk1}, \tau_{tk2}, \ldots, \tau_{tkn}\}$ denotes the set of arrival times at the locations in $L_{t,k}$, $V_{t,k} = \{v_{tk1}, v_{tk2}, \ldots, v_{tkn}\}$ is the set of durations of stay at each location in $L_{t,k}$, and $P_{tkij} \in P_{t,k}$ represents the transition probability from location l_{tki} to location l_{tkj}, $i \neq j$.

- *Output*: The output $l_{t+1,k} \in L_{t,k}$ is the predicted location of d_k in time-slot $t+1$.

- *Context*: The context is $h{=}L_{t,k}(n{-}m{+}1, n){=}\{l_{tk(n-m+1)}, l_{tk(n-m+2)}, \ldots, l_{tk(n-1)}, l_{tkn}\}$.

Markov Predictor extracts the context h from the input set $H_k(t)$ and examines the duration of stay V_l at a location l that follows h. Mathematically,

$$V_l = \{v_{tki} | v_{tki} = \tau_{tk(i+1)} - \tau_{tki}, \text{ where } L_{t,k}(i-m+1, i+1) = hl\} \tag{6.18}$$

From each V_l, we compute the conditional probability $P_l(\tau \leq v < \tau + \Delta\tau | h, \tau)$ that the device shifts to location l within $\Delta\tau$ time beyond the current elapsed time τ. We consider $\Delta\tau$ as the remaining time of the current time-slot. Therefore, for a given h and

τ, the probability of each device moving to each possible location l within $\Delta\tau$ time is:

$$P(l|h,\tau) = P(l)P_l(\tau \leq v < \tau + \Delta\tau|h,\tau), \tag{6.19}$$

where $P(l)$ is the transition probability of every possible next location l which is:

$$P(l_{tk(n+1)} = l|L_{t,k}) \approx \widehat{P}(l_{tk(n+1)} = l|L_{t,k}) = \frac{N(hl, L_{t,k})}{N(h, L_{t,k})}, \tag{6.20}$$

where $N(hl, L_{t,k})$ signifies the number of occurrences of hl in the set $L_{t,k}$. Therefore, the output of the Markov Predictor which is the most likely next location of d_k is:

$$l_{tk(n+1)} = \arg\max_{l \in L_{t,k}} P(l_{tk(n+1)} = l) \tag{6.21}$$

If $N(h, L_{t,k}) = 0$, the $O(m)$ Markov Predictor fails to return a result. Therefore, we use fallback Markov Predictor [58] which backtracks to an $O(m-1)$ Markov predictor whenever an $O(m)$ Markov Predictor fails to return a result. The $O(0)$ Markov Predictor yields the location that occurs most frequently in the location history set $L_{t,k}$.

Algorithm 6.1: CORE: Mobility Prediction Algorithm

Inputs : $H_k(t-1)$, h
Output: $z(t)$

1 Extract $L_{t-1,k}$ from $H_k(t-1)$
2 Compute V_l at possible locations l using (6.18)
3 Calculate $P(l|h,\tau)$ using (6.19)
4 Predict the next location using (6.21)
5 Select the nearest AP which covers the predicted location and matches the radio access capability of the d_k
6 Set $z_{ik}(t) = 1$ if s_i is associated with the selected AP

Algorithm 6.1 presents the steps required for mobility prediction of a device d_k and the formulation of device-switch association $z_{ik}(t)$. The Mobility Prediction Algorithm (MPA) is executed for each device d_k. An $O(m)$ Markov Predictor returns a location where the device is predicted to be present in time-slot t. We consider that d_k associates

with the nearest AP that covers the predicted location and matches its radio access capability. Let s_i be the switch associated with the selected AP. Therefore, MPA predicts the device-switch association $z_{ik}(t) = 1$.

6.2.2 Rule-Caching

To estimate rule popularity, rule-caching module sorts the flow-rules in each switch in descending order of the received packet count. Let R_i be the set of flow-rules in s_i. Therefore, the rule popularity is denoted by $\Theta = \{\theta_1, \theta_2, \theta_3, \ldots, \theta_{|R_i|}\}$, where $\theta_j \in [0, 1]$ is the probability that an incoming flow matches with the j^{th} flow-rule. In this work, we assume that rule popularity satisfies the Zipf distribution [36]. Therefore, the popularity of the j^{th} ordered flow-rule is $\theta_j = \frac{\frac{1}{j^\gamma}}{\sum_{a=1}^{|R_i|} \frac{1}{a^\gamma}}$, where $\gamma \in [0, 1]$ denotes the skewness of the rule popularity. The value $\gamma = 0$ signifies uniform popularity distribution and a larger γ implies more uneven rule popularity.

Algorithm 6.2: CORE: Rule-Caching Algorithm

Inputs : R_i, γ, $z(t)$
Output: $w(t)$

```
1 Compute popularity of the rules in R_i
2 Sort the flow-rules in descending order of popularity
3 foreach rule r_e in the sorted list do
4 |   Select the device d_k whose flow type maps to r_e
5 |   if z_ik(t) == 1 and r_e not cached then
6 |   |   Delete the least popular rule from cache if cache is full
7 |   |   Set T_e = 1/Q_k(t-1) + (T_0 - δ_k^max), w_ik(t) = 1
8 |   end
9 end
```

Algorithm 6.2 presents the steps of the proposed greedy solution for caching rules in each switch s_i. For each switch s_i, the Rule-Caching Algorithm (RCA) sorts the flow-rules present in the rule-space of the switch based on the rule popularity. For each flow-rule r_e which maps to the flow type of d_k, RCA checks whether $z_{ik}(t) == 1$ from the output of MPA. In addition, RCA checks whether the rule is already cached by s_i.

If the cache size reaches its maximum limit R_{cache}, RCA deletes the least popular rule from the cache by setting its timeout as the default timeout T_0. For caching r_e, RCA sets the timeout value as $T_e = \frac{1}{Q_k(t-1)} + (T_0 - \delta_k^{max})$. This timeout value ensures that latency-sensitive flows are prioritized over other flows as a larger timeout value signifies lower chance of flow-table miss.

6.2.3 Master Controller Assignment

We derive the optimization problem for minimum cost master controller assignment from the joint optimization problem of cache-enabled minimum cost master controller assignment stated in (6.11). Hence, for a given caching policy $w(t)$, the optimization problem P_0 for minimum cost master controller assignment (MCA) is given by:

$$\underset{x(t)}{\text{Minimize}} \quad \alpha \sum_{j=1}^{|C|} \rho_j(t) + (1-\alpha) \sum_{j=1}^{|C|} \Gamma_j(t) \tag{6.22}$$

subject to

$$\lambda_j(t) \leq \Omega_j, \forall c_j \in C, \tag{6.23}$$

$$\sum_{j=1}^{|C|} x_{ij}(t) = 1, \forall s_i \in S, \tag{6.24}$$

$$x_{ij}(t) = x_{ij}(t-1) + y_{ij}(t-1),$$

$$\forall s_i \in S, \forall c_j \in C \tag{6.25}$$

$$\sum_{i=1}^{|S|} w_{ik}(t) \leq 1, \forall d_k \in D(t), \tag{6.26}$$

$$\delta_k \leq \delta_k^{max}, \forall d_k \in D(t) \tag{6.27}$$

The MCA problem is non-convex because of the presence of binary decision variables. For solving the MCA problem, we use the branch and bound technique [57] which defines a common structure to solve a wide range of non-convex optimization problems. Therefore, the master controller assignment scheme for the MCA problem has two significant components — 1) branching method and 2) lower-bounding method.

6.2.3.1 Branching Method

Let P_0 denote the MCA problem stated in (6.22). The branching method starts with P_0 as the root of the search tree. The total number of levels in the tree is $|S| + 1$ starting from level 0. Each level corresponds to the selection of a master controller for each switch s_i. For example, level 1 corresponds to the selection of a master controller for switch s_1. Therefore, each node at a level denotes a subproblem. At each level, we partition the leaves or subproblems. Each child node of a node P_v at level l corresponds to a feasible master controller for s_{l+1}. Let C_v be the set of feasible master controllers for s_{l+1}. From constraint (6.25), we find that a controller $c_j \in C$ is a member of C_v if $x_{l+1,j}(t-1) + y_{l+1,j}(t-1) = 1$. Therefore, the number of children of P_v is $|C_v|$.

6.2.3.2 Lower-Bounding Method

Initially, we construct a lower bound for the original MCA problem. To find the initial lower bound, we construct a relaxed problem MCA-R by removing the controller capacity constraint in (6.23). Therefore, each switch freely selects the master controller so that the control plane cost is minimum. For a given switch s_i, the cost for the assignment to a master controller c_j is $U_{ij} = \alpha\rho_j(t) + (1-\alpha)\Gamma_j(t)$, where $x_{ij} = 1$ and $x_{ij'} = 0$ for all $j \neq j^{'}$. Therefore, the cost of a minimum cost controller-switch association for a given switch s_i is expressed as $U_{ij^i} = \min_{\forall c_j}\{U_{ij}\}$. Hence, the LB for problem P_0 is $LB_0 = \sum_{i=1}^{|S|} U_{ij^i}$. Subsequently, we find the LB for each subproblem P_v where $v \neq 0$. Let $x^v(t)$ be the allocation matrix for the branch ending at a node P_v at level l. Therefore, the initial value of Lower Bound (LB) is $LB_v^0 = \sum_{\forall c_j \in C, s_i \in S} U_{ij}x^v(t)$. $S^{'} = s_{l+1}, s_{l+2}, \ldots, s_{|S|}$ denotes the set of unassigned switches for the current branch. For each switch $s_i \in S^{'}$, we find the minimum cost controller-switch association that satisfies the constraints (6.23), (6.24), (6.25) and (6.27). Therefore, the LB of P_v is $LB_v = LB_v^0 + \sum_{s_i \in S^{'}} \min_{\forall c_j}\{U_{ij}\}$.

Algorithm 6.3: CORE: Master Controller Assignment Algorithm

Inputs : P_0, C, S, $z(t)$, $w(t)$
Output: $\{x^*(t), u^*\}$

```
1  P ← {P_0}, u* = +inf, x*(t) = 0
2  while P ≠ φ do
3      Select a node P_v ∈ P
4      P ← P − {P_v}
5      Apply branching method to P_v and generate subproblems P_{v_1}, P_{v_2}, ..., P_{v_|C_v|}
6      foreach P_{v_a} do
7          Compute LB_{v_a}
8          if LB_{v_a} > u* then
9              Delete P_{v_a}
10             if P_{v_a} gives a complete solution {x'(t), u'} then
11                 u* = u'
12                 x*(t) = x'(t)
13             else
14                 P ← P ∪ {P_{v_a}}
15             end
16         end
17     end
18 end
```

6.2.3.3 Master Controller Assignment Algorithm

Algorithm 3 shows the branch and bound procedure to solve the MCA problem. The Master Controller Assignment Algorithm (MCAA) initializes the values of optimal solution $x^*(t)$ and the optimal objective value u^*. In addition, MCAA adds the root node P_0 to the set of live nodes P. For each live node $P_v \in P$, MCAA applies branching method to generate child nodes or subproblems. A subproblem is deleted if it has a LB greater than the optimal objective value u^*. The values $x^*(t)$ and u^* are updated when a subproblem generates a complete solution with each switch assigned to a master controller. Otherwise, the subproblem is added to the set of live nodes P. The output of MCAA signifies an optimal master controller assignment $x^*(t)$ for time-slot t. At time-slot $t-1$, we compute $x^*(t)$ and change the controller-switch assignments accordingly

by using Role-Change messages [34]. Additionally, we deactivate the controllers which have no assigned switches.

6.3 Performance Evaluation

6.3.1 Simulation Settings

For the simulation, we consider random controller placement. In addition, we consider an equal number of randomly and periodically activated devices. We conduct two sets of experiments for performance evaluation. In the first experiment, we consider that 80% devices generate high traffic. This experiment evaluates the performance of the proposed scheme in the presence of high IoT traffic volume. In the second experiment, we set the percentage of latency-sensitive devices as 80% to analyze the performance for time-critical IoT applications. Table 6.2 shows the specific parameters considered for categorizing high traffic generating and latency-sensitive devices. The simulation parameters are depicted in Table 6.1.

6.3.2 Benchmark Schemes

We compare CORE with existing switch migration-based schemes — DCP-SA [22] and ESMLB [23]. DCP-SA considers flow setup delay and inter-controller communication in the presence of dynamic traffic. ESMLB considers the control traffic generated by the switches as primary criteria for switch migration-based load balancing in SDIoT control plane. On the other hand, CORE considers flow setup delay, inter-controller communication, dynamic network traffic, device mobility, and heterogeneous QoS demands to determine feasible controller-switch assignment.

6.3.3 Performance Metrics

The performance metrics considered for evaluating the proposed scheme are as follows:

Table 6.1: CORE: Simulation Parameters

Parameter	Value
Network topology	8−pod Fat-tree [60]
Simulation Area	500 m × 500 m
Mobility model	Gauss-Markov [61]
Number of IoT devices	$200 - 2500$
Speed of IoT devices	$1 - 2$ m/s [62]
Number of switches	20
Flow-rule default timeout T_0	10 s
Number of controllers	5
Controller capacity	$7200 - 10800$ K req/time-slot [63]
Average packet size	$94 - 234$ bytes [64]
Mean data rate	$462 - 11388$ bytes/s [64]
Maximum allowable delay	$0.001 - 1$ s [46]
Time-slot duration ϵ	1 hour
Skewness of rule popularity γ	0.56
Shape parameter β_1	3 [53]
Shape parameter β_2	4 [53]
Weighing factor α	0.8

Table 6.2: Device Category

Category	Average packet size (bytes)	Mean data rate (bytes/s)	Maximum allowable delay (s)
High traffic generating	234 [64]	11388 [64]	$0.001 - 1$ [46]
Latency-sensitive	94 [64]	462 [64]	$0.001 - 0.25$ [46]

- **Prediction accuracy**: Prediction accuracy shows the correctness of mobility prediction for the IoT devices.

- **Control plane cost**: Control plane cost is the cumulative cost of controller-switch communication cost and inter-controller communication cost, as mentioned in (6.11). We evaluate this metric to estimate the load on the control plane as a high controller load increases the cost.

- **Peak traffic intensity**: We calculate the peak traffic intensity across all controllers to analyze the distribution of control traffic. Mathematically, the peak

traffic intensity is given as $max\left(\rho_j(t)\right), \forall c_j \in C$.

- **QoS violated flows**: QoS violated flows are the flows which do not satisfy end-to-end delay requirement of the flow type. We evaluate this metric to show the efficiency of CORE in terms of QoS.

6.3.4 Observations and Results

6.3.4.1 Prediction Accuracy

For the simulation, we fix the order of the Markov predictor as $k = 3$. We use a Twitter dataset [65] involving $200-1000$ devices to analyze the prediction accuracy of the Markov predictor. Figure 6.4 shows that the average prediction accuracy is 83.72%. From the simulation, we infer that CORE is capable of correctly predicting the device locations for a significant number of cases, although the mobility pattern and speed of the devices are highly dynamic.

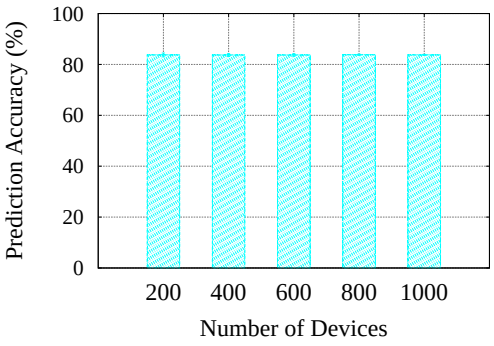
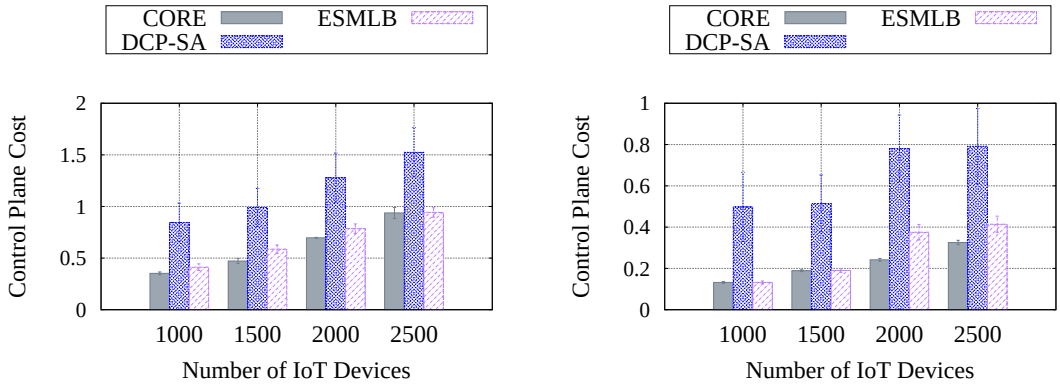


Figure 6.4: CORE: Prediction Accuracy

6.3.4.2 Control Plane Cost

Figure 6.5(a) shows that CORE achieves 46.94% and 9.82% reduction in control plane cost compared to DCP-SA and ESMLB, respectively, for high traffic load. Figure 6.5(b) shows that CORE achieves 65.63% and 20.14% reduction in control plane cost compared

to DCP-SA and ESMLB, respectively, when the majority of the devices are latency-sensitive.



(a) Experiment 1

(b) Experiment 2

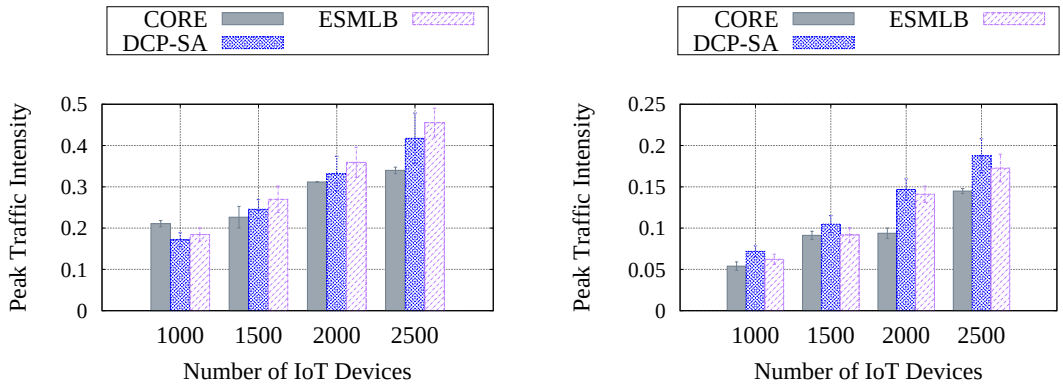
Figure 6.5: CORE: Control Plane Cost

6.3.4.3 Peak Traffic Intensity

Figure 6.6(a) shows that for 2500 devices, the peak traffic intensity of CORE is 18.66% and 25.27% less as compared to DCP-SA and ESMLB, respectively, for the first experiment. Figure 6.6(b) shows that CORE achieves 23.08% and 16.67% reduction in peak traffic intensity compared to DCP-SA and ESMLB, respectively, for the second experiment.

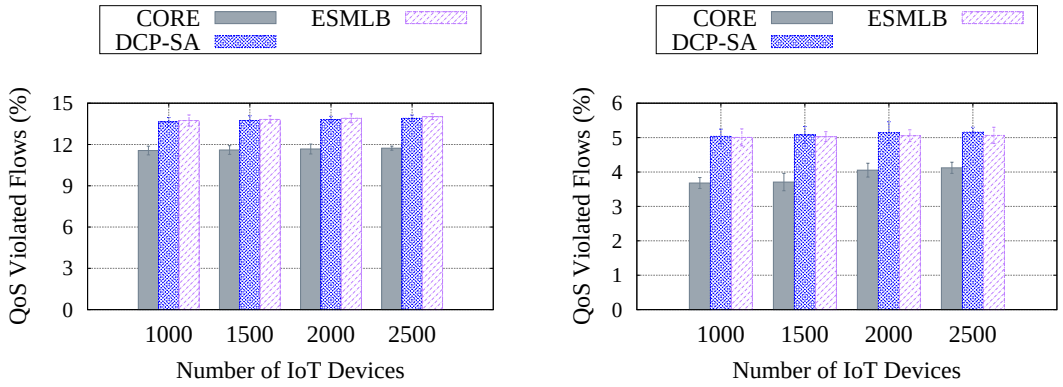
6.3.4.4 QoS Violated Flows

From Figure 6.7(a), we observe that the percentage of QoS violated flows is less for CORE even when the number of devices is high. Figure 6.7(b) shows the amount of QoS violated flows with high number of latency-sensitive devices. For this experiment, CORE achieves 23.73% and 22.82% better performance as compared to DCP-SA and ESMLB, respectively.



(a) Experiment 1 (b) Experiment 2

Figure 6.6: CORE: Peak Traffic Intensity



(a) Experiment 1 (b) Experiment 2

Figure 6.7: CORE: QoS Violated Flows

6.3.5 Discussion

From the simulation result, we observe that CORE significantly outperforms the benchmarks. The majority of the IoT flows are latency-sensitive, and CORE has low control plane cost for a high number of latency-sensitive devices. This is because the rule-caching module prioritizes latency-sensitive flows and reduces controller-switch communication. It is noteworthy that with less number of IoT devices, the peak traffic intensity of CORE is similar to the benchmark schemes. This is because, at a lower load, the control traffic is well-distributed across the controllers. However, IoT networks expect the presence of

a massive number of IoT devices, and CORE reduces the peak traffic intensity for a high number of IoT devices. Therefore, we deduce that CORE is more suitable for reducing control plane load in the IoT environment than the benchmark scheme.

6.4 Concluding Remarks

This chapter presents a prediction-based approach to reduce the control plane load in SDIoT. In this scheme, we designed rule-caching and master controller assignment algorithms considering heterogeneous attributes of IoT devices. Simulation results indicate that the proposed scheme reduces the average control plane cost for varying traffic load and varying QoS demand compared to the benchmarks. Specifically, for high traffic load, the average control plane cost decreased approximately by 46.94% and 9.82% as compared to DCP-SA and ESMLB, respectively.

Chapter 7

QoS-Aware Switch and Controller Placement

In this chapter, we propose a scheme for QoS-aware *Switch and Controller Placement (SCOPE)* in hybrid SDN. Hybrid SDN is an intermediate step of transforming a traditional backbone network into pure SDN. For hybrid SDN, QoS is a primary concern for ensuring service guarantee of a traditional network, while providing additional benefits of softwarization. The positions of SDN controllers determine the QoS parameters, such as network throughput and flow-processing delays. SCOPE addresses the joint switch and controller placement problem in hybrid SDN by (a) selecting legacy switches for upgrade, and (b) determining the locations and the number of controllers based on the upgraded switches.

This chapter consists of four sections. The system model of SCOPE is discussed in Section 7.1. Section 7.2 describes the proposed scheme. Section 7.3 depicts the experimental results. Finally, Section 7.4 concludes the proposed work.

7.1 System Model

The system includes a collection of service requests, a set of devices, and network elements. Figure 7.1 shows the hybrid SDN architecture. We consider a service request as a

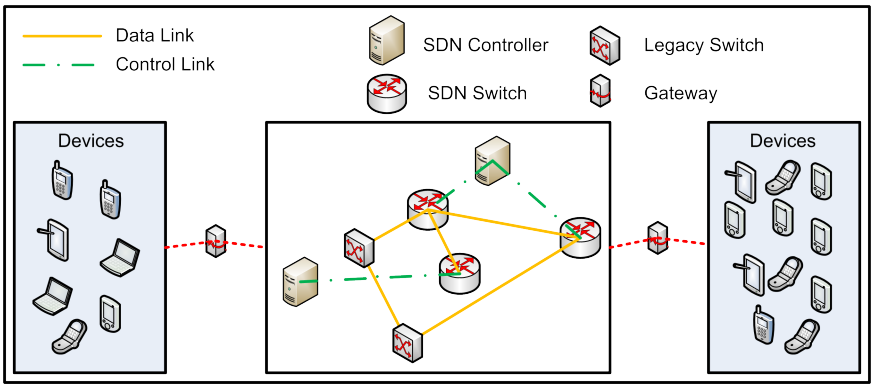


Figure 7.1: SCOPE: Hybrid SDN Architecture

flow denoted by f_i. A flow $f_i \in F$ is denoted by a tuple $< id(f_i), src(f_i), dest(f_i), rate(f_i) >$, where $id(f_i)$ represents the flow identification number, $src(f_i)$ denotes the source, $dest(f_i)$ is the destination, and $rate(f_i)$ is the traffic rate. A flow f_i is termed *new*, if no matching flow-rule is present in the ingress switch. For a time-slot t, the set of devices is denoted by $D(t) = \{d_1, d_2, \ldots, d_m\}$. Let $F_k(t)$ denote the set of service requests generated by d_k. We model a hybrid SDN as a connected graph $G(N, E)$ having a set of nodes $N = \mathcal{C} \cup S \cup \mathcal{R}$. The set $\mathcal{C} = \{c_1, c_2, \ldots c_a\}$ is the set of available locations for placing the controllers. The selection of a location for controller placement is expressed as:

$$CP(j,t) = \begin{cases} 1 & \text{if at time-slot } t, \text{ an active controller is present at location } c_j, \\ 0 & \text{otherwise.} \end{cases} \tag{7.1}$$

$S = \{s_1, s_2, \ldots s_b\}$ and $\mathcal{R} = \{r_1, r_2, \ldots r_n\}$ denote the sets of SDN and legacy switches, respectively. The legacy switches forward packets based on traditional routing protocols, including OSPF. In this work, we consider heterogeneous legacy switches having different

expected lifetime. At time-slot t, the controller-switch association is expressed as:

$$SC(i,j,t) = \begin{cases} 1 & \text{if } CP(j,t) = 1 \text{ and } c_j \text{ is the master controller of } s_i, \\ 0 & \text{otherwise.} \end{cases} \tag{7.2}$$

The weight of a data link $e_{ij} \in E$ is denoted by $LW(e_{ij})$, which specifies the priority of the link for the shortest path routing protocols such as OSPF. A data link having at least one SDN switch is termed as an SDN link. Otherwise, the data link is called a non-SDN link. Therefore, a traffic or flow is termed as *programmable* if it passes through at least one SDN link. Programmable traffic is expressed as:

$$\Upsilon(i) = \begin{cases} 1 & \text{if } f_i \in F \text{ is programmable,} \\ 0 & \text{otherwise.} \end{cases} \tag{7.3}$$

The number of Packet-In requests generated by $s_i \in S$ at time-slot t is given by:

$$\Psi_i(t) = \sum_{d_k \in D^i(t)} \sum_{f_l \in F_k(t)} (1 - p_{match}), \tag{7.4}$$

where $0 \leq p_{match} \leq 1$ is the rule matching probability, and $D^i(t) \subseteq D(t)$ is the set of devices sending service requests to s_i at time-slot t. The maximum number of Packet-In requests of s_i which are processed by the associated controller during time-slot t is $\Psi_i^{succ}(t) = \frac{1}{\delta_i(t)}$, where $\delta_i(t)$ is the average flow-setup delay for a flow originating from s_i at time-slot t. The flow-setup delay consists of — (1) switch-to-controller delay for transmitting the Packet-In request to the controller and receiving the new flow-rule from the controller ($\delta_i^{tr}(t)$), (2) queueing delay at the controller ($\delta_j^{que}(t)$), and (3) processing delay at the controller for deciding the forwarding path ($\delta^{pr}(t)$). Therefore, $\delta_i(t) = 2\delta_i^{tr}(t) + \delta_j^{que}(t) + \delta^{pr}(t)$. For simplicity, we assume that all controllers are homogeneous in terms of capacity and the maximum number of Packet-In requests handled by a controller

during a time-slot is Ω. For time-slot t, the number of Packet-In requests received by an active controller placed at location c_j is $\Omega_j(t) = \sum_{i=1}^{|S|} SC(i,j,t)\Psi_i(t)$. Therefore, the queueing delay at the corresponding controller is $\delta_j^{que}(t) = \frac{1}{\Omega - \Omega_j(t)}$. The time required for calculating single source route depends on the network size [66]. Therefore, the processing delay is $\delta^{pr}(t) = \frac{1}{\Omega} O(|S|^2)$.

Definition 19 (Effective SDN Throughput)**.** *The effective SDN throughput consists of the programmable service requests for which new flow-rules are installed as well as the service requests having matching flow-rules. Accordingly, we define the effective SDN throughput at time-slot t as:*

$$Th^{eff}(t) = \sum_{i=1}^{|S|} \Psi_i^{act}(t) + \left(\sum_{d_k \in D^i(t)} |F_k(t)| - \Psi_i(t) \right), \tag{7.5}$$

where $\Psi_i^{act}(t) \leq \Psi_i^{succ}(t)$ is the actual number of processed service requests that have no matching flow-rules and $\left(\sum_{d_k \in D^i(t)} |F_k(t)| - \Psi_i(t) \right)$ is the number of service requests having matching flow-rules.

7.1.1 Budget Model

For hybrid SDN, the number of upgraded switches and placed controllers depend on the available upgrade budget. Let $\mathbb{B}$ denote the total budget for upgrading a traditional network to SDN and T denote the number of time-slots allocated for the upgrade process. Therefore, the upgrade budget is $\mathbb{B} = \mathbb{B}^s + \mathbb{B}^c$, where $\mathbb{B}^s$ is the budget allocated for replacing all legacy switches with SDN switches and $\mathbb{B}^c$ is the budget allocated for the installation of controllers. We consider that the switch upgrade budget at each time-slot is different. Let $\mathbb{B}^{s(t)}$ denote the switch upgrade budget and $\mathbb{B}^{c(t)}$ denote the controller placement budget for time-slot $t \leq T$. Additionally, we consider that replacing a legacy switch with an SDN switch costs $\theta_{s,t}$ unit at time-slot $t \leq T$ and placement of a controller

costs $\theta_{c,t}$ unit at time-slot $t \leq T$. The upgrade of a legacy switch r_j is expressed as:

$$v_{jt} = \begin{cases} 1 & \text{if } r_j \in \mathcal{R} \text{ upgrades at time-slot } t, \\ 0 & \text{otherwise.} \end{cases} \tag{7.6}$$

Therefore, the consumed switch upgrade budget at time-slot t is $\sum_{j=1}^{|\mathcal{R}|} v_{jt}\theta_{s,t}$. The placement of a controller at location c_j is expressed as:

$$v_{jt}^{'} = \begin{cases} 1 & \text{if } CP(j,t) = 1 \text{ and } CP(j,t-1) = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{7.7}$$

Therefore, the consumed controller installation budget at time-slot t is $\sum_{j=1}^{|\mathcal{C}|} v_{jt}^{'}\theta_{c,t}$.

7.1.2 Problem Formulation

Objective 1 The first objective of this work is finalizing the switch upgrade policy which maximizes the programmable traffic. The switch upgrade policy specifies a set of to-be-upgraded legacy switches for each time-slot. Therefore, we formulate the switch placement problem (SPP) as:

$$\underset{v}{\text{Maximize}} \quad \sum_{i=1}^{|F|} \Upsilon(i) \tag{7.8}$$

subject to

$$\sum_{t \in T} v_{jt} \leq 1, \forall r_j \in \mathcal{R}, \tag{7.9}$$

$$\sum_{j=1}^{|\mathcal{R}|} v_{jt}\theta_{s,t} \leq \mathbb{B}^{s(t)}, \forall t \in T, \tag{7.10}$$

$$\sum_{t \in T} \mathbb{B}^{s(t)} \leq \mathbb{B}^{s} \tag{7.11}$$

Equation (7.9) ensures that a legacy switch is upgraded only once. Equation (7.10)

ensures that the total expense for switch upgrade does not exceed the switch upgrade budget for each time-slot. Equation (7.11) ensures that the total upgrade expenditure is within the switch upgrade budget. The inputs of the SPP are $\mathcal{R}$, t, $\theta_{s,t}$, and $\mathbb{B}^{s(t)}$.

Theorem 7. *SPP is NP-hard.*

Proof. To prove the NP-hardness of SPP, we reduce the $0-1$ knapsack problem, which has been proven as NP-hard, to SPP. The $0-1$ knapsack problem involves a set of items so that each item has a weight and a value. The goal is to add items in a knapsack of fixed capacity so that the total value is the maximum. However, the decision for including an item in a knapsack is binary, i.e., an item can be added to the knapsack as a whole or not added at all.

Let us consider a specific instance of the SPP by limiting the number of time-slots T to unity. We reduce an instance of the $0-1$ knapsack problem to this instance of SPP. In this case, each item in the $0-1$ knapsack problem refers to a legacy switch $r_j \in \mathcal{R}$. The weight and value of each item correspond to the upgrade cost $\theta_{s,1}$ and the traffic volume $Vol(r_j)$ that traverses the switch r_j, respectively. The capacity of the knapsack is mapped to the total switch upgrade budget $\mathbb{B}^{s(1)}$. In SPP, the value of the decision variable v_{j1} is restricted to 1 or 0, depending on whether $r_j \in \mathcal{R}$ is selected for upgrade or not. The goal of the SPP is to find a feasible solution that maximizes the total programmable traffic without exceeding the switch upgrade budget. Therefore, the optimal solution to the $0-1$ knapsack problem is also the optimal solution of the SPP. Hence, the SPP is also NP-hard.

□

As SPP is NP-hard, we propose a greedy algorithm that computes priorities of the legacy switches and selects the switches accordingly.

Objective 2 The second objective of this work is finalizing the controller placement policy which maximizes the effective SDN throughput. The controller placement pol-

icy specifies controller locations, given the set of upgraded switches in each time-slot. Therefore, we formulate the QoS-aware controller placement problem (QCPP) as:

$$\underset{v'}{\text{Maximize}} \quad \sum_{t=1}^{T} Th^{eff}(t) \tag{7.12}$$

subject to

$$CP(j,t)\Omega_j(t) < \Omega, \forall c_j \in \mathcal{C}, \forall t \in T, \tag{7.13}$$

$$\sum_{j=1}^{|\mathcal{C}|} v'_{jt}\theta_{c,t} \leq \mathbb{B}^{c(t)}, \forall t \in T, \tag{7.14}$$

$$\sum_{t \in T} \mathbb{B}^{c(t)} \leq \mathbb{B}^{c}, \tag{7.15}$$

$$\sum_{j=1}^{|\mathcal{C}|} SC(i,j,t) = 1, \forall s_i \in S, \forall t \in T, \tag{7.16}$$

$$\delta_i(t) \leq \delta^{max}, \forall s_i \in S, \forall t \in T, \tag{7.17}$$

where Equation (7.13) states the controller capacity constraint. Equation (7.14) ensures that the total expense for controller placement does not exceed the controller placement budget for each time-slot. Equation (7.15) ensures that the total expenditure for controller placement is within the controller installation budget. Equation (7.16) expresses that each SDN switch has single controller. Equation (7.17) states the QoS requirement of a service request in terms of flow-setup delay, where δ^{max} denotes the maximum allowable delay. The inputs of the QCPP are $\mathcal{C}$, t, and $S'(t) \in S$ that denotes the set of upgraded switches in time-slot t.

Theorem 8. *QCPP is NP-hard.*

Proof. To prove the NP-hardness of QCPP, we reduce the well-known facility location problem to QCPP. The facility location problem, which has been proven as NP-hard, involves a set of potential locations for opening a facility. In addition, there exists a set of demand points. The goal of the problem is to find a set of locations to open facilities which minimizes the distance of each demand point to the nearest facility and the total

facility opening cost.

Let us consider a particular instance of QCPP by limiting the number of time-slots T to unity. In this case, we consider each potential controller location $c_j \in \mathcal{C}$ as a facility location. The installation of a controller at a potential location costs $\theta_{c,1}$, given the total controller placement budget $\mathbb{B}^{c(1)}$. The demand points are the SDN switches. For the unmatched service requests, SDN switches send Packet-In messages to the connected controllers. The flow-setup delay of a service request depends on the switch-to-controller distance. The goal of QCPP is to find a set of controller locations for placing active controllers which maximizes the overall utility for each device by minimizing the switch-to-controller distance for each SDN switch without exceeding the controller placement budget and controller capacity. Therefore, the optimal solution of the facility location problem is also the optimal solution of QCPP. Hence, QCPP is also NP-hard. □

As QCPP is NP-hard, we propose a coalition game-based algorithm which forms coalitions of SDN switches to select the locations for the placement of controllers.

7.2 SCOPE: The Proposed Scheme

The process of transforming a traditional network into a pure SDN involves multiple rounds or time-slots. In each time-slot, a new set of legacy switches are swapped with SDN switches. Based on the current set of SDN switches, we formulate a coalition game to determine the placement policy of new SDN controllers.

7.2.1 SDN Switch Placement

For SDN switch placement, we design a priority-based algorithm which assigns priority values to the legacy switches and selects switches for upgrade considering the upgrade budget. The priority value of a legacy switch $r_j \in \mathcal{R}$ depends on the following parameters:

1. **Number of Non-SDN Links**: Replacing a legacy switch having more number of non-SDN links results into higher programmable traffic.

2. **Traffic Volume**: Priority of a switch is directly proportional to the traffic volume that traverses the switch. This is because the upgrade of heavily used switches produces higher programmable traffic.

3. **Link Weightage**: A switch is likely to be a part of the shortest route if it has the lowest average weight for the adjacent links. Upgrading a legacy switch which is a part of the shortest route increases programmable traffic. Therefore, the priority of a switch is inversely proportional to the average link weightage.

4. **Residual Lifetime of the Switch**: The expected lifetime of a legacy switch is 3 to 5 years and these switches are very expensive [67]. Moreover, the initial upgrade cost is directly proportional to the residual lifetime of a legacy switch because the switch upgrade cost decreases over time-slots [67]. Therefore, replacing a legacy switch which has a high residual lifetime is not cost-effective.

Definition 20 (Legacy Switch Utilization)**.** *The utilization of each legacy switch r_j before upgrade is defined as:*

$$SU_j = \frac{\mathcal{T}^c(r_j)}{\mathcal{T}(r_j)}, \tag{7.18}$$

where $\mathcal{T}(r_j)$ is the total lifespan of r_j and $\mathcal{T}^c(r_j) \leq \mathcal{T}(r_j)$ denotes the consumed lifespan of r_j.

Definition 21 (Priority of a Legacy Switch)**.** *The priority of a legacy switch r_j is:*

$$PR(r_j) = \quad z_1 NS(r_j) + z_2 \frac{Vol(r_j)}{Vol} + z_3 \frac{1}{\overline{W}(r_j)} + z_4 SU_j, \tag{7.19}$$

where $NS(r_j)$ denotes the number of non-SDN links for r_j, $Vol(r_j)$ is the average traffic that traverses r_j, Vol is the total traffic volume of the network, $\overline{W}(r_j)$ is the

average weight of the links of r_j*, and* $z_i \in [0,1]$ *terms denote user-defined weighting constants.*

Definition 22 (Switch Upgrade Budget)**.** *The switch upgrade budget for time-slot* $t \leq T$ *is:*

$$\mathbb{B}^{s(t)} = \begin{cases} 100\frac{1}{\xi^t} & \text{if } t < T, \\ 100 - \sum_{t=1}^{T-1} \mathbb{B}^{s(t)} & \text{otherwise,} \end{cases} \tag{7.20}$$

where $\xi > 1$ *is a constant that controls the budget allocation in each time-slot.*

Algorithm 7.1: SCOPE: SDN Switch Placement Algorithm

Inputs : $G(N, L)$, t, $\theta_{s,t}$
Output: $\{S^{'}(t), v\}$: Upgraded switches
Compute $\mathbb{B}^{s(t)}$ using Equation (7.20) and set $\mathbb{B}^{con}_{s,t} \leftarrow 0$
while $\mathbb{B}^{con}_{s,t} < \mathbb{B}^{s(t)}$ **do**
  Select the maximum priority switch $r_j \in \mathcal{R}$
  Set $S \leftarrow S \cup \{r_j\}$, $v_{jt} \leftarrow 1$, $S^{'}(t) \leftarrow S^{'}(t) \cup \{r_j\}$, $\mathcal{R} \leftarrow \mathcal{R} - \{r_j\}$, $\mathbb{B}^{con}_{s,t} \leftarrow \mathbb{B}^{con}_{s,t} + \theta_{s,t}$
end
foreach $r_j \in S^{'}(t)$ **do**
  Set $LW(e_{ij}) \leftarrow \frac{LW(e_{ij})}{2}$, $\forall r_i \in \mathcal{R}$
end
return $\{S^{'}(t), v\}$

Algorithm 7.1 describes the SDN switch placement process at time-slot $t \leq T$. The *SDN Switch Placement Algorithm* (SSPA) selects the legacy switches in priority order without exceeding the specified upgrade budget. After completion of each upgrade schedule, SSPA decreases the weights of the new SDN links by half to redirect more traffic through SDN links.

7.2.2 Coalition Game Formulation for Controller Placement

Coalition formation game is a form of distributed cooperation algorithm which is used for a wide variety of network problems such as fair rate allocation in an interference channel,

energy-aware cooperation in routing protocols, and resource allocation [42]. In SDN architecture, a group of SDN switches is connected to each controller. So, the capacity of a controller is shared by the connected switches. Therefore, each SDN switch behaves cooperatively and decides its optimum strategy to achieve Pareto optimal distribution of controller capacity. Hence, a coalition formation game approach is the most appropriate approach for the placement of controllers in hybrid SDN, where the newly placed SDN switches form cooperative groups to select suitable controller locations. We formulate a coalition game with non-transferable utility (NTU) because each player's utility depends on the joint actions chosen by the other players in the coalition. In this game, SDN switches act as rational players who decide the preferable coalition for them. Each coalition represents a set of switches. The switches in a coalition are associated with a controller location. The proposed game ensures that each player is part of precisely one coalition at any time. The main components of the proposed game are as follows:

- The SDN switches in set S are the players of the game.
- The strategy of each switch $s_i \in S$ corresponds to the flow processing rate that is the maximum amount of processed service requests $\Psi_i^{succ}(t)$ in a time-slot t.
- The utility function u_i represents the benefit resulted from the choice of s_i.

Definition 23 (Pseudo-Price Coefficient)**.** *At time-slot t, the pseudo-price coefficient for c_j is defined as:*

$$\alpha_j(t) = \left(1 - \frac{\Omega_j^{res} \left(\mathbb{B}^{c(t)} - \sum_{j=1}^{|\mathcal{C}|} v_{jt}^{'} \theta_{c,t} \right)}{\Omega \mathbb{B}^{c(t)}} \right), \tag{7.21}$$

where $\Omega_j^{res} \geq 0$ is the residual capacity of the controller placed at location c_j.

The value Ω_j^{res} is determined based on the demand of the associated SDN switches. Mathematically, $\Omega_j^{res} = \Omega - \Omega_j(t)$.

7.2.2.1 Utility Function of a Coalition

At time t, each switch $s_i \in S$ uses its utility function $u_i(\cdot)$ to determine its optimal coalition which in turn determines the location for the master controller. In particular, $u_i(\cdot)$ is expresses the willingness of s_i to be in a coalition. Let, A_k denote the k^{th} coalition which is associated with a controller location c_j. The utility function $u_i(\cdot)$ for coalition $A_k(t)$ must satisfy the following properties:

1. Each SDN switch s_i tries to maximize the flow processing rate and we refer this number $\Psi_i^{succ}(t)$ as the demand of s_i. So, the utility function of the SDN switches is formulated as a non-decreasing function. Mathematically,

$$\frac{\partial u_i(\cdot)}{\partial \Psi_i^{succ}(t)} \geq 0 \tag{7.22}$$

2. The utility of a switch s_i is inversely proportional to the switch-to-controller delay $\delta_i^{tr}(t)$. Therefore, we get:

$$\frac{\partial u_i(\cdot)}{\partial \delta_i^{tr}(t)} < 0 \tag{7.23}$$

3. The utility value decreases if the pseudo-price coefficient $\alpha_j(t)$ for c_j increases. Mathematically,

$$\frac{\partial u_i(\cdot)}{\partial \alpha_j(t)} < 0 \tag{7.24}$$

Therefore, we formulate the utility function of an SDN switch s_i as a concave function, which is represented as follows:

$$u_i(\cdot) = \sum_{d_k \in D^i(t)} |F_k(t)| \log \left(\Psi_i^{succ}(t) + \frac{\delta^{max} - \delta_i^{tr}(t)}{\delta^{max}} - \alpha_j(t) \right), \tag{7.25}$$

where $\Psi_i^{succ}(t) \in \left[0, max\left(\Psi_i(t), \Omega - \sum_{x=1, x\neq i}^{|A_k(t)|} \Psi_x^{succ}(t)\right)\right]$. Therefore, the utility of a coalition $A_k(t)$ is given by:

$$U\left(A_k(t), c_j\right) = \sum_{i=1}^{|A_k(t)|} u_i(\cdot) \tag{7.26}$$

The utility function conforms to the objective stated in Equation (7.12) as $Th^{eff}(t)$ depends on $\Psi_i^{succ}(t)$ and $\alpha_j(t)$ addresses the controller capacity and budget constraints.

Definition 24 (Coalition Structure)**.** *A coalition structure V_w is defined as:*

$$V_w(t) = \{A_1(t), A_2(t), \ldots, A_m(t)\}, \tag{7.27}$$

where $\bigcup_{k=1}^{z} A_k(t) = S$, $A_i(t) \cap A_j(t) = \phi$, $\forall i \neq j$, and m denotes the total number of coalitions for $V_w(t)$.

The total number possible coalition structures for m coalitions is calculated using the Bell number, which is expressed as:

$$\Gamma_m = \sum_{q=0}^{m-1} \binom{m-1}{q} \Gamma_q, \tag{7.28}$$

where $m \geq 1$ and $\Gamma_0 = 1$.

Definition 25 (Stable Coalition)**.** *A coalition $A_k(t) \in V_w(t)$ is stable if*

1. *no player s_i can improve its utility by leaving its coalition $A_k(t)$ and acting individually.*

2. *no other coalition $A_l(t) \in V_w(t)$ can improve its utility by joining $A_k(t)$.*

Definition 26 (Stable Coalition Structure)**.** *A coalition structure $V_w(t)$ is stable if $A_i \in V_w(t), \forall i \in [1, m]$ is stable.*

We consider that the proposed coalition formation game is *hedonic*, which implies that a player has a preference for the choice of coalition.

Definition 27 (Preference Relation)**.** *The relation* $V_p(t) \succ_{S_a} V_{\mathbb{B}}$ *denotes that the way* $V_p(t)$ *partitions* S_a *is preferred to the way* $V_q(t)$ *partitions* S_a*, where* $S_a \subseteq S$ *is a set of players.*

The coalitions are updated periodically based on merge and split rules.

Definition 28 (Merge Rule)**.** *Merge any set of coalitions* $\{A_1(t), A_2(t), \ldots, A_k(t)\}$*, where* $\{\bigcup_{i=1}^{k} A_i(t)\} \succ_{S_a} \{A_1(t), A_2(t), \ldots, A_k(t)\}$*,* $S_a = \bigcup_{i=1}^{k} A_i(t)$*. Therefore,* $\{A_1(t), A_2(t), \ldots, A_k(t)\} \rightarrow \bigcup_{i=1}^{k} A_i(t)$*.*

Definition 29 (Split Rule)**.** *Split any set of coalitions* $\bigcup_{i=1}^{k} A_i(t)$*, where* $\{A_1(t), A_2(t), \ldots, A_k(t)\} \succ_{S_a} \{\bigcup_{i=1}^{k} A_i(t)\}$*,* $S_a = \bigcup_{i=1}^{k} A_i(t)$*. Therefore,* $\bigcup_{i=1}^{k} A_i(t) \rightarrow \{A_1(t), A_2(t), \ldots, A_k(t)\}$*.*

Let $A_p(t)$ and $A_q(t)$ be two coalitions having associated controller locations c_e and c_f, respectively. The associated controller location for a merged coalition $A_p(t) \cup A_q(t)$ is:

$$c_{pq} = \begin{cases} c_e & \text{if } U\left(A_p(t) \cup A_q(t), c_e\right) \geq U\left(A_p(t) \cup A_q(t), c_f\right), \\ c_f & \text{otherwise.} \end{cases} \tag{7.29}$$

Let $A_l(t)$ be a coalition having an associated controller location c_g. Let $A_l(t)$ is split into two coalitions $A_p(t)$ and $A_q(t)$ having associated controller locations c_e and c_f, respectively. Mathematically,

$$c_e = \begin{cases} c_g & \text{if } U\left(A_p(t), c_g\right) \geq U\left(A_q(t), c_g\right), \\ c_x & \text{otherwise,} \end{cases} \tag{7.30}$$

and

$$c_f = \begin{cases} c_g & \text{if } U\left(A_p(t), c_g\right) < U\left(A_q(t), c_g\right), \\ c_x & \text{otherwise,} \end{cases} \tag{7.31}$$

where $c_x \in \mathcal{C} \setminus \{c_g\}$ is the nearest controller location and $CP(x) = 0$. If $c_e = c_x$ or $c_f = c_x$, we update $CP(x) = 1$.

Algorithm 7.2: SCOPE: Coalition Formation Algorithm

Inputs : $G(N, L)$, t, $S^{'}(t) \subseteq S$, $\mathcal{C}$
Output: $V_w^*(t)$: Stable coalition structure

```
if t == 1 then
    Form coalition A_j(t) for each controller location c_j ∈ C
end
foreach s_i ∈ S'(t) do
    Select the nearest controller location c_j with CP(j,t) = 1
    if Ω_j^res ≥ Ψ_i(t) then
        A_j(t) ← A_j(t) ∪ {s_i}
    else
        Select the nearest c_j with CP(j,t) = 0
        A_j(t) ← A_j(t) ∪ {s_i}
    end
end
Add the non-empty coalitions to initial coalition structure V_w(t)
while V_w(t) is not stable do
    Form new coalition structure using Merge and Split rules
end
V_w^*(t) ← V_w(t)
Update CP(j,t) for each c_j ∈ C
return V_w^*(t)
```

7.2.2.2 Coalition Formation

SDN switches decide their strategies to form an optimal stable coalition structure or equivalently the locations for placing active controllers. Algorithm 7.2 describes the process of forming a stable coalition structure. Each coalition is a non-empty subset of S, having attached to single controller location. For the first time-slot, an empty coalition is associated with each controller location. In each time-slot, each newly placed SDN

switch selects the nearest controller location having an active controller and enough residual capacity to handle the service requests. If no such active controller is available, the corresponding switch selects the nearest controller location, which does not have an active controller. Stable coalition structure is achieved using merge and split rules, as mentioned in Definitions (28) and (29), respectively. Finally, the controller locations with active controllers are determined based on the stable coalition structure.

Algorithm 7.3: SCOPE: Controller Placement Algorithm

Inputs : $\mathcal{C}$, t, $\theta_{c,t}$, $\mathbb{B}^{c(t)}$, $V_w^*(t)$
Output: $v^{'}$

$\mathbb{B}_{c,t}^{con} \leftarrow 0$
while $\mathbb{B}_{c,t}^{con} < \mathbb{B}^{c(t)}$ **do**
  Select the coalition $A_k(t) \in V_w^*(t)$ having maximum utility
  Select the associated controller location c_j
  Set $CP(j,t) = 1$, $v_{jt}^{'} = 1$, $\mathbb{B}_{c,t}^{con} = \mathbb{B}_{c,t}^{con} + \theta_{c,t}$
end
return $v^{'}$

7.2.2.3 Controller Placement

Algorithm 7.3 describes the process of controller placement after Coalition Formation Algorithm (CFA) computes a stable coalition partition. The Controller Placement Algorithm (CPA) checks the availability of budget and selects the controller location associated with the coalition which has the maximum utility. Accordingly, a controller is placed in the selected location and the controller placement budget is updated.

7.3 Performance Evaluation

7.3.1 Simulation Settings

We evaluate the performance of SCOPE by performing two experiments for each objective. In the first experiment, we perform the simulations on the Abilene dataset [67],

with 30 directed links and 12 switches to evaluate the performance of the proposed switch placement scheme. The Abilene dataset records the data transferred between each pair of nodes in every 5 minutes for six months. For the simulation, we consider the traffic matrix of 144 flows from the dataset of Day 1, 8:00 pm. We use Abilene dataset because it is publicly available, and it provides an accurate description of the network setup required for the first objective. Abilene is a small-scale topology with limited network traffic. However, the performance evaluation of the proposed controller placement scheme requires large-scale topology to assess the impact of high network traffic on the control plane load. Therefore, in the second experiment, we perform simulations on a large-scale topology, an 8-pod Fat-tree topology [60], having 80 switches. The simulation parameters are depicted in Table 7.1.

Table 7.1: SCOPE: Simulation Parameters

Parameter	Value
Network topology	Abilene [45], Fat-tree [60]
Number of switches	12 (Abilene) [67], 80 (Fat-tree)
Number of flows	144 (Abilene) [67], $0.1 - 0.5$ million (Fat-tree)
Number of time-slots	$1 - 5$
Duration of each time-slot	1 year [67]
Lifetime of a legacy switch	[3, 4, 5] years [67]
Initial switch upgrade cost	[$36K, $60K, $100K] [67]
Switch upgrade budget	$200K-$1M [67]
Traffic rate increment	22% per year [67]
Switch upgrade cost decrement	40% per year [67]
ξ	2
Initial controller placement cost	$1465 [26]
Controller placement cost decrement	10% per year
Controller placement budget	$10K-$30K
Controller capacity	$0.02 - 0.03$ mfps [63]
Maximum allowable delay	$0.001 - 1$ s [46]

In the first experiment, we vary switch upgrade budget and the number of time-slots to analyze the effect on the performance metrics. In the second experiment, we vary the number of flows, controller placement budget, the number of time-slots, and controller

capacity because these are the significant parameters that affect the performance metrics. In each subset of the second experiment, we vary only one parameter while keeping the other three parameters static. For each static parameter, we set the number of flows, controller placement budget, the number of time-slots, and the controller capacity as 0.5 million, $30K, 5, and 0.03 mfps, respectively.

7.3.2 Benchmark Schemes

We compare the switch upgrade performance of SCOPE with DEG, VOL [68], and Local Search [67]. DEG upgrades legacy switches in decreasing order of the number of adjacent links. VOL upgrades switches in decreasing order of traffic volume that traverses the switch. Local Search maximizes the volume of programmable traffic by selecting locally feasible solutions. On the other hand, for SDN switch placement, SCOPE considers heterogeneous parameters such as the number of non-SDN links, traffic volume, link weightage, and the residual lifetime of a legacy switch. We select the aforementioned schemes as the benchmarks to highlight the efficacy of SCOPE, where the parameter domain is more holistic. To the best of our knowledge, there exists no incremental controller placement scheme exclusively for hybrid SDN. Therefore, we select the benchmarks based on the existing controller placement schemes in pure SDN, which is a subset of hybrid SDN. We compare the controller placement performance of SCOPE with LiDy+ [26] and Greedy. LiDy+ activates or deactivates controllers in each controller module based on dynamic traffic load. The Greedy approach activates the minimum number of controllers conforming to the available budget, and each switch selects the nearest active controller as the master controller. However, in SCOPE, SDN switches cooperatively decide the locations for placing active controllers so that the effective SDN throughput is the maximum.

7.3.3 Performance Metrics

- **Programmable traffic**: We evaluate the programmable traffic achieved by each upgrade scheme. This metric quantifies the amount of network upgrade.
- **Legacy switch utilization**: This metric shows the effectiveness of SCOPE in terms of the usage of expensive legacy switches.
- **Effective SDN throughput**: High effective SDN throughput signifies large number of processed programmable service requests. Therefore, high effective SDN throughput is one of the preferable criteria of IoT networks which is loss-sensitive.
- **QoS violated flows**: QoS violated flows are the flows which do not satisfy the maximum allowable delay requirement. This metric shows the efficiency of SCOPE in terms of QoS.

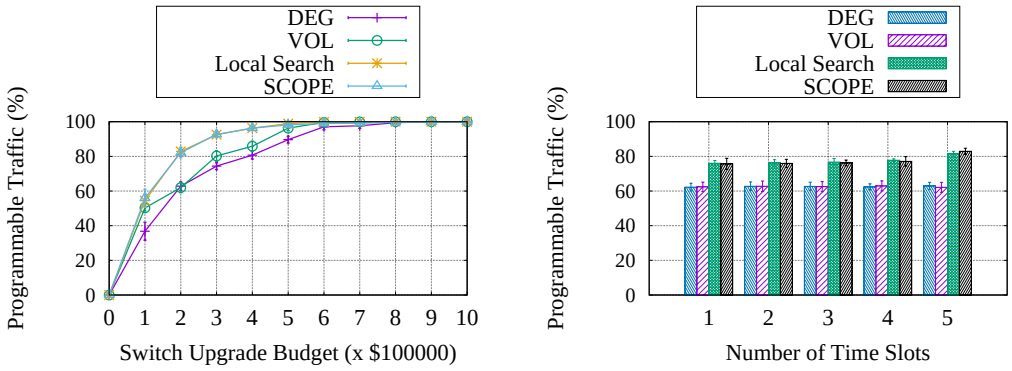
In the first experiment, we measure programmable traffic and legacy switch utilization. In the second experiment, we measure effective SDN throughput and QoS violation.

7.3.4 Result and Discussion

7.3.4.1 Programmable Traffic

Figure 7.2(a) shows the programmable traffic achieved by each scheme for different switch upgrade budgets. For this simulation, we set the number of time-slots as 1 and vary the switch upgrade budget from \$0K to \$1M. From the simulation result, we observe that SCOPE performs 10.08% and 5.59% better than DEG and VOL, respectively. For SCOPE, the amount of programmable traffic is high because SCOPE reduces the OSPF weights of the SDN links so that more flows are forwarded thorough the SDN links. Figure 7.2(b) shows the programmable traffic achieved by each scheme for different numbers of time-slots. For this simulation, we set the switch upgrade budget as \$200K and vary the number of time-slots from 1 to 5. From the simulation result, we observe

that SCOPE performs better than DEG and VOL because DEG and VOL upgrade a limited number of switches as all upgrades are performed in the first time-slot. On the other hand, SCOPE increases the programmable traffic by upgrading more switches in each subsequent time-slot.



(a) Effect of Budget

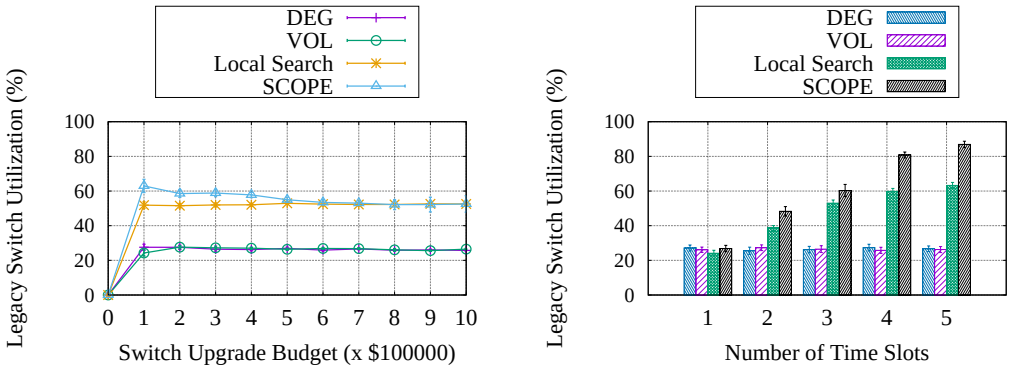
(b) Effect of Upgrade Duration

Figure 7.2: SCOPE: Programmable Traffic

7.3.4.2 Legacy Switch Utilization

To estimate legacy switch utilization, we set the weighting constant z_4 in Equation (7.19) to a value higher than other weighing constants. Figure 7.3(a) shows the legacy switch utilization for different switch upgrade budgets. For this simulation, we set the number of time-slots as 3. From the simulation result, we observe that SCOPE performs better for low budget conditions. A low budget allows the upgrade of less number of legacy switches. In this case, SCOPE priorities switches, which have a less residual lifetime. Moreover, in SCOPE, legacy switch utilization reduces with the increasing budget because, with more budget, more switches are upgraded even if their consumed lifetime is less. Figure 7.3(b) shows the legacy switch utilization for different number of time-slots. For this simulation, we set the switch upgrade budget as $100K. From the simulation result, we observe that SCOPE performs better than the benchmark schemes, and SCOPE's performance improves with an increasing number of time-slots. In the case

of larger time-slots, SCOPE upgrades legacy switches with a lesser residual lifetime in the early stages of the upgrade process, and legacy switches with higher residual lifetimes are upgraded at the final stages. Therefore, the consumed lifetime in SCOPE is higher for a larger number of time-slots.



(a) Effect of Budget

(b) Effect of Upgrade Duration

Figure 7.3: SCOPE: Legacy Switch Utilization

7.3.4.3 Effective SDN Throughput

From Figure 7.4, we observe that SCOPE performs better than the benchmarks in terms of effective SDN throughput. This is because SCOPE optimizes the maximum number of processed service requests $\Psi_i^{succ}(t)$ for each SDN switch $s_i \in S$. Figure 7.4(a) shows that the performance of the Greedy approach and LiDy+ degrades with an increasing number of flows. For 0.5 million flows, SCOPE performs 14.76% and 3.72% better than LiDy+ and Greedy, respectively. The Greedy approach is not scalable, as SDN switches select master controllers based on switch-to-controller delay only, and some controllers experience a high queueing delay when the network traffic is high. LiDy+ aims to increase the switch count per active controller. Therefore, for high traffic load, queueing delay of the controllers is high, and the effective SDN throughput is low. From Figure 7.4(b), we observe that effective SDN throughput increases with the increasing budget for all schemes. However, in this case, SCOPE performs better than the benchmarks

even for low budget.

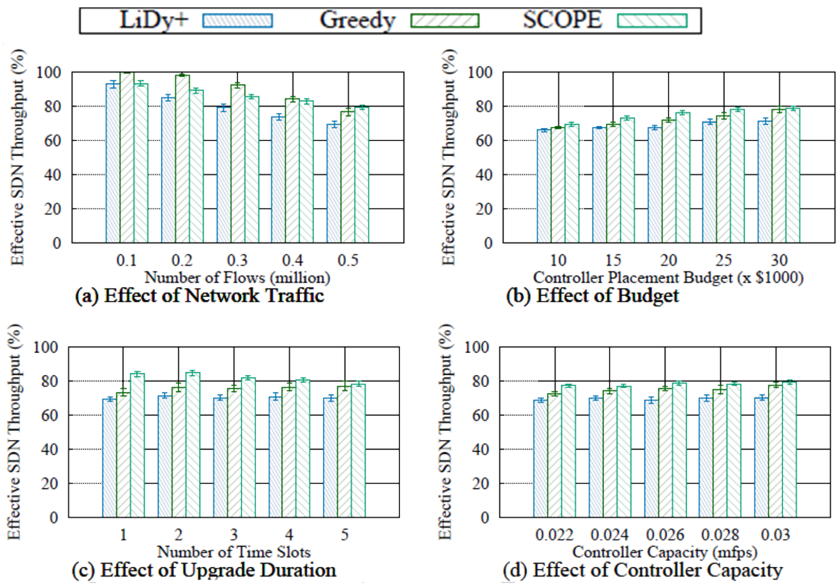


Figure 7.4: SCOPE: Effective SDN Throughput

7.3.4.4 QoS Violated Flows

From Figure 7.5(a), we observe that for a higher number of flows, the performance of SCOPE improves more than the benchmark schemes. For low network traffic, Greedy performs better than SCOPE because Greedy selects the nearest active controller, which reduces delay. However, with the increase of network traffic, control plane delay increases in Greedy, and a significant number of flows fail to meet the latency bound. Figure 7.5(b) depicts that the number of QoS violated flows is less in SCOPE than the benchmark schemes for different controller placement budget. SCOPE performs better even for a low budget. We observe that the QoS violation stabilizes as the controller placement budget reaches $25K. This signifies that the number of active controllers installed within this budget is sufficient to address the QoS requirements of the service requests. Therefore, in this case, $25K is the QoS-optimal controller placement budget for SCOPE. From Figure 7.5(c), we observe that QoS violation is less in SCOPE than the benchmarks

irrespective of the number of time-slots. In each time-slot, SCOPE refines the stable coalition structure to address the QoS requirements of more service requests or flows. Therefore, SCOPE has a steady performance even for large upgrade duration. Figure 7.5(d) shows that the number of QoS violated flows for different controller capacity is approximately 51.28% and 24.43% less than LiDy+ and Greedy, respectively. Additionally, we observe that the performance of SCOPE is uniform for different controller capacity. This is because SCOPE forms the coalitions based on the pseudo-price coefficient of the controller locations, which is formulated using the controller capacity.

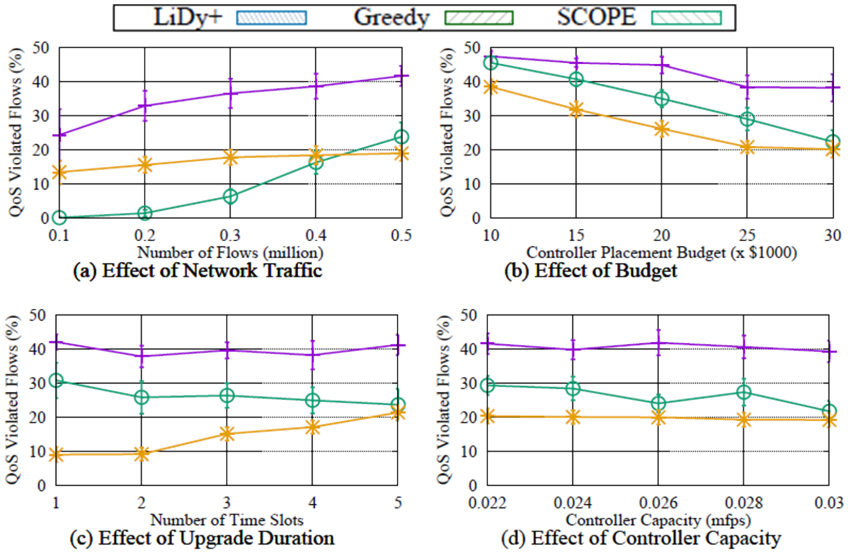


Figure 7.5: SCOPE: QoS Violated Flows

7.4 Concluding Remarks

This chapter presents a cost-efficient QoS-aware switch and controller placement approach for hybrid SDN. The proposed solution prioritizes the provision of QoS-guaranteed service to the users in the presence of dynamic network traffic and restricted upgrade budget. The proposed scheme increases the legacy switch utilization and the effective SDN throughput compared to benchmarks. We compared the proposed scheme, SCOPE, with

existing solutions. If the number of time slot is 5, SCOPE increases the legacy switch utilization approximately by 37.40% as compared to Linear Search. In addition, for 0:5 million flows, SCOPE increases the effective SDN throughput approximately by 14.76% and 3.72% as compared to LiDy+ and Greedy, respectively.

Chapter 8

Energy-Aware Traffic Engineering

In this chapter, we present an *energy-aware traffic engineering scheme in hybrid SDN (ETHoS)*. The lack of centralized control over the power states of legacy switches impedes energy-aware traffic engineering in hybrid SDN. On the other hand, there exists a trade-off between energy-aware routing and programmable traffic as traffic rerouting may transform programmable traffic to a non-programmable one, if not rerouted carefully. In this paper, we propose a scheme for dynamic activation of SDN links and optimal route selection of existing flows. Different from previous works, we focus on reducing energy consumption while maximizing the programmable traffic as it is the primary purpose of transforming a legacy network to an SDN.

This chapter consists of four sections. The system model of ETHoS is presented in Section 8.1. Section 8.2 describes the proposed scheme. Section 8.3 depicts the experimental results. Finally, Section 8.4 concludes the proposed work.

8.1 System Model

We consider a hybrid SDN environment consisting of multiple controllers and both legacy IP switches and SDN switches. The link between a controller and an SDN switch is termed as a control link. On the other hand, the data links are the links between SDN

switches and IP switches. The data links are categorized as SDN links and non-SDN links. A data link is an SDN link if it connects at least one SDN switch. Otherwise, the link is termed as a non-SDN link. A flow is termed as *programmable traffic* if it passes through at least one SDN link. Controllers have direct access to the SDN switches and SDN links [44]. Therefore, only SDN switches and SDN links can be turned off to reduce energy consumption. Traffic routing in legacy switches follows traditional routing protocols such as Open Shortest Path First (OSPF) [69]. The schematic diagram of the hybrid SDN architecture is shown in Figure 8.1.

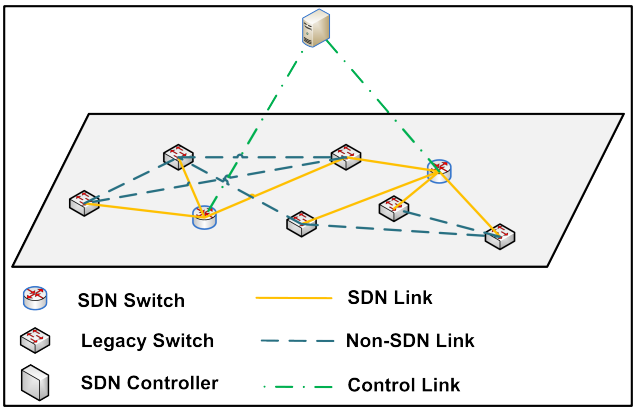


Figure 8.1: ETHoS: Hybrid SDN Architecture

We represent the hybrid SDN as a graph $G = (N, E)$, where N denotes the set of switches, and E denotes the set of links. We define a binary variable α_i to denote whether a switch n_i is a legacy switch or an SDN switch. Therefore,

$$\alpha_i = \begin{cases} 1 & \text{if } n_i \in N \text{ is an SDN switch,} \\ 0 & \text{otherwise.} \end{cases} \tag{8.1}$$

In addition, we express the type of links as:

$$\beta_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in E \text{ is an SDN link,} \\ 0 & \text{otherwise.} \end{cases} \tag{8.2}$$

The activation status of the switches is expressed as:

$$x_i = \begin{cases} 1 & \text{if } n_i \in N \text{ is active,} \\ 0 & \text{otherwise.} \end{cases} \tag{8.3}$$

The activation status of the links is expressed as:

$$y_{ij} = \begin{cases} 1 & \text{if } e_{ij} \in E \text{ is active,} \\ 0 & \text{otherwise.} \end{cases} \tag{8.4}$$

Let b_{ij} and w_{ij} denote the bandwidth usage and capacity of $e_{ij} \in E$. The bandwidth usage of the data links consists of data packets of the traffic flows. On the other hand, the bandwidth usage of the control links consists of control messages.

8.1.1 Traffic Flow Model

Let F denote the set of traffic flows. A flow $f_a \in F$ is represented by a tuple $< src_a, dest_a, E_a, N_a >$, where src_a, $dest_a$, E_a, and N_a denote the source, destination, the set of edges signifying the routing path, and the set of switches along the routing path of f_a. For each link $e_{ij} \in E$, we express the traffic matrix as:

$$g_{ij}^a = \begin{cases} 1 & \text{if } e_{ij} \in E_a, \\ 0 & \text{otherwise.} \end{cases} \tag{8.5}$$

A flow f_a is programmable if and only if at least one switch in N_a is an SDN switch.

Definition 30 (Network State)**.** *The state of the network is defined as:*

$$\Omega = \{x, y, \{E_a, N_a | f_a \in F\}\} \tag{8.6}$$

8.1.2 Power Consumption Model

The power consumption of a hybrid SDN has two parts — (1) power consumption of the links and (2) power consumption of the switches. The power consumption of a link $e_{ij} \in E$ is expressed as:

$$\mathbb{P}_{ij}^{E} = y_{ij}\mathbb{P}_{ij} + b_{ij}\Theta_{ij}, \tag{8.7}$$

where $\mathbb{P}_{ij}$ is the baseline power usage when not transmitting, and Θ_{ij} is the power coefficient [33].

Therefore, the power consumption of a switch $n_i \in N$ is estimated as:

$$\mathbb{P}_{i}^{N} = x_i \left(\mathbb{P}_{i}^{act} + \sum_{n_j \in N} \mathbb{P}_{ij}^{E} \right) + (1 - x_i)\mathbb{P}_{i}^{inact}, \tag{8.8}$$

where $\mathbb{P}_{i}^{act}$ and $\mathbb{P}_{i}^{inact}$ are the power consumption by a switch in active and inactive states, respectively.

Definition 31 (Link Utility)**.** *The route utility of a link e_{ij} is defined as:*

$$\mathbb{U}_{ij}^{E} = (\alpha_i + \alpha_j) \left(1 - \frac{\mathbb{P}_{ij}^{E}}{y_{ij}\mathbb{P}_{ij} + w_{ij}\Theta_{ij}} \right) \tag{8.9}$$

Definition 32 (Route Utility)**.** *The route utility of a traffic flow f_a is defined as:*

$$\mathbb{U}_a = \sum_{e_{ij} \in E_a} \mathbb{U}_{ij}^{E} \tag{8.10}$$

8.1.3 Problem Formulation

The objective of this work is to maximize the route utility for all traffic flows. Therefore, we formulate the energy-aware traffic engineering problem (ETEP) as:

$$\underset{\Omega}{\text{Maximize}} \sum_{f_a \in F} \mathbb{U}_a \tag{8.11}$$

subject to

$$b_{ij} \leq w_{ij}, \forall e_{ij} \in E, \tag{8.12}$$

$$\sum_{n_i \in N_a} x_i = |N_a|, \forall f_a \in F, \tag{8.13}$$

$$y_{ij} \leq x_i \text{ and } y_{ij} \leq x_j, \forall e_{ij} \in E, \tag{8.14}$$

$$\sum_{n_j \in N} g_{ij}^a - \sum_{n_j \in N} g_{ji}^a = \begin{cases} 1 & \text{if } n_i = src_a, \\ -1 & \text{if } n_i = dest_a, \\ 0 & \text{otherwise} \end{cases},$$

$$\forall f_a \in F, n_i \in N \tag{8.15}$$

Equation (8.12) expresses the link capacity constraint. Equation (8.13) states that all switches in the path of a flow are active. Equation (8.14) ensures that a link can not be active if it is connected to an inactive switch. Equation (8.15) expresses the flow conservation constraint. The objective of ETEP is a combinatorial problem having high complexity for large-scale network topologies. This is because energy-aware routing is a NP-hard problem [9]. Therefore, we design heuristic algorithms for solving ETEP.

8.2 ETHoS: The Proposed Scheme

In this section, we present two heuristic approaches — (1) a greedy heuristic approach, named ETHoS-G, and (2) a simulated annealing (SA) based approach, named ETHoS-SA. The goal of the proposed heuristic approaches is to maximize the objective function expressed in Equation (8.11).

8.2.1 ETHoS-G: Energy-Aware Traffic Engineering in Hybrid SDN with Greedy Heuristic

ETHoS-G generates a feasible network state by deactivating under-utilized SDN links. We consider e_{ij} as an under-utilized link if the power consumption $\mathbb{P}_{ij}^{E}$ is less than a pre-defined threshold $\mathbb{P}^{th}$. The value of $\mathbb{P}^{th}$ depends on network-specific parameters such as traffic load, and type of applications (latency-sensitive or throughput-sensitive). Algorithm 8.1 shows the steps of the Feasible State Generation Algorithm (FSGA). The input to FSGA includes network topology G and the current network state $\Omega^0 = \{x^0, y^0, \{E_a^0, N_a^0 | f_a \in F\}\}$.

Algorithm 8.1: ETHoS: Feasible State Generation Algorithm

Inputs : G , Ω^0
Output: Ω^1: Feasible network state

$\Omega^1 \leftarrow \Omega^0$
$E^{'} \leftarrow$ Set of links with $\beta_{ij} = 1$ and $y_{ij} = 1$ and $\mathbb{P}_{ij}^{E} \leq \mathbb{P}^{th}$
Sort the links in $E^{'}$ in ascending order of power usage
for $e_{ij} \in E^{'}$ **do**
    if G *remains strongly connected with* $E \setminus e_{ij}$ **then**
        $E^{''} \leftarrow E^{''} \cup \{e_{ij}\}$, $y_{ij}^1 \leftarrow 0$
    end
end
Set $x_i^1 = 0$ if an SDN switch n_i has no link e_{ij} with $y_{ij}^1 = 1$
$F^{'} \leftarrow$ Set of flows passing through any link in $E^{''}$
for $f_a \in F^{'}$ **do**
    Select the path with the maximum route utility and update N_a^1, E_a^1
end
return $\Omega^1 \leftarrow \{x^1, y^1, \{E_a^1, N_a^1 | f_a \in F\}\}$

FSGA selects the set of under-utilized SDN links $E^{'}$, which have power consumption less than $\mathbb{P}^{th}$. From the set $E^{'}$, links are selected in ascending order of power consumption. A selected link is deactivated if G remains strongly connected without that link. An SDN switch is deactivated if it is associated with no active link. Accordingly, FSGA generates an alternate path for each flow involving the deactivated links. An alternative

routing path is a path having the maximum route utility.

Theorem 9. *The number of SDN links in a fully connected hybrid SDN is* $e_s^{max} = \sum_{n_j \in N} \alpha_j \left(|N| - \frac{\left(\sum_{n_j \in N} \alpha_j + 1 \right)}{2} \right)$.

Proof. Each switch in a fully connected network with $|N|$ switches has $|N| - 1$ links. Therefore, if a single SDN switch is present in the network, i.e. $\sum_{n_j \in N} \alpha_j = 1$, the number of SDN links is $|N| - 1$. Subsequently, when another legacy switch is upgraded to SDN switch, the SDN link count becomes $(|N| - 1) + (|N| - 2)$ because the link between the first and the second SDN switch is already upgraded to an SDN link in the previous stage. Therefore, the number of SDN links in a fully connected hybrid SDN is given by:

$$e_s^{max} = \sum_{s=1}^{\sum_{n_j \in N} \alpha_j} (|N| - s) = \sum_{n_j \in N} \alpha_j \left(|N| - \frac{\left(\sum_{n_j \in N} \alpha_j + 1 \right)}{2} \right) \tag{8.16}$$

□

The time complexity of FSGA is estimated based on three parts of the algorithm. The first part takes $O(e_s^{max})$ time to select the set of under-utilized SDN links. The second part constructs set $F^{'}$ in $O(|F|)$ time. The last part involves the selection of alternate routes for the flows in $F^{'}$ and takes $|F^{'}|\,(|E| + |N| \log |N|)$ time. However, this time reduces further if the controller stores all the available routes between each pair of nodes in descending order of route utility.

8.2.2 ETHoS-SA: Energy-Aware Traffic Engineering in Hybrid SDN with Simulated Annealing

ETHoS-SA generates a final network state considering a current network state. The feasible network state Ω^1 generated by FSGA, may not be optimal in terms of the joint criteria of programmable traffic and energy consumption. Therefore, we use SA to generate a final network state Ω, which is better than Ω^1. We select SA for optimal network state generation because it is a meta-heuristic optimization algorithm, which is time-efficient [70]. SA can generate a globally optimal solution. Therefore, SA is used for a wide range of applications such as signal processing, production scheduling, and control engineering [71].

Algorithm 8.2: ETHoS: Optimal State Generation Algorithm

Inputs : G, Ω^1, T_0, z, L, p
Output: Ω: Final network state
$T \leftarrow T_0$: Current temperature
$\Omega \leftarrow \Omega^1$: Current state
while $T > 0$ **do**
  while $L > 0$ **do**
    $\Omega^{next} \leftarrow GenerateNextState(\Omega)$
    if $exp\left(\frac{Cost(\Omega)-Cost(\Omega^{next})}{T}\right) > p$ **then**
      $\Omega \leftarrow \Omega^{next}$
    end
    $L \leftarrow L - 1$
  end
  $T \leftarrow z \times T$
end
return $\Omega \leftarrow \{x, y, \{E_a, N_a | f_a \in F\}\}$

Algorithm 8.2 shows the steps of the SA-based Optimal State Generation Algorithm (OSGA). The inputs of OSGA include network-specific parameters and parameters required for SA. The network-specific parameters are network topology G, the feasible network state $\Omega^1 = \{x^1, y^1, \{E_a^1, N_a^1 | f_a \in F\}\}$. The required parameters for SA are initial temperature T_0, the rate of cooling z, the length of Markov chain L, and accep-

tance probability p. The initial temperature determines the convergence time of the algorithm, where a high value of T_0 signifies that the time to reach the global optimal solution is high, and a low T_0 may direct the algorithm to a local optimal solution. The cooling rate determines the amount of decrease in the temperature, and the algorithm terminates when the temperature reaches 0. The length of Markov chain signifies the maximum number of iterations before decreasing the temperature. The acceptance probability determines whether a solution is acceptable or not.

OSGA aims to find the optimal routes for the flows that balance the trade-off between energy-aware routing and programmable traffic. Maximum L iterations are performed for each value of the current temperature. In each iteration, OSGA generates a next state using the *GenerateNextState* method, as shown in Algorithm 8.3. Algorithm 8.4 calculates the cost of the current state and the next state. The next state is selected as the current state if $exp\left(\frac{Cost(\Omega)-Cost(\Omega^{next})}{T}\right) > p$. After the completion of L iterations, the current temperature is reduced. In this work, we use an exponential function as the cooling method and set the new temperature as $T \leftarrow z \times T$.

Algorithm 8.3: GenerateNextState

Inputs : Ω: Current state
Output: Ω^{next}: Next state

1 $\Omega^{next} \leftarrow \Omega$
2 Randomly select an SDN link e_{ij} and set $y_{ij}^{next} \leftarrow \left(1 - y_{ij}^{next}\right)$
3 Set $x_i^{next} = 0$ if SDN switch n_i has no link e_{ij} with $y_{ij}^{next} = 1$
4 Select the alternate shortest path for each affected flow f_a and update E_a^{next}, N_a^{next}
5 **return** $\Omega^{next} \leftarrow \{x^{next}, y^{next}, \{E_a^{next}, N_a^{next} | f_a \in F\}\}$

Algorithm 8.3 shows the steps of generating the next state given a current state. We alter the activation status of a randomly selected links. Based on the change of activation status, the available shortest path is selected as the new route for each affected flow, and the next network state is formed.

Algorithm 8.4 estimates the cost of a given network state. The cost of a network

Algorithm 8.4: Cost

Input : Ω
Output: *stateCost*: Cost for state Ω

```
for f_a ∈ F do
    stateCost ← stateCost + (|N| − 𝕌_a)
end
return stateCost
```

state is the cumulative cost of all the traffic flows in the network. Therefore, the cost of a network state Ω is defined as:

$$Cost(\Omega) = \sum_{f_a \in F} |N| - \mathbb{U}_a \tag{8.17}$$

The cost of a traffic flow increases with the decrease in route utility. Therefore, the *Cost* method aims to maximize the total route utility of all traffic flows, which is the objective represented in Equation (8.11).

The time complexity of OSGA depends on the initial temperature T_0 and the cooling rate z. A high T_0 and low z increases the possibility of finding a global optimal solution at the cost of time complexity.

8.2.3 Summary of the Proposed Approach

In each such time-period, FSGA detects the under-utilized SDN links. If any under-utilized link is detected, FSGA generates a feasible network state. Subsequently, OSGA computes a final network state Ω. The final network state Ω portrays the activation status of the SDN links and the final routes of the flows. An SDN switch is selected for deactivation if all the associated links are in an inactive state as per the network state Ω. Based on Ω, the required switches and links are activated or deactivated. To deactivate a link, we set the OSPF weight of the link to infinity so that the link is not selected in the shortest path computation. Before the deactivation of an SDN switch, ETHoS ensures that the legacy switches which follow traditional routing protocols such as OSPF must

not transmit any packet to the SDN switch. OSPF detects the active switches based on the *Hello* messages sent by the switches. OSPF considers a switch to be an inactive switch if the switch sends no *Hello* message for a time interval called the *dead interval*. Therefore, in the proposed scheme, an SDN switch, marked for deactivation according to Ω, stops sending *Hello* messages for a duration δ greater than the *dead interval*. After this duration, the switch is deactivated. We term this duration δ as the *initiation interval*. Additionally, the OSPF weights of the links are adjusted based on the link utility metric.

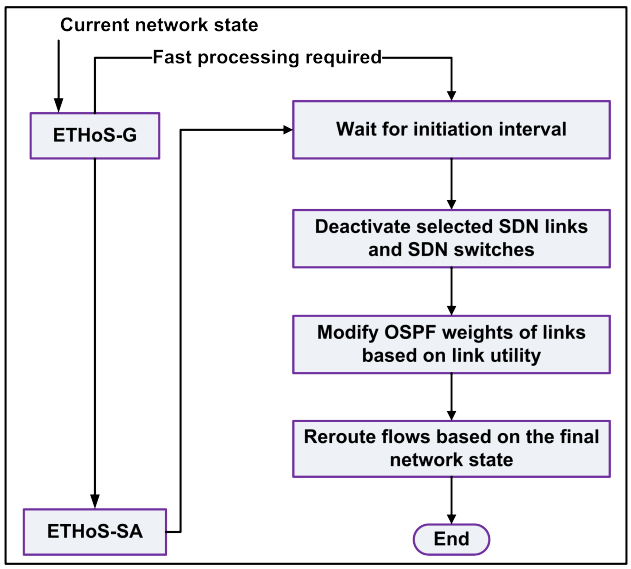


Figure 8.2: Execution of ETHoS by an SDN Controller

Figure 8.2 shows the flowchart for the execution of ETHoS by an SDN controller. ETHoS-G generates a less accurate solution with less convergence time. Therefore, if fast processing is required and a less precise solution is acceptable, the final network state is the network state generated by ETHoS-G. However, ETHoS-SA refines the solution generated by ETHoS-G and produces a more accurate solution.

Definition 33 (Energy Savings)**.** *The energy savings by transition from an initial network state* Ω^0 *to a final network state* Ω *is defined as:*

$$\xi = \frac{\sum\limits_{n_i \in N} \mathbb{P}_i^N(\Omega^0) - \sum\limits_{n_i \in N} \mathbb{P}_i^N(\Omega)}{\sum\limits_{n_i \in N} \mathbb{P}_i^N(\Omega^0)}, \tag{8.18}$$

where $\mathbb{P}_i^N(\Omega^0)$ *and* $\mathbb{P}_i^N(\Omega)$ *represent the power consumption of* n_i *in network states* Ω^0 *and* Ω*, respectively.*

Theorem 10. *The maximum energy savings achieved by link deactivation in a pure SDN is* $\frac{\mathbb{P}(|N|-1)(|N|-2)}{|N|\mathbb{P}^{act}+|N|(|N|-1)(\mathbb{P}+b\Theta)}$*, where* $b_{ij} = b, \mathbb{P}_{ij} = \mathbb{P}, \Theta_{ij} = \Theta, \forall e_{ij} \in E$*,* $\mathbb{P}_i^{act} = \mathbb{P}^{act}, \forall n_i \in N$ *and* $|N| > 2$*.*

Proof. The maximum and the minimum number of links in a pure SDN with $|N|$ switches are $\frac{|N|(|N|-1)}{2}$ and $|N|-1$, respectively. Let Ω^0 and Ω represent the initial network state with the maximum number of links and the final network state with the minimum number of links, respectively. Therefore, the maximum number of links that can be deactivated is given by:

$$e^{inact} = \frac{|N|(|N|-1)}{2} - (|N|-1) = \frac{(|N|-1)(|N|-2)}{2} \tag{8.19}$$

Let the total bandwidth usage of e^{inact} links are equally distributed among $|N|-1$ active links. Therefore, additional bandwidth usage in each active link is expressed as:

$$b^{add} = \frac{be^{inact}}{|N|-1} = \frac{b(|N|-1)(|N|-2)}{2(|N|-1)} = \frac{b(|N|-2)}{2} \tag{8.20}$$

The total power consumption in network state Ω^0 is:

$$\begin{aligned}\sum_{n_i \in N} \mathbb{P}_i^N(\Omega^0) &= |N|\mathbb{P}^{act} + 2\frac{|N|(|N|-1)}{2}(\mathbb{P} + b\Theta) \\ &= |N|\mathbb{P}^{act} + |N|(|N|-1)(\mathbb{P} + b\Theta) \end{aligned} \tag{8.21}$$

The total power consumption in network state Ω is:

$$\begin{aligned}\sum_{n_i \in N} \mathbb{P}_i^N(\Omega) &= |N|\mathbb{P}^{act} + 2(|N|-1)\left(\mathbb{P} + (b + b^{add})\Theta\right) \\ &= |N|\mathbb{P}^{act} + (|N|-1)(2\mathbb{P} + bn\Theta) \end{aligned} \tag{8.22}$$

Hence, the energy savings is given by:

$$\begin{aligned}\xi &= \frac{|N|\mathbb{P}^{act} + |N|(|N|-1)(\mathbb{P} + b\Theta) - |N|\mathbb{P}^{act} - (|N|-1)(2\mathbb{P} + bn\Theta)}{|N|\mathbb{P}^{act} + |N|(|N|-1)(\mathbb{P} + b\Theta)} \\ &= \frac{\mathbb{P}(|N|-1)(|N|-2)}{|N|\mathbb{P}^{act} + |N|(|N|-1)(\mathbb{P} + b\Theta)} \end{aligned} \tag{8.23}$$

□

8.3 Performance Evaluation

8.3.1 Simulation Settings

For performance evaluation ETHoS, we use Abilene topology [38] as the default topology for performing the simulations. For the simulations performed on Abilene topology, we use the traffic matrix provided by the Abilene dataset [45], which records traffic between each pair of switches in Abilene topology in every 5 minute for 6 months. We use the traffic data for Day 1 from the Abilene dataset. This dataset also provides OSPF weights of the links. We use the OSPF weights for the calculation of the shortest path. The

Abilene topology is a publicly available topology that has 12 switches and 30 directed links. We use the Abilene topology because it is a small-scale topology, where the deactivation of links is more restricted due to fewer alternate routes. However, Abilene is a sparse topology with limited paths between nodes. Therefore, we select a 4−pod Fat-tree for a topology-based comparison. Fat-tree is a dense topology with redundant paths between nodes [60]. For the simulation performed on Fattree topology, we randomly generate traffic flows between each pair of switches. The simulation parameters are shown in Table 8.1. The value of $\mathbb{P}^{th}$ is set to the average power consumption of the links.

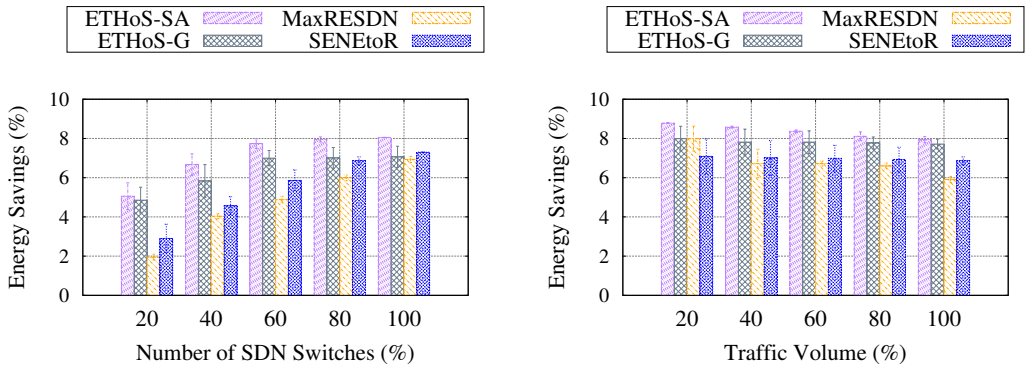
Table 8.1: ETHoS: Simulation parameters

Parameter	Value
Topology	Abilene [38], 4− pod Fat-tree [60]
Maximum traffic volume	144 flows
Bandwidth of a traffic flow	$0.0001 - 0.39$ Gbps [45]
Maximum link capacity	9.92 Gbps [45]
Number of switches	12 (Abilene), 20 (Fat-tree)
Percentage of SDN switches	$0 - 100$
Power consumption of a switch n_i in active state ($\mathbb{P}_i^{act}$)	150 W [72]
Power consumption of a switch n_i in inactive state ($\mathbb{P}_i^{inact}$)	95 W [72]
Baseline power usage of a link e_{ij} ($\mathbb{P}_{ij}$)	30 W [73]
Power coefficient (Θ_{ij})	10 W [73]
Initial temperature for SA (T_0)	90 [54]
Cooling rate (z)	0.97 [54]
Length of Markov chain (L)	200 [54]
Acceptance probability (p)	0.85 [54]

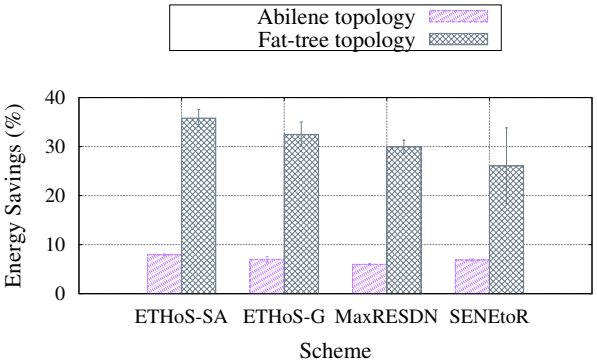
8.3.2 Benchmark Schemes

For performance evaluation, we consider ETHoS-G, MaxRESDN [32], and SENEToR [33] as the benchmark schemes. We consider ETHoS-G as a benchmark to show the necessity of final network state generation by ETHoS-SA. ETHoS-G reroutes the flows based

on the feasible network state generated by the greedy FSGA. We select RESDN as a benchmark scheme because it reroutes the flows based on the RESDN metric similar to ETHoS, which reroutes flows based on route utility metric. We select SENEToR as a benchmark scheme because it considers hybrid SDN similar to ETHoS.



(a) Abilene Topology with 100% Traffic Volume (b) Abilene Topology with 80% SDN Switches



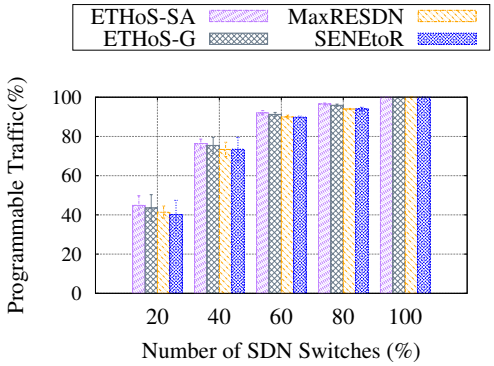
(c) Comparison between Abilene and Fat-tree Topology with 100% Traffic Volume and 80% SDN Switches

Figure 8.3: ETHoS: Energy Savings

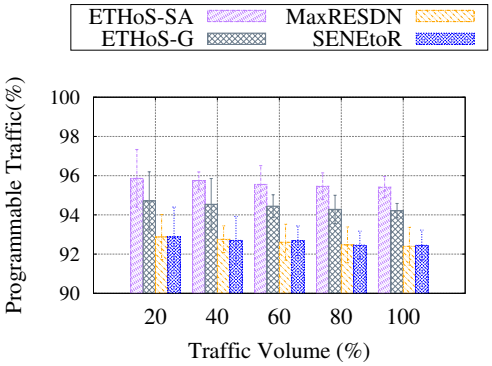
8.3.3 Performance Metrics

We consider the following metric to evaluate the performance of ETHoS:

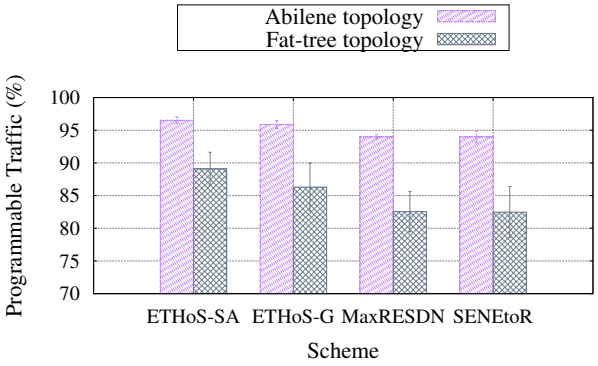
- **Energy savings**: The amount of energy savings is evaluated by comparing the cumulative power usage by the switches in the final network state with that of the initial network state. Therefore, this metric shows the energy efficiency of the



(a) Abilene Topology with 100% Traffic Volume



(b) Abilene Topology with 80% SDN Switches



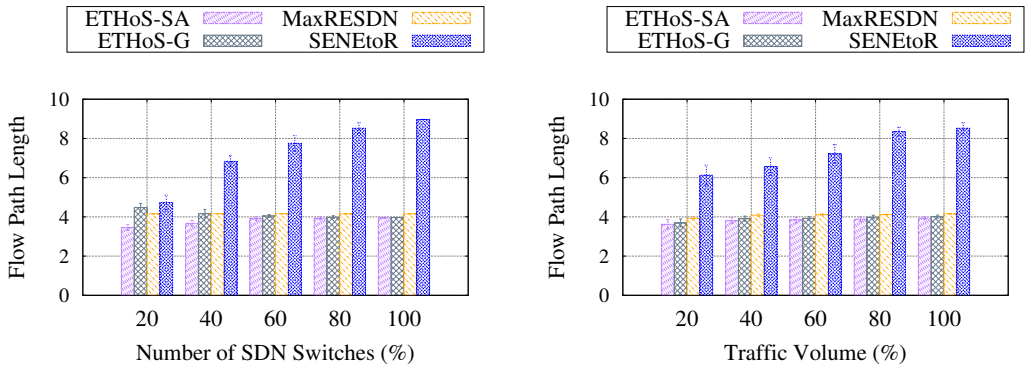
(c) Comparison between Abilene and Fat-tree Topology with 100% Traffic Volume and 80% SDN Switches

Figure 8.4: ETHoS: Programmable Traffic

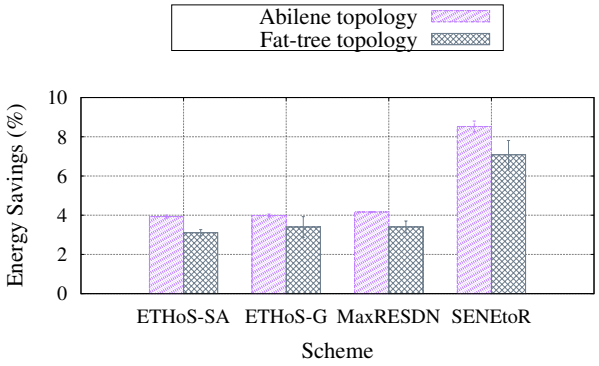
proposed scheme in comparison with the benchmarks.

- **Programmable traffic**: We use this metric to quantify the trade-off between energy-aware routing and programmable traffic.

- **Flow path length**: We estimate the average path length of the traffic flows as a performance metric to assess the overhead caused by traffic rerouting.



(a) Abilene Topology with 100% Traffic Volume (b) Abilene Topology with 80% SDN Switches



(c) Comparison between Abilene and Fat-tree Topology with 100% Traffic Volume and 80% SDN Switches

Figure 8.5: ETHoS: Flow Path Length

8.3.4 Results and Discussion

8.3.4.1 Energy Savings

Figure 8.3(a) portrays the comparison between the energy savings of ETHoS and that of the benchmarks with 100% traffic and varying SDN deployment. ETHoS-SA saves 11.65%, 48.95%, and 28.91% more energy than ETHoS-G, MaxRESDN, and SENEtoR, respectively. As shown in Figure 8.3(b), ETHoS-SA saves 6.91%, 23.12%, and 19.94% more energy than ETHoS-G, MaxRESDN, and SENEtoR, respectively. Figure 8.3(c) illustrates that energy savings in Fat-tree topology is significantly high for all schemes.

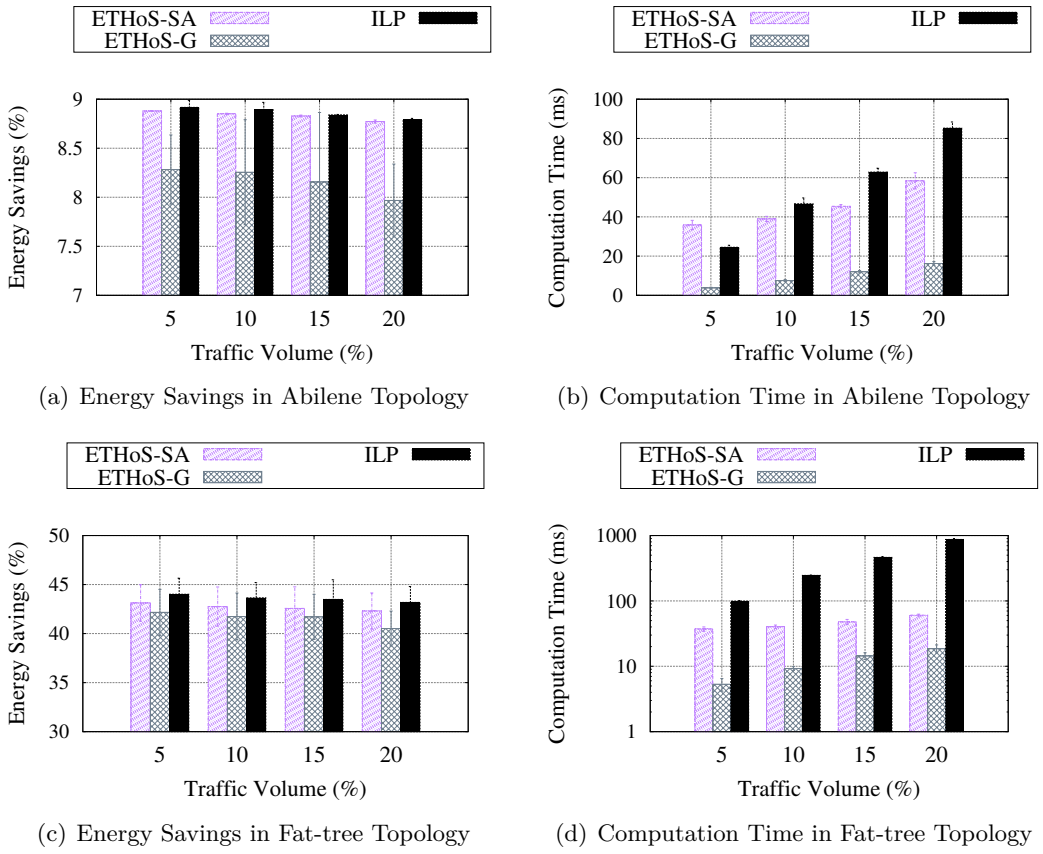
Inference: ETHoS-G performs local optimization and deactivates SDN links that

have low power usage. MaxRESDN reroutes flows based on RESDN value and deactivated unused links. However, with low SDN deployment, most of the selected links are non-SDN links and cannot be deactivated. Therefore, the energy savings of MaxRESDN is less, especially for less number of SDN switches. SENEtoR does not select the alternative routing path or tunnel based on link usage. Therefore, some links experience high traffic and high energy consumption due to tunneling or rerouting. On the other hand, ETHoS-SA considers activating an inactive SDN link to obtain a globally optimal performance that minimizes the cost of the network state. With fixed SDN deployment, the difference between the performance of ETHoS-SA and the benchmarks is almost constant with varying traffic because the bandwidth usage of the links has a limited contribution to energy consumption. Energy conservation is high in Fat-tree topology because many alternative paths are available in the Fat-tree topology than the Abilene topology. This is also the reason for the better performance of ETHoS that performs repeated analysis to select the best alternative path.

8.3.4.2 Programmable Traffic

As depicted in Figure 8.4(a), for fixed traffic volume, programmable traffic in ETHoS is 0.92%, 2.77%, and 3.08% higher than that of ETHoS-G, MaxRESDN, and SENEtoR, respectively. On the other hand, for fixed SDN deployment, ETHoS performs 1.24%, 3.23%, and 3.22% better than ETHoS-G, MaxRESDN, and SENEtoR, respectively as shown in Figure 8.4(b). From Figure 8.4(c), we notice that the programmable traffic with Fat-tree topology is 10.51% less than that with Abilene topology.

Inference: The volume of programmable traffic in ETHoS (both ETHoS-SA and ETHoS-G) is high because ETHoS uses route utility metric for optimal route selection of the flows. The amount of programmable traffic is low in Fat-tree because Fat-tree has more routing paths between the pair of nodes, which enables some flows to be routed through non-SDN links.



(a) Energy Savings in Abilene Topology

(b) Computation Time in Abilene Topology

(c) Energy Savings in Fat-tree Topology

(d) Computation Time in Fat-tree Topology

Figure 8.6: ETHoS: Comparison between ILP Solution and ETHoS with 80% SDN Switches

8.3.4.3 Flow Path Length

Figure 8.5(a) and Figure 8.5(b) show the average flow path length for fixed traffic volume and fixed SDN deployment, respectively. For fixed traffic volume, the average flow path length is 9.26%, 9.13%, and 48.64% less than that of ETHoS-G, MaxRESDN, and SENEtoR, respectively. On the other hand, for fixed SDN deployment, the average flow path length is 2.30%, 6.37%, and 48.10% less than that of ETHoS-G, MaxRESDN, and SENEtoR, respectively. From Figure 8.5(c), we observe that the average control path length is less for the Fat-tree topology than the Abilene topology.

Inference: The average flow path length is less in ETHoS-SA because OSGA selects the shortest path available at the time of a new network state generation. Addition-

ally, lower flow path length in Fat-tree topology signifies that the overhead due to flow rerouting is less in Fat-tree as the availability of alternative minimal length path is high in Fat-tree.

We solve the ILP formulated in Equation (8.11) using Gurobi Optimizer [48]. Figure 8.6 shows that ETHoS-SA achieves performance similar to the ILP solution while having low computation time.

8.4 Concluding Remarks

In this chapter, we presented a traffic engineering scheme to address the trade-off between programmable traffic and energy-aware routing for hybrid SDN. The proposed scheme, ETHoS, optimizes the network's energy consumption by selective deactivation of SDN switches and careful rerouting of the affected flows so that the optimal amount of programmable traffic is retained after traffic rerouting. We proposed a faster and less accurate greedy solution, named ETHoS-G, along with an optimized solution having high accuracy, called ETHoS-SA. Simulation results show that ETHoS-SA is capable of saving a significant amount of energy as compared to the benchmarks. In particular, with 80% SDN deployment, ETHoS-SA consumes 23.12% less energy than the existing scheme MaxRESDN.

Chapter 9

Conclusion

In this thesis, we proposed a scalable framework for SDN to handle a large number of traffic flows. We considered several challenges, such as rule-space capacity constraint, controller capacity constraint, and the trade-off between energy management and programmable traffic. In Chapters 3-8 , we discussed the schemes proposed in this thesis. We present the summary of the thesis in Section 9.1. In Section 9.2, we enlist the primary contributions of the thesis work. The limitations of our work is presented in Section 9.3. Finally, we conclude the thesis and cite future directions in Section 9.4.

9.1 Summary

This thesis was presented in six chapters. Chapter 1 presented a brief discussion on scalability challenges in SDN, the scope of the work, and the main objectives of this thesis.

Chapter 2 surveyed the existing literature on data plane scalability, control plane scalability, and energy-aware traffic engineering in SDN. Additionally, we summarized the shortcomings of the existing schemes and the motivation of this thesis.

Chapter 3 presented a scheme for *consistent update with redundancy reduction* (*CURE*) in SDN. The proposed scheme, CURE, ensures consistent rule update without the stor-

age of multiple versions of flow-rules. CURE prioritizes switches according to their usage pattern and schedules updates based on priority zones. We performed extensive simulations to evaluate the performance of the proposed scheme.

In Chapter 4, we presented an approach for *data plane load reduction for traffic flow migration (DART)* that formulates a coalition graph game to generate a QoS-aware flow migration schedule. Based on the initial schedule, DART verifies the possibility of link congestion and prepares a feasible migration schedule. We compared the proposed scheme with relevant benchmarks to analyze its performance.

Chapter 5 presented a *tensor-based rule-space management scheme (TERM)* that applies the concept of tensor decomposition to aggregate flow-rules. TERM also employs a rule caching mechanism for better throughput. We evaluated the proposed scheme through simulations and compared the results for various performance metrics with relevant benchmark schemes.

In Chapter 6, we presented a scheme for *control plane load reduction* (*CORE*) in the presence of IoT traffic. CORE uses a Markov Predictor to predict device-switch associations based on device mobility. Additionally, CORE computes optimal controller-switch assignments to prevent control plane overload. We compared the performance of CORE with relevant benchmarks to show its effectiveness.

Chapter 7 presented a scheme for *switch and controller placement* (*SCOPE*) in hybrid SDN. SCOPE ranks legacy switches according to different network parameters and selects a set of legacy switches for an upgrade in the current round while considering the upgrade budget. Additionally, SCOPE forms a coalition game involving the SDN switches to decide the locations for the controllers. The proposed scheme was evaluated through simulations, and the results for various performance metrics were compared with benchmark schemes.

In Chapter 5, we presented an *energy-aware traffic engineering scheme in hybrid SDN* (*ETHoS*). The proposed scheme, ETHoS, focuses on reducing energy consumption

in hybrid SDN while maximizing the programmable traffic as it is the primary purpose of transforming a legacy network to an SDN. We evaluated ETHoS by implementing a discrete event simulator and compared the performance metrics with relevant benchmarks.

9.2 Contributions

In this thesis, multiple approaches were proposed to enhance the scalability of SDN data and control planes. The proposed schemes address several issues, such as limited rule-space, control plane load optimization, and energy-aware routing. We list the major contributions of this thesis as follows.

Consistent Update with Reduced Rule-Redundancy: We emphasize reduction of rule-space usage and propose a rule update scheme that ensures packet-level consistency using a multilevel queue-based policy.

Traffic-Aware Data Plane Load Reduction during Flow Migration: We propose a traffic-aware flow-migration scheme that migrates traffic flows in different update stages. Each update stage is formed based on the QoS demand of the flows, and bandwidth usage of the links.

Rule-Space Management: We propose a rule-space management system which aims to reduce flow-table miss by increasing the available capacity of switches in SDN.

Control Plane Load Reduction for IoT Flows: We propose a prediction-based approach to reduce the control plane load in SDIoT. This approach includes rule-caching and master controller assignment considering heterogeneous attributes of IoT devices.

Switch and Controller Placement in Hybrid SDN: We propose a cost-efficient QoS-aware switch and controller placement approach for hybrid SDN. The proposed solution prioritizes the provision of QoS-guaranteed service to the users in the presence of dynamic network traffic and restricted upgrade budget.

Energy-Aware Traffic Engineering in Hybrid SDN: We propose a traffic engineering scheme to address the trade-off between programmable traffic and energy-aware routing for hybrid SDN. The proposed scheme optimizes the network's energy consumption by selective deactivation of SDN switches and careful rerouting of the affected flows to retain the optimal amount of programmable traffic after traffic rerouting.

9.3 Limitations

One of the critical challenges faced in this work is the lack of suitable datasets. Therefore, it would be interesting to implement the proposed schemes in a real testbed. Additionally, we made some assumptions while designing the proposed schemes.

- In CURE, we assumed centralized control plane.
- In DART, we assumed that controllers determine the new path for the migration of each traffic flow.
- We assumed exact match flow-rules where the mapping between a flow-rule and a flow type is one-to-one.
- We assumed that mobile IoT devices follow the Gauss-Markov mobility model.
- In SCOPE, we assumed that exactly one SDN switch replaces a legacy switch.

9.4 Future Work

- Future extension of the proposed scheme, CURE, includes an extension of the proposed scheme in the distributed SDN control plane, where multiple controllers perform network update concurrently. Additionally, flow-level consistency can be considered along with packet-level consistency.
- Future extension of the work, DART, includes consideration of energy consumption at the switches during traffic flow migration. Additionally, the impact of network disruptions such as link failure and traffic spike can be analysed.
- Future extension of the problem, TERM, includes optimizing the rule caching procedure and the placement of flow-rules in SDN switches.
- Future extension of the proposed work, CORE, includes increasing the prediction accuracy further.
- Future extension of the scheme, SCOPE, includes minimizing the overall energy consumption considering IoT networks.
- Future extension of the proposed approach, ETHoS, includes consideration varying traffic load, traffic with different QoS demands, and fault tolerance.

Dissemination of Research Works

Journal

- **I. Maity**, A. Mondal, S. Misra, and C. Mandal, "CURE: Consistent Update With Redundancy Reduction in SDN," *IEEE Transactions on Communications*, vol. 66, no. 9, pp. 3974-3981, Sep. 2018.
- **I. Maity**, A. Mondal, S. Misra and C. Mandal, "Tensor-Based Rule-Space Management System in SDN," *IEEE Systems Journal*, vol. 13, no. 4, pp. 3921-3928, Dec. 2019.
- **I. Maity**, S. Misra and C. Mandal, "Prediction-Based Control Plane Load Reduction in Software-Defined IoT Networks," *IEEE Transactions on Communications* (Under Review).
- **I. Maity**, S. Misra and C. Mandal, "SCOPE: Cost-Efficient QoS-Aware Switch and Controller Placement in Hybrid SDN," *IEEE Transactions on Emerging Topics in Computing* (Under Review).
- **I. Maity**, S. Misra and C. Mandal, "DART: Data Plane Load Reduction for Traffic Flow Migration in SDN," *IEEE Transactions on Communications* (Under Review).
- **I. Maity**, S. Misra and C. Mandal, "ETHoS: Energy-Aware Traffic Engineering in Hybrid SDN," *IEEE Transactions on Green Communications and Networking* (Under Review).

Conference

- **I. Maity**, S. Misra and C. Mandal, "Traffic-Aware Consistent Flow Migration in SDN," in Proceedings of *IEEE International Conference on Communications (ICC)*, Dublin, Ireland, June 2020, pp. 1-6, DOI: 10.1109/ICC40277.2020.9148983.

References

[1] W. Rafique, L. Qi, I. Yaqoob, M. Imran, R. u. Rasool, and W. Dou, "Complementing IoT Services through Software Defined Networking and Edge Computing: A Comprehensive Survey," *IEEE Commun. Surveys Tuts.*, pp. 1–1, 2020, doi: 10.1109/COMST.2020.2997475.

[2] H. Alameddine, M. H. K. Tushar, and C. Assi, "Scheduling of Low Latency Services in Softwarized Networks," *IEEE Trans. Cloud Comput.*, pp. 1–1, 2019, doi: 10.1109/TCC.2019.2907949.

[3] B. A. A. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, and T. Turletti, "A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 3, pp. 1617–1634, 2014.

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

[5] L. Cui, F. R. Yu, and Q. Yan, "When big data meets software-defined networking: SDN for big data and big data for SDN," *IEEE Netw.*, vol. 30, no. 1, pp. 58–65, Jan. 2016.

[6] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, J. Zolla, U. Hölzle, S. Stuart, and A. Vahdat, "B4: Experience with a Globally-deployed Software Defined Wan," in *Proc. ACM SIGCOMM*, New York, NY, USA, 2013, pp. 3–14.

[7] F. Giroire, J. Moulierac, and T. K. Phan, "Optimizing rule placement in software-defined networks for energy-aware routing," in *Proc. IEEE GLOBECOM*, Dec. 2014, pp. 2523–2529.

[8] A. Ruiz-Rivera, K. W. Chin, and S. Soh, "GreCo: An Energy Aware Controller Association Algorithm for Software Defined Networks," *IEEE Commun. Lett.*, vol. 19, no. 4, pp. 541–544, Apr. 2015.

[9] A. Fernández-Fernández, C. Cervello-Pastor, and L. Ochoa-Aday, "Achieving Energy Efficiency: An Energy-Aware Approach in SDN," in *Proc. IEEE GLOBECOM*, Dec. 2016, pp. 1–7.

[10] P. Francois and O. Bonaventure, "Avoiding Transient Loops During the Convergence of Link-state Routing Protocols," *IEEE/ACM Transactions on Networking*, vol. 15, no. 6, pp. 1280–1292, 2007.

[11] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, "Abstractions for Network Update," in *Proc. of ACM SIGCOMM*, New York, NY, USA, 2012, pp. 323–334.

[12] T. Mizrahi, E. Saat, and Y. Moses, "Timed Consistent Network Updates in Software-Defined Networks," *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3412–3425, Dec. 2016.

[13] R. McGeer, "A Safe, Efficient Update Protocol for Openflow Networks," in *Proc. of HOT SDN*, New York, NY, USA, 2012, pp. 61–66.

[14] C. R. Meiners, A. X. Liu, and E. Torng, "Bit Weaving: A Non-Prefix Approach to Compressing Packet Classifiers in TCAMs," *IEEE/ACM Trans. Netw.*, vol. 20, no. 2, pp. 488–500, Apr. 2012.

[15] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *Proc. of IEEE INFOCOM*, Apr. 2013, pp. 545–549.

[16] T. Kosugiyama, K. Tanabe, H. Nakayama, T. Hayashi, and K. Yamaoka, "A flow aggregation method based on end-to-end delay in SDN," in *Proc. of IEEE ICC*, May 2017, pp. 1–6.

[17] S. Bera, S. Misra, and M. S. Obaidat, "Mobility-Aware Flow-Table Implementation in Software-Defined IoT," in *Proceedings of IEEE Global Communications Conference (GLOBECOM)*, December 2016, pp. 1–6.

[18] F. Clad, S. Vissicchio, P. Mérindol, P. Francois, and J. J. Pansiot, "Computing Minimal Update Sequences for Graceful Router-Wide Reconfigurations," *IEEE/ACM Transactions on Networking*, vol. 23, no. 5, pp. 1373–1386, October 2015.

[19] M. Canini, P. Kuznetsov, D. Levin, and S. Schmid, "Software Transactional Networking: Concurrent and Consistent Policy Composition," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, 2013, pp. 1–6.

[20] D. A. Applegate, G. Calinescu, D. S. Johnson, H. Karloff, K. Ligett, and J. Wang, "Compressing Rectilinear Pictures and Minimizing Access Control Lists," in *Proc. of ACM-SIAM SODA*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 1066–1075.

[21] M. Moshref, M. Yu, A. Sharma, and R. Govindan, "vCRIB: Virtualized Rule Management in the Cloud," in *Proc. of HotCloud*, 2012.

[22] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic Controller Provisioning in Software Defined Networks," *Proc. IEEE CNSM*, pp. 18–25, Oct. 2013.

[23] K. S. Sahoo, D. Puthal, M. Tiwary, M. Usman, B. Sahoo, Z. Wen, B. P. S. Sahoo, and R. Ranjan, "ESMLB: Efficient Switch Migration-based Load Balancing for Multi-Controller SDN in IoT," *IEEE Internet Things J.*, pp. 1–1, 2019, doi: 10.1109/JIOT.2019.2952527.

[24] B. Heller, R. Sherwood, and N. McKeown, "The Controller Placement Problem," *Proc. ACM SIGCOMM Workshop on HotSDN*, pp. 7–12, 2012.

[25] L. F. Müller, R. R. Oliveira, M. C. Luizelli, L. P. Gaspary, and M. P. Barcellos, "Survivor: An enhanced controller placement strategy for improving SDN survivability," *Proc. IEEE GLOBECOM*, pp. 1909–1915, Dec. 2014.

[26] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli, "Large-Scale Dynamic Controller Placement," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 1, pp. 63–76, Mar. 2017.

[27] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in SDN-based core networks," *Proc. ITC*, pp. 1–9, Sep. 2013.

[28] A. Ksentini, M. Bagaa, T. Taleb, and I. Balasingham, "On using bargaining game for Optimal Placement of SDN controllers," *Proc. IEEE ICC*, pp. 1–6, May 2016.

[29] A. Dixit, F. Hao, S. Mukherjee, T. V. Lakshman, and R. Kompella, "Towards an Elastic Distributed SDN Controller," *Proc. ACM SIGCOMM*, vol. 43, no. 4, pp. 7–12, Aug. 2013.

[30] M. Tanha, D. Sajjadi, and J. Pan, "Enduring Node Failures through Resilient Controller Placement for Software Defined Networks," *Proc. IEEE GLOBECOM*, pp. 1–7, Dec. 2016.

[31] A. Sallahi and M. St-Hilaire, "Optimal Model for the Controller Placement Problem in Software Defined Networks," *IEEE Commun. Lett.*, vol. 19, no. 1, pp. 30–33, Jan. 2015.

[32] B. G. Assefa and Ö. Özkasap, "RESDN: A Novel Metric and Method for Energy Efficient Routing in Software Defined Networks," *IEEE Trans. Netw. Service Manag.*, pp. 1–1, 2020, doi: 10.1109/TNSM.2020.2973621.

[33] N. Huin, M. Rifai, F. Giroire, D. Lopez Pacheco, G. Urvoy-Keller, and J. Moulierac, "Bringing Energy Aware Routing Closer to Reality With SDN Hybrid Networks," *IEEE Trans. Green Commun. Netw.*, vol. 2, no. 4, pp. 1128–1139, 2018.

[34] "OpenFlow Switch Specification (Version 1.5.1): Open Networking Foundation," Mar. 2015.

[35] R. Anand, K. Mehrotra, C. K. Mohan, and S. Ranka, "Efficient classification for multiclass problems using modular neural networks," *IEEE Trans. Neural Netw.*, vol. 6, no. 1, pp. 117–124, Jan. 1995.

[36] A. F. Tayel, S. I. Rabia, and Y. Abouelseoud, "An Optimized Hybrid Approach for Spectrum Handoff in Cognitive Radio Networks With Non-Identical Channels," *IEEE Trans. Commun.*, vol. 64, no. 11, pp. 4487–4496, Nov. 2016.

[37] D. Li, W. Saad, I. Guvenc, A. Mehbodniya, and F. Adachi, "Decentralized Energy Allocation for Wireless Networks With Renewable Energy Powered Base Stations," *IEEE Trans. Commun.*, vol. 63, no. 6, pp. 2126–2142, Jun. 2015.

[38] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet Topology Zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.

[39] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," in *Proc. of the IEEE*, vol. 103, no. 1, Jan. 2015, pp. 14–76.

[40] S. Azodolmolky, R. Nejabati, M. Pazouki, P. Wieder, R. Yahyapour, and D. Simeonidou, "An Analytical Model for Software Defined Networking: A Network Calculus-Based Approach," in *Proc. of IEEE GLOBECOM*, Dec. 2013, pp. 1397–1402.

[41] S. Shirali-Shahreza and Y. Ganjali, "Delayed Installation and Expedited Eviction: An Alternative Approach to Reduce Flow Table Occupancy in SDN Switches," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1547–1561, Aug. 2018.

[42] W. Saad, Z. Han, M. Debbah, and A. Hjorungnes, "A Distributed Coalition Formation Framework for Fair User Cooperation in Wireless Networks," *IEEE Trans. Wireless Commun.*, vol. 8, no. 9, pp. 4580–4593, Sep. 2009.

[43] M. Ahmed, M. Peng, M. Abana, S. Yan, and C. Wang, "Interference Coordination in Heterogeneous Small-Cell Networks: A Coalition Formation Game Approach," *IEEE Syst. J.*, vol. 12, no. 1, pp. 604–615, Mar. 2018.

[44] I. Maity, A. Mondal, S. Misra, and C. Mandal, "CURE: Consistent Update With Redundancy Reduction in SDN," *IEEE Trans. Commun.*, vol. 66, no. 9, pp. 3974–3981, Sep. 2018.

[45] Abilene Dataset, Accessed: May, 2020. [Online]. Available: http://www.cs.utexas.edu/~yzhang/research/AbileneTM

[46] S. F. Abedin, M. G. R. Alam, S. M. A. Kazmi, N. H. Tran, D. Niyato, and C. S. Hong, "Resource Allocation for Ultra-Reliable and Enhanced Mobile Broadband IoT Applications in Fog Network," *IEEE Trans. Commun.*, vol. 67, no. 1, pp. 489–502, Jan. 2019.

[47] A. Basta, A. Blenk, S. Dudycz, A. Ludwig, and S. Schmid, "Efficient Loop-Free Rerouting of Multiple SDN Flows," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 948–961, Apr. 2018.

[48] Gurobi Optimizer, *Gurobi Optimizer Reference Manual*, Accessed: May, 2020. [Online]. Available: http://www.gurobi.com

[49] T. G. Kolda and B. W. Bader, "Tensor Decompositions and Applications," *SIAM Review*, 2009.

[50] E. Henry and J. Hofrichter, "Singular value decomposition: Application to analysis of experimental data," in *Numerical Computer Methods.* Academic Press, 1992, vol. 210, pp. 129 – 192.

[51] T. Wu, S. A. N. Sarmadi, V. Venkatasubramanian, A. Pothen, and A. Kalyanaraman, "Fast SVD Computations for Synchrophasor Algorithms," *Trans. Power Syst.*, vol. 31, no. 2, pp. 1651–1652, Mar. 2016.

[52] K. Sood, S. Yu, and Y. Xiang, "Performance Analysis of Software-Defined Network Switch Using $M/Geo/1$ Model," *IEEE Commun. Lett.*, vol. 20, no. 12, pp. 2522–2525, Dec. 2016.

[53] M. Mozaffari, W. Saad, M. Bennis, and M. Debbah, "Mobile Unmanned Aerial Vehicles (UAVs) for Energy-Efficient Internet of Things Communications," *IEEE Trans. Wireless Commun.*, vol. 16, no. 11, pp. 7574–7589, Nov. 2017.

[54] D. Wu, J. Yan, H. Wang, and R. Wang, "User-centric Edge Sharing Mechanism in Software-Defined Ultra-Dense Networks," *IEEE J. Sel. Areas Commun.*, pp. 1–1, 2020.

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

[56] L. Õzbakir, A. Baykasoğlu, and P. Tapkan, "Bees algorithm for generalized assignment problem," *Applied Mathematics and Computation*, vol. 215, no. 11, pp. 3782 – 3795, 2010.

[57] A. Jarray and A. Karmouch, "Decomposition Approaches for Virtual Network Embedding With One-Shot Node and Link Mapping," *IEEE/ACM Trans. Netw.*, vol. 23, no. 3, pp. 1012–1025, Jun. 2015.

[58] S. Bera, S. Misra, and M. S. Obaidat, "Mobi-Flow: Mobility-Aware Adaptive Flow-Rule Placement in Software-Defined Access Network," *IEEE Trans. Mobile Comput.*, vol. 18, no. 8, pp. 1831–1842, Aug. 2019.

[59] S. Sigg, D. Gordon, G. v. Zengen, M. Beigl, S. Haseloff, and K. David, "Investigation of Context Prediction Accuracy for Different Context Abstraction Levels," *IEEE Trans. Mobile Comput.*, vol. 11, no. 6, pp. 1047–1059, Jun. 2012.

[60] M. Al-Fares, A. Loukissas, and A. Vahdat, "A Scalable, Commodity Data Center Network Architecture," in *Proc. ACM SIGCOMM*, 2008, pp. 63–74.

[61] T. Camp, J. Boleng, and V. Davies, "A Survey of Mobility Models for Ad Hoc Network Research," *Wireless Communications and Mobile Computing*, vol. 2, no. 5, pp. 483–502, Aug. 2002.

[62] R. W. Bohannon, "Comfortable and maximum walking speed of adults aged 20-79 years: reference values and determinants," *Age and Ageing*, vol. 26, no. 1, pp. 15–19, Jan. 1997.

[63] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, "On Controller Performance in Software-defined Networks," *Proc. USENIX Conf. Hot Topics Manage. Internet, Cloud, Enterprise Netw. Services*, pp. 1–6, 2012.

[64] A. Sivanathan, D. Sherratt, H. H. Gharakheili, A. Radford, C. Wijenayake, A. Vishwanath, and V. Sivaraman, "Characterizing and classifying IoT traffic in smart cities and campuses," in *Proc. IEEE INFOCOM Workshops*, May 2017, pp. 559–564.

[65] X. Liu, Y. Liu, Y. Chen, and L. Hanzo, "Trajectory Design and Power Control for Multi-UAV Assisted Wireless Networks: A Machine Learning Approach," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7957–7969, Aug. 2019.

[66] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec. 1959.

[67] K. Poularakis, G. Iosifidis, G. Smaragdakis, and L. Tassiulas, "Optimizing Gradual SDN Upgrades in ISP Networks," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 288–301, Feb. 2019.

[68] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann, "Panopticon: Reaping the Benefits of Incremental SDN Deployment in Enterprise Networks," in *USENIX Annual Technical Conference*, 2014.

[69] B. Fortz and M. Thorup, "Internet traffic engineering by optimizing OSPF weights," in *Proc. IEEE INFOCOM*, vol. 2, 2000, pp. 519–528 vol.2.

[70] B. K. Saha, S. Misra, and S. Pal, "SeeR: Simulated Annealing-Based Routing in Opportunistic Mobile Networks," *IEEE Trans. Mobile Comput.*, vol. 16, no. 10, pp. 2876–2888, 2017.

[71] H. M. E. Abdelsalam and H. P. Bao, "A simulation-based optimization framework for product development cycle time reduction," *IEEE Trans. Eng. Manag.*, vol. 53, no. 1, pp. 69–85, 2006.

[72] T. Vu, "Sleep mode and wakeup method for OpenFlow switches," *J. Low Power Electron.*, vol. 10, no. 3, pp. 347–353, 2014.

[73] Cisco 1941 Series Integrated Services Routers Data Sheet, Accessed: Aug, 2014. [Online]. Available: http://www.cisco.com/c/en/us/products/collateral/routers/1900-series-integrated-services-routers-isr

BIO-DATA

1. **Bio-data**

   - *Name*: Ms. Ilora Maity
   - *Roll No.*: 15IT91R03
   - *Date of Birth*: 7^{th} September, 1986
   - *Permanent Address*: N-119/3, H. C. Banerjee Lane,
     P.O. - Konnagar, Dist. - Hooghly,
     State - West Bengal, India, PIN - 712235
   - *E-Mail*: iloramaity7@gmail.com, imaity@iitkgp.ac.in
   - *Homepage*: https://iloramaity.wixsite.com/ilora

2. **Present Status**: PhD Research Scholar, IIT Kharagpur, India.

3. **Academic Qualification**:

   - Master of Engineering (M. E.) in Computer Science and Engineering, Bengal Engineering and Science University, Shibpur, West Bengal, India, 2011.
   - Bachelor of Technology (B. Tech) in Computer Science and Engineering, West Bengal University of Technology, West Bengal, India, 2008.

4. **Professional Experience**:

   - September, 2011 – July, 2015, Technical Analyst, Cognizant Technology Solutions India Private Ltd.

5. **Journal Publications**:

   - **I. Maity**, A. Mondal, S. Misra, and C. Mandal, "Tensor-Based Rule-Space Management System in SDN," *IEEE Systems Journal*, vol. 13, no. 4, pp. 3921-3928, December 2019.
   - **I. Maity**, A. Mondal, S. Misra, and C. Mandal, "CURE: Consistent Update with Redundancy Reduction in SDN," *IEEE Transactions on Communications*, vol. 66, no. 9, pp. 3974-3981, September 2018.

- A. Mondal, S. Misra, and **I. Maity**, "Buffer Size Evaluation of OpenFlow Systems in Software-Defined Networks," *IEEE Systems Journal*, vol. 13, no. 2, pp. 1359-1366, June 2019.
- A. Mondal, S. Misra, and **I. Maity**, "AMOPE: Performance Analysis of OpenFlow Systems in Software-Defined Networks," *IEEE Systems Journal*, vol. 14, no. 1, pp. 124-131, March 2020.

6. **Conference Publications**:

- **I. Maity**, S. Misra, and C. Mandal, "Traffic-Aware Consistent Flow Migration in SDN," in Proceedings of IEEE International Conference on Communications (ICC), Dublin, Ireland, June 2020, pp. 1-6.
- **I. Maity**, G. Bhattacharya, S. Das, and B. K Sikdar, "A Cellular Automata based Scheme for Diagnosis of Faulty Nodes in WSN," in *Proceedings of IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, Anchorage, AK, pp. 1212-1217, October 2011.
- N. Khan, **I. Maity**, S. Das, and B. K. Sikdar, "Cellular Automata Based Scheme for Energy Efficient Fault Diagnosis in WSN", In Proceedings of *International Conference on Cellular Automata*, pp. 234-243, 2012.
- B. Das, G. Bhattacharya, **I. Maity**, and B. K. Sikdar, "Impact of Inaccurate Design of Branch Predictors on Processors' Power Consumption," in Proceedings of *IEEE International Conference on Dependable, Autonomic and Secure Computing*, Sydney, NSW, pp. 335-342, December 2011.
- G. Bhattacharya, **I. Maity**, B. K Sikdar, and B. Das, "Exploring Impact of Faults on Branch Predictors' Power for Diagnosis of Faulty Module," in *Proceedings of Asian Test Symposium*, New Delhi, pp. 226-231, November 2011.

7. **Referee Services**:

- **Journal**:
  - Peer reviewed articles for IEEE Transactions on Network and Service Management
  - Peer reviewed articles for IEEE Transactions on Mobile Computing
  - Peer reviewed articles for Pervasive and Mobile Computing
  - Peer reviewed articles for International Journal of Communication Systems
  - Peer reviewed articles for IEEE Transactions on Network Science and Engineering
- Conference:
  - Peer reviewed articles for IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), 2016

8. **Achievements**:

- Received **recognition** for review contributions to *International Journal of Communication Systems* (2020).
- Received **fellowship** for pursuing Ph.D degree from *Ministry of Human Resource Development (MHRD)*, India (2015).
- Recipient of **Assimilator of the Quarter (Best Newcomer) award**, *Cognizant Technology Solutions India Private Ltd.*, Kolkata, Q1 (2013).
- Recipient of **award for** 1^{st} **rank** in M.E. in *Department of Computer Science and Engineering* in Bengal Engineering and Science University, Shibpur (2012).
- **Qualified** *GATE* (The Graduate Aptitude Test in Engineering) with a percentile score of 97.52 (2009).
- **Qualified** the 3rd *National IT Aptitude Test* (NIIT) with a percentile of 88.00 (2007).
- Recipient of **award for** 2^{nd} **rank** in Serampore Girls High School in *Higher Secondary Examination* (2004).

9. **Professional Affiliations**

- Student member, IEEE
- Student member, IEEE Computer Society
- Student member, IEEE Communications Society