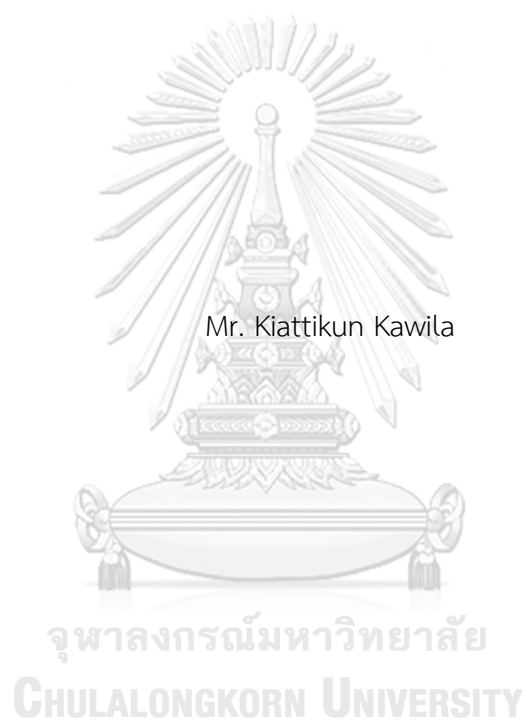


An SDN-Coordinated Multipath Transmission Steering Framework



A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy (Computer Engineering) in Computer Engineering
Department of Computer Engineering
FACULTY OF ENGINEERING
Chulalongkorn University
Academic Year 2021
Copyright of Chulalongkorn University

กรอบงานการขับเคลื่อนการส่งข้อมูลหลายเส้นทางแบบประสานเอสดีเอ็น



วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์

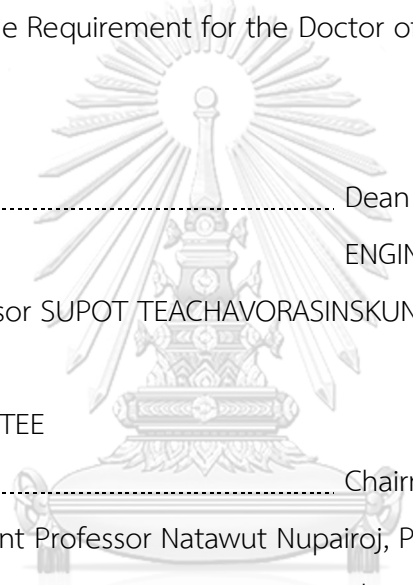
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย

ปีการศึกษา 2564

ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย

Thesis Title An SDN-
 Coordinated Multipath Transmission Steering Framework
By Mr. Kiattikun Kawila
Field of Study Computer Engineering
Thesis Advisor Associate Professor Kultida Rojviboonchai, Ph.D.
Thesis Co Advisor Professor JongWon Kim, Ph.D.

Accepted by the FACULTY OF ENGINEERING, Chulalongkorn University in
Partial Fulfillment of the Requirement for the Doctor of Philosophy (Computer
Engineering)



..... Dean of the FACULTY OF
ENGINEERING
(Professor SUPOT TEACHAVORASINSKUN, Ph.D.)

DISSERTATION COMMITTEE

..... Chairman
(Assistant Professor Natawut Nupairoj, Ph.D.)

..... Thesis Advisor
(Associate Professor Kultida Rojviboonchai, Ph.D.)

..... Thesis Co-Advisor
(Professor JongWon Kim, Ph.D.)

..... Examiner
(Professor Prabhas Chongstitvatana, Ph.D.)

..... Examiner
(Associate Professor KRERK PIROMSOPA, Ph.D.)

..... External Examiner
(Associate Professor Anan Phonphoem, Ph.D.)

เกียรติคุณ กาวีละ : กรอบงานการขับเคลื่อนการส่งข้อมูลหลายเส้นทางแบบประสานเอสดีเอ็น. (An SDN-Coordinated Multipath Transmission Steering Framework)

อ.ที่ปรึกษาหลัก : รศ. ดร.กุลธิดา โรจนวิบูลย์ชัย, อ.ที่ปรึกษาร่วม : ศ. ดร.จงวอน คิม

การเคลื่อนย้ายข้อมูลเป็นกลไกหลักที่สามารถส่งผลกระทบโดยตรง ต่อประสิทธิภาพโดยรวมของระบบการวิเคราะห์ข้อมูลขนาดใหญ่ เนื่องจากข้อมูลส่วนมากถูกสร้างขึ้นจากหลายตำแหน่งที่ตั้ง การที่จะเก็บรวบรวม และเคลื่อนย้ายข้อมูลระหว่างบริเวณที่จัดเก็บ จึงเป็นสิ่งที่กำลังท้าทาย การใช้วิธีการส่งผ่านเส้นทางเดียวแบบดั้งเดิม ไม่มีประสิทธิภาพที่จะตอบสนองต่อความต้องการต่าง ๆ ของแอปพลิเคชันข้อมูลขนาดใหญ่ ในงานวิจัยนี้เราจึงนำเสนอ กรอบงานการขับเคลื่อนการส่งข้อมูลหลายเส้นทางแบบประสานเอสดีเอ็นสำหรับแอปพลิเคชันการเคลื่อนย้ายข้อมูลขนาดใหญ่ โพรโทคอลที่ซีพีแบบหลายเส้นทางหรือเอ็มพีทีซีพี และสถาปัตยกรรมแบบเอสดีเอ็นจะเป็นส่วนสำคัญที่ถูกใช้พิจารณาเพื่อออกแบบ และพัฒนากรอบงานการส่งข้อมูลหลายเส้นทางของเรา เพื่อที่จะจัดเตรียมวิธีการกำหนดเส้นทางที่ใช้งานได้เชิงปฏิบัติ เราจึงนำเสนอ อัลกอริทึมกำหนดเส้นทางแบบสถิติโอเพ่นโพร ในอัลกอริทึมของเรานั้น จะมีการใช้เทคนิคของการพรุนิงโทโพลยี และข้อมูลสถิติจากพอร์ตของสวิตช์เพื่อเลือกเส้นทางการส่ง กรอบงานของเรานั้น จะถูกประเมินผลโดยใช้โปรแกรมจำลองมินิเน็ตกับโปรแกรมควบคุมไอนอส ผลลัพธ์จากการทดลองแสดงให้เห็นว่ากรอบงานที่เรานำเสนอนั้น สามารถลดเวลาของการเคลื่อนย้ายข้อมูลขนาดใหญ่สูงถึง ๙๐ เปอร์เซ็นต์ เมื่อเปรียบเทียบกับการกำหนดเส้นทางแบบดั้งเดิมด้วยเส้นทางที่แยกจากกัน และสามารถลดเวลาของการเคลื่อนย้ายข้อมูลขนาดใหญ่สูงถึง ๓๕ เปอร์เซ็นต์ เมื่อเปรียบเทียบกับงานวิจัยก่อนหน้า นอกจากนี้การกำหนดเส้นทางที่เรานำเสนอนั้น มีความสามารถในการปรับขนาดได้มากกว่างานวิจัยก่อนหน้า อีกทั้งยังให้ความซับซ้อน และค่าใช้จ่ายของระบบที่ต่ำกว่า ผลลัพธ์จากการทดลองแสดงให้เห็นว่าการกำหนดเส้นทางที่เรานำเสนอนั้นก่อให้เกิดค่าใช้จ่ายที่น้อยกว่างานวิจัยก่อนหน้าถึง ๕๗ เปอร์เซ็นต์

สาขาวิชา วิศวกรรมคอมพิวเตอร์

ปีการศึกษา 2564

ลายมือชื่อนิสิต

ลายมือชื่อ อ.ที่ปรึกษาหลัก

ลายมือชื่อ อ.ที่ปรึกษาร่วม

5771460721 : MAJOR COMPUTER ENGINEERING

KEYWORD: Big Data, Data transfer, MPTCP, Multipath TCP, OpenFlow, OpenFlow network, SDN, Software-Defined Networking

Kiattikun Kawila : An SDN-Coordinated Multipath Transmission Steering Framework. Advisor: Assoc. Prof. Kultida Rojviboonchai, Ph.D. Co-advisor: Prof. JongWon Kim, Ph.D.

Data transfer is a primary mechanism that can directly affect the overall performance of Big Data analytic systems. This is because most data are generated from several locations. It has been challenging to collect and transfer data among multiple storage regions. Using the traditional single-path transfer approaches is not efficient to serve several requirements of Big Data applications. In this dissertation, we propose an SDN-coordinated multipath transmission steering framework for Big Data transfer application. Multipath TCP protocol (MPTCP) and SDN architecture are mainly considered to design and develop our multipath transmission framework. To provide a practical routing solution, we propose a novel OpenFlow-Stats routing algorithm. In our algorithm, a new topology-pruning technique is applied, and the transmission paths are selected based on switch-port statistics. Our proposed framework is evaluated using the Mininet emulator and ONOS controller. The results show that our routing scheme can reduce the completion time of Big Data transfer up to 90% compared with the traditional routing scheme with disjoint paths and up to 35% compared with the previous work. Moreover, our proposed routing is more scalable than other previous works in that it can provide lower complexity and system overhead. The results show that our routing scheme produces 57% less overhead compared with the previous work.

Field of Study: Computer Engineering

Student's Signature

Academic Year: 2021

Advisor's Signature

Co-advisor's Signature

ACKNOWLEDGEMENTS

First, I would like to express my deepest gratitude to my advisor, Assoc. Prof. Kultida Rojviboonchai, for her patient guidance, enthusiastic encouragement, and useful critiques of this Ph.D. dissertation. She has advised me the way to study and also the way to live. I am so proud of being her supervision since my master's degree.

I would like to express my deepest gratitude to my co-advisor, Prof. JongWon Kim, for his patient guidance and useful critiques of this Ph.D. dissertation. It is an honor to learn from him.

I would also like to thank my committee members, Prof. Prabhas Chongstitvatana, Assoc. Prof. Anan Phonphoem, Asst. Prof. Natawut Nupairoj, and Assoc. Prof. Kerk Piromsopa, who gave many useful comments and suggestions to improve this dissertation.

I am also thankful to Assoc. Prof. Chaodit Aswakul who provides an opportunity for me to be a part of OF@TEIN+ project, an SDN-Cloud R&D collaboration among TEIN partners. It is an invaluable experience that I learnt from this project.

For all tuition fee and monthly expense, this dissertation is supported by the 100th Anniversary Chulalongkorn University Fund for Doctoral Scholarship from the Graduate School at Chulalongkorn University, and Wireless Network and Future Internet Research Unit at Chulalongkorn University.

I thank all members of ISEL laboratory and all members of MASS research group for any comments and discussion on my dissertation. It was a wonderful time that we spent together.

Finally, I wish to thank my family for their support and encouragement throughout my study.

Kiattikun Kawila

TABLE OF CONTENTS

	Page
.....	iii
ABSTRACT (THAI).....	iii
.....	iv
ABSTRACT (ENGLISH).....	iv
ACKNOWLEDGEMENTS.....	v
TABLE OF CONTENTS.....	vi
LIST OF TABLES.....	x
LIST OF FIGURES.....	xi
CHAPTER 1 Introduction.....	1
1.1 Design Goals.....	6
1.2 Scope and Assumption.....	6
1.3 Summary of Contributions.....	7
1.4 Dissertation Organization.....	8
CHAPTER 2 Background Knowledge, Related Work, and Motivation.....	9
2.1 Background Knowledge.....	9
2.1.1 Software-Defined Networking.....	9
2.1.2 Controller Platform.....	11
2.1.3 OpenFlow Protocol.....	12
2.1.4 OpenFlow Specification (Based on Version 1.0.0).....	14
2.1.4.1 Flow Table.....	14
2.1.4.2 OpenFlow Message.....	18

2.2 Related Work.....	19
2.2.1 Multipath Transfer Technologies over the Traditional Networking.....	19
2.2.2 Multipath Transfer Approaches Using MPTCP over the Software-Defined Networking.	25
2.3 Motivation.....	30
CHAPTER 3 An SDN-Coordinated Multipath Transmission Steering Framework for Big Data Transfer Application.....	33
3.1 Definition of Big Data in This Dissertation.....	33
3.2 Framework Overview.....	33
3.3 Practical Issues on Using MPTCP Protocol and OpenFlow Protocol.....	38
3.3.1 Issue on MPTCP Protocol.....	38
3.3.2 Issue on OpenFlow Protocol.....	40
3.4 Our Framework Design.....	41
3.4.1 Multipath Transmission Manager.....	42
3.4.2 Multipath Topology Manager.....	45
3.4.3 OpenFlow-Stats Analyzer.....	46
CHAPTER 4 Performance Evaluation.....	48
4.1 Experimental Setup.....	48
4.1.1 Simulation Configuration.....	48
4.1.2 Simulation Scenario.....	48
4.1.3 Benchmark and Metric.....	51
4.1.3.1 Benchmark.....	51
4.1.3.2 Metric.....	52
4.2 Experimental Results.....	53

4.2.1 Scenario 1: Vary the Number of MPTCP Connections	53
4.2.2 Scenario 2: Vary the Size of Data	60
4.2.3 Scenario 3: Overhead in the Control Plane	65
4.3 Comparison of Multipath Transport Protocols in Our SDN Environment.....	69
CHAPTER 5 Conclusion.....	72
5.1 Dissertation Summary.....	72
5.2 Cost Analysis of Using the Proposed Framework.....	73
5.2.1 Cost of SDN Deployment.....	73
5.2.2 Cost of Multipath Configuration at the Endpoint.....	73
5.2.3 Cost of Modifying the Network Component of Computing Software to Support Open vSwitch.....	73
5.2.4 Cost of Maintaining the OpenFlow Controller.....	73
5.2.5 Cost of Measuring the Maximum Capacity of Links.....	74
5.3 Complexity Analysis of Our OpenFlow-Stats Routing.....	74
5.4 Discussion on the Congestion Control Algorithms	75
5.4.1 Impact on Using the Traditional Congestion Control Algorithm.....	75
5.4.2 Impact on Using the Coupled Congestion Control Algorithm.....	75
5.5 Discussion on the Real-World Scenario Using OF@TEIN+ Testbed.....	77
5.5.1 Experimental Setup.....	77
5.5.2 Results of Deploying the Proposed Framework.....	80
5.6 Discussions on Limitations and Future Works	81
5.7 Concluding Remark.....	82
REFERENCES	83
APPENDIX.....	89

APPENDIX A Technical Details of Our Proposed Framework Being Applied to An
Overlay Network 90

VITA..... 92



LIST OF TABLES

	Page
Table 1 Flow table entry.....	14
Table 2 Match field of flow table entry.....	15
Table 3 Field lengths and the way they must be applied to flow entries.....	15
Table 4 Required list of counters for use in statistics messages.....	17
Table 5 Comparison of multipath transfer techniques over the traditional networking.....	24
Table 6 Comparison of MPTCP-based routing decision in Software-Defined Networking.	32
Table 7 Structure of MPTCP table.....	43
Table 8 Experimental parameter settings.....	50
Table 9 Completion time (in seconds) and percentage of improvement compared with traditional routing in multipath topology.....	57
Table 10 Completion time (in seconds) and percentage of improvement compared with traditional routing in COST239 topology.....	57
Table 11 Experimental parameter settings.....	77

LIST OF FIGURES

	Page
Figure 1 Comparison of the traditional network architecture and Software-Defined Networking [32].	10
Figure 2 Software-Defined Networking in (a) planes, (b) layers, and (c) system design architecture [31].	11
Figure 3 Application programming interfaces (APIs) of SDN controller [38].	12
Figure 4 OpenFlow architecture [32].	13
Figure 5 OpenFlow specification (a) version 1.0.0 [39] (b) version 1.3.0 [40].	14
Figure 6 MPTCP architecture.	20
Figure 7 MPTCP establishment.	21
Figure 8 MPQUIC architecture.	22
Figure 9 MFQUIC architecture [34].	22
Figure 10 The system architecture of Multiflow [22].	26
Figure 11 The system architecture of SMOC [23].	27
Figure 12 The system architecture of SFO [25].	28
Figure 13 The system architecture of S-MPTCP [26].	29
Figure 14 The system architecture of GCLR [27].	30
Figure 15 Distributed data storage system in the traditional network.	34
Figure 16 Distributed data storage system in our proposed framework with full-range deployment.	35
Figure 17 Distributed data storage system in our proposed framework being applied to an overlay network	36

Figure 18 Sequential diagram of data request and data transfer in our proposed framework.	37
Figure 19 MPTCP connection establishment and subflow creation mechanisms.	39
Figure 20 Example of flow rule structure with basic match fields.	40
Figure 21 Our proposed functional modules over the ONOS controller.	42
Figure 22 Pseudocode of our OpenFlow-Stats routing algorithm.	45
Figure 23 Pseudocode of our topology-pruning algorithm.	46
Figure 24 Pseudocode of our OpenFlow-Stats monitoring algorithm.	47
Figure 25 Multipath topology.	49
Figure 26 COST239 topology.	49
Figure 27 Average throughput per connection in multipath topology (File size = 200MB).	53
Figure 28 Average throughput per connection in COST239 topology (File size = 200MB).	54
Figure 29 Throughput variation at N = 15 connections with 95% confidence interval. (a) multipath topology. (b) COST239 topology.	55
Figure 30 CDF of completion time in multipath topology (File size = 200MB).	59
Figure 31 CDF of completion time in COST239 topology (File size = 200MB).	59
Figure 32 Average throughput per connection in multipath topology (N = 10 connections).	60
Figure 33 Average throughput per connection in COST239 topology (N = 10 connections).	61
Figure 34 Throughput variation at file size = 500MB with 95% confidence interval. (a) multipath topology. (b) COST239 topology.	62
Figure 35 CDF of completion time in multipath topology (N = 10 connections).	63
Figure 36 CDF of completion time in COST239 topology (N = 10 connections).	64

Figure 37 Overhead in the control plane (multipath topology) for the traditional routing with disjoint paths.	65
Figure 38 Overhead in the control plane (multipath topology) for k-Max disjoint routing.....	66
Figure 39 Overhead in the control plane (multipath topology) for our OpenFlow-Stats routing.....	66
Figure 40 Overhead in the control plane (COST239 topology) for the traditional routing with disjoint paths.	67
Figure 41 Overhead in the control plane (COST239 topology) for k-Max disjoint routing.....	67
Figure 42 Overhead in the control plane (COST239 topology) for our OpenFlow-Stats routing.....	68
Figure 43 Performance comparison between MPQUIC (Dash-dotted line) and MPTCP (Solid line).	70
Figure 44 Complexity analysis of OpenFlow-Stats routing algorithm.....	74
Figure 45 Experimental Results of using (a) CUBIC protocol and (b) LIA protocol.....	76
Figure 46 Experimental topology over OF@TEIN+ testbed.	78
Figure 47 Available bandwidth measurement between GIST-South Korea and HUST-Vietnam.....	79
Figure 48 Path latency measurement of the shortest path, alternative path 1, and alternative path 2 respectively.....	80
Figure 49 Comparison of experimental results between OF@TEIN+ testbed and Mininet emulator (File Size = 500MB).	81
Figure 50 An example of our proposed framework being applied to an overlay network.....	90
Figure 51 Examples of forwarding rules in the overlay-network environment.	91



จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

CHAPTER 1

Introduction

Big data analytics is a powerful process which has gained a lot of interests not only from scientific and medical fields but also business companies [1-3]. This is because of the growth of Internet services, the usage of IoT devices and IT-related technologies. A lot of real-time data can be generated from customers' smart devices and can be sent to cloud computing through the Internet. This data can be processed in order to analyze the customers' behaviors and predict the customers' demands in the future. The core components of data analytics are related to data collecting, data cleansing, and data analysis. To achieve the high performance of data analytics, the data gathering process must be concerned. How to collect a lot of data and how to transfer them to the big data analytic system are the key successes.

In general, a distributed data storage system is applied to manage the data [4]. This is because most data can be generated from several locations, for example, GIS data, Geospatial health, Continuous Glucose Monitoring, Disease-and-Health Monitoring, Bioinformatics, Electric Power Industry, and etc. With this characteristic, it has been challenging to collect and transfer data among multiple storage regions [5]. As mentioned above, to support the data analytic systems, the approach for big data transmission must be seriously concerned.

In order to cope with the big data transfer among multiple storage regions, several data-transfer techniques have been proposed, which can be classified into two categories. In the first category, data transfer using single-path fashion has been studied and optimized [6, 7]. In the second category, the techniques for improving performance of data transfer using multipath fashion have been investigated [8-27]. In this dissertation, we focus on the second category.

According to the benefits of using multipath fashion, several multipath-transfer techniques have been proposed, which can be classified into two categories. In the first category, several multipath routing techniques have been studied and optimized [8-13]. In the second category, not only multipath routing but also multipath transport protocols have been considered and exploited in the framework together [14-27]. This

is more beneficial to enhance the performance of the overall system. In order to support the big data transfer over the Internet environment that is filled with heterogeneous traffic, in this dissertation, we propose our framework that belongs to the second category.

Based on the traditional networking with network multihoming [8], in RFC 8684, multipath transmission control protocol (MPTCP) [28], is proposed to increase the throughput performance and improve resilience to network failure. Instead of single flow transmission of TCP, MPTCP uses multiple subflows to transfer the data. These subflows are generated based on all available pairs of network interfaces between endpoints. However, using MPTCP over the traditional networks cannot efficiently utilize the network capacity because multiple subflows normally share the same bottleneck link [29, 30]. This is because the IP-based routing provides the shortest path for every subflow.

Software-defined networking (SDN) [31] is a key networking paradigm which manages the network elements in a centralized fashion. The control plane is decoupled from the data plane. Data plane elements are replaced by simple forwarder devices with rule tables. Each incoming traffic flow is manipulated by the table rules. SDN controller is responsible for programming the rule tables based on its routing decisions. OpenFlow protocol [31, 32] is the most popular control protocol which is used for communication between the forwarding devices and the controller.

In SDN, a routing decision can be more flexible with the capability to use global knowledge and endpoint coordination. Many researches have been proposed to use MPTCP over SDN for big data transfer, which can be categorized into two groups.

The first group has focused on the data center environment. Due to the particular characteristics such as path diversity and bandwidth demands on elephant flows, most solutions aim to increase the network utilization. In [14-16], the available bandwidth was the primary parameter which was considered by routing algorithms. The authors in [17] applied the number of active flows on each link to define the path cost against the link capacity. Exploiting path diversity, the authors in [18] proposed to modify the path manager for using multiple subflows per IP-address pair. The authors

in [19-21] focused on how to find the appropriate number of subflow creations based on the flow demand and network conditions.

The second group has focused on the Internet environment in general. To avoid the shared-bottleneck problem, a multipath OpenFlow controller [22-27] has been proposed.

In [22], the controller can provide the disjoint shortest paths to subflows that are generated by the same MPTCP connection. However, this method did not consider background traffic and traffic distribution throughout the overall networks. Subflows of different connections then mostly shared the same disjoint shortest paths, leading to inefficient network utilization.

In [23], a simple multipath OpenFlow controller using topology-based routing algorithm has been proposed. To compose a path set, the shortest path is used for the first subflow. Other paths for other subflows will be calculated and prioritized by minimizing the number of the shared links with the first subflow. However, in a large-scale incoming-traffic scenario, the algorithm leads to high complexity. The controller needs to repeat the prioritization on each path set and store those sets for every MPTCP connection.

In [24], a multipath routing algorithm has been proposed. The algorithm considers the remaining bandwidth on possible links to calculate the optimal set of disjoint paths. In [25], the subflow optimizer for MPTCP has been proposed. In this scheme, the source needs to inform the required bandwidth. Then, the algorithm will select a set of disjoint paths in which the total capacity can be satisfied. In [26, 27], a cross-layer technique has been used to make a cooperation between the controller and MPTCP protocol stack in order to maximize the performance of multipath transmission. It can be seen that in [24-27], the remaining bandwidth is mainly used as a key metric to calculate the optimal set of disjoint paths. In practice, it is difficult and complicated to do so because the network links can be filled with heterogeneous traffic in terms of traffic type, traffic size, and etc. A lot of flow-rule tracking mechanisms need to be deployed and processed in real-time on every node and link. This increases complexity to the system.

In addition to MPTCP, there are also other multipath protocols recently proposed in the literature such as Multipath QUIC (MPQUIC) [33] and Multiflow QUIC (MFQUIC) [34]. MPQUIC is an extension of QUIC protocol [35]. It enables a QUIC connection to spread data over multiple networks. MFQUIC is a variant of QUIC protocol that is aware of network asymmetries. It focuses on using multiple unidirectional flows instead of bidirectional flows. However, currently there is only MPTCP that is defined as a standard transport protocol in RFC.

As mentioned above, to design a practical and efficient solution for big data multipath transmission, the following three issues should be taken into consideration.

- 1) *Traffic distribution*: A routing decision without the consideration of traffic distribution leads to inefficient utilization in the overall network. To support a large number of MPTCP traffic flows, how to design multiple paths for the flows based on the existing network resources and background traffic should be concerned.
- 2) *Complexity of the routing algorithm*: For multipath transmission using MPTCP, the number of messages to the controller will increase according to the number of negotiating messages that are generated by subflows. The processing time of those messages depends on the complexity of the routing algorithm. High complexity leads to a lot of waiting states of incoming traffic at the controller. To manipulate a large amount of the messages, the complexity of the routing algorithm needs to be considered.
- 3) *Practicality with OpenFlow protocol*: The OpenFlow protocol has been designed as a standard to provide a secure communication between the controller and the data-plane devices. Currently, some network-related metrics used in [24-27] such as the remaining bandwidth and delay cannot be directly provided by the OpenFlow messages. To obtain those metrics, a lot of flow-rule tracking mechanisms for heterogeneous traffic need to be deployed at the controller. These mechanisms induce high complexity and a lot of polling messages in the control plane.

In previous works, those solutions were focused on how to optimize the performance of the routing algorithm only. Most of them do not consider the

practicality with the OpenFlow protocol. This is a main issue that needs to be considered when we deploy a routing algorithm in a real environment. More discussion can be found in Section 2.2.2 and the comparison between our proposed framework and the previous works is shown in TABLE 6.

In this dissertation, we propose a new practical framework using SDN for big data transfer applications. In our framework, the MPTCP serves as the primary multipath transmission protocol. It is utilized among endpoints within distributed data storage systems. All subflows will be manipulated by our SDN controller. To address three aforementioned issues, our framework contains three modules including *Multipath Transmission Manager*, *Multipath Topology Manager* and *OpenFlow-Stats Analyzer*.

First, to detect the background traffic, our *OpenFlow-Stats Analyzer* gathers the statistics obtained from the OpenFlow messages and then estimates the resource usage on every node and every link. Based on this information, *Multipath Topology Manager* creates an OpenFlow-Stats graph that will be used by *Multipath Transmission Manager* for traffic distribution. *Multipath Transmission Manager* is the main module to manipulate incoming subflows and assign paths for the subflows.

Second, to reduce the complexity of the routing decision, our *Multipath Topology Manager* performs a topology-pruning technique. The mechanism of topology pruning will be operated in the background process at the controller. Based on the OpenFlow-Stats graph, links that are associated with the highly-utilized ports will be temporarily removed from the original topology graph. The resulting graph will then be used in order to select paths for subflows. This helps reduce the complexity of the routing decision.

Third, instead of using the remaining bandwidth which requires the complicated flow-rule tracking mechanisms, our *OpenFlow-Stats Analyzer* uses the information of port usage at every switch. Such information can be easily provided by the standard OpenFlow protocol. Then, our *OpenFlow-Stats Analyzer* will determine the level of port utilization and inform our *Multipath Topology Manager* about highly utilized ports. After that, our *Multipath Topology Manager* will find links associated with the highly-utilized ports and then remove them from the original graph.

Our framework has been evaluated by the well-known network emulator called Mininet [36]. The SDN controller is implemented based on ONOS controller [37]. According to our experiments, the results showed that our proposed framework significantly outperforms the traditional routing with disjoint paths in terms of throughput and completion time. In addition to MPTCP, the results showed that our proposed framework can work efficiently and compatible with other multipath transport protocols such as MPQUIC [33].

1.1 Design Goals

The goals are set as guidelines for our design and development. The desirable properties are traffic distribution, low complexity, and practicality with the standard control protocol. The details of properties are mentioned as following explanations.

- *Traffic distribution:* According to SDN capabilities, our framework should efficiently distribute all Big Data traffic throughout the whole network based on the existing network resources and background traffic.
- *Low complexity:* Our framework has to reduce any routing complexities in the SDN controller as much as possible. Our framework should also minimize the number of probing messages that are utilized for gathering global information from SDN switches.
- *Practicality with OpenFlow protocol:* Our framework should operate well using OpenFlow protocol over the Internet environment that the network links are frequently filled with unstable and heterogenous traffic.

1.2 Scope and Assumption

The scope of this dissertation is limited to the followings:

- This dissertation considers improving the performance of Big Data transfer application that relies on a distributed data storage system over the Internet environment.
- In terms of data type, the proposed framework is designed to support batch data only.

- The proposed framework focused on using the standard protocols in both data plane and control planes.
- In the data plane, multipath TCP is mainly evaluated for Big Data transfer among data storage servers.
- In the control plane, OpenFlow 1.0 and 1.3 is mainly utilized for controlling OpenFlow-enabled devices. Other SDN solutions is not concern in this dissertation.
- In order to reduce the complexity of multipath routing algorithm, the proposed controller separates the routing tasks from the routing algorithm into three proposed modules.
- The complexity of topology pruning is supposed to be $O(1)$.
- The tasks that are separated from the routing algorithm are designed to be process in background at the controller.
- In order to evaluate a link quality, the proposed controller is relied on the port-stats of OpenFlow-enabled devices.
- In order to evaluate the proposed framework, the environment of SDN environment is simulated by using Mininet emulator.
- The data size for evaluating the proposed controller is generated by *iPerf* tool.
- The issue on node/link failures is not taken into consideration in this dissertation.

1.3 Summary of Contributions

Our contribution in this article is fourfold. First, we proposed a new multipath transmission framework using SDN for big data transfer applications. This is used as a guideline to boost the performance of the networking component in a distributed data storage system. Second, we proposed a new practical OpenFlow controller to extract and manipulate subflows of the MPTCP protocol. The controller consists of three modules as follows. (1) *Multipath Transmission Manager* module is the main module to process incoming subflows and make a routing decision based on the OpenFlow-

Stats graph. (2) *Multipath Topology Manager* module is responsible for maintenance the OpenFlow-Stats graph based on the information that is received by the *OpenFlow-Stats Analyzer* module. (3) *OpenFlow-Stats Analyzer* module is responsible for collecting port statistics on each switch and determining the level of port utilization associated with each link. Third, we proposed a new OpenFlow-Stats routing algorithm. The algorithm with our pruning technique can provide a disjoint path to a subflow with low congestion level and low complexity. Fourth, we proposed a new OpenFlow-Stats monitoring algorithm. The algorithm is designed to be processed in the background at the controller. The output of the algorithm will be sent to the *Multipath Topology Manager* to update the OpenFlow-Stats graph. This process can reduce the complexity at the routing decision and also provide a traffic distribution.

1.4 Dissertation Organization

The rest of the dissertation is organized as follows. The next chapter describes background knowledge to understand the emerging networking architecture and how does it work in practice. This chapter also includes the literature review that contribute to this dissertation. Chapter 3 explains the traditional data-transfer approach and our proposed multipath transmission framework including our proposed controller. In Chapter 4, we evaluate the performance of our proposed framework compared with the previous solutions. Finally, Chapter 5 concludes the dissertation and discussion for further research.

CHAPTER 2

Background Knowledge, Related Work, and Motivation

2.1 Background Knowledge

2.1.1 Software-Defined Networking

Software-Defined Networking (SDN) is a new networking paradigm which manage network elements in a centralized fashion using software. Basically, SDN architecture is completely different from the traditional networking architecture. The control plane is physically separated from the data plane. The existing complex functions in data plane elements are replaced by simple forwarding functions with rule tables. Each incoming traffic flow is manipulated by the table rules. SDN controller(s) is/are responsible for creating the table rules based on its routing decisions. Figure 1 illustrates a comparison between the traditional networking architecture and SDN architecture. From the left to the right, the embedded control including Middleboxes are migrated to the SDN controller. Each complex function (e.g., routing algorithm, Firewall) is transformed to a network control application running on top of the controller. The existing control space of forwarding devices is replaced by simple forwarding functions with programmable tables. According to the Open Networking Foundation (ONF), an SDN-driven community, OpenFlow protocol is defined as a standard control protocol which is used for communication between the forwarding devices and the controller.

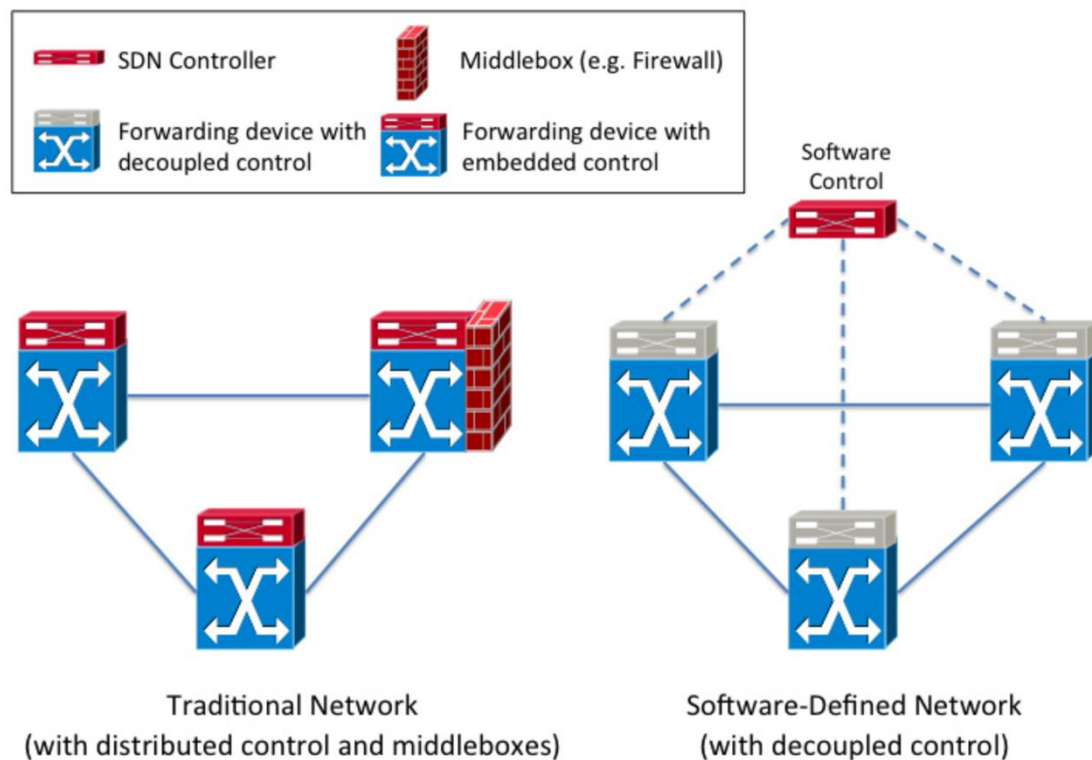


Figure 1 Comparison of the traditional network architecture and Software-Defined Networking [32].

Similarly, to the traditional networking, SDN can be depicted as a composition of different layers, as shown in Figure 2. These layers are the management plane, the control plane, and the data plane. From the left to the right, the controller represents a network operating system in the control plane. Each network control application (e.g., routing, access control, load balancer) represents an application in the management plane. As previously mentioned, the decoupling of the control plane and the data plane induce several advantages. For example, it becomes easier to develop and deploy a new networking feature, a network control application can take the global network information, it become easier to integrate different network control applications, and etc.

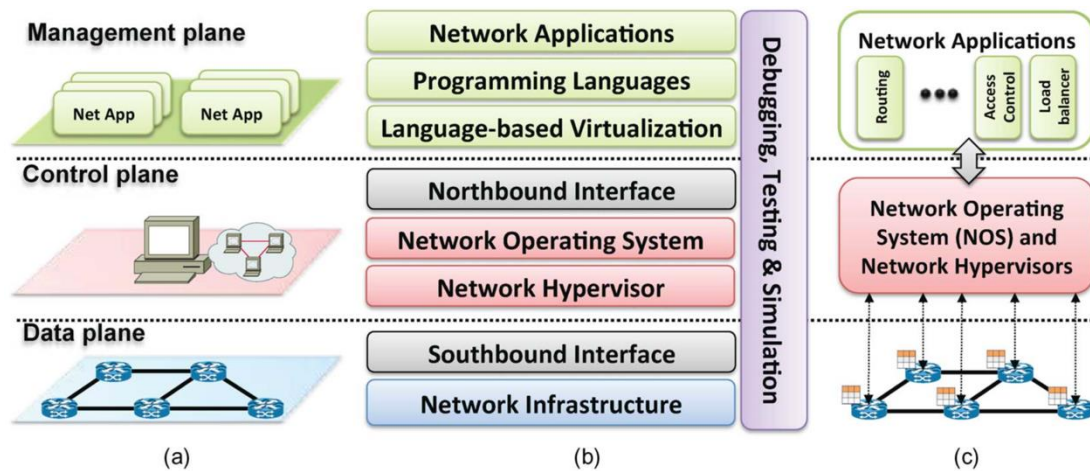


Figure 2 Software-Defined Networking in (a) planes, (b) layers, and (c) system design architecture [31].

2.1.2 Controller Platform

In general, SDN controller consists of three interfaces or APIs as shown in Figure

3.

- *Northbound interface:* This interface is used for communication between network control applications and the controller. Technically, the programming language and the instruction set for developing applications depend on the controller software. According to the discussion in survey papers [30, 31], the REST API is the most popular programming interface that is widely supported among the controller software.
- *Southbound interface:* This interface is used for communication between the controller and forwarding devices. This interface involves in the low-level instruction set that are used for programming data plane elements. According to ONF, OpenFlow is defined as a standard southbound protocol in SDN.
- *East-West bound interface:* In cases of multiple-domain or multiple-controller scenarios, each domain/controller can use this interface in order to share their control plane to others based on network policy and agreement. This interface is also used for communication between the controller and external servers. For example, a developer who need to

integrate his/her applications with a network control application, he/she can use this interface to send a specific request to the controller.

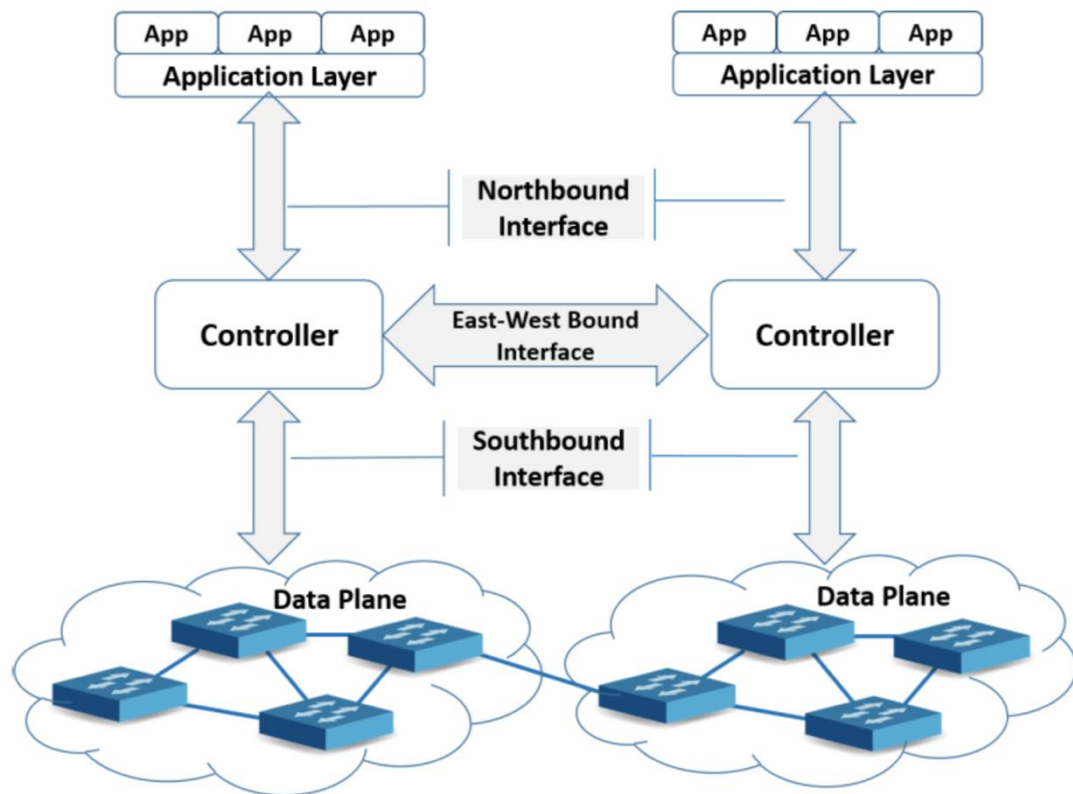


Figure 3 Application programming interfaces (APIs) of SDN controller [38].

2.1.3 OpenFlow Protocol

The OpenFlow protocol is a standard control protocol which is designed for supporting the centralized management in SDN. The protocol defines messaging mechanisms between controller and switches including structures of flow rules. Figure 4 demonstrates OpenFlow architecture. The forwarding devices, OpenFlow-enabled switches contain flow tables and OpenFlow client. The OpenFlow client is responsible for communication with the OpenFlow controller through the secure channel. The flow tables are used to process incoming packets. Typically, the flow entries consist of three following fields: (1) Match field, used to match incoming packets (2) Action field, used to handle matching packets and (3) Statistics field, used to collect statistics

for matching flow such as number of received packets, number of received bytes, duration, and etc.

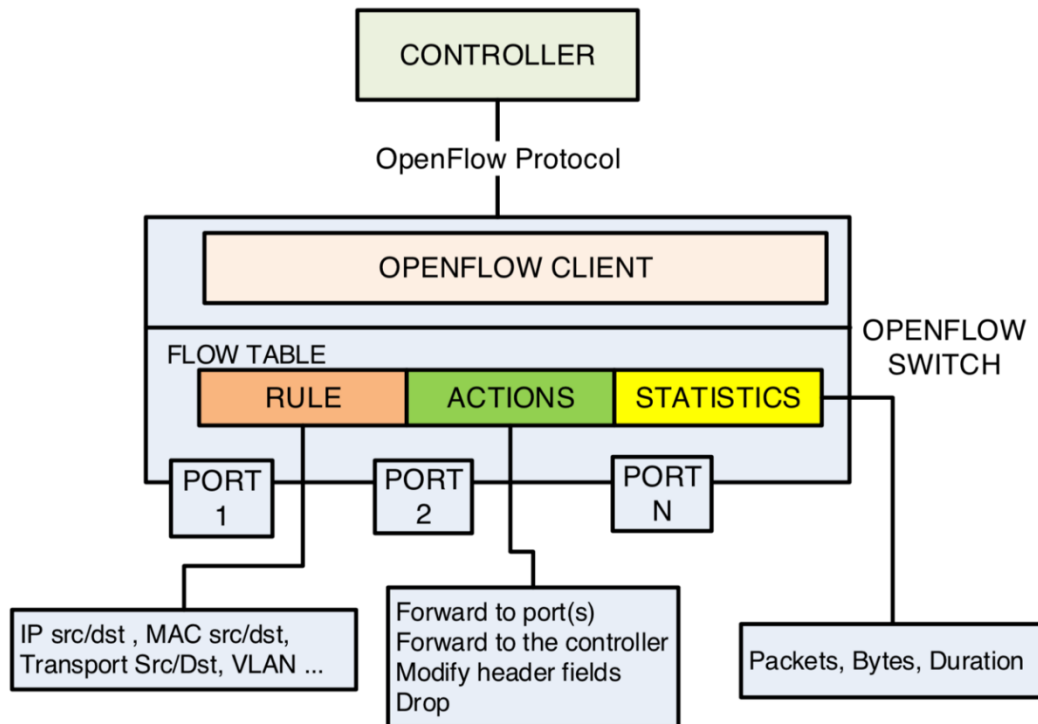


Figure 4 OpenFlow architecture [32].

Since March 2008, several versions of OpenFlow specification have been released such as 0.2.0, 0.8.9, 1.0.0, 1.3.0, 1.5.0, and etc. Some of them are now deprecated. However, the popular OpenFlow versions which are widely deployed is 1.0.0 and 1.3.0. The current commercial OpenFlow-enabled switches and open-source SDN controller software support at least version 1.0.0 and 1.3.0. In general, an OpenFlow specification consists of the requirements of OpenFlow switch, the definitions of Switch-to-Controller message, and the definitions of Controller-to-Switch message. In the Figure 5 demonstrates OpenFlow specification version 1.0.0 [39] and 1.3.0 [40].

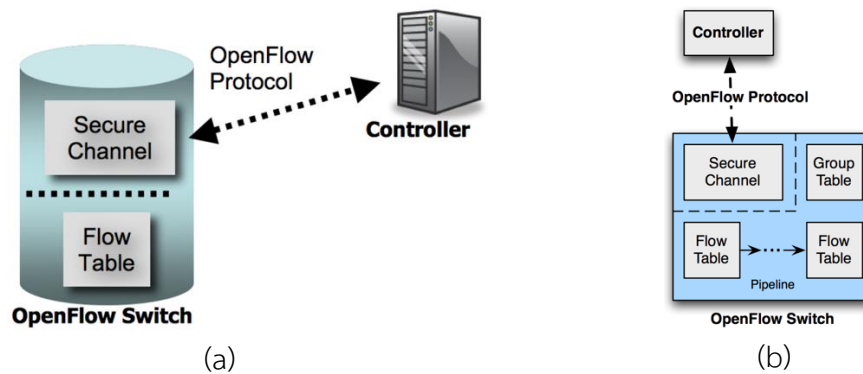


Figure 5 OpenFlow specification (a) version 1.0.0 [39] (b) version 1.3.0 [40].

The requirements of version 1.0.0 of OpenFlow switch consists of Secure Channel and one Flow Table but in case of version 1.3.0, there are multiple Flow table and one Group Table. The Flow Table will be modified by SDN controller via OpenFlow protocol over the Secure Channel. In addition, the version 1.3.0 supports Pipeline matching of multiple Flow Table. This provides for flexibility and Scalability in practice. However, the common specifications of message types such as Controller-Switch and Asynchronous message, Symmetric message between two versions are almost the same. Thus, the following explanation, it will be focused on the version 1.0.0 only.

2.1.4 OpenFlow Specification (Based on Version 1.0.0)

2.1.4.1 Flow Table

The Flow Table of OpenFlow switch is represented to forwarding rules. An incoming packet will be processed by these rules. A flow table entry consists of three components: Header Fields, Counters, and Actions followed by Table 1.

Table 1 Flow table entry.

HEADER FIELD	COUNTERS	ACTIONS
--------------	----------	---------

- **Header Fields** to match against packets.

This field is compared with the content of packet headers which are coming to a switch port. Each entry contains a specific value, or any which matches any values. The Field in OpenFlow 12-tuple are listed in Table 2 and the details of each field are described in Table 3.

Table 2 Match field of flow table entry.

INGRESS PORT	ETHER SRC	ETHER DST	ETHER TYPE	VLAN ID	VLAN PRIORITY	IP SRC	IP DST	IP PROTOCOL	IP TOS BITS	TCP/UDP SRC PORT	TCP/UDP DST PORT
-----------------	--------------	--------------	---------------	------------	------------------	-----------	-----------	----------------	-------------------	------------------------	------------------------

Table 3 Field lengths and the way they must be applied to flow entries.

Field	Bits	When Applicable	Notes
Ingress Port	(Implementation dependent)	All packets	Numerical representation of incoming port, starting at 1.
Ethernet source address	48	All packets on enabled ports	
Ethernet destination address	38	All packets on enabled ports	
Ethernet type	16	All packets on enabled ports	An OpenFlow switch is required to match the type in both standard Ethernet and 802.2 with a SNAP header and OUI of 0x000000. The special value of 0x05FF is used to match all 802.3 packets without SNAP headers.

VLAN id	12	All packets of Ethernet type 0x8100	
VLAN priority	3	All packets of Ethernet type 0x8100	VLAN PCP field
IP source address	32	All IP and ARP packets	Can be subnet masked
IP destination address	32	All IP and ARP packets	Can be subnet masked
IP protocol	8	All IP and IP over Ethernet, ARP packets	Only the lower 8 bits of the ARP opcode are used
IP ToS bits	6	All IP packets	Specify as 8-bit value and place ToS in upper 6 bits.
Transport source port / ICMP Type	16	All TCP, UDP, and ICMP packets	Only lower 8 bits used for ICMP Type
Transport destination port / ICMP Code	16	All TCP, UDP, and ICMP packets	Only lower 8 bits used for ICMP Code

- **Counters** to update for matching packet.

Counters are statistic values of packet matching which are maintained in terms of per-table, per-flow, per-port, and per-queue. A counter value is wrapped around with no overflow indicator. The Table 4 shows that required list of counters for use in statistic messages.

Table 4 Required list of counters for use in statistics messages.

Counter	Bits
Per Table	
Active Entries	32
Packet Lookups	64
Packet Matches	64
Per Flow	
Received Packets	64
Received Bytes	64
Duration (seconds)	32
Duration (nanoseconds)	32
Per Port	
Received Packets	64
Transmitted Packets	64
Received Bytes	64
Transmitted Bytes	64
Receive Drops	64
Transmit Drops	64
Receive Errors	64
Transmit Errors	64
Receive Frame Alignment Errors	64
Receive Overrun Errors	64
Receive CRC Errors	64
Collisions	64
Per Queue	
Transmit Packets	64
Transmit Bytes	64
Transmit Overrun Errors	64

- **Actions** to apply for matching packets.

Each Flow Table entry can be associated with zero or more actions to handle matching packets. The actions of OpenFlow switches can be divided by two types.

1) Required Action

- **ALL:** Send incoming packet out all interfaces excluding the incoming interface.
- **CONTROLLER:** Encapsulate and sent incoming packet to controller.
- **LOCAL:** Send incoming packet to local networking stack of switch.
- **TABLE:** Perform actions in flow table for packet-out message.
- **IN_PORT:** Send incoming packet out the input interface

2) Optional Action:

- **NORMAL:** Process incoming packet based on the traditional switch forwarding such as traditional L2, VLAN, and L3 processing.
- **FLOOD:** Flood incoming packet along minimum spanning tree.

2.1.4.2 OpenFlow Message

Message types of OpenFlow protocol consists of three types. First, Controller-to-Switch messages which is initiated by a controller for management and inspection. Second, Asynchronous messages is initiated by switches for network event notification and switch state changing. Final, Symmetric messages which is initiated by either a switch or a controller.

- Example of structures of Controller-to-Switch message

Read State Messages are used for collecting statistics from a switch. Several statistic levels that can be collected by this message type such as

Individual Flow Statistics, Table Statistics, Port Statistics, Queue Statistics, and etc.

- Example of structures of Asynchronous message

Packet-In Message is used for sending a packet which does not match any Flow Table Entries in a OpenFlow switch.

2.2 Related Work

2.2.1 Multipath Transfer Technologies over the Traditional Networking

In the traditional networking, the previous multipath transfer technologies can be categorized into three groups as follows.

- **Multipath transfer at the network layer:** Many routing protocols are allowed to use multipath via Equal Cost Multipath Protocol (ECMP) [41]. When a router has multiple destinations with same cost, the protocol will use one of them to provide a load balancing based on computing a hash function over packet headers with a selected transfer path. However, the technique leads to some drawbacks such as no aggregated capacity, uneven load balancing due to difference between big flow and small flow.
- **Multipath transfer at the transport layer:** Multipath TCP (MPTCP) [28] is an extension of Transmission Control Protocol (TCP) [42] which allows an application process to transfer any datagrams with multiple sockets over one or more network interfaces. Figure 6 illustrates MPTCP architecture. MPTCP is processed within Kernel space. The path manager is responsible for scheduling the incoming data stream to multiple subflows using TCP. MP_CAPABLE option will be included for establishing the Multipath TCP connection, if the receiver side also supports this protocol, it replies to the sender with MP_CAPABLE option. This is normal process of the traditional TCP protocol to establish their connection. Otherwise, it will use the traditional TCP instead. After the first TCP flow connection setup, MPTCP will use MP_JOIN option to create more subflows using Fullmesh approach. For example, an PC host which consists of two difference IP addresses with

different subnets to connect the internet, and the destination server also consists of two different addresses. The possible subflows that are initiated by the MPTCP protocol are four TCP subflows. Figure 7 demonstrates an example of MPTCP establishment. In addition to MPTCP, Concurrent Multipath Transmission for SCTP (CMT-SCTP) [43] is proposed in the literature. Although, CMT-SCTP can support multipath transmission using multiple network interfaces, its performance relies on the combination of source/destination IP addresses that are chosen. Whereas MPTCP creates a full mesh of possible paths among available addresses.

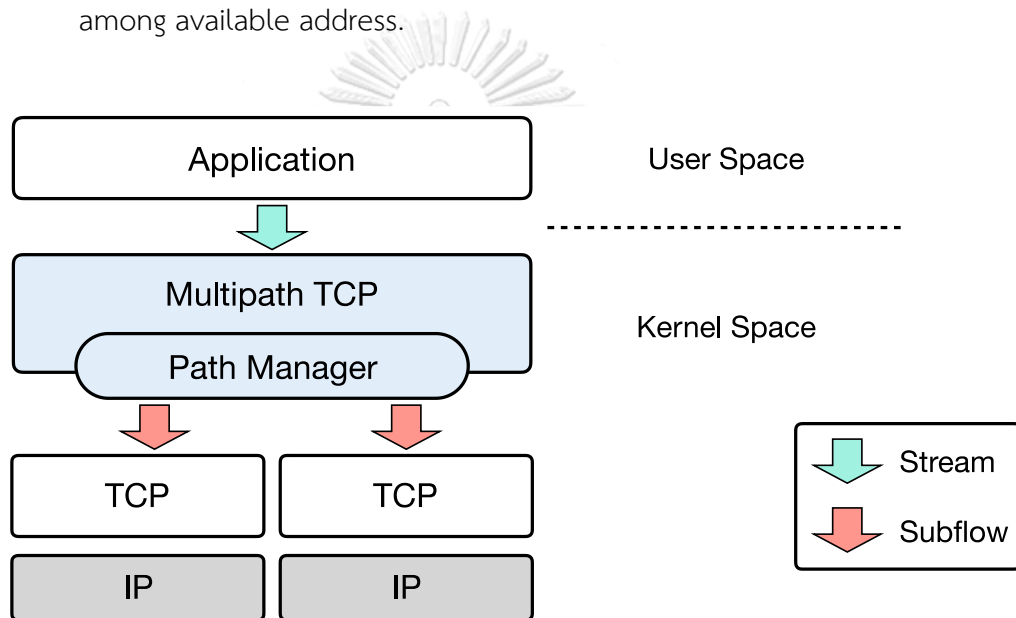


Figure 6 MPTCP architecture.

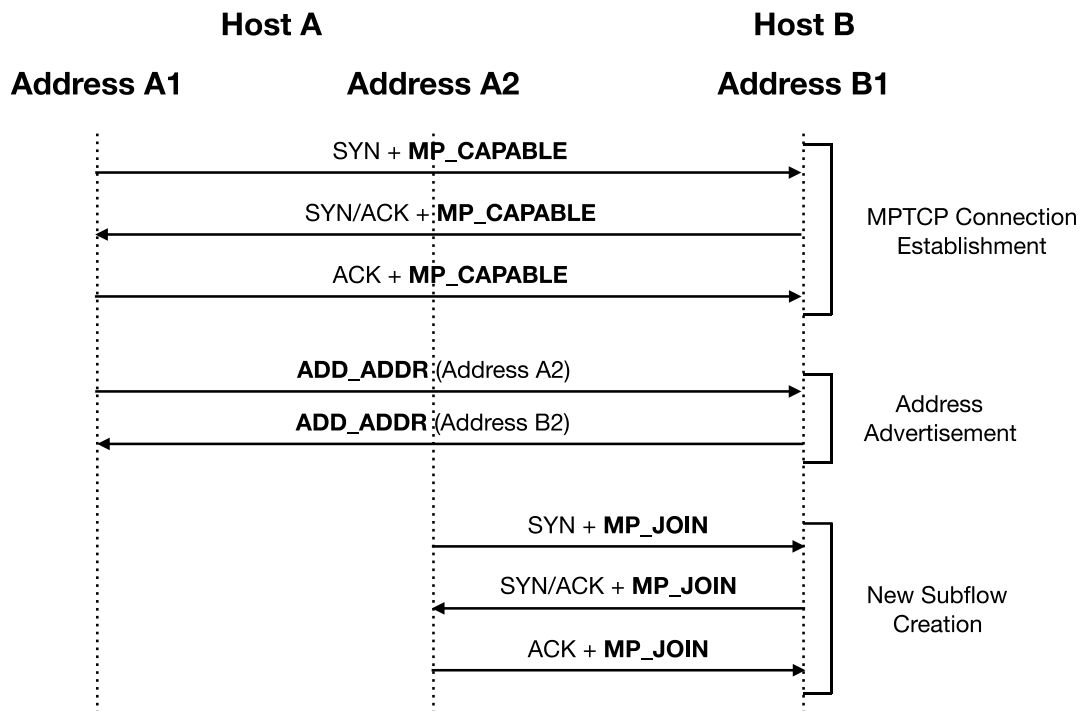


Figure 7 MPTCP establishment.

- Multipath transfer at the application layer:** Some applications support for multipath transferring which are implemented not only commercial software such as GridFTP [44] for high-speed data transfer, Internet Download Manager (IDM) but also several open-source web browsers which create the parallel TCP connections to increase the download speed of web content. Multipath QUIC (MPQUIC) [33] and Multiflow QUIC (MFQUIC) [34]. MPQUIC is an extension of QUIC protocol. It enables a QUIC connection to spread data over multiple network interface. Figure 8 illustrates MPQUIC architecture. According to the QUIC design, MPQUIC is also processed within User space. The path manager is responsible for scheduling multiple data stream to multiple subflows using UDP. MFQUIC is a variant of QUIC protocol that is aware of network asymmetries. It focuses on using multiple unidirectional flows instead of bidirectional flows as shown in Figure 9.

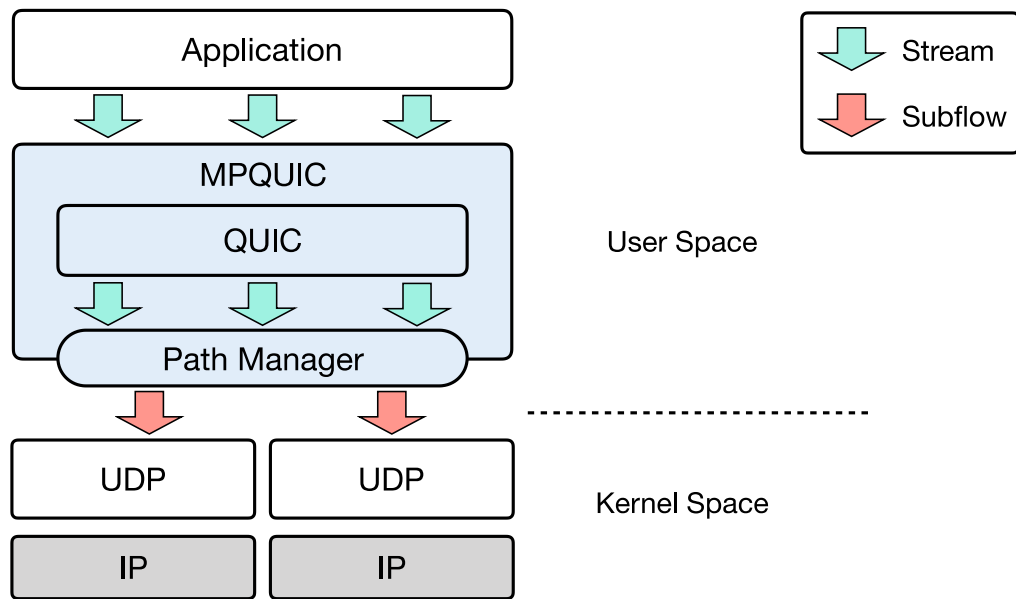


Figure 8 MPQUIC architecture.

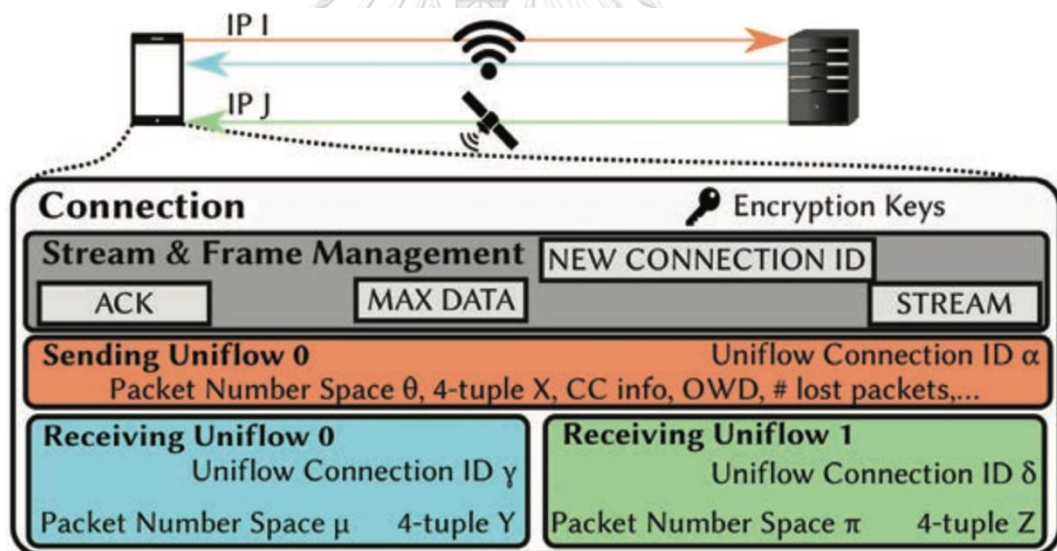


Figure 9 MFQUIC architecture [34].

We summarize the previous multipath technologies over the traditional networking in Table 5. GridFTP and Parallel Connections can provide high speed data transfer with low resource consumption. However, using those techniques induces unfairness and unfriendliness to other traffic in the network. Whereas Multipath QUIC and Multiflow QUIC are fair and friendly to other due to QUIC's congestion control.

However, they suffer from high resource consumption. This is because the encryption process that belong to QUIC mechanism is frequently executed on every packet and every subflow. CMT-SCTP and MPTCP do not suffer from the consumption issue. Moreover, both protocols are fair and friendly to other traffic as same as the QUIC-based protocols. However, currently there is only MPTCP that is defined as a standard transport protocol in RFC. Thus, in this dissertation, MPTCP represent a prototype technology for multipath data transfer that is mainly used to design and develop our proposed framework.



Table 5 Comparison of multipath transfer techniques over the traditional networking.

Multipath Transfer Approach	Non-Commercial Use	Low Resource Consumption	Be Fair and Friendly to Other Traffic Flows	Defined as a Standard Transport Protocol in RFC
GridFTP [44]	X	✓	X	X
Parallel TCP	X	✓	X	X
Multipath QUIC [33]	✓	X	✓	X
Multiflow QUIC [34]	✓	X	✓	X
CMT-SCTP [43]	✓	✓	✓	X
Multipath TCP [28]	✓	✓	✓	✓

2.2.2 Multipath Transfer Approaches Using MPTCP over the Software-Defined Networking.

According to the benefits of MPTCP and SDN, many researches have been proposed to use MPTCP over SDN for big data transfer, which can be categorized into two groups.

The first group has focused on the data center environment. Due to the characteristics such as full access control and high path-diversity, most solutions aim to increase the network utilization. In [14-16], the available bandwidth was the primary parameter which was considered by routing algorithms. The authors in [17] applied the number of active flows on each link to define the path cost against the link capacity. Exploiting path diversity, the authors in [18] proposed to modify the path manager of MPTCP for using multiple subflows per IP-address pair. The authors in [19-21] focused on how to find the appropriate number of subflow creations based on the flow demand and network conditions.

The second group has focused on the Internet environment in general. To avoid the shared-bottleneck problem, a multipath OpenFlow controller [22-27] has been proposed.

Multiflow [22] was proposed to manipulate the transmission paths for MPTCP subflows in disjoint fashion as shown in Figure 10. The topology-pruning technique is applied to the Dijkstra algorithm. When an MPTCP endpoint negotiates to add a new subflow, the links that have been utilized by the previous subflow will be removed from the original topology. Then, the resulting topology will be used for the routing algorithm to select the shortest path for the new subflow. This makes two subflows of the same MPTCP connection use two different and disjoint paths. However, when there is a new incoming connection, the original topology will be used again at the starting point of the algorithm without consideration of paths that the current traffic flows are using. As a result, it is most likely that all traffic flows will eventually share the same paths, leading to inefficient network utilization in the overall network.

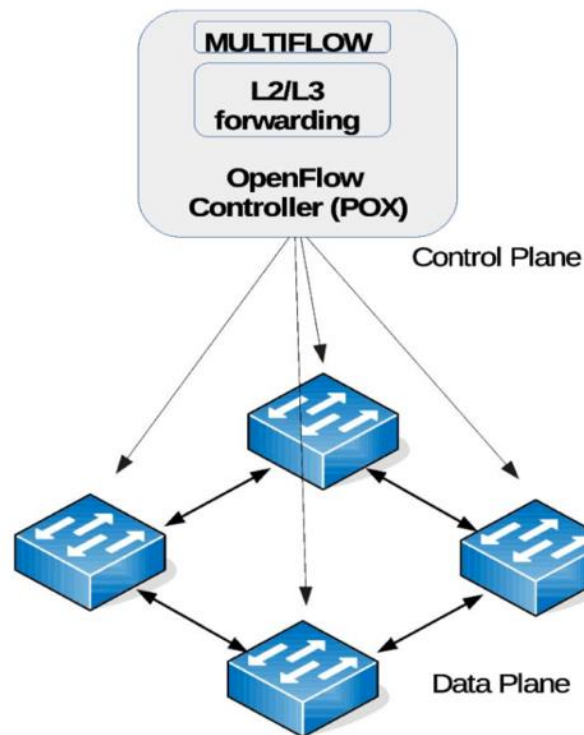


Figure 10 The system architecture of Multiflow [22].

SMOC [23], a simple multipath OpenFlow controller, was proposed to find the transmission paths for an MPTCP connection. Figure 11 illustrates the SMOC architecture. SMOC will use the NetworkX package, an open-source tool from NetworkX project [45], to find the set of all possible paths between a sender and a receiver. The first shortest path is defined to be a primary path with the highest priority. Other paths will be prioritized by the number of edges shared with the primary path and the path length in ascending. Then, SMOC will assign the set to each subflow to such a connection. However, using the routing algorithm, the controller needs to calculate all possible paths and prioritize those paths for every connection, leading to high complexity at the controller. This also increases the waiting time of incoming traffic.

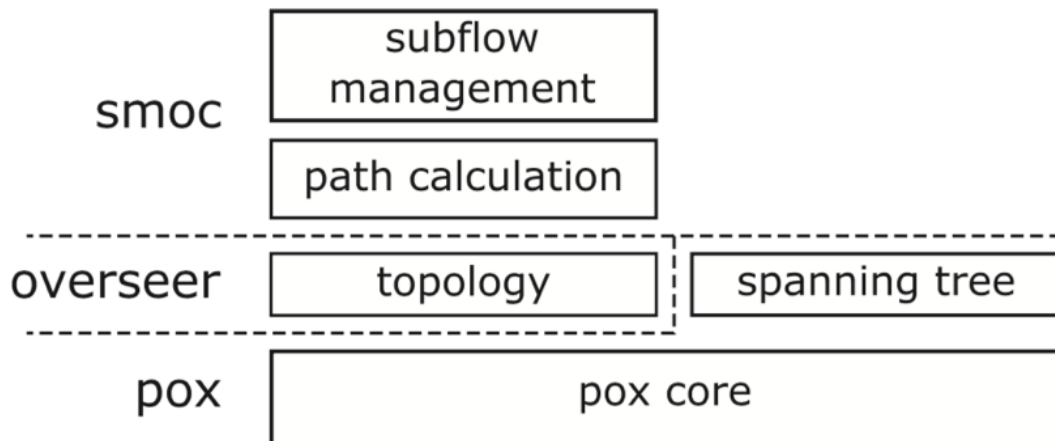


Figure 11 The system architecture of SMOC [23].

A heuristic algorithm for multipath routing was proposed [24]. The algorithm focused on how to find the optimal set of k disjoint paths from the candidate disjoint paths by maximizing the minimum bottleneck bandwidth path. The algorithm consists of two phases. First phase, the modifying Dijkstra algorithm is applied to find the set of candidate disjoint paths. Second phase, a greedy algorithm will be applied to select the optimal set based on the algorithm's objective. However, in practice, to support the algorithm with the remaining bandwidth information, it is difficult and complicated to do by using the standard OpenFlow protocol. Moreover, the Additive Increase Multiplicative Decrease (AIMD) algorithm in the standard transport protocol can lead to the variance issue on the bandwidth usage in the network.

SFO [25], a subflow optimizer for MPTCP, was proposed. In SFO architecture as shown in Figure 12, the controller will estimate the demand of an MPTCP session in terms of data rate requirement based on the type of traffic. Then, the routing algorithm will use the information to calculate the optimal k disjoint paths based on the network condition. This algorithm focuses on how to select the k disjoint path with the lowest total remaining bandwidth that is still satisfied to the data-rate requirement. However, the performance of the routing algorithm depends on the accuracy of the remaining-bandwidth measurement. In practice, this measurement metric is varied all the time and difficult to be precise. A lot of flow-rule tracking mechanisms must be deployed at the controller. This increases complexity to the system.

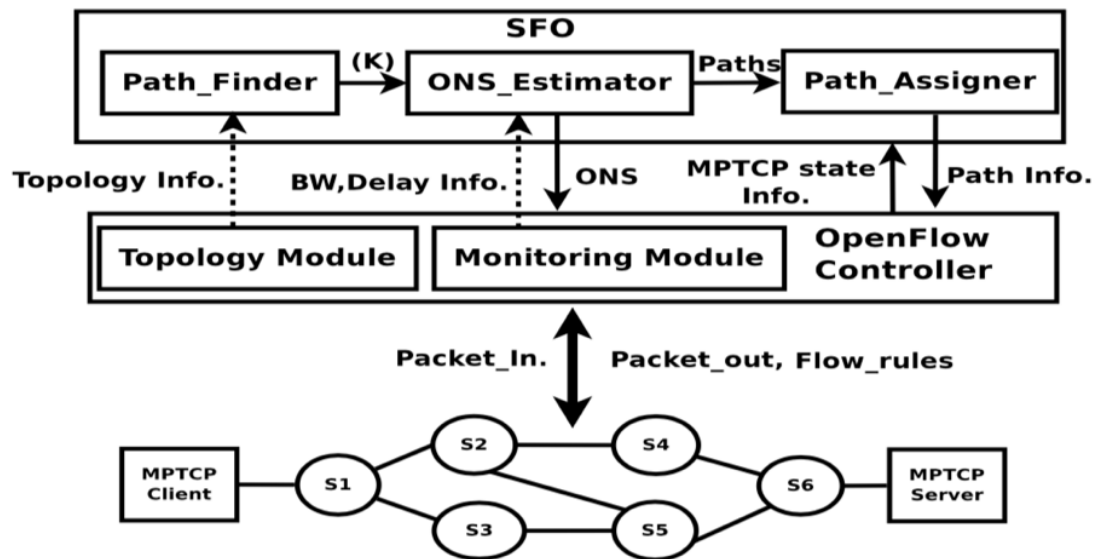


Figure 12 The system architecture of SFO [25].

S-MPTCP [26], a cross-layer technique between SDN controller and MPTCP protocol stack was proposed. The system design as shown in Figure 13 focused on efficient resource exploration and fair resource allocations among existing MPTCP connections. The system consists of three phases. In the first phase, the controller explores the bandwidth resource according to network statistics. Then, in the second phase, the optimal route set will be calculated and deployed at the data plane. Finally, in the last phase, the controller will calculate the expected throughputs for each connection and send those values to the corresponding hosts in order to set parameters for the congestion control mechanism. In practice, to obtain the network metric, several flow-rule tracking mechanisms need to be deployed. This leads to high complexity and a lot of polling messages in the control plane.

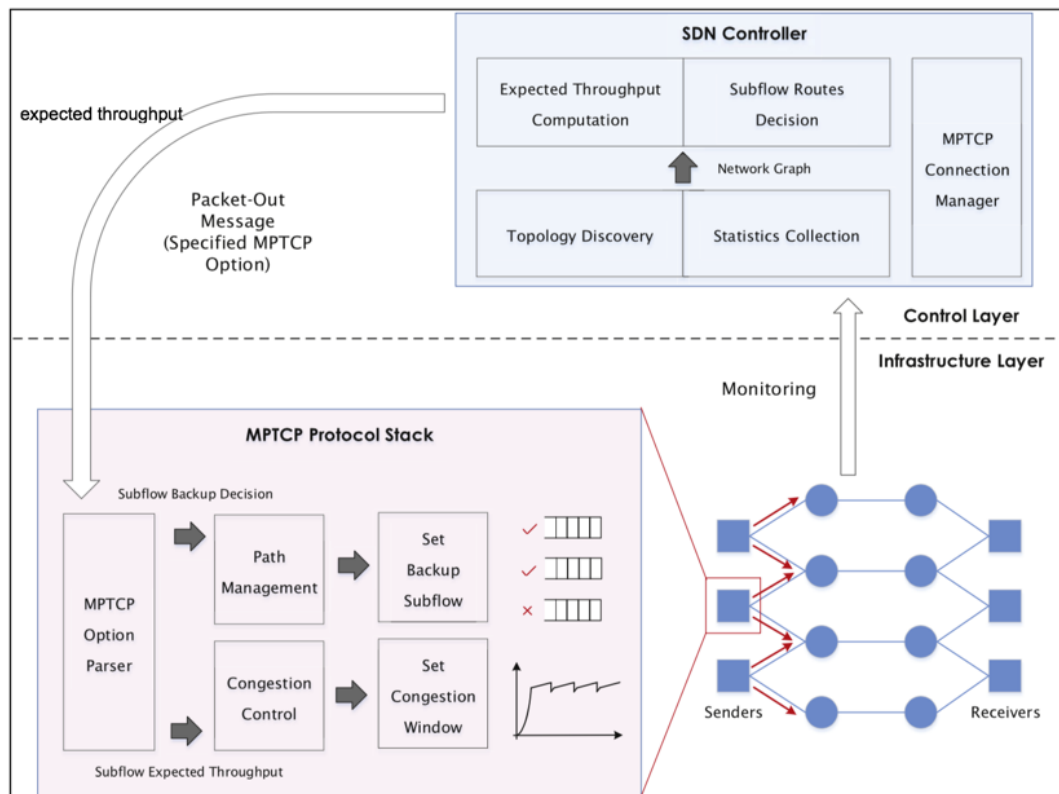


Figure 13 The system architecture of S-MPTCP [26].

GCLR [27], an GNN-based Cross Layer Optimization for MPTCP, was proposed. Figure 14 illustrates the system architecture of GCLR. The solution focuses on how to predict the expected throughput of an MPTCP connection using the GNN model. Three network-related metrics including link delay, remaining bandwidth, and packet loss rate are considered in the GNN model. The output of the model can be used as a guidance to adjust the congestion window at the endpoint. However, to collect those network-related metrics, both flow-rule and link tracking mechanisms must be deployed. This leads to a huge polling traffic at the control plane.

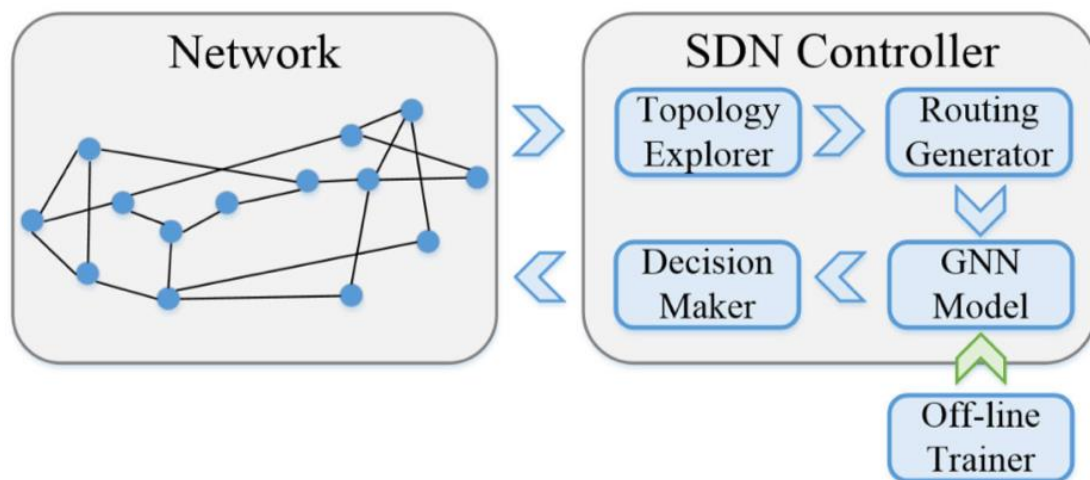


Figure 14 The system architecture of GCLR [27].

2.3 Motivation

We summarize the related works mentioned above in Table 6. As can be seen, although Multiflow [22] and SMOC [23] are practical with the OpenFlow protocol, they do not consider the traffic distribution issue. Additionally, SMOC [23] suffers from extremely-high complexity with a factorial function. The heuristic algorithm [24], SFO [25], S-MPTCP [26], and GCLR [27] consider the issue of traffic distribution and attempt to optimize the remaining bandwidth. However, their complexities are very high. Moreover, they are not practical with the standard OpenFlow protocol.

In this dissertation, our proposed framework focuses on how to provide a practical multipath transmission scheme for big data transfer applications using MPTCP. The traffic distribution, the complexity of the routing algorithm, and the practicality of the OpenFlow protocol are mainly considered for the design of our multipath OpenFlow controller. The OpenFlow statistics from the OpenFlow messages will be collected and processed in order to evaluate the level of congestion. In addition, the topology-pruning technique will be used in our routing algorithm to reduce the time complexity of path calculation. Three following approaches are implemented to compare the performance with our algorithm. First, the traditional routing is represented as the existing approach on the Internet. Second, the traditional routing with disjoint paths is represented as the solutions taken in [22, 23]. Third, the k -max disjoint routing is represented as a group of solutions [24-27] that require the network-

related metrics. It focuses on selecting the first k paths that have the maximum remaining bandwidth among candidate disjoint paths.



Table 6 Comparison of MPTCP-based routing decision in Software-Defined Networking.

Multipath OpenFlow Controller	Traffic Distribution	Complexity of Routing Algorithm*	Complexity Function	Parameters of Routing Algorithm	Practicality with the Standard OpenFlow Protocol
Multiflow [22]	X	$O(E \log(V))$	Linearithmic	Hop Count	✓
SMOC [23]	X	$O(V!)$	Factorial	Hop Count, No. of Shared Edges	✓
Heuristic Algorithm [24]	✓	$O(E \log(V) + V^2)$	Quadratic	Hop Count, Remaining Bandwidth	X
SFO [25]	✓	$O(K(KV(E + V \log(V))))$	Quadratic	Hop Count, Remaining Bandwidth	X
S-MPTCP [26]	✓	$O(V^2E)$	Quadratic	Hop Count, Remaining Bandwidth	X
GCLR [27]	✓	$O(V^2)$	Quadratic	Hop Count, Remaining Bandwidth, Link Delay, Packet Loss	X
Our Proposed Framework	✓	$O(E \log(V))$	Linearithmic	Hop Count, Port statistics	✓

*Complexity notation; V = number of vertices, E = number of edges, K = number of feasible paths.

CHAPTER 3

An SDN-Coordinated Multipath Transmission Steering Framework for Big Data Transfer Application

3.1 Definition of Big Data in This Dissertation

The definition of Big Data in our proposed framework is mentioned as the following explanations.

- Big Data refers to extremely large, fast growing, and complex data sets that it is difficult to manage and process using commonly used software tools.
- In terms of data type, the proposed framework is designed for transferring data with batch type only.
- The batch data is a set of data that has been stored or collected over a period of time.
- In terms of data size, the proposed framework can be performed with no limitation of data size.
- The data arrival must be a single batch file. It must not be Poisson distribution.

3.2 Framework Overview

Data transmission is a fundamental mechanism of distributed storage systems which is frequently used for data synchronization and data pulling. These big data traffic flows can be transferred across Local Area Networks (LANs) or Wide Area Networks (WANs). Basically, the architecture of the systems consists of three components: data servers, data clients, and network devices. Figure 15 is an example of a distributed data storage system over the Internet. The data servers are located in several domains or regions. A particular computing software is used in order to control and manage all data servers. The network interface that is attached to each server is used to communicate with each other and transfer data. All data traffic flows that are generated within the system will be steered by network routers through the Internet.

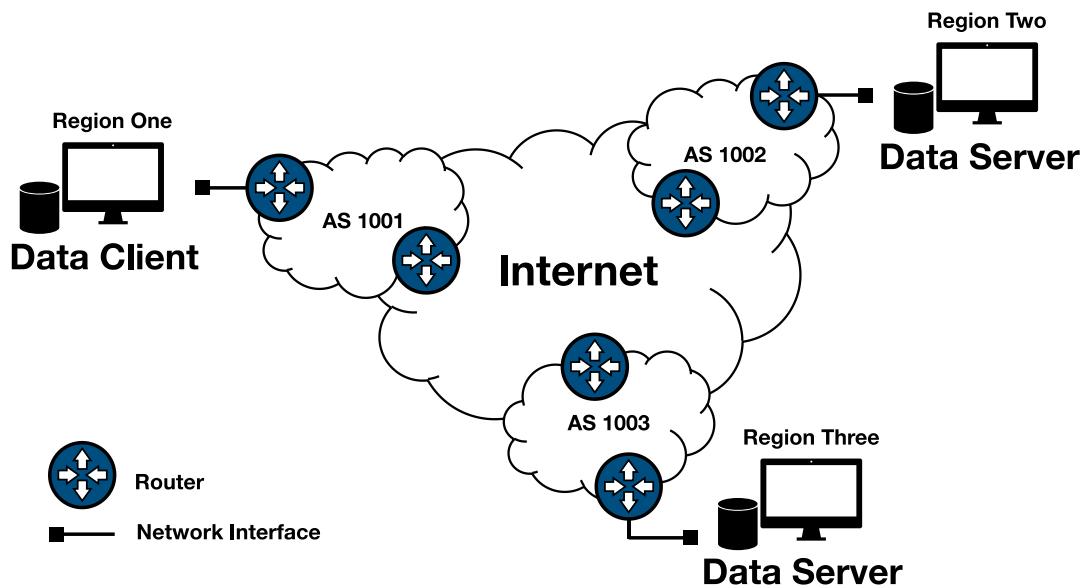


Figure 15 Distributed data storage system in the traditional network.

In practice, the data servers consist of one main server and multiple sub-servers. The main server is represented by a master node which is responsible for managing and controlling accessibility of the system. The others are represented by slave nodes which collect and protect the data based on storage policy. Thus, a data request needs to be processed by the master node before data loading. In terms of big data transmission, it can be divided into two major cases. The former is a transmission that is performed based on a specific request. The latter is a transmission that is related to the mechanisms of data exchanging due to the maintenance process. However, to find and create a transfer path for both use cases, it relies on the control logic inside the network routers. In the Internet case, the shortest path is always provided to all traffic flows.

In our framework, we apply SDN to manage and control those data traffic flows in the network. The existing network routers are replaced by OpenFlow switches which are controlled by the OpenFlow controller. Figure 16 illustrates an example of our framework with full-range deployment. Each OpenFlow switch is responsible for handling incoming data traffic based on OpenFlow rules within flow tables. The OpenFlow controller is responsible for making a routing decision and creating the rules for the OpenFlow switches. All control messages that are generated by the controller

and switches are encrypted by TLS over TCP connection and sent through the secure channel. In practice, the channel can be deployed by two types: out-of-band and in-band. For the first type, each OpenFlow switch allocates a dedicated interface to communicate with the controller. All control messages will be transferred by this interface only. For the second type, each OpenFlow switch uses the existing interfaces that are used by the data traffic to transfer the control messages. Each endpoint of data transmission such as data server and data client are attached with multiple network interfaces and configured to support the network multihoming and MPTCP protocol.

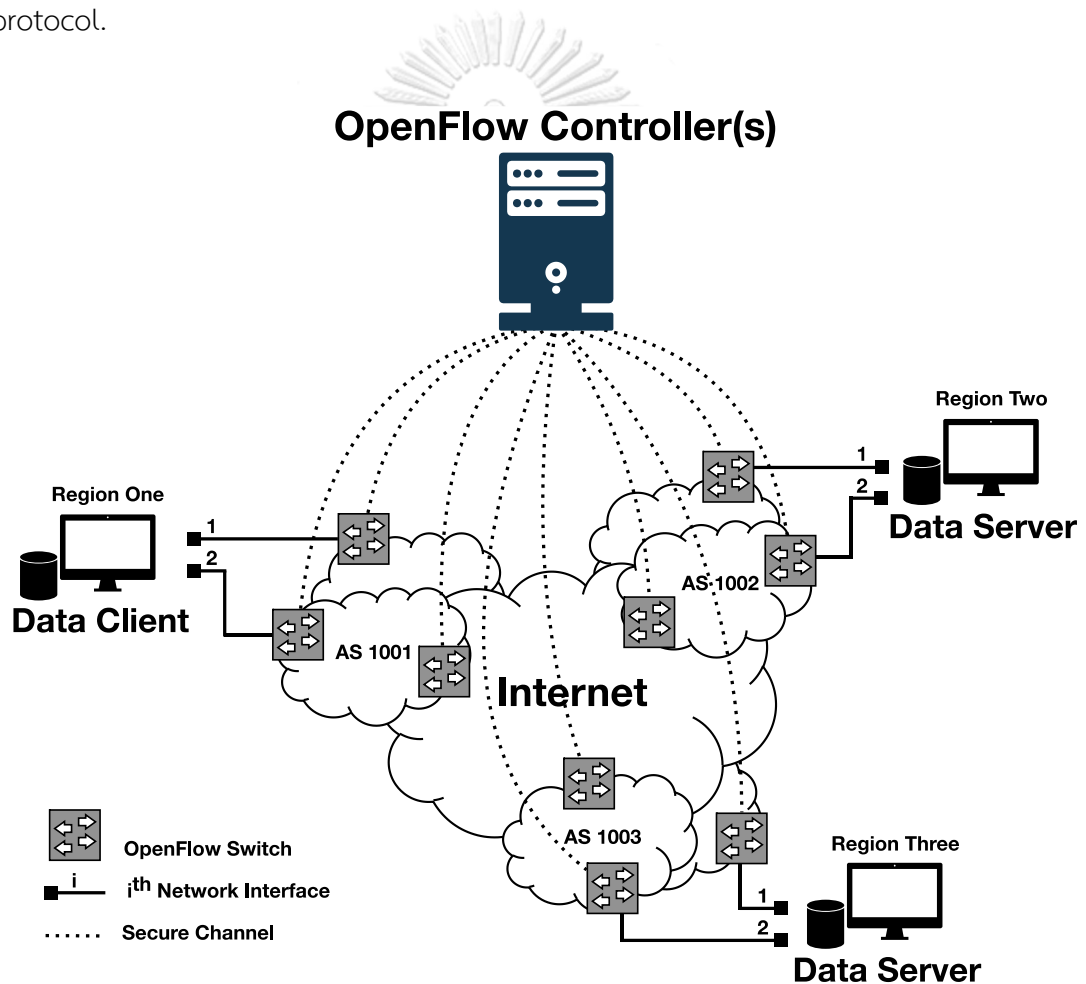


Figure 16 Distributed data storage system in our proposed framework with full-range deployment.

In practice, shrinking the applicability of the proposed solution to selected scenarios, the adoption of *VXLAN* tunnels is useful to avoid the full-range deployment

of OpenFlow-enabled networking devices. Figure 17 illustrates an example of our proposed framework being applied to an overlay network. OpenFlow switches can be put between data servers/clients and the gateway routers. These switches are operated by our controller and can communicate to each other through VXLAN tunnels. For more technical details, please see *Appendix A*. Thus, our proposed framework can also be applied to any overlay network or private cloud WAN, for instance, Google's B4 [46].

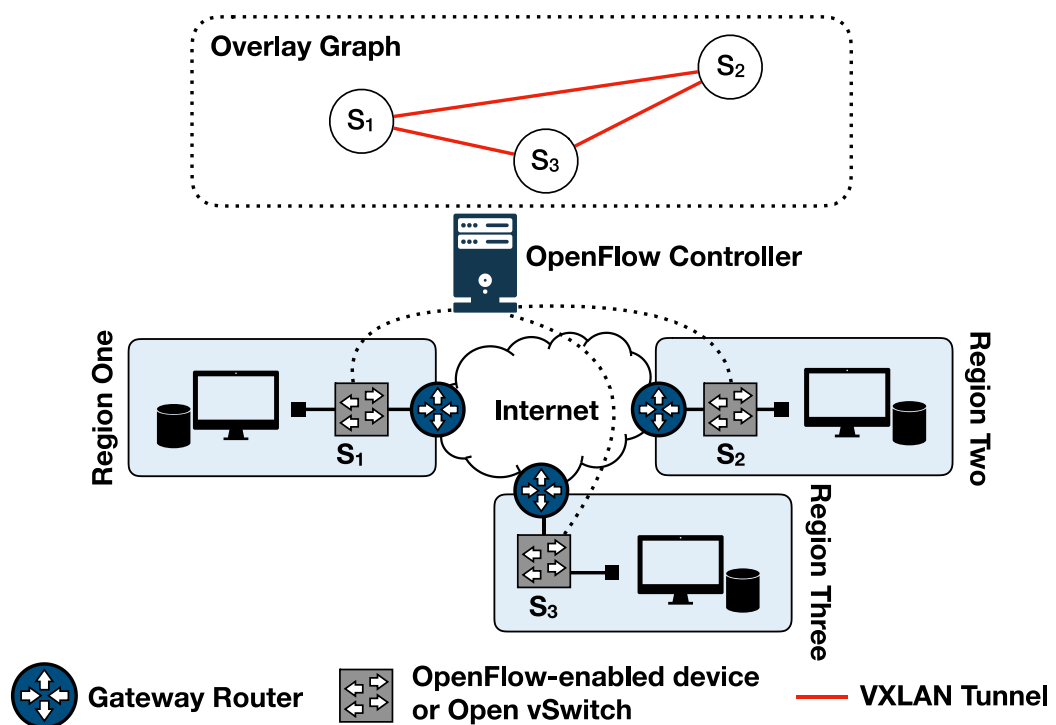


Figure 17 Distributed data storage system in our proposed framework being applied to an overlay network

Data transmission consists of two phases: data request phase and data transfer phase. Figure 18 demonstrates a sequence diagram of data request and data transfer phases. In the data request phase, the data client generates a request message with metadata and sends it to the main data server through SDN (Step 1). The OpenFlow switch that is directly connected to the client creates a Packet_In message and sends it to the OpenFlow controller. This mechanism will be automatically processed by

OpenFlow protocol when a switch cannot find a matching rule for an incoming traffic (Step 2). The controller extracts the message to make a routing decision and create OpenFlow rules to the related switches (Step 3). Then the request message is transferred to the main server (Step 4). When the main server receives the request message, it assigns the appropriate sub-server that depends on the storage policy to transfer the data (Step 5). A notification message and response message will be sent to the sub-server and the data client, respectively (Step 6 - Step 7).

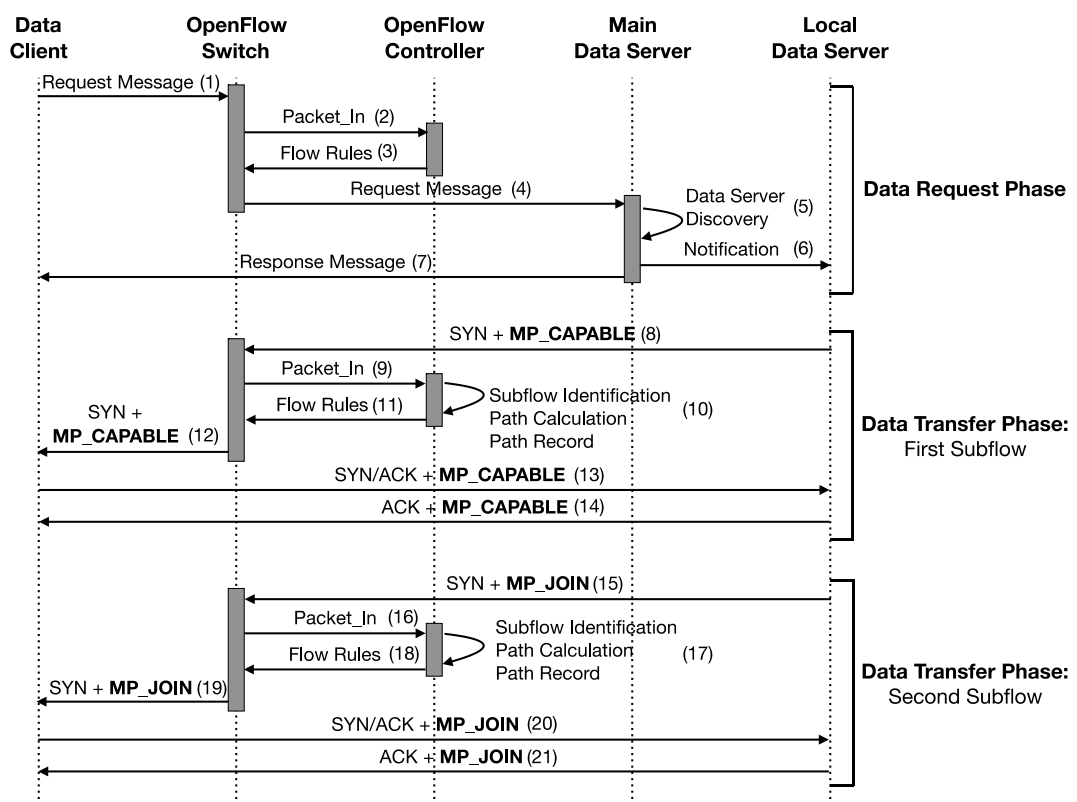


Figure 18 Sequential diagram of data request and data transfer in our proposed framework.

In the data transfer phase, the MPTCP at the sub-server starts to negotiate with the data client for connection establishment. Based on the RFC 8684, the MP_CAPABLE options will be used in order to create the first subflow. Other subflows will use MP_JOIN options as shown in Figure 18. Firstly, the SYN packet with MP_CAPABLE option will be sent to SDN (Step 8). The OpenFlow switch that is directly connected

to the sub-server generates a Packet_In message to the controller (Step 9). The controller calculates the transfer path and installs the OpenFlow rules to related switches (Step 10 - Step 11). The details of the routing algorithm are explained in Section 3.4. When the SYN packet arrives at the data client, the SYN/ACK packet with MP_CAPABLE will be sent back to the sub-server (Step 12 - Step 13). Then the server responds with the ACK packet and starts to transfer data respectively (Step 14).

In terms of additional subflow creation, the sub-server will negotiate with the same mechanism as the first subflow but the MP_JOIN options will be used instead (Step 15 - Step 21). In our framework, the MPTCP options that are extracted during the negotiation mechanism will be recorded and considered for routing decisions.

3.3 Practical Issues on Using MPTCP Protocol and OpenFlow Protocol

This section, we explain the key mechanisms of Multipath TCP (MPTCP) protocol and OpenFlow protocol including their practicality issues.

3.3.1 Issue on MPTCP Protocol

MPTCP is a standard transport protocol for multipath transmission that supports the network multihoming at end hosts. Based on the RFC 8684, the MPTCP is designed to be compatible with the traditional TCP. The control mechanisms of MPTCP will be processed within the option space of the transport header. Each TCP connection that is generated by MPTCP is called subflow. The number of subflow creation per one connection is relied on the parameter setting of the path manager [47]. For example, with the default setting of the Fullmesh parameter, the maximum number of subflows that will be negotiated is the number of all available pairs of network addresses between two end hosts.

Figure 19 demonstrates a sequence diagram of MPTCP establishment. Each subflow will be created by using a traditional three-way handshake, the same as the TCP establishment. During the handshaking, the control messages will be exchanged between two end hosts by appending MPTCP options to the existing transport header. For example, to create the first subflow, the MP_CAPABLE option is appended to the transport header of the SYN, SYNACK, and ACK packets. After that, the ADD_ADDR

messages will be sent for advertising the additional IP addresses to the path manager of MPTCP. Then, Host A starts to transfer the data. To generate a new subflow, Host A negotiates with the three-way handshake again but the MP_JOIN options will be used instead. Then, the new subflow starts to transfer the data.

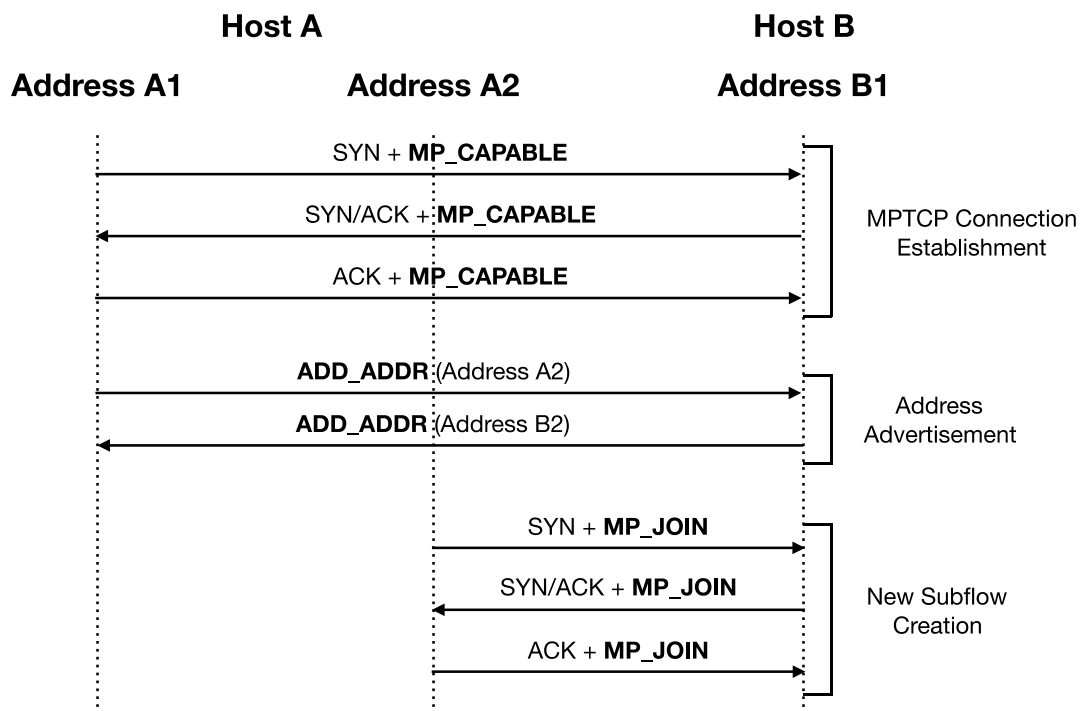


Figure 19 MPTCP connection establishment and subflow creation mechanisms.

Although MPTCP can improve the throughput of data transfer by using multiple subflows simultaneously, serving a lot of traffic flows without global knowledge considerations of network topology and link conditions leads to two problems: the shared-bottleneck problem and the low-network-utilization problem. This is because the existing routing approach normally provides only the shortest path for all incoming traffic flows. Moreover, in the traditional network architecture, it is not possible to collect the global knowledge or route the traffic based on the flow requirement. To deal with such problems, our framework uses the capabilities of SDN and the OpenFlow protocol to recognize the MPTCP traffic and select the transfer paths for subflows based on network topology and switch-port statistics.

3.3.2 Issue on OpenFlow Protocol

The OpenFlow protocol is a standard control protocol which is designed for supporting the centralized management in SDN. The protocol defines messaging mechanisms between controller and switches including structures of flow rules. Figure 20 demonstrates an example of flow rule structure. The match fields are responsible for the conditions of that rule which is flexibly defined based on the headers in L2, L3 or L4. This enhances the capabilities of routing decisions. For example, a packet can be steered by 5-tuple packet header (Protocol, source/destination IPs, and source/destination ports) instead of depending upon the destination IP address only.

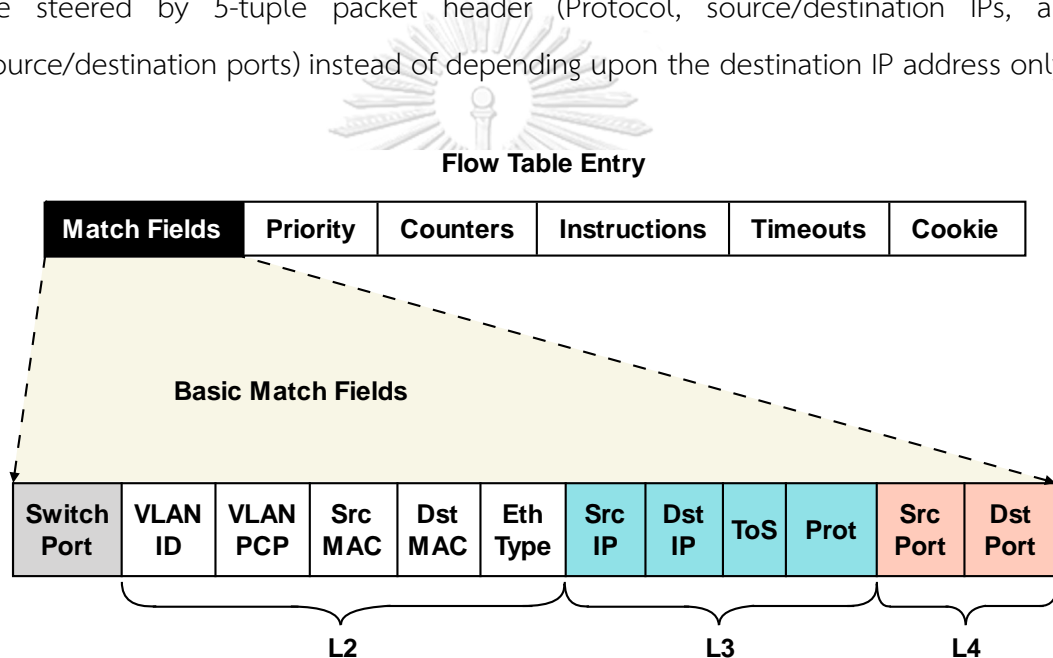


Figure 20 Example of flow rule structure with basic match fields.

In addition, the protocol provides the capability to collect network information such as network topology, link status, port statistics, and etc. Although the protocol can provide the flexibility of rule creation, the range of matching rules are limited. There are only a few existing header fields that can be used to match the rules on switches. In other words, the protocol does not support a new end-to-end protocol such as MPTCP. To cope with the OpenFlow limitations with MPTCP traffic, those MPTCP packets will be extracted and recorded in our controller. This information helps to identify and manage all MPTCP subflows in the network.

3.4 Our Framework Design

In this section, the design of our proposed framework and functionalities are explained in detail. Our framework is designed to facilitate big data transmission applications in multipath fashion using MPTCP. This is our unique transfer technique that we proposed for using in distributed data storage systems. All incoming MPTCP traffic flows can be detected and managed by our controller. The proposed framework consists of three new functional modules which can be deployed on the top of the existing controller. Figure 21 demonstrates our proposed modules running on the ONOS controller. As can be seen, there are three new modules in our framework: *Multipath Transmission Manager*, *Multipath Topology Manager* and *OpenFlow-Stats Analyzer*. These modules can be deployed as network control applications on top of any SDN controllers. In order to communicate with the controller, there are six fundamental functions needed in our framework as follows.

- 1) *Packet_In()*: This function is used to deliver a *Packet_In* message to our framework. The function is automatically executed by the controller when it receives a new *Packet_In* message from switches.
- 2) *Get_Topology()*: This function is used to discover the current network topology.
- 3) *Topology_Change()*: This function is used to acquire the notification of network failure in terms of nodes and links.
- 4) *Get_Port_Stats()*: This function is used to collect the statistics of switches' ports.
- 5) *Push_Flow_Rules()*: This function is used to assign flow rules to switches.
- 6) *Get_Host_Location()*: This function is used to acquire the location information of hosts in the network topology.

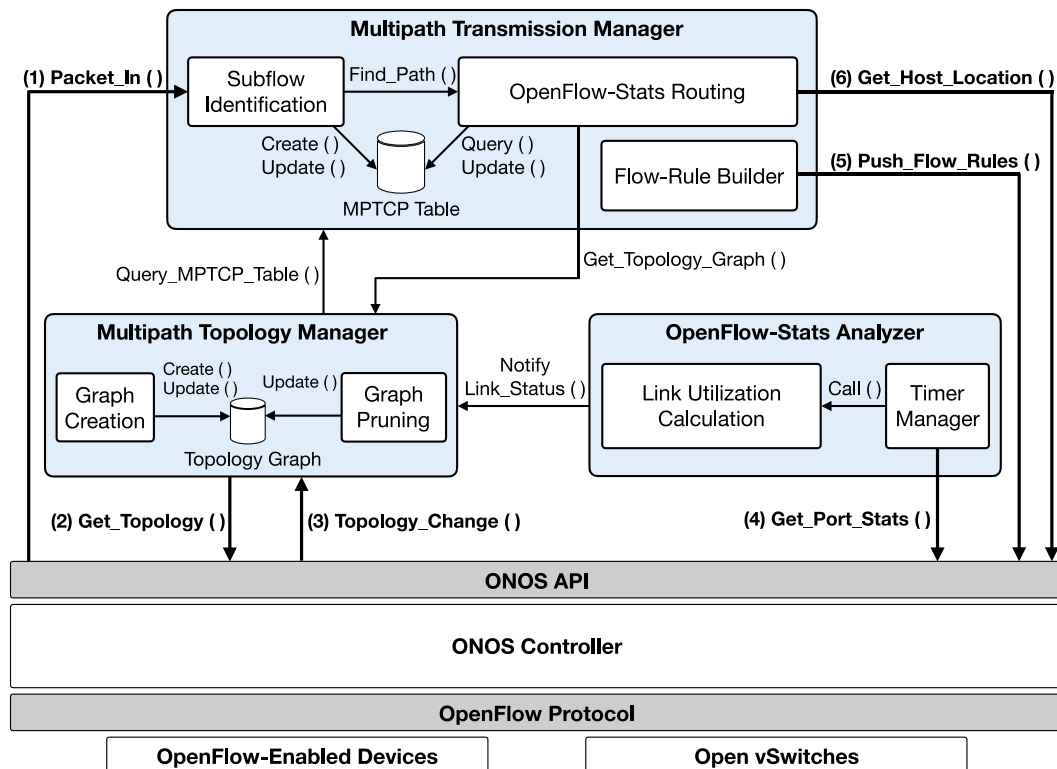


Figure 21 Our proposed functional modules over the ONOS controller.

3.4.1 Multipath Transmission Manager

This module is the main functional module which is used for network management. The module consists of two tasks.

The first task is to process incoming packets. This is related to the extraction and identification of packet header. When this module receives a new packet by the Packet_In() function, it extracts the MPTCP options from the packet header. The MPTCP options including 5-tuple header and TCP flag will be recorded in our MPTCP table. Table 7 shows the structure of our MPTCP table. The table is primarily used to identify an incoming MPTCP flow and perform the topology pruning in *Multipath Topology Manager*. As can be seen, there are three identifiers for each subflow that consists of end-to-end communication ID, connection ID, and subflow No.

Table 7 Structure of MPTCP table.

End-to_End Communication ID	Connection ID	Subflow No.	5-Tuples	TCP Flag	MPTCP Type	Key	Token	Random No.	HMAC	Route
1	1.1	1.1.1	...	SYN	MP_CAPABLE	Key1	-	-	-	{ S1, S2, S3 }
			...	SYN/ACK	MP_CAPABLE	Key2	Token1	-	-	{ S3, S4, S1 }
		1.1.2	...	SYN	MP_JOIN	-	Token1	Random1	-	{ S1, S5, S3 }
			...	SYN/ACK	MP_JOIN	-	-	Random2	HMAC1	{ S3, S4, S1 }
...										

In MPTCP establishment, SYN and SYN/ACK packets will be recorded in the MPTCP table. Similarly to the traditional TCP approach, MPTCP uses the same 5-tuple header in connection establishment phase, datagram transfer phase, and connection closing phase. When forwarding rules for SYN packet and SYN/ACK packet are installed, the following packets are processed by the same rules. In addition, the necessary information of MPTCP protocol such as key, token and etc. is recorded to identify subflows among MPTCP connections.

The second task is related to path calculation and rule installation. The path calculation is the most important mechanism which can affect the performance of MPTCP traffic. It can be seen that in [22-27], the performance of MPTCP can be significantly improved by using disjoint paths and considering the remaining bandwidth. However, in practice, it is difficult and complicated to measure the remaining bandwidth correctly. This is because the network links can be filled with unstable and heterogeneous traffic. A lot of flow-rule tracking mechanisms need to be deployed and processed in real-time on every node and link. This increases complexity to the system.

To mitigate the above-mentioned problem, in this dissertation, we propose a new OpenFlow-Stats routing algorithm. In our routing decision, the port stats of switches is considered as a link quality instead of the remaining bandwidth. This is our unique design that makes our routing algorithm compatible with the OpenFlow protocol. Our *OpenFlow-Stats Analyzer* is responsible for collecting the port statistics and evaluating the level of port utilization. Our *Multipath Topology Manager* uses the information of the level of port utilization to create an OpenFlow-Stats graph which is an important input of our OpenFlow-Stats routing algorithm. The details of evaluating the link quality are explained in Section 3.4.3. In order to reduce the complexity of disjoint-path consideration, the topology-pruning technique is applied. Specifically, the links that have been utilized by the previous subflows will be removed from the topology graph. The resulting graph will then be used by Dijkstra's algorithm [48], a well-known shortest-path algorithm, to select a disjoint path. Our *Multipath Topology Manager* uses the routes which are recorded in the MPTCP table to perform the topology pruning.

The pseudocode for OpenFlow-Stats routing algorithm is shown in Figure 22. The idea is to use the resulting OpenFlow-Stats graph to find the shortest path first. However, if there is no available shortest path in the OpenFlow-Stats graph, the original graph will be used instead. The complexity of Algorithm 1 is $O(E \log(V))$ where E is the number of edges and V is the number of vertices in the graph.

Algorithm 1: OpenFlow-Stats Routing Algorithm

Input: Switch1 $S1$,
Switch2 $S2$,
MPTCP Option MP_opt ,
MPTCP Table MP_table ,
OpenFlow-Stats Graph OS_graph ,
Original Graph O_graph

Output: Path $Path$

- 1: **if** MP_opt is $MP_CAPABLE$ **do:**
- 2: $Path \leftarrow$ Dijkstra ($S1, S2, OS_graph$)
- 3: **if** $Path = \emptyset$ **do:**
- 4: $Path \leftarrow$ Dijkstra ($S1, S2, O_graph$)
- 5: **end if**
- 6: **end if**
- 7: **if** MP_opt is MP_JOIN **do:**
- 8: $l_set \leftarrow$ get_used_links (MP_opt, MP_table)
- 9: $new_graph \leftarrow$ prune (l_set, OS_graph)
- 10: $Path \leftarrow$ Dijkstra ($S1, S2, new_graph$)
- 11: **if** $Path = \emptyset$ **do:**
- 12: $new_graph \leftarrow$ prune (l_set, O_graph)
- 13: $Path \leftarrow$ Dijkstra ($S1, S2, new_graph$)
- 14: **if** $Path = \emptyset$ **do:**
- 15: $Path \leftarrow$ Dijkstra ($S1, S2, O_graph$)
- 16: **end if**
- 17: **end if**
- 18: **end if**
- 19: update ($Path, MP_opt, MP_table$)
- 20: **return** $Path$

Figure 22 Pseudocode of our OpenFlow-Stats routing algorithm.

3.4.2 Multipath Topology Manager

This functional module is designed for graph maintenance. The information of network topology from the existing network discovery module of the controller will

be collected and transformed by using the `Get_Topology()` and `Topology_Change()` function. To create the OpenFlow-Stats graph, this module uses the information of port utilization which is analyzed by our *OpenFlow-Stats Analyzer* in order to determine the status of network links. If the port stats is attached to Disable label, the links will be removed from the OpenFlow-Stats graph. Otherwise, the link will be added to the graph. In addition, this module is responsible for performing topology pruning based on the demand of OpenFlow-Stats routing in the *Multipath Transmission Manager*.

The pseudocode for Topology-Pruning algorithm is shown in Figure 23. The complexity of Algorithm 2 is $O(E)$ where E is the number of edges that relates to the $Switch_{ID}$.

Algorithm 2: Topology-Pruning Algorithm

Input: Link Set with Switch ID $L_{switchID}$,
Topology Graph G

Output: Topology Graph G

```

1:   for each  $l$  in  $L_{switchID}$  do
2:      $G \leftarrow \text{remove}(l, G)$ 
3:   end for
4:   return  $G$ 

```

Figure 23 Pseudocode of our topology-pruning algorithm.

3.4.3 OpenFlow-Stats Analyzer

In our framework, the level of port utilization is utilized to specify the link quality. In order to evaluate a link quality, this module is responsible for collecting the port statistics of switches, evaluating the level of port utilization, and notifying the *Multipath Topology Manager*.

To collect the statistics, *OpenFlow-Stats Analyzer* creates a set of probing timers. One timer is attached to one switch. When a timer expires, this module executes the `Get_Port_Stats()` function to the controller. Smaller probing interval can provide higher accuracy of port-stats information. However, it can lead to higher

complexity at the control plane. In practice, this value is set based on the network administrator and the network policy.

To evaluate the level of port utilization, in this dissertation, we propose a new OpenFlow-Stats monitoring algorithm. In our monitoring algorithm, the status of each port is estimated based on the number of bytes which are processed within a probing interval. If the value is above a certain threshold, the port is attached to Disable label. Otherwise, the port is attached to Enable label.

The pseudocode for OpenFlow-Stats monitoring algorithm is shown in Figure 24. The complexity of Algorithm 3 is $O(E)$ where E is the number of edges that relates to the $Switch_{ID}$.



Algorithm 3: OpenFlow-Stats Monitoring Algorithm

Input: Switch ID $Switch_id$,
Output: Port Set of Switch ID $P_{Switch\ ID}$

```

1:    $P_{Switch\ ID} \leftarrow \emptyset$ 
2:   for each  $p \in Switch\_id$  do:
3:      $\Delta \leftarrow \text{calculate\_delta\_bytes}(p)$ 
4:     if  $\Delta > Threshold$  do:
5:        $status \leftarrow \{Disable\}$ 
6:     else do:
7:        $status \leftarrow \{Enable\}$ 
8:     end if
9:      $P_{Switch\ ID} \leftarrow P_{Switch\ ID} \cup \{p, status\}$ 
10:  end for
11:  return  $P_{Switch\ ID}$ 

```

Figure 24 Pseudocode of our OpenFlow-Stats monitoring algorithm.

CHAPTER 4

Performance Evaluation

4.1 Experimental Setup

4.1.1 Simulation Configuration

To evaluate performance of our proposed framework, we implemented it on the Mininet v2.3.0 [36], a popular network emulator. OpenFlow switches, OpenFlow links, and hosts were created using this emulator. All switches were controlled by a remote controller. We used the Open Network Operating System (ONOS v1.13) [37] as the controller in our framework for discovering and configuring the OpenFlow switches. The discovery process and the switch-configuration process relied on the standard versions of OpenFlow protocol (Version 1.3). Our proposed modules described in Section 3.4 were implemented as a new control application running on top of the controller. For MPTCP implementation, we created two network interfaces and installed the MPTCP v0.93 [47] for every host. All MPTCP parameters were set according to the default setting.

4.1.2 Simulation Scenario

The performance evaluation was done using two network topologies as follows.

The first topology is a multipath topology as illustrated in Figure 25. This topology represents the future Internet infrastructure where several completely-disjoint paths exist among Internet Service Providers (ISPs).

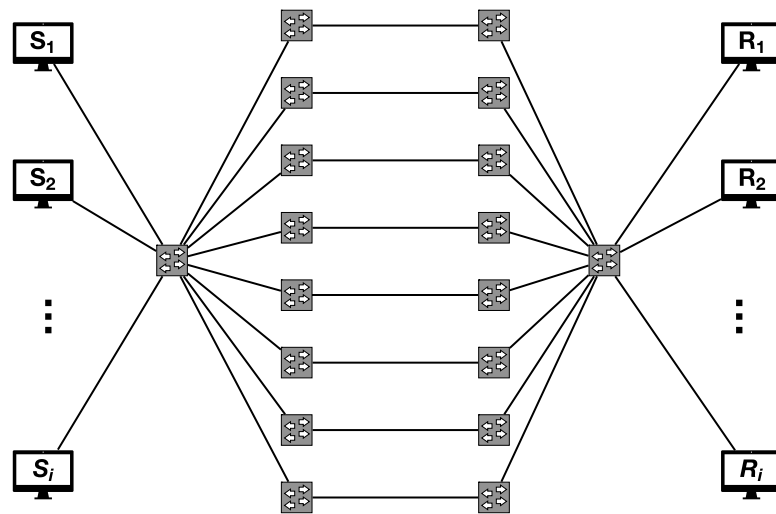


Figure 25 Multipath topology.

The second topology is the COST239 topology as depicted in Figure 26. This topology is an optical-WAN topology developed in Europe [49]. It is one of the most popular topologies used in this research field.

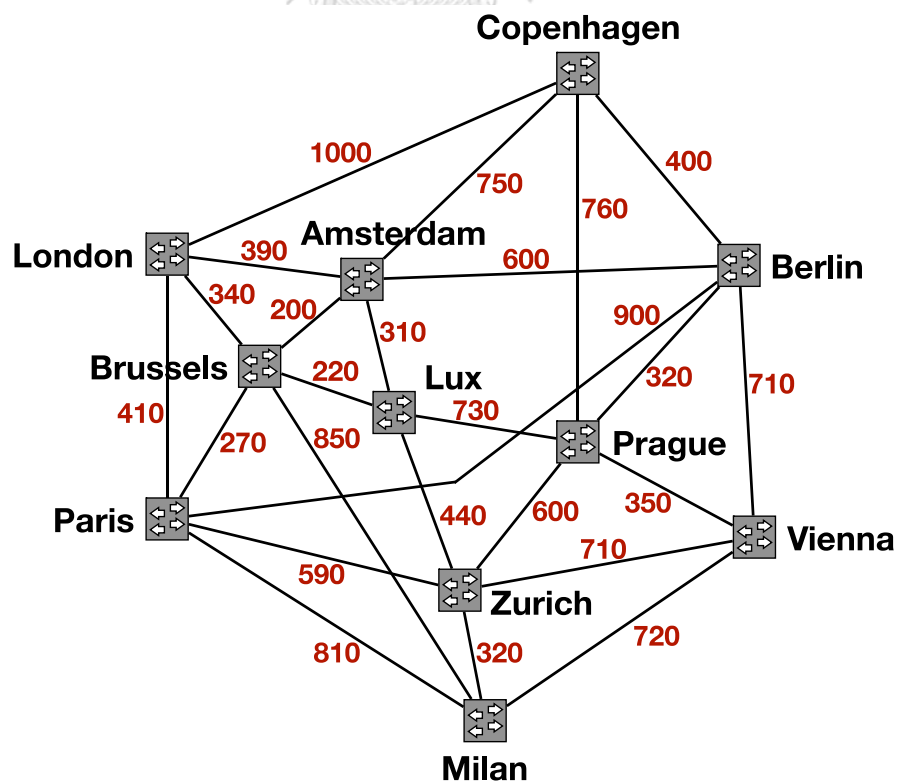


Figure 26 COST239 topology.

In all scenarios, each MPTCP connection randomly started to transfer data with a given data size. The TCPDUMP [50] was used to collect experimental results for each connection. For the multipath topology (Figure 25), the senders (S_i) at the left side were transferring data to the receivers (R_i) at the right side of the topology. For the COST239 topology (Figure 26), the senders and the corresponding receivers were randomly selected from all possible pairs with two-hop distance. In our simulation, we varied the number of MPTCP connections to represent different levels of network congestion and varied the size of data to represent different levels of traffic load per connection. Unless stated otherwise, our parameter settings are shown in Table 8.

Table 8 Experimental parameter settings.

Multipath Topology	
Link capacity	10 Mbps.
Link delay	0 ms.
COST239 Topology	
Link capacity	10 Mbps.
Link delay	Estimated by link distance [49] in ms.
Controller Parameters	
Maximum number of subflows per connection	2
Interval of flow-stats polling	1 s.
Interval of port-stats polling	1 s.
Threshold of link pruning	80% of link capacity
Simulation Parameters	
Traffic injection interval in multipath topology	Randomly set in the range of 1 to 80 s.
Traffic injection interval in COST239 topology	Randomly set in the range of 1 to 40 s.
Number of connections	1 to 20 connections

File size	50MB, 100MB, 200MB, 500MB, and 1GB
Number of runs	5
Routing parameter for the k-max disjoint routing	
Number of candidate paths	3

4.1.3 Benchmark and Metric

4.1.3.1 Benchmark

For performance comparison, we implemented the following routing algorithms. Each of them was implemented as a new control application running on the well-known SDN controller (ONOS).

1) *Traditional Routing*

This represents the shortest-path IP-based routing which is being used in today's Internet such as OSPF.

2) *Traditional Routing with Disjoint Paths*

This represents the multipath OpenFlow controllers in [22, 23]. Each subflow of the same connection was assigned to use a disjoint path.

3) *k-Max Disjoint Routing*

This represents the multipath OpenFlow controllers in [24-27]. The routing decision focused on selecting the first k disjoint paths that provide the highest bottleneck bandwidth from a set of candidate disjoint paths.

4) *OpenFlow-Stats Routing*

This represents our proposed routing algorithm. The OpenFlow-port stats were utilized to evaluate the congestion level of all switches' ports. The routing decision focused on selecting disjoint paths with the lowest congestion level.

4.1.3.2 Metric

1) Average Throughput per Connection

Average throughput per connection (\overline{TP}) is measured as an average of all connections' throughput using Equation (1), where n is a number of MPTCP connections, k is a number of subflows, and $TP_{subflow_{i,j}}$ is a throughput of subflow.

$$\overline{TP} = \frac{1}{n} \sum_{i=1}^n \sum_{j=1}^k TP_{subflow_{i,j}} \quad (1)$$

2) CDF of Completion Time

CDF of completion time is measured as a distribution of completion time of data transfer as in Equation (2), where x is a completion time. x_n is the largest possible value of X that is less than or equal to x .

$$F(x) = P(X \leq x) = \sum_{k=1}^n P(X = x_k) \quad (2)$$

3) Overhead in the Control Plane

Overhead in the control plane is measured as a total cost of message polling for collecting the OpenFlow stats in bytes using Equation (3), where T_{port} is a period of port-stats polling, T_{flow} is a period of flow-stats polling, i is a packet type including request and reply, n is a number of switches, t is a simulation time, $w_{i,k}$ is a packet size of type i at time k , and x is an enabled flag of the polling mechanism for switch j .

$$Overhead = \sum_{i \in \{req, rep\}} \sum_{j=1}^n \sum_{k=1}^t \left(w_{i,k} \left(\frac{x_{port,j}}{T_{port}} + \frac{x_{flow,j}}{T_{flow}} \right) \right) \quad (3)$$

, for $T \in \mathbb{R}, w \in \mathbb{Z}^+, x \in \{0, 1\}$

4.2 Experimental Results

4.2.1 Scenario 1: Vary the Number of MPTCP Connections

In order to evaluate the impact of different levels of big data traffic, we varied the number of MPTCP connections from 1 to 20 connections. Figure 27 and Figure 28 show the results of average throughput per connection in the multipath topology and those in the COST239 topology, respectively.

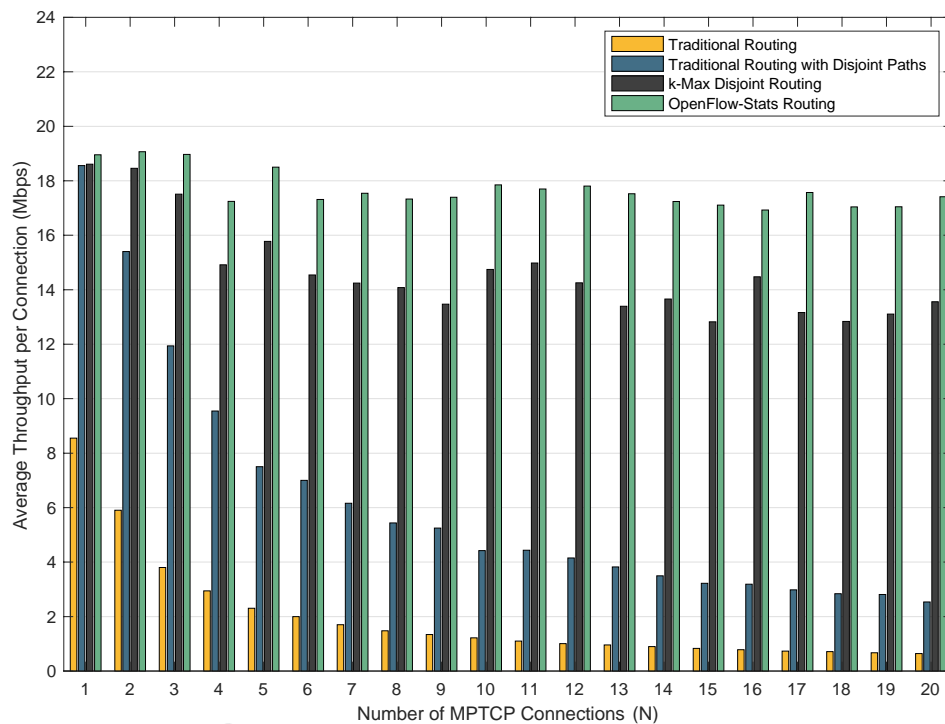


Figure 27 Average throughput per connection in multipath topology (File size = 200MB).

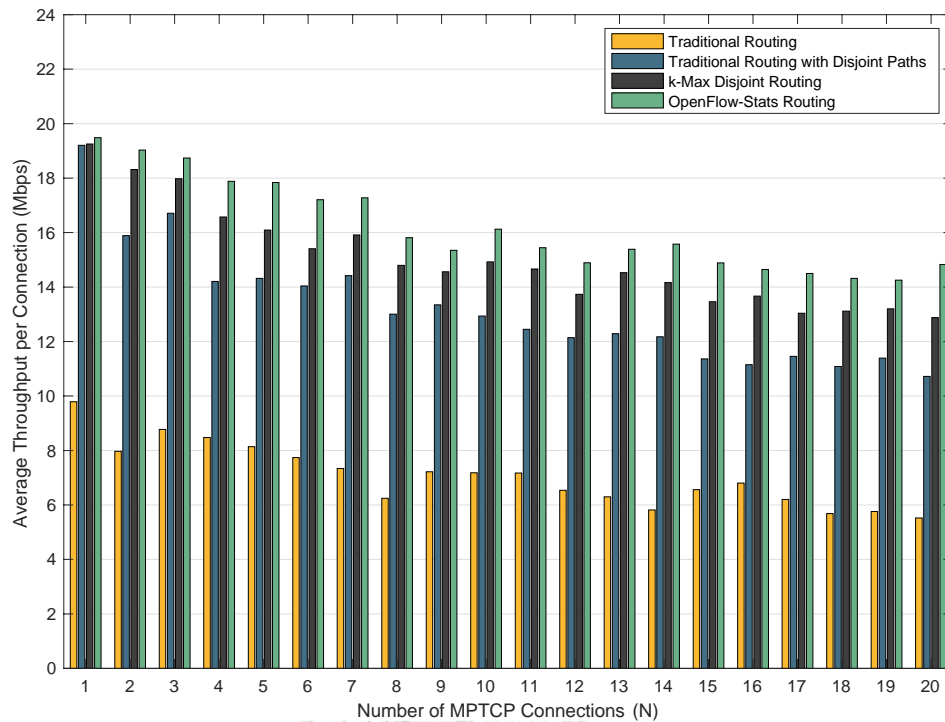


Figure 28 Average throughput per connection in COST239 topology (File size = 200MB).

As can be seen from Figure 27, when the number of MPTCP connections is 1 connection, our proposed OpenFlow-Stats routing scheme achieves relatively the same throughput as other disjoint-path routing schemes do. However, as the number of MPTCP connections increases, the throughput improvement of our proposed scheme increases much higher than other routing schemes do. This is because as the number of MPTCP connections increases, the level of network congestion increases and in such a high-congestion situation our proposed scheme can distribute traffic throughout the whole network. This leads to greater throughput achievement compared to other schemes. Specifically, as described in Section 3.4, our OpenFlow-Stats graph is used to illustrate the congestion level of all links in the network. The links carrying high amounts of background traffic will be pruned from the topology and will not be used for the incoming MPTCP connections. This helps the incoming MPTCP connections to avoid the congested links and then utilize other available paths instead. Therefore, the efficient network-resource utilization can be achieved. The k -max disjoint routing achieves less throughput than our scheme because its performance is limited based on the predefined number of candidate disjoint paths. Increasing this

number will help it perform better but will make the algorithm complexity even higher. The traditional routing schemes perform worse than our proposed scheme and the k -max disjoint routing because they use the shortest-path algorithm and cannot deal with the traffic-distribution issue.

As can be seen in Figure 28, the experimental results reveal the same trend as previously discussed in Figure 27. However, the throughput improvement of our proposed scheme in the COST239 topology is lower than that in the multipath topology. This is because the number of possible disjoint paths for each source-destination pair in the COST239 topology is less than that in the multipath topology.

In order to see the details of throughput variation among all connections in our experiments, Figure 29 (a) and (b) show an example of throughput variation at $N = 15$ connections on the multipath topology and that on the COST239 topology, respectively. The simulation runs that provide the lowest and the highest standard deviation are selected and the margin of error with 95% confidence interval are shown.

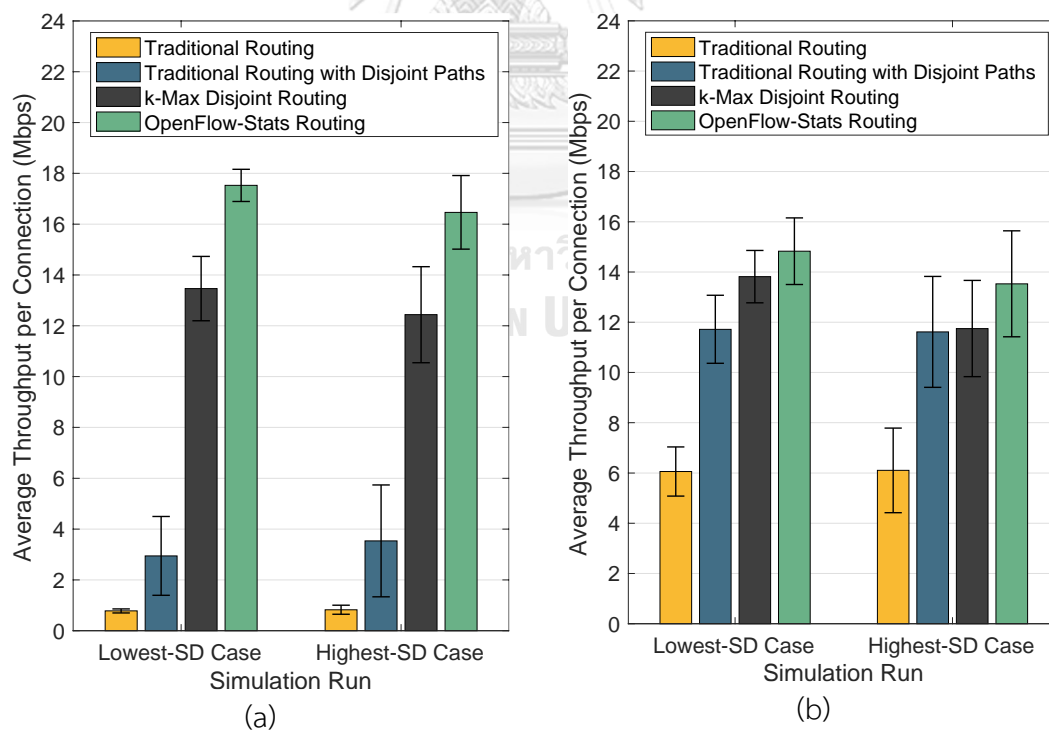


Figure 29 Throughput variation at $N = 15$ connections with 95% confidence interval. (a) multipath topology. (b) COST239 topology.

As can be seen from Figure 29 (a), the traditional routing scheme provides the smallest margin but the lowest throughput. The other routing schemes provide relatively the same margin. However, the results show that our proposed scheme outperforms other routing schemes in terms of throughput significantly.

As can be seen from Figure 29 (b), every routing scheme provides relatively the same margin. This is because the number of possible disjoint paths for each source-destination pair in the COST239 topology is less than that in the multipath topology.

Table 9 and Table 10 show average flow completion time in the multipath topology and that in the COST239 topology, respectively.



Table 9 Completion time (in seconds) and percentage of improvement compared with traditional routing in multipath topology.

Routing Algorithm	Number of Connections (N)																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1)	207 (53%)	303 (61%)	474 (65%)	622 (64%)	792 (63%)	933 (64%)	1093 (65%)	1252 (64%)	1401 (65%)	1560 (64%)	1708 (66%)	1867 (66%)	2002 (65%)	2138 (65%)	2304 (65%)	2471 (66%)	2623 (66%)	2764 (66%)	2931 (66%)	3086 (66%)
2)	97 (53%)	118 (61%)	165 (65%)	223 (64%)	290 (63%)	331 (64%)	381 (65%)	443 (64%)	477 (65%)	550 (64%)	580 (66%)	625 (66%)	684 (65%)	748 (65%)	805 (65%)	836 (66%)	887 (66%)	935 (66%)	977 (66%)	1048 (66%)
3)	95 (54%)	97 (67%)	104 (78%)	125 (79%)	118 (85%)	129 (86%)	131 (88%)	133 (89%)	141 (89%)	127 (91%)	125 (92%)	133 (92%)	141 (92%)	140 (93%)	150 (93%)	131 (94%)	148 (94%)	154 (94%)	149 (94%)	144 (95%)
4)	93 (55%)	93 (69%)	94 (80%)	105 (83%)	97 (87%)	104 (88%)	103 (90%)	105 (91%)	105 (92%)	101 (93%)	104 (93%)	102 (94%)	104 (94%)	107 (94%)	107 (95%)	110 (95%)	104 (96%)	107 (96%)	108 (96%)	104 (96%)

Table 10 Completion time (in seconds) and percentage of improvement compared with traditional routing in COST239 topology.

Routing Algorithm	Number of Connections (N)																			
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1)	179 (48%)	246 (53%)	213 (47%)	224 (39%)	233 (42%)	246 (43%)	266 (50%)	323 (52%)	293 (49%)	285 (46%)	293 (46%)	328 (50%)	353 (55%)	397 (58%)	336 (47%)	317 (41%)	334 (44%)	385 (49%)	365 (50%)	412 (51%)
2)	92 (48%)	115 (53%)	111 (47%)	135 (39%)	133 (42%)	138 (43%)	133 (50%)	153 (52%)	149 (49%)	152 (46%)	158 (46%)	163 (50%)	158 (55%)	164 (58%)	176 (47%)	186 (41%)	187 (44%)	195 (49%)	180 (50%)	198 (51%)
3)	92 (48%)	97 (60%)	100 (53%)	110 (50%)	115 (50%)	119 (51%)	116 (56%)	125 (61%)	131 (55%)	125 (56%)	128 (56%)	141 (57%)	130 (63%)	133 (66%)	141 (58%)	139 (56%)	148 (55%)	150 (61%)	144 (60%)	147 (64%)
4)	91 (49%)	94 (61%)	95 (55%)	102 (54%)	102 (56%)	109 (55%)	108 (59%)	120 (62%)	126 (56%)	116 (59%)	122 (58%)	128 (60%)	124 (64%)	121 (69%)	129 (61%)	132 (58%)	132 (60%)	137 (64%)	135 (63%)	131 (68%)



As can be seen in Table 9, the average time spent by MPTCP connections to finish the data transfer using our proposed OpenFlow-Stats routing is shorter than that using the k -max disjoint routing, the traditional routing with disjoint paths and the traditional routing. Specifically, our proposed scheme provides up to 96% of improvement compared with the traditional routing scheme. The k -max disjoint routing provides up to 95% of improvement compared with the traditional routing scheme. The traditional routing with disjoint paths provides up to 66% of improvement compared with the traditional routing scheme. Although our proposed scheme and the k -max disjoint routing provide a similar value of % improvement, our proposed scheme still outperforms the k -max routing in terms of the algorithm complexity as shown in Table 6 and the overhead at the control plane as will be shown and discussed in this section (Scenario 3).

As can be seen in Table 10, the experimental results reveal the same trend as previously discussed in Table 9. The average time spent by MPTCP connections to finish the data transfer using our proposed OpenFlow-Stats routing is shorter than that using the k -max disjoint routing, the traditional routing with disjoint paths and the traditional routing. Specifically, our proposed scheme provides up to 68% of improvement compared with the traditional routing scheme. The k -max disjoint routing provides up to 64% of improvement compared with the traditional routing scheme. The traditional routing with disjoint paths provides up to 51% of improvement compared with the traditional routing scheme. It can be seen that the average time spent in the COST239 topology is lower than that in the multipath topology. This is because the traffic distribution in the COST239 topology is better than that in the multipath topology.

In order to see the detail of completion time for all MPTCP connections, Figure 30 and Figure 31 are plotted to show Cumulative Distribution Function (CDF) of flow completion time of all connections in the multipath topology and that in the COST239 topology, respectively.

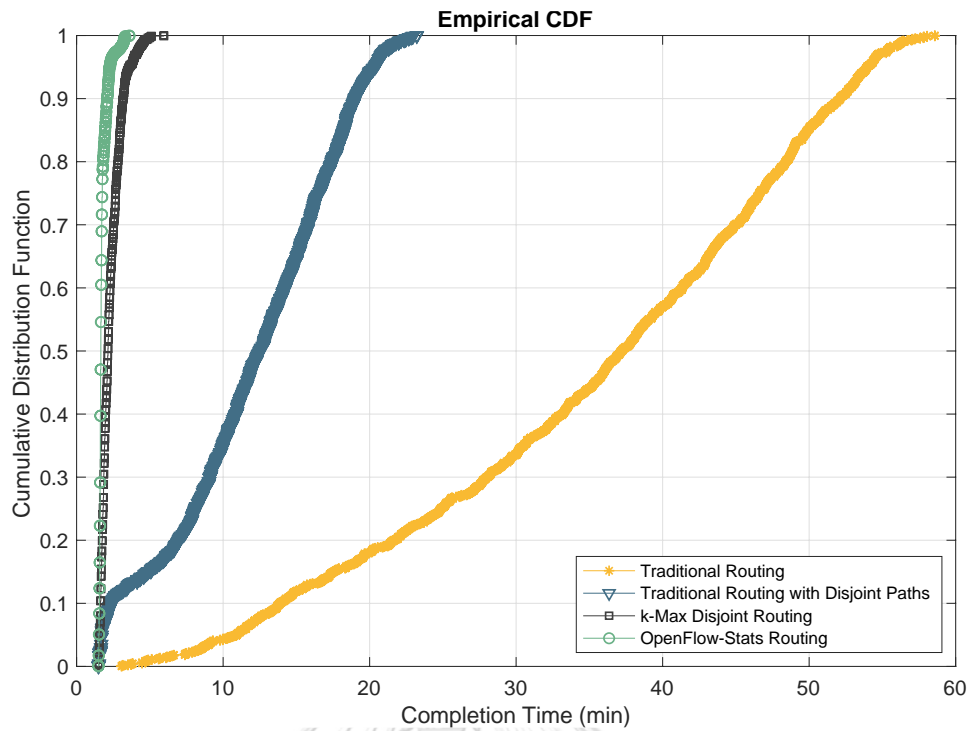


Figure 30 CDF of completion time in multipath topology (File size = 200MB).

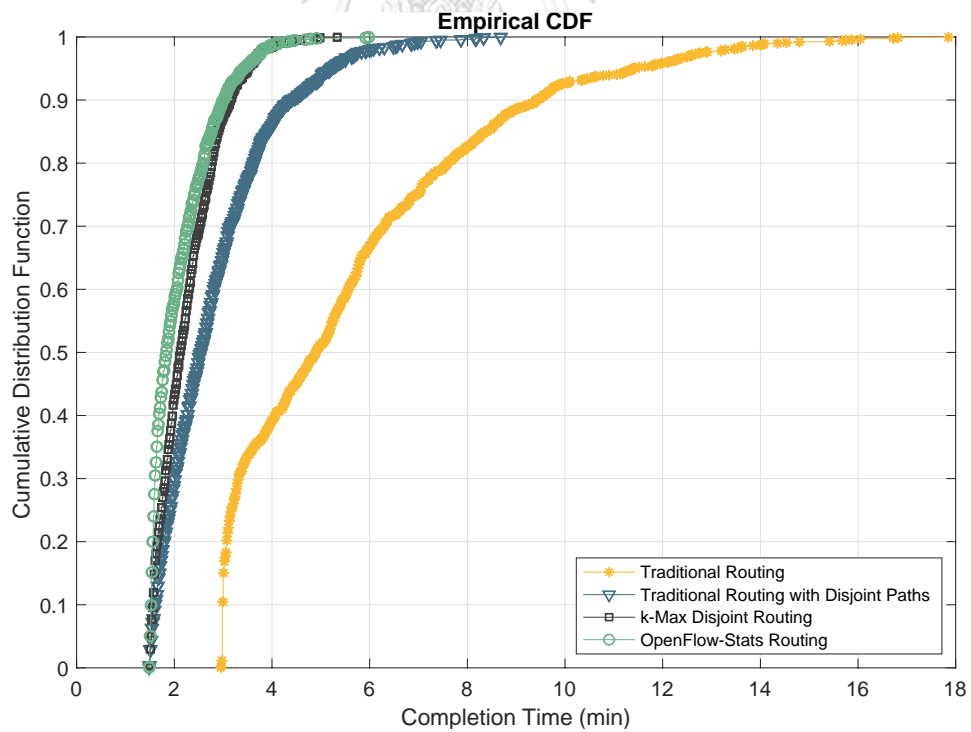


Figure 31 CDF of completion time in COST239 topology (File size = 200MB).

4.2.2 Scenario 2: Vary the Size of Data

In order to evaluate the impact of different levels of big data traffic, we varied the data size that each MPTCP connection needs to transfer. The number of MPTCP connections is fixed at 10 connections. Figure 32 and Figure 33 show the results of average throughput per connection in the multipath topology and those in the COST239 topology, respectively.

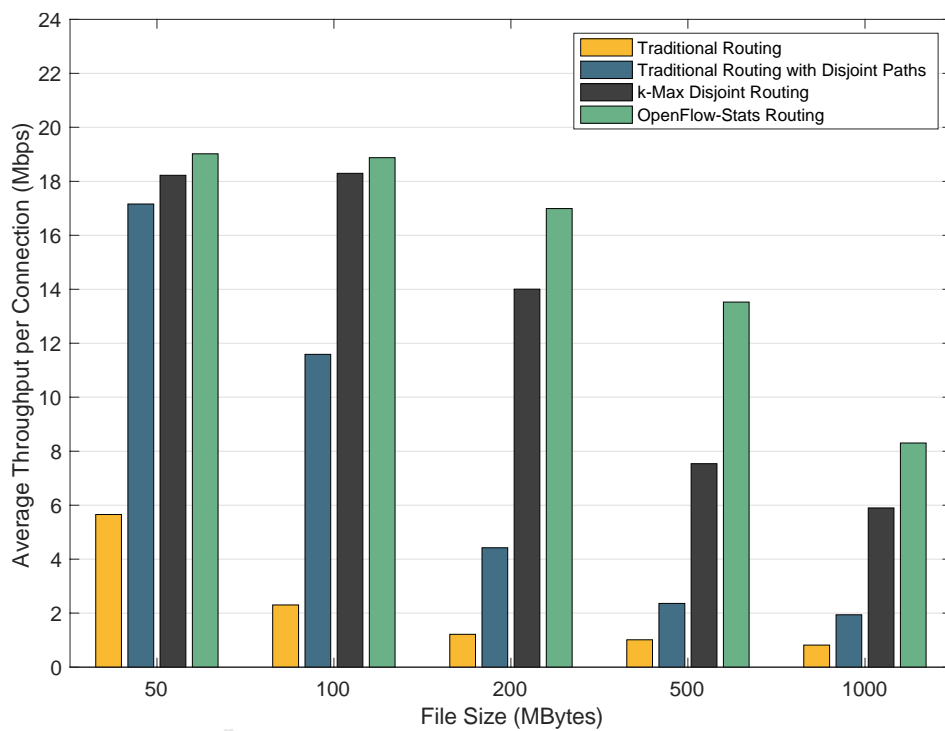


Figure 32 Average throughput per connection in multipath topology ($N = 10$ connections).

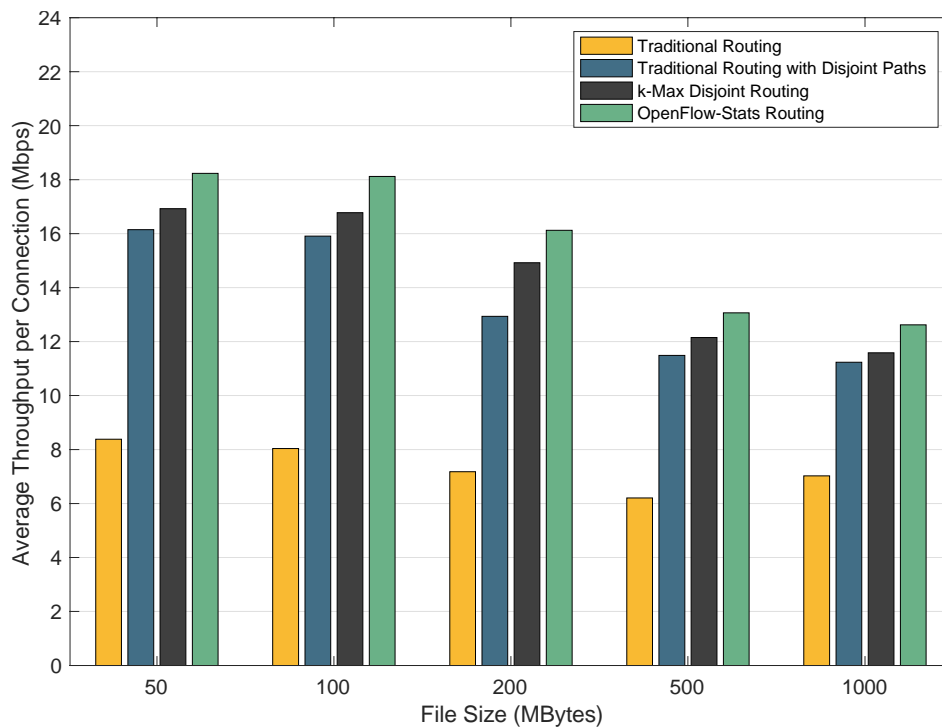


Figure 33 Average throughput per connection in COST239 topology ($N = 10$ connections).

According to Figure 32, as the data size increases, the average throughput of all routing schemes decreases. This is because the congestion level in the network increases. Nevertheless, our proposed OpenFlow-Stats routing scheme still achieves higher average throughput than other disjoint-path routing schemes do. This is because as the data size increases, the level of network congestion increases and in such a high-congestion situation our proposed scheme can distribute traffic throughout the whole network. This leads to greater throughput achievement compared to other schemes. Specifically, as described in Section 3.4, our OpenFlow-Stats graph is used to depict the congestion level of all links in the network. The links carrying high amounts of background traffic will be pruned from the topology and will not be used for the incoming MPTCP connections. This helps the traffic be distributed throughout the whole network. Therefore, the efficient network-resource utilization can be achieved. The k -max disjoint routing achieves less throughput than our scheme because its performance is limited based on the predefined number of candidate disjoint paths.

Increasing this number will help it perform better but will make the algorithm complexity even higher. The traditional routing schemes perform worse than our proposed scheme and the k -max disjoint routing because they use the shortest-path algorithm and cannot deal with the traffic-distribution issue.

According to Figure 33, the experimental results reveal the same trend as previously discussed in Figure 32. However, the average throughput of all routing schemes in the COST239 topology is higher than that in the multipath topology. This is because the number of MPTCP connections is fixed at 10 connections and in the COST239 topology the number of possible source-destination pairs is much greater than that in the multipath topology.

In order to see the details of throughput variation in this scenario, Figure 34 (a) and (b) show an example of throughput variation at File Size = 500 MB on the multipath topology and that on the COST239 topology, respectively.

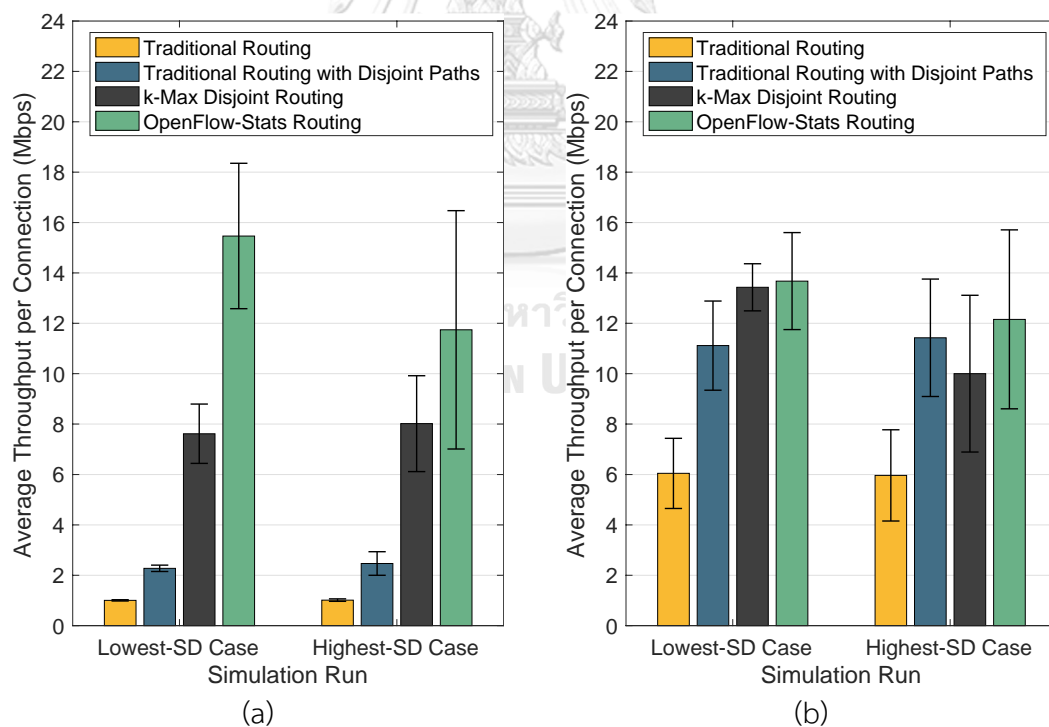


Figure 34 Throughput variation at file size = 500MB with 95% confidence interval.

(a) multipath topology. (b) COST239 topology.

As can be seen from Figure 34 (a), the traditional routing, the traditional routing with disjoint paths, and the k -max disjoint routing provide a smaller margin than our proposed scheme. However, our proposed scheme outperforms other routing schemes in terms of throughput significantly. Our proposed scheme provides the biggest margin because it can distribute all big-data traffic to the whole network by selecting low-congested links. This makes each connection utilize different paths, resulting in different throughput.

As can be seen from Figure 34 (b), every routing scheme provides relatively the same margin. This is because the number of possible disjoint paths for each source-destination pair in the COST239 topology is less than that in the multipath topology.

In order to see the detail of completion time for all MPTCP connections, Figure 35 and Figure 36 are plotted to show Cumulative Distribution Function (CDF) of flow completion time of all connections in the multipath topology and that in the COST239 topology, respectively.

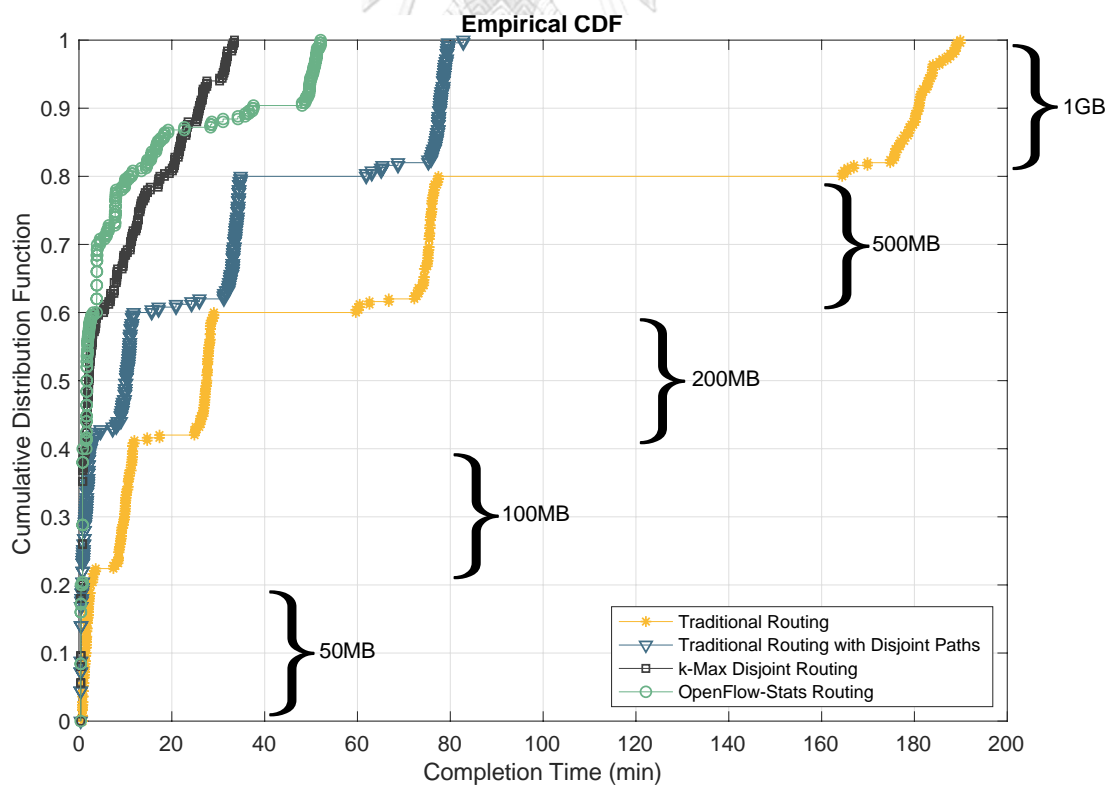


Figure 35 CDF of completion time in multipath topology (N = 10 connections).

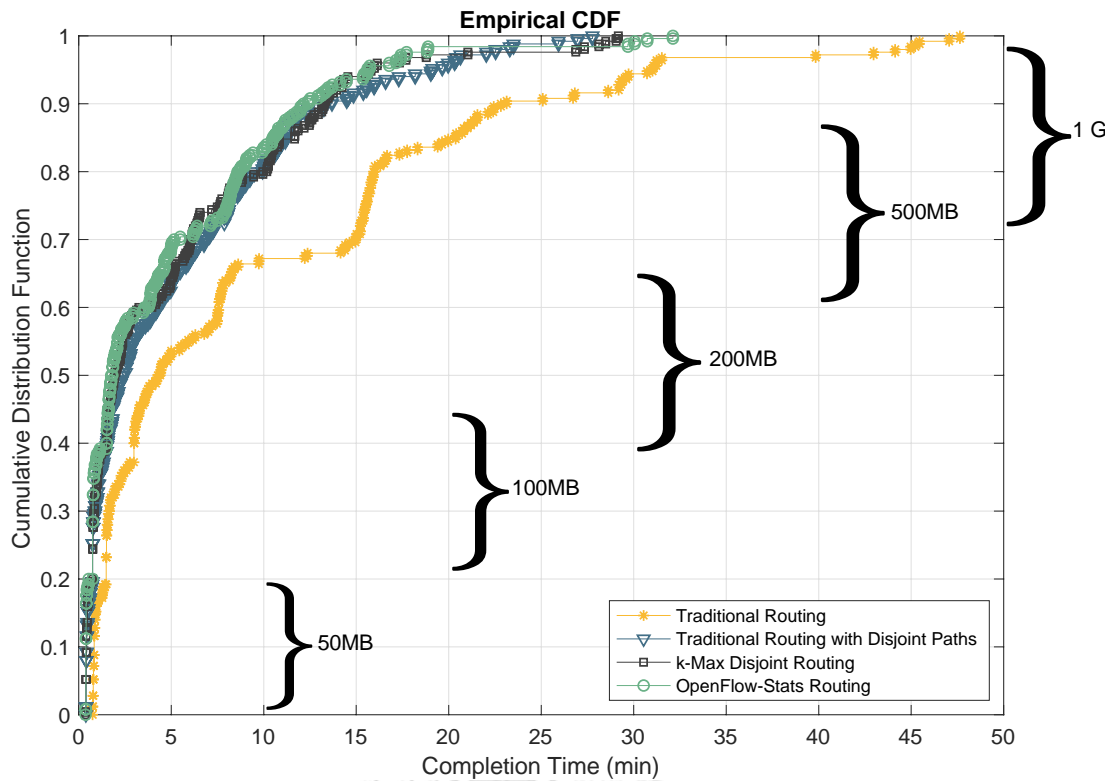


Figure 36 CDF of completion time in COST239 topology ($N = 10$ connections).

As can be seen in Figure 35, when the data size is 50 MB, our OpenFlow-Stats routing scheme provides relatively the same completion time with other schemes. However, when the data size increases, both the traditional routing scheme and the traditional routing with disjoint paths suffer from the heavy traffic load much more than our proposed scheme and the k -max disjoint routing scheme. This is because our proposed scheme considers the port-stats parameter and the k -max disjoint scheme considers the remaining bandwidth, making the heavy traffic load be distributed throughout the whole network. Nevertheless, at the data size of 1 GB, some connections using our proposed scheme start suffering. This shows the limitation of our proposed scheme when being used in the condition of extremely heavy traffic load.

As can be seen in Figure 36, the time spent in the COST239 topology is lower than that in the multipath topology. This is because the traffic in the COST239 topology is more distributed than that in the multipath topology. As a result, the possibility of traffic sharing a path in the COST239 is lower.

4.2.3 Scenario 3: Overhead in the Control Plane

In order to evaluate the overhead in the control plane generated by each routing scheme, we used TCPDUMP to observe the amount of the OpenFlow messages between the controller and switches in the Mininet emulator. Figure 37 to Figure 42 show the amount of the OpenFlow messages in the multipath topology and that in the COST239 topology, respectively.

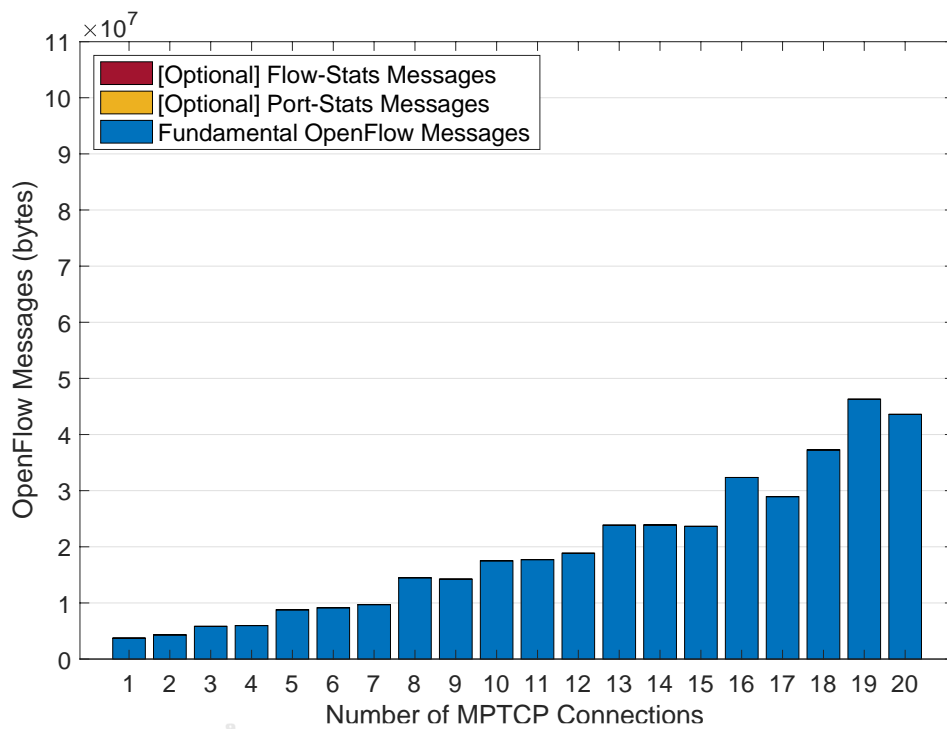


Figure 37 Overhead in the control plane (multipath topology) for the traditional routing with disjoint paths.

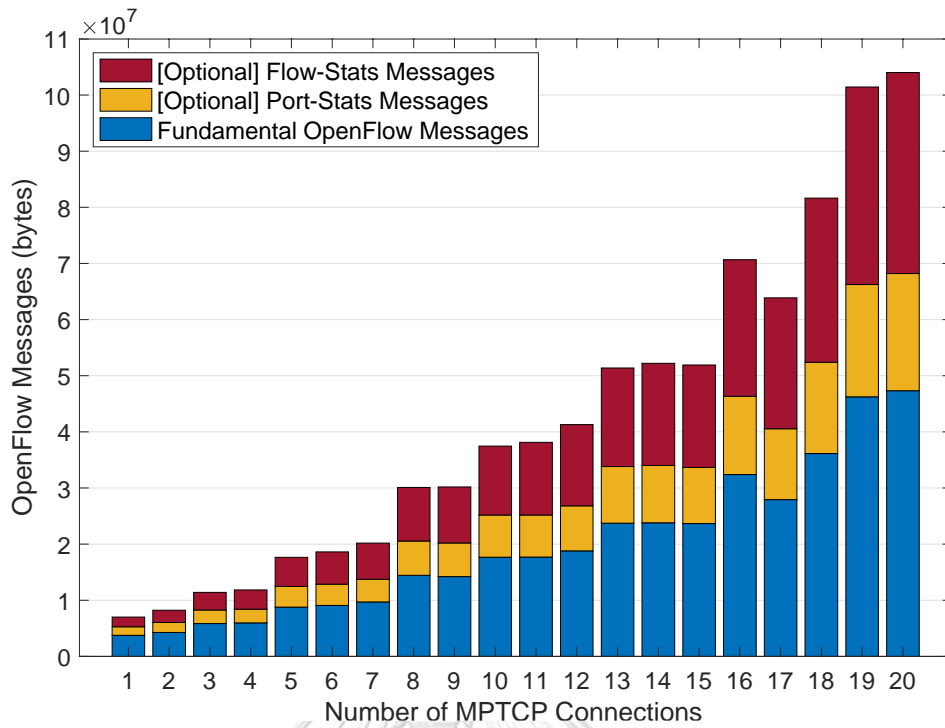


Figure 38 Overhead in the control plane (multipath topology) for k -Max disjoint routing.

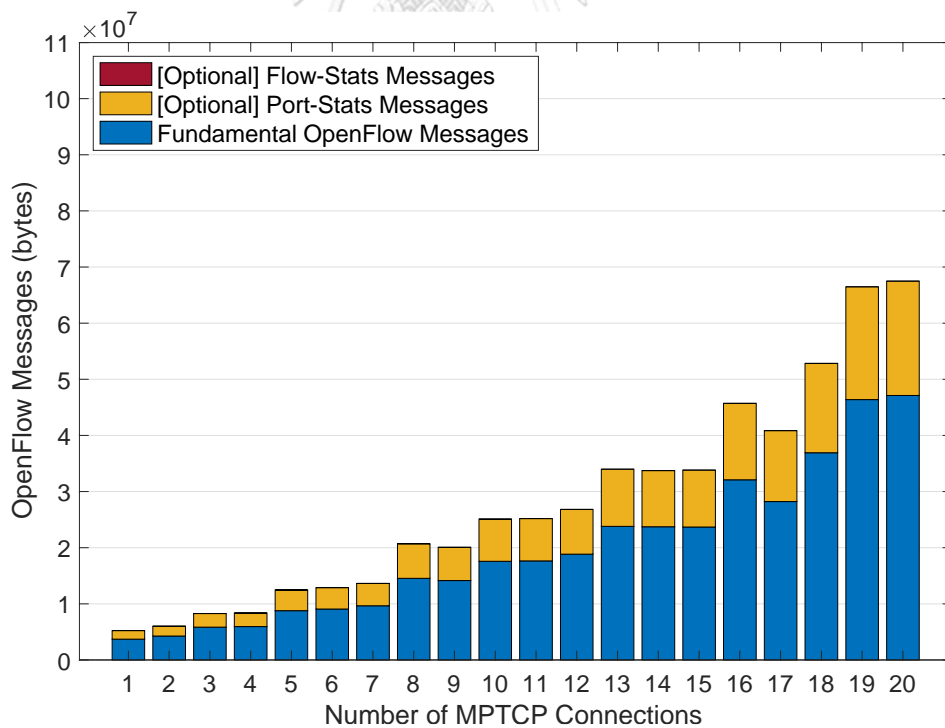


Figure 39 Overhead in the control plane (multipath topology) for our OpenFlow-Stats routing.

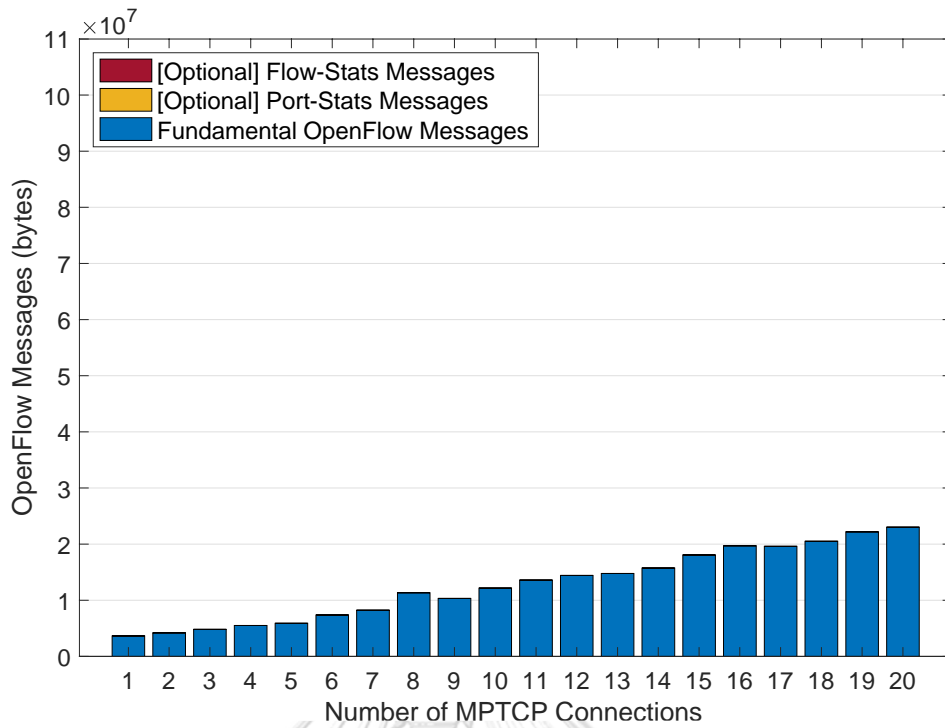


Figure 40 Overhead in the control plane (COST239 topology) for the traditional routing with disjoint paths.

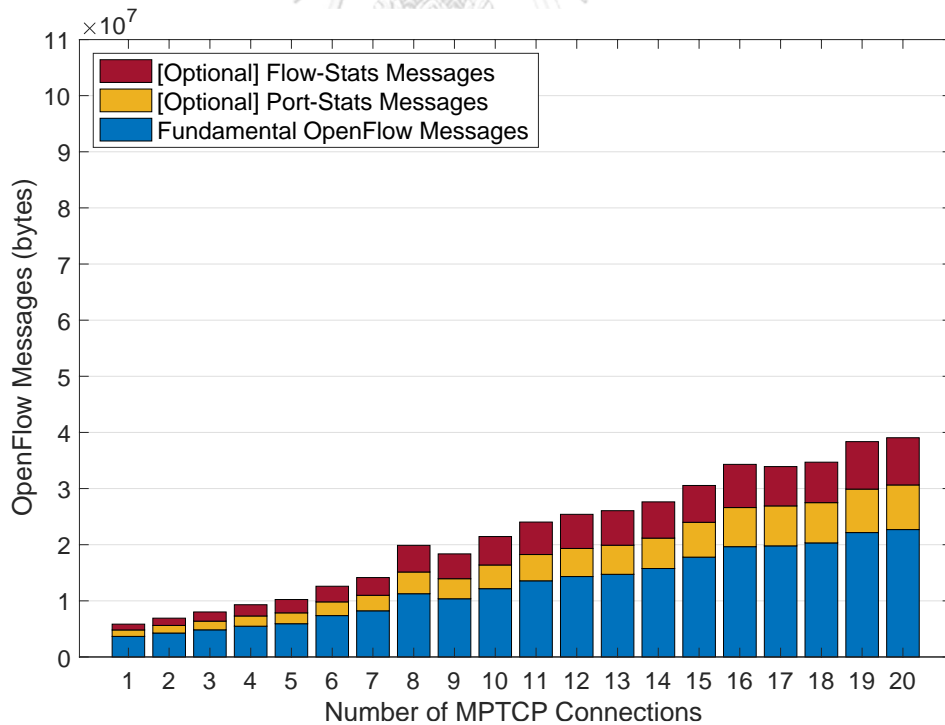


Figure 41 Overhead in the control plane (COST239 topology) for k-Max disjoint routing.

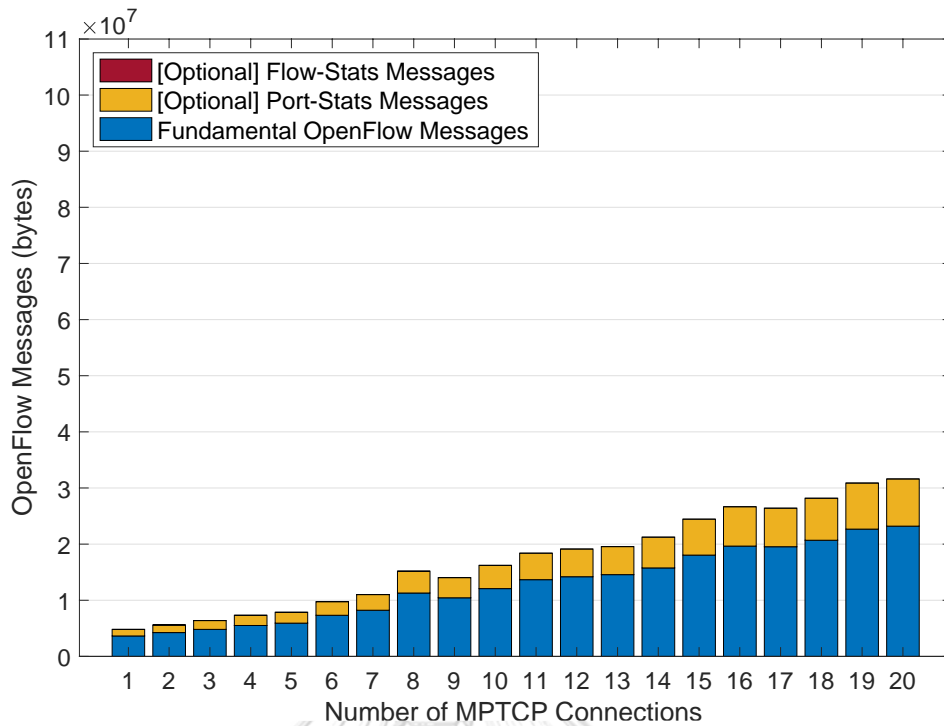


Figure 42 Overhead in the control plane (COST239 topology) for our OpenFlow-Stats routing.

Basically, the OpenFlow messages are classified into two categories. The first one is fundamental messages which are necessary for the basic communication between the controller and switches. These messages include hello messages, feature messages, config messages, packet-in messages, packet-out messages, flow mod messages, etc. The second one is optional messages which are necessary for the controller to acquire specific statistical information from switches. These messages include port stats, flow stats, table stats, meter stats, etc.

As can be seen in Figures 37 to Figure 42, each routing scheme generates a different amount of control-plane overhead. Specifically, as shown in Figures 37 and 40, the traditional routing with disjoint paths needs only the fundamental messages for operation. As shown in Figures 38 and 41, the k -max disjoint routing requires not only the fundamental messages, but also port-stats and flow-stats messages for estimating the remaining bandwidth of available paths. As shown in Figures 39 and 42, our OpenFlow-Stats routing needs the fundamental messages and port-stats messages for evaluating level of port utilization.

For comparison in the multipath topology, Figure 37, Figure 38, and Figure 39 reveal the results that our OpenFlow-Stats routing generates 42% overhead more than the traditional routing with disjoint paths. However, our OpenFlow-Stats routing generates 57% overhead less than the k -max disjoint routing. The k -max disjoint routing generates a lot more overhead because it needs flow-stats messages, and the amount of the messages increases proportionally with the number of flow rules.

For comparison in the COST239 topology, Figure 40, Figure 41, and Figure 42 reveal the similar results as discussed in the previous paragraph. Briefly, our OpenFlow-Stats routing generates 34% overhead more than the traditional routing with disjoint paths. But our OpenFlow-Stats routing generates 32% overhead less than the k -max disjoint routing.

4.3 Comparison of Multipath Transport Protocols in Our SDN Environment

In order to compare the performances of multipath transport protocols in our SDN environment, we implemented the Multipath QUIC (MPQUIC) [33], an extension to the QUIC protocol [35], and compared its performances with MPTCP. For the experimental setup, we used the same settings as described in Scenario 1 (Section 4.2.1). Figure 43 shows the results of average throughput per connection of MPQUIC (Dash-dotted line) and MPTCP (Solid line) for each type of routing scheme.

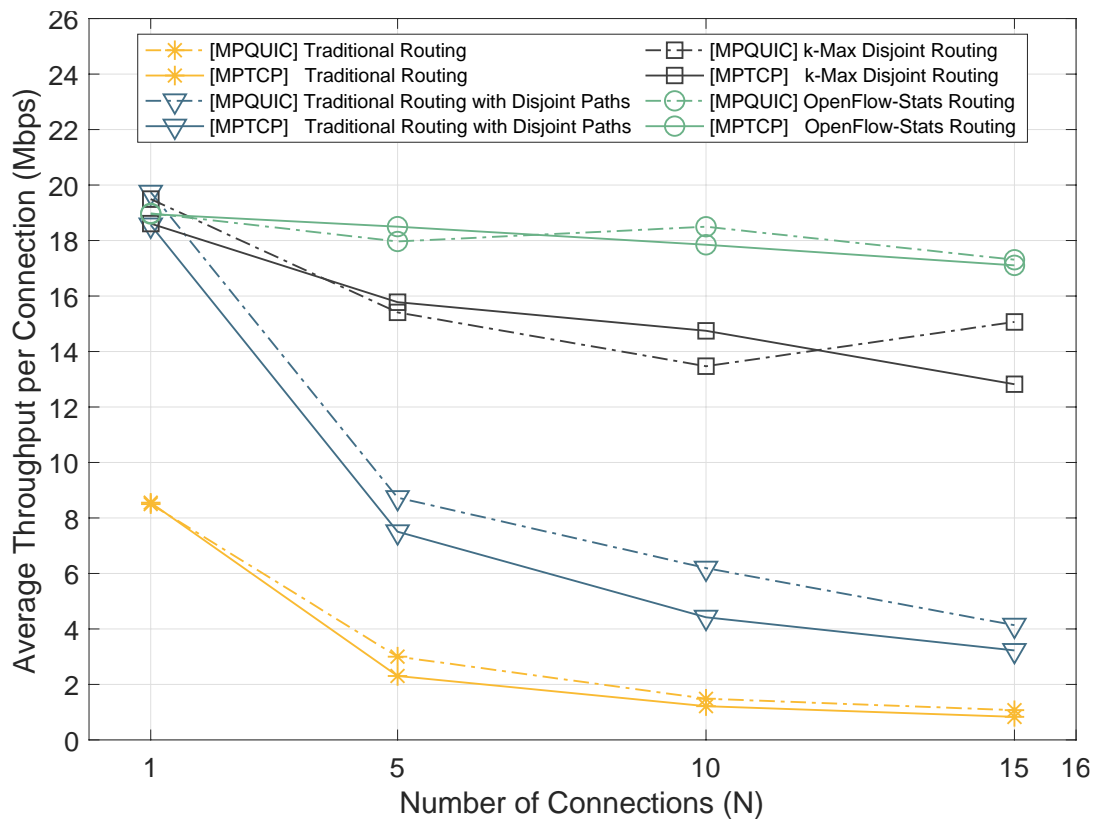


Figure 43 Performance comparison between MPQUIC (Dash-dotted line) and MPTCP (Solid line).

As can be seen in Figure 43, when the number of connections is 1, the MPQUIC protocol achieves relatively the same throughput as MPTCP protocol does for all routing schemes. However, when considering each routing scheme, as the number of connections increases, MPQUIC can achieve slightly-higher throughput than MPTCP. Specifically, for the cases of the traditional routing and the traditional routing with disjoint paths, MPQUIC can outperform MPTCP at every number of connections. This is because MPQUIC relies on UDP protocol and its congestion control is maintained at the application level. When the network is congested, MPQUIC can estimate the path latencies and adjust its congestion windows precisely, whereas MPTCP is suffering from the multiplicative decrease and slow start. For the case of k -max Disjoint routing and our proposed OpenFlow-Stats routing, MPQUIC achieves relatively the same throughput as MPTCP does at every number of connections. This is because the k -max Disjoint routing focuses on selecting paths that provide the highest available bandwidth

and our proposed OpenFlow-Stats routing focuses on selecting paths that provide the low congestion level based on OpenFlow-Stats graph. This helps both MPQUIC and MPTCP run on less-congested paths, thus less suffering from network congestion. In overall, our proposed OpenFlow-Stats routing scheme outperforms other routing schemes.

In conclusion, as can be seen and discussed above, our proposed routing scheme outperforms other routing schemes. Furthermore, it can work efficiently and is compatible with several multipath protocols such as MPQUIC and MPTCP.



CHAPTER 5

Conclusion

5.1 Dissertation Summary

In this dissertation, we proposed an SDN-coordinated steering framework for big data transfer application. The MPTCP protocol is primarily used to transfer the data. Our framework consists of *Multipath Transmission Manager*, *Multipath Topology Manager*, and *OpenFlow-Stats Analyzer*. Each module is deployed as a network control application running on the top of the SDN framework. Our proposed OpenFlow-Stats routing algorithm performs the topology-pruning technique based on the switch-port statistics. Specifically, the congestion level of each link will be evaluated based on the utilization of the associated ports. The topology pruning helps reduce the system complexity.

Using our framework, the performance of the big data analytic system that relies on the distributed data storage system can be significantly improved. The results show that our proposed routing scheme outperforms other previous works in several aspects in both multipath topology and COST239 topology. It can provide the highest throughput improvement with low complexity. It can reduce the completion time of data transfer up to 90% compared with the traditional routing with disjoint paths and up to 35% compared with the k -max disjoint routing. In addition, it can reduce the system overhead at the control plane up to 57% in the multipath topology and up to 32% in the COST239 topology. However, we observe the performance degradation of our proposed routing algorithm in the extremely-high traffic load. This can be improved by adding mechanisms in order to identify the traffic with high load and avoid the shared bottleneck among those traffic flows. This is included in our future work.

In addition to OpenFlow, other solutions of opening up the forwarding plane such as P4 [51] can also benefit from our proposed framework. Specifically, since port statistics and network topology are basic information generally provided by forwarding devices, the technique of using port statistics and choosing multiple paths proposed in this dissertation can be applied.

5.2 Cost Analysis of Using the Proposed Framework

This section, we explain the cost of deploying our proposed framework in the real environment. The details of our cost are explained as follows.

5.2.1 Cost of SDN Deployment

Currently the Internet architecture relies on the network routers. The routing schemes among Autonomous Systems (ASes) relied on static routing. Thus, if one wants to implement this framework over the traditional Internet, basically he/she needs to put OpenFlow switches between the data servers/clients and the gateway routers. Then, the *VXLAN* technology can be applied to each switch in order to create the *VXLAN* tunnels among the data servers/clients. These tunnels are managed and controlled as an overlay network by the proposed controller.

5.2.2 Cost of Multipath Configuration at the Endpoint

In order to use the MPTCP protocol, we need to modify the Kernel of the operating system (OS) of all data servers including data clients. Although MPTCP is currently defined as a standard multipath transport protocol, most operating systems have not supported the protocol yet. There are only the Linux-based OSes [47] that can be manually installed by users. In addition, we need to add more network interfaces and manually configure them for supporting the network multihoming.

5.2.3 Cost of Modifying the Network Component of Computing Software to Support Open vSwitch.

In case of cloud-based systems, we need to modify the network component of the computing software to support the *Open vSwitch* [52] that is mainly used for creating OpenFlow switches.

5.2.4 Cost of Maintaining the OpenFlow Controller

In order to control the Big Data traffic, we need to set up a dedicated server or a cloud server in order to deploy the OpenFlow controller and our proposed modules. This induces additional cost of maintaining the server and the connections between OpenFlow controller and OpenFlow switches.

5.2.5 Cost of Measuring the Maximum Capacity of Links

In order to set up the threshold of evaluating the link quality, we need to collect the information about the maximum capacity of links first. In practice, the maximum capacity depends on the network policy among ASes. Thus, we need to manually measure the link capacity by using a network tool such as *iPerf* and *NetFlow* among data servers.

5.3 Complexity Analysis of Our OpenFlow-Stats Routing

This section, we explain the complexity of our proposed routing. According to the pseudocode as shown in Figure 22, the sub-tasks of our OpenFlow-Stats routing can be divided into two major cases. The first case is a sub-task for processing the first subflow. The second case is a sub-task for processing the next subflow. Figure 44 illustrates the details of sub-tasks that $h_1(n)$, $h_2(n)$ are routing functions for the first subflow and the next subflow respectively.

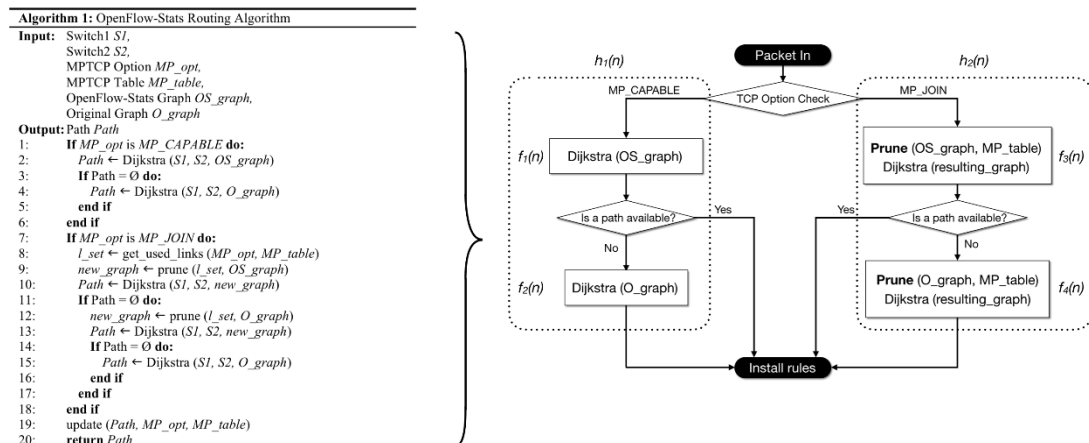


Figure 44 Complexity analysis of OpenFlow-Stats routing algorithm.

According to our framework design in Section 3.4, the complexity of topology pruning is $O(E)$ where E is the number of edges. The complexity of Dijkstra's algorithm is $O(E \log(V))$ [48] where E is the number of edges and V is the number of vertices. Then, the complexity of OpenFlow-Stats graph can be obtained as follows:

$$\text{Complexity of OpenFlow-Stats routing} \in \max(h_1(n), h_2(n))$$

<i>First Subflow ($h_1(n)$)</i>	<i>Next Subflow ($h_2(n)$)</i>
$h_1(n) \in O(f_1(n) + f_2(n))$	$h_2(n) \in O(f_3(n) + f_4(n))$
$h_1(n) \in O(E \log(V) + E \log(V))$	$h_2(n) \in O((O(E) + E \log(V)) + (O(E) + E \log(V)))$
$h_1(n) \in O(2E \log(V))$	$h_2(n) \in O(2O(E) + 2E \log(V))$
$h_1(n) = O(E \log(V))$	$h_2(n) = O(E \log(V))$

Thus,

Complexity of OpenFlow-Stats routing $\in \max(O(E \log(V)), O(E \log(V)))$

Complexity of OpenFlow-Stats routing = $O(E \log(V))$

5.4 Discussion on the Congestion Control Algorithms

5.4.1 Impact on Using the Traditional Congestion Control Algorithm

According to the current speed of Internet links, the traditional congestion controls such as Reno are too slow to fill up the network links due to the Bandwidth-Delay-Product (BDP) problem [53]. This induces high completion time of a Big Data transfer among data storage regions. In order to deal with the problem, most operating systems (OS) have already changed the default congestion control algorithm to CUBIC and CTCP [54]. This configuration also affects MPTCP because the path manager of MPTCP relied on the TCP congestion control. Thus, in order to avoid the BDP problem, we suggest to use the OS default congestion control algorithms in our framework.

5.4.2 Impact on Using the Coupled Congestion Control Algorithm

In order to avoid the unfairness problem between MPTCP and regular TCP, the coupled congestion controls named Linked Increase Algorithm (LIA) [55] was proposed for MPTCP, in RFC 6356. The following three goals capture the desirable properties of a practical multipath congestion control algorithm:

- *Goal 1 (Improve Throughput)* A multipath flow should perform at least as well as a single path flow would on the best of the paths available to it.
- *Goal 2 (Do no harm)* A multipath flow should not take up more capacity from any of the resources shared by its different paths than if it were a single flow

using only one of these paths. This guarantees it will not unduly harm other flows.

- *Goal 3 (Balance congestion)* A multipath flow should move as much traffic as possible off its most congested paths, subject to meeting the first two goals.

In addition to LIA, there are also other coupled congestion control protocols recently proposed in the literature such as Opportunistic Linked Increase Algorithm (OLIA) [56], Delay-based Congestion Control for MPTCP (wVegas) [57] and Balanced Linked Adaptation Congestion Control Algorithm (BALIA) [58]. However, currently there is only LIA that is defined as a standard coupled congestion control protocol in RFC.

In this dissertation, we also evaluated the performance between CUBIC and LIA in our framework. Figure 45 (a) and (b) show the results of using the regular congestion control named CUBIC and the coupled congestion control named LIA. As can be seen in Figure 45 (a) and (b), the experimental results of using LIA reveal the same trend as the results of using CUBIC. In addition, the performance of each routing scheme reveals the same trend as previously discussed in Section 4.2.1. Because, in this dissertation, we focused on providing a set of disjoint paths for serving a multipath transport protocol such as MPTCP. The disjoint consideration can benefit both regular congestion control and coupled congestion control. Thus, the coupled congestion control can work together with our framework.

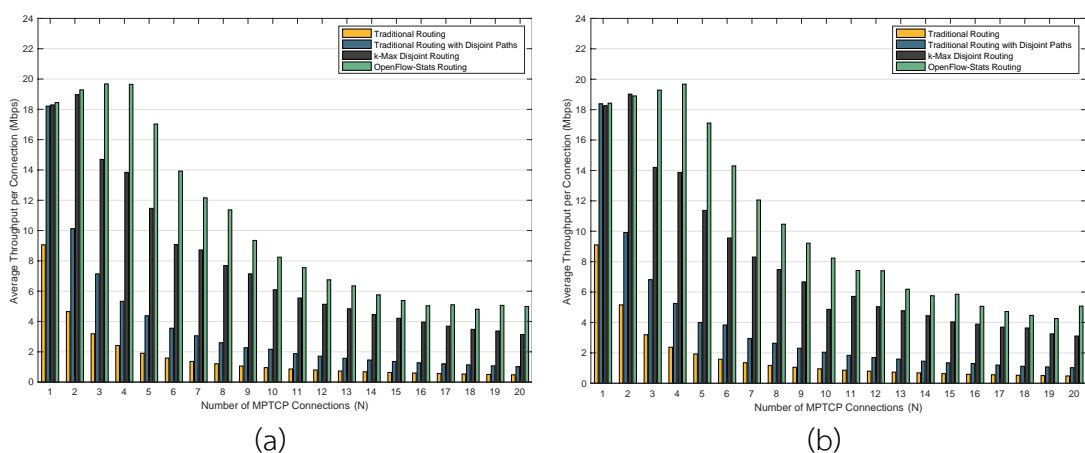


Figure 45 Experimental Results of using (a) CUBIC protocol and (b) LIA protocol.

5.5 Discussion on the Real-World Scenario Using OF@TEIN+ Testbed

5.5.1 Experimental Setup

In order to evaluate the proposed framework over the real environment, we setup a multipath topology as an overlay network over the Open/Federated Playground for Future Networks (OF@TEIN+) [59], an SDN-Cloud R&D collaboration among TEIN partners. Figure 46 illustrates our multipath topology which consists of four nodes: Gwangju Institute of Science and Technology (GIST) in South Korea, Hanoi University of Science & Technology (HUST) in Vietnam, Chulalongkorn University (CU) in Thailand, National Cheng Kung University (NCKU) in Taiwan. According to the architecture of OF@TEIN+ that relied on the *OpenStack platform* [60] and OpenFlow networking, we used the *Open vSwitch software* [52] for installing OpenFlow switches on the selected sites. In order to create a set of overlay links, the *VXLAN* tunnels were applied on each node. GIST and HUST sites were set as a source and a destination respectively.

As can be seen in Figure 46, our overlay network consists of one shortest path and two alternative paths between GIST site and HUST site. In our overlay network, the shortest path has only one-hop distance. The first alternative path has a two-hop distance (GIST→CU→HUST). The second alternative path has a two-hop distance (GIST→NCKU→HUST). In order to create a Big Data flow, we created a virtual machine that represents a data server running on GIST site and another virtual machine that represents a data client running on HUST site. In this experiment, the data server transferred a batch of data with *500MB* to the data client. For MPTCP implementation, we created two network interfaces and installed the *MPTCP v0.93* [47] on both virtual machines. All MPTCP parameters were set according to the default setting. Unless stated otherwise, our parameter settings are shown in Table 11.

Table 11 Experimental parameter settings.

Controller Parameters	
Maximum number of subflows per connection	2
Interval of port-stats polling	1 s.

Threshold of link pruning	80% of link capacity
Simulation Parameters	
Number of connections	1 connection
File size	500MB
Number of runs	5

In order to control the multipath topology, we create a virtual machine for deploying our SDN controller at the GIST site. For performance comparison, we implemented two routing schemes: the traditional routing and our OpenFlow-Stats routing.



Figure 46 Experimental topology over OF@TEIN+ testbed.

As can be seen in Figure 46, the path characteristics can be varied by the real-world background traffic. Figure 47 illustrates the available bandwidth that was measured about 1,200 seconds. According to heterogeneous traffic and the network policies of the Internet, each path suffers a high variation including the bandwidth shaping. The average bandwidth of the shortest path, the first alternative path, and the second alternative path are about 36, 19, and 14 Mbps respectively. Figure 48

illustrates the results of path latency measurement of the shortest path, the first alternative path, and the second alternative path respectively. The average of path latency of the shortest path, the first alternative path, and the second alternative path are about 67, 321, and 524 milliseconds respectively.

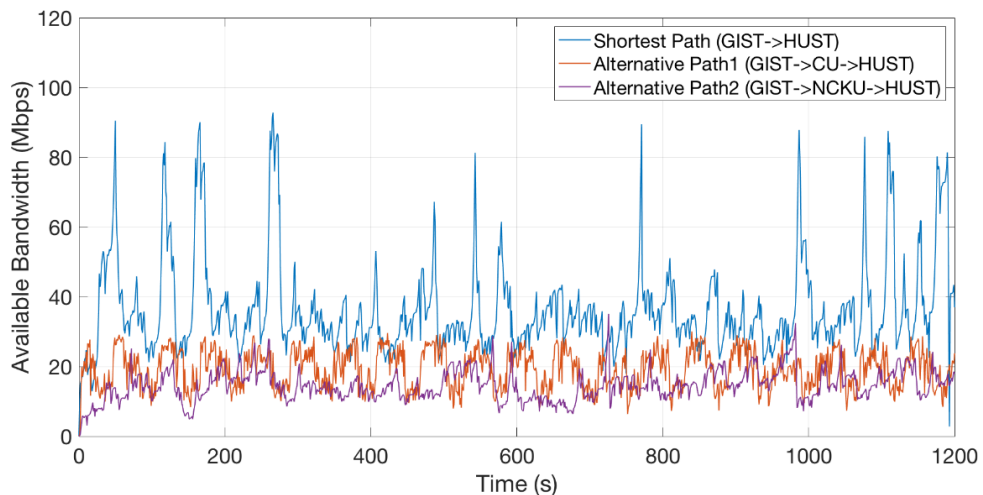


Figure 47 Available bandwidth measurement between GIST-South Korea and HUST-Vietnam.

GIST→HUST

```
ubuntu@gist1-vm1:~$ ping hust1-2 -c 10
PING hust1-2 (192.168.125.125) 56(84) bytes of data.
64 bytes from hust1-2 (192.168.125.125): icmp_seq=1 ttl=64 time=67.1 ms
64 bytes from hust1-2 (192.168.125.125): icmp_seq=2 ttl=64 time=65.0 ms
64 bytes from hust1-2 (192.168.125.125): icmp_seq=3 ttl=64 time=68.5 ms
64 bytes from hust1-2 (192.168.125.125): icmp_seq=4 ttl=64 time=64.8 ms
64 bytes from hust1-2 (192.168.125.125): icmp_seq=5 ttl=64 time=64.9 ms
64 bytes from hust1-2 (192.168.125.125): icmp_seq=6 ttl=64 time=77.7 ms
64 bytes from hust1-2 (192.168.125.125): icmp_seq=7 ttl=64 time=64.9 ms
64 bytes from hust1-2 (192.168.125.125): icmp_seq=8 ttl=64 time=64.9 ms
64 bytes from hust1-2 (192.168.125.125): icmp_seq=9 ttl=64 time=69.2 ms
64 bytes from hust1-2 (192.168.125.125): icmp_seq=10 ttl=64 time=65.0 ms

--- hust1-2 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9013ms
rtt min/avg/max/mdev = 64.869/67.256/77.760/3.831 ms
```

GIST→CU→HUST

```
ubuntu@gist1-vm1:~$ ping hust1-1 -c 10
PING hust1-1 (192.168.115.129) 56(84) bytes of data.
64 bytes from hust1-1 (192.168.115.129): icmp_seq=1 ttl=64 time=343 ms
64 bytes from hust1-1 (192.168.115.129): icmp_seq=2 ttl=64 time=317 ms
64 bytes from hust1-1 (192.168.115.129): icmp_seq=3 ttl=64 time=316 ms
64 bytes from hust1-1 (192.168.115.129): icmp_seq=4 ttl=64 time=317 ms
64 bytes from hust1-1 (192.168.115.129): icmp_seq=5 ttl=64 time=318 ms
64 bytes from hust1-1 (192.168.115.129): icmp_seq=6 ttl=64 time=320 ms
64 bytes from hust1-1 (192.168.115.129): icmp_seq=7 ttl=64 time=317 ms
64 bytes from hust1-1 (192.168.115.129): icmp_seq=8 ttl=64 time=317 ms
64 bytes from hust1-1 (192.168.115.129): icmp_seq=9 ttl=64 time=328 ms
64 bytes from hust1-1 (192.168.115.129): icmp_seq=10 ttl=64 time=319 ms

--- hust1-1 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 9011ms
rtt min/avg/max/mdev = 316.827/321.570/343.026/7.952 ms
```


GIST→NCKU→HUST

```

ubuntu@gist1-vm1:~$ ping hust1-4 -c 10
PING hust1-4 (192.168.145.122) 56(84) bytes of data.
64 bytes from hust1-4 (192.168.145.122): icmp_seq=1 ttl=64 time=525 ms
64 bytes from hust1-4 (192.168.145.122): icmp_seq=2 ttl=64 time=522 ms
64 bytes from hust1-4 (192.168.145.122): icmp_seq=3 ttl=64 time=522 ms
64 bytes from hust1-4 (192.168.145.122): icmp_seq=4 ttl=64 time=522 ms
64 bytes from hust1-4 (192.168.145.122): icmp_seq=5 ttl=64 time=522 ms
64 bytes from hust1-4 (192.168.145.122): icmp_seq=6 ttl=64 time=532 ms
64 bytes from hust1-4 (192.168.145.122): icmp_seq=7 ttl=64 time=522 ms
64 bytes from hust1-4 (192.168.145.122): icmp_seq=8 ttl=64 time=522 ms
64 bytes from hust1-4 (192.168.145.122): icmp_seq=9 ttl=64 time=530 ms
64 bytes from hust1-4 (192.168.145.122): icmp_seq=10 ttl=64 time=522 ms

--- hust1-4 ping statistics ---
10 packets transmitted, 10 received, 0% packet loss, time 8999ms
rtt min/avg/max/mdev = 522.614/524.813/532.113/3.450 ms

```

Figure 48 Path latency measurement of the shortest path, alternative path 1, and alternative path 2 respectively.

5.5.2 Results of Deploying the Proposed Framework

Figure 49 (a) and (b) show the results of throughput performance of Big Data transfer from OF@TEIN+ testbed. As can be seen in Figure 49 (a), our OpenFlow-Stats routing scheme significantly outperforms the traditional routing scheme. Our routing scheme can reduce the completion time by about 26%.

In order to validate the experimental results over the real environment, we also repeated the experiment on the Mininet emulator. We created the same topology and also configured the path characteristics by using the average values of bandwidth and path latency that were estimated from the previous section. Figure 49 (b) shows the results of the throughput performance of a Big Data transfer from Mininet. As can be seen in Figure 49 (b), the throughput results reveal the same trend as previously discussed in Figure 49 (a). Our routing scheme can reduce the completion time by about 32%.

In conclusion, as can be seen and discussed above, our proposed framework can be deployed on the Internet environment and also provides the improvement of a Big Data transfer with the same trend as done in Mininet.

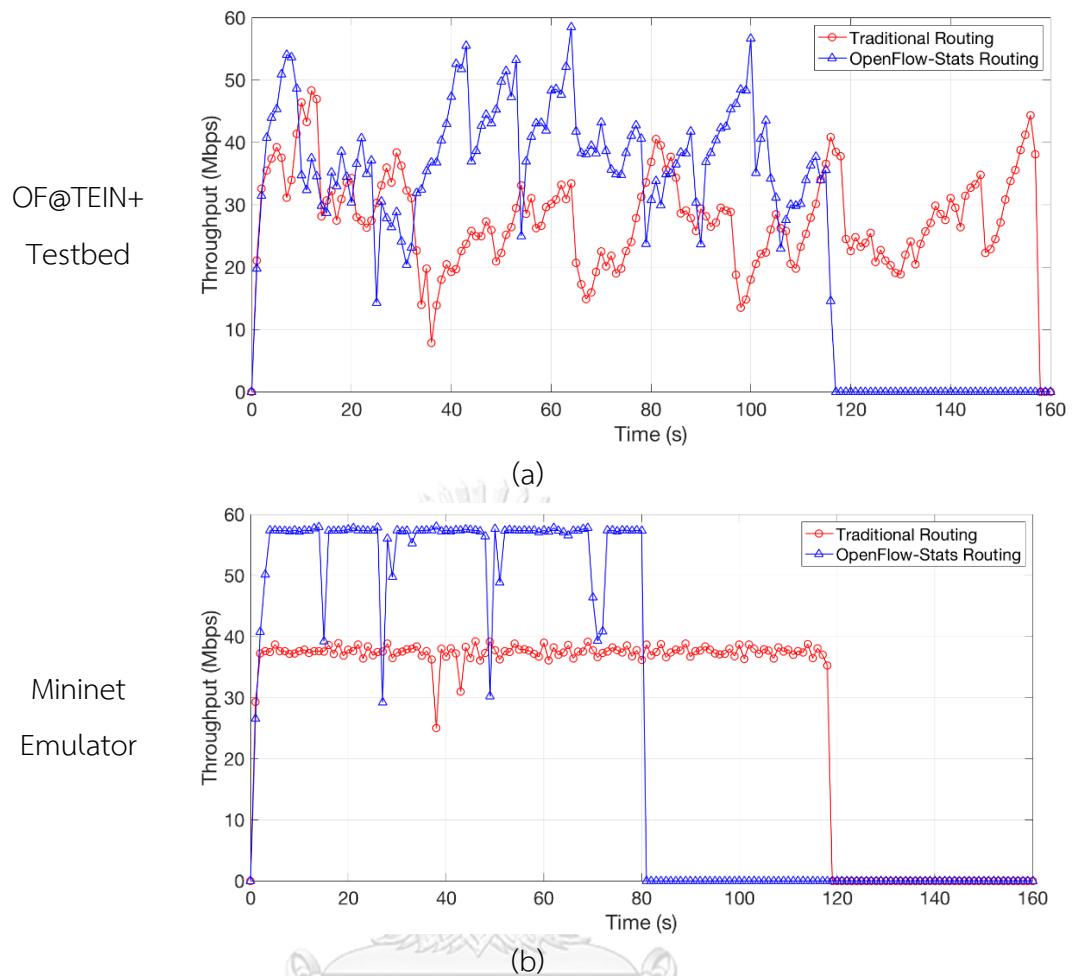


Figure 49 Comparison of experimental results between OF@TEIN+ testbed and Mininet emulator (File Size = 500MB).

5.6 Discussions on Limitations and Future Works

Despite several benefits, there are limitations that should be mentioned.

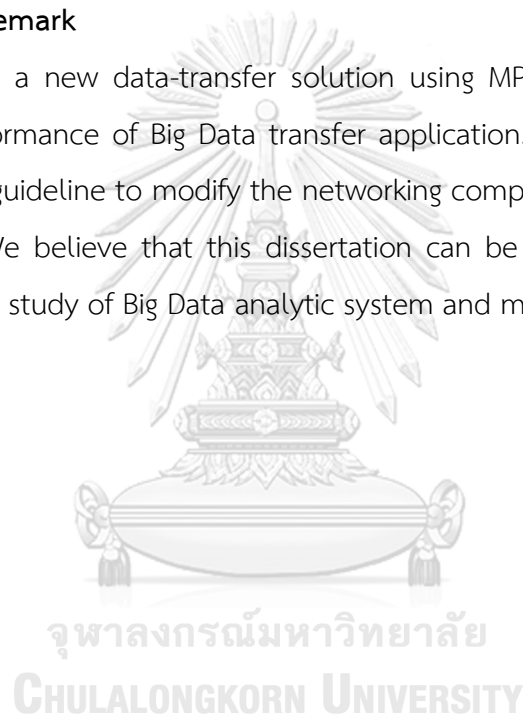
Our proposed framework can provide the highest throughput performance compared with other routing schemes. However, in the condition of extremely heavy traffic load, our OpenFlow-Stats routing induce higher completion time compared with the k -max disjoint routing. This is because all links are fully utilized above the threshold. All links are completely removed from our OpenFlow-Stats graph. Such a situation, the performance of our routing scheme is nearly the same as the traditional routing with disjoint paths. We believe that the adoption of adaptive threshold that need to be investigate for algorithm improvement.

The practicality is a main issue that is taken into consideration for developing our multipath transmission framework. Although, the proposed framework was evaluated by the real controller platform with the emulated OpenFlow-enabled switches, the actual distributed storage environment needs to be investigated.

Although, our framework is designed to support batch data only, we believe that using QUIC-based multipath transfer protocols such as MPQUIC and MFQUIC can be investigated for supporting other data types.

5.7 Concluding Remark

We induce a new data-transfer solution using MPTCP and SDN in order to improve the performance of Big Data transfer application. Our proposed framework can be used as a guideline to modify the networking component in a distributed data storage system. We believe that this dissertation can be the beginning of Big Data transfer for further study of Big Data analytic system and more complex data types.



REFERENCES

1. Philip Chen CL, Zhang C-Y. Data-intensive applications, challenges, techniques and technologies: A survey on Big Data. *Information Sciences*. 2014;275:314-47.
2. Chen M, Mao S, Liu Y. Big Data: A Survey. *Mobile Networks and Applications*. 2014;19(2):171-209.
3. Yang C, Huang Q, Li Z, Liu K, Hu F. Big Data and cloud computing: innovation opportunities and challenges. *International Journal of Digital Earth*. 2017;10(1):13-53.
4. Oussous A, Benjelloun F-Z, Ait Lahcen A, Belfkih S. Big Data technologies: A survey. *Journal of King Saud University - Computer and Information Sciences*. 2018;30(4):431-48.
5. Yu S, Liu M, Dou W, Liu X, Zhou S. Networking for Big Data: A Survey. *IEEE Communications Surveys & Tutorials*. 2017;19(1):531-49.
6. Liu Z, Kettimuthu R, Foster I, Beckman PH. Toward a smart data transfer node. *Future Generation Computer Systems*. 2018;89:10-8.
7. Qin Y, Rodero I, Simonet A, Meertens C, Reiner D, Riley J, et al. Leveraging user access patterns and advanced cyberinfrastructure to accelerate data delivery from shared-use scientific observatories. *Future Generation Computer Systems*. 2021;122:14-27.
8. Singh SK, Das T, Jukan A. A Survey on Internet Multipath Routing and Provisioning. *IEEE Communications Surveys & Tutorials*. 2015;17(4):2157-75.
9. Doshi M, Kamdar A, editors. Multi-constraint QoS disjoint multipath routing in SDN. 2018 Moscow Workshop on Electronic and Networking Technologies (MWENT); 2018 14-16 March 2018.
10. Tapolcai J, Rétvári G, Babarczi P, Bérczi-Kovács ER. Scalable and Efficient Multipath Routing via Redundant Trees. *IEEE Journal on Selected Areas in Communications*. 2019;37(5):982-96.
11. Zuo L, Zhu M, Wu C, Hou A, Cao L, editors. Bandwidth Reservation for Data Transfers through Multiple Disjoint Paths of Dynamic HPNs. 2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th

International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS); 2019 10-12 Aug. 2019.

12. Martín B, Sánchez Á, Beltran-Royo C, Duarte A. Solving the edge-disjoint paths problem using a two-stage method. *International Transactions in Operational Research*. 2020;27(1):435-57.

13. Lopez-Pajares D, Rojas E, Carral JA, Martinez-Yelmo I, Alvarez-Horcajo J. The Disjoint Multipath Challenge: Multiple Disjoint Paths Guaranteeing Scalability. *IEEE Access*. 2021;9:74422-36.

14. Hussein A, Elhadj IH, Chehab A, Kayssi A, editors. SDN for MPTCP: An enhanced architecture for large data transfers in datacenters. 2017 IEEE International Conference on Communications (ICC); 2017 21-25 May 2017.

15. Alharbi F, Fei Z, editors. An SDN Architecture for Improving Throughput of Large Flows Using Multipath TCP. 2018 5th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2018 4th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom); 2018 22-24 June 2018.

16. Lee SSW, Li K, Chan KY, YwiChi J, Lee T, Liu W, et al., editors. Design of SDN based large multi-tenant data center networks. 2015 IEEE 4th International Conference on Cloud Networking (CloudNet); 2015 5-7 Oct. 2015.

17. Deepshikha, Dave M, editors. An Efficient Traffic Management Solution in Data Center Networking Using SDN. 2018 International Conference on Power Energy, Environment and Intelligent Control (PEEIC); 2018 13-14 April 2018.

18. Zannettou S, Sirivianos M, Papadopoulos F, editors. Exploiting path diversity in datacenters using MPTCP-aware SDN. 2016 IEEE Symposium on Computers and Communication (ISCC); 2016 27-30 June 2016.

19. Duan J, Wang Z, Wu C, editors. Responsive multipath TCP in SDN-based datacenters. 2015 IEEE International Conference on Communications (ICC); 2015 8-12 June 2015.

20. Pang J, Xu G, Fu X. SDN-Based Data Center Networking With Collaboration of Multipath TCP and Segment Routing. *IEEE Access*. 2017;5:9764-73.

21. Chattopadhyay S, Shailendra S, Nandi S, Chakraborty S, editors. Improving MPTCP Performance by Enabling Sub-Flow Selection over a SDN Supported Network. 2018 14th

International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob); 2018 15-17 Oct. 2018.

22. Sandri M, Silva A, Rocha LA, Verdi FL, editors. On the Benefits of Using Multipath TCP and Openflow in Shared Bottlenecks. 2015 IEEE 29th International Conference on Advanced Information Networking and Applications; 2015 24-27 March 2015.

23. Nakasan C, Ichikawa K, Iida H, Uthayopas P. A simple multipath OpenFlow controller using topology-based algorithm for multipath TCP. *Concurrency and Computation: Practice and Experience*. 2017;29(13):e4134.

24. Jang-Ping S, Lee-Wei L, Jagadeesha R, Yeh-Cheng C, editors. An efficient multipath routing algorithm for multipath TCP in Software-Defined Networks. 2016 European Conference on Networks and Communications (EuCNC); 2016 27-30 June 2016.

25. Joshi KD, Kataoka K, editors. SFO: SubFlow Optimizer for MPTCP in SDN. 2016 26th International Telecommunication Networks and Applications Conference (ITNAC); 2016 7-9 Dec. 2016.

26. Liu Y, Qin X, Zhu T, Chen X, Wei G. Improve MPTCP with SDN: From the perspective of resource pooling. *Journal of Network and Computer Applications*. 2019;141:73-85.

27. Zhu T, Chen X, Chen L, Wang W, Wei G. GCLR: GNN-Based Cross Layer Optimization for Multipath TCP by Routing. *IEEE Access*. 2020;8:17060-70.

28. Ford A, Raiciu C, Handley M, Bonaventure O, Paasch C. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 8684; 2020.

29. Ferlin S, Ö A, Dreibholz T, Hayes DA, Welzl M, editors. Revisiting congestion control for multipath TCP with shared bottleneck detection. IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications; 2016 10-14 April 2016.

30. Habib S, Qadir J, Ali A, Habib D, Li M, Sathiseelan A. The past, present, and future of transport-layer multipath. *Journal of Network and Computer Applications*. 2016;75:236-58.

31. Kreutz D, Ramos FMV, Verissimo PE, Rothenberg CE, Azodolmolky S, Uhlig S. Software-Defined Networking: A Comprehensive Survey. *Proceedings of the IEEE*. 2015;103(1):14-76.

32. Nunes BAA, Mendonca M, Nguyen XN, Obraczka K, Turetli T. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Communications Surveys & Tutorials*. 2014;16(3):1617-34.
33. Coninck QD, Bonaventure O. Multipath QUIC: Design and Evaluation. *Proceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*; Incheon, Republic of Korea: Association for Computing Machinery; 2017. p. 160–6.
34. Coninck QD, Bonaventure O. Multiflow QUIC: A Generic Multipath Transport Protocol. *IEEE Communications Magazine*. 2021;59(5):108-13.
35. J. Iyengar E, M. Thomson E. QUIC: A UDP-Based Multiplexed and Secure Transport. RFC 9000; 2021.
36. Team M. Mininet [Available from: <http://mininet.org/>].
37. al. Tve. Open Network Operating System (ONOS) [Available from: <https://opennetworking.org/onos/>].
38. Cox JH, Chung J, Donovan S, Ivey J, Clark RJ, Riley G, et al. Advancing Software-Defined Networks: A Survey. *IEEE Access*. 2017;5:25487-526.
39. Foundation ON. OpenFlow Switch Specification Version 1.0.0. Open Networking Foundation; 2013.
40. Foundation ON. OpenFlow Switch Specification Version 1.3.0. Open Networking Foundation; 2014.
41. Hopps C. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992; 2000.
42. Postel J. Transmission Control Protocol. RFC 793; 1981.
43. Iyengar JR, Amer PD, Stewart R. Concurrent Multipath Transfer Using SCTP Multihoming Over Independent End-to-End Paths. *IEEE/ACM Transactions on Networking*. 2006;14(5):951-64.
44. Allcock W, Bresnahan J, Kettimuthu R, Link M, editors. The Globus Striped GridFTP Framework and Server. *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*; 2005 12-18 Nov. 2005.
45. Hagberg A, Swart P, S Chult D, editors. Exploring network structure, dynamics, and function using networkx2008 2008-01-01; United States. Research Org.: Los Alamos National Lab. (LANL), Los Alamos, NM (United States) Sponsor Org.: USDOE.

46. Jain S, Kumar A, Mandal S, Ong J, Poutievski L, Singh A, et al. B4: experience with a globally-deployed software defined wan. 2013;43(4 SIGCOMM Comput. Commun. Rev.):3–14.
47. ICTEAM. MultiPath TCP - Linux Kernel implementation: ICTEAM; [Available from: <https://www.multipath-tcp.org/>].
48. Dijkstra EW. A note on two problems in connexion with graphs. *Numerische Mathematik*. 1959;1(1):269-71.
49. Batchelor P, Daino B, Heinzmann P, Hjelme DR, Inkret R, Ja'ger HA, et al. Study on the Implementation of Optical Transparent Transport Networks in the European Environment—Results of the Research Project COST 239. *Photonic Network Communications*. 2000;2(1):15-32.
50. Group TT. TCPDUMP & LIBPCAP: The TCPDUMP Group; [Available from: <https://www.tcpdump.org/>].
51. Bosshart P, Daly D, Gibb G, Izzard M, McKeown N, Rexford J, et al. P4: programming protocol-independent packet processors. 2014;44(3 SIGCOMM Comput. Commun. Rev.):87–95.
52. Project LFC. Open vSwitch: Linux Foundation Collaborative Project; [Available from: <https://www.openvswitch.org/>].
53. Ha S, Rhee I, Xu L. CUBIC: a new TCP-friendly high-speed TCP variant. 2008;42(5 %J SIGOPS Oper. Syst. Rev.):64–74.
54. Afanasyev A, Tilley N, Reiher P, Kleinrock L. Host-to-Host Congestion Control for TCP. *IEEE Communications Surveys & Tutorials*. 2010;12(3):304-42.
55. Raiciu C, Handley MJ, Wischik D. Coupled Congestion Control for Multipath Transport Protocols. RFC 6356; 2011.
56. Khalili R, Gast N, Popovic M, Boudec JL. MPTCP Is Not Pareto-Optimal: Performance Issues and a Possible Solution. *IEEE/ACM Transactions on Networking*. 2013;21(5):1651-65.
57. Yu C, Mingwei X, Xiaoming F, editors. Delay-based congestion control for multipath TCP. 2012 20th IEEE International Conference on Network Protocols (ICNP); 2012 30 Oct.-2 Nov. 2012.

58. Hwang J, Low SH, Walid A, Peng Q, editors. Balanced Linked Adaptation Congestion Control Algorithm for MPTCP2016.
59. Ling TC, Kim J. OF@TEIN+: Open/Federated Playground for Future Networks: OF@TEIN+ Project GitHub; 2018 [Available from: <https://github.com/OFTEIN-NET/OFTEIN-Plus>].
60. Project OF. OpenStack OpenInfra Foundation Project; [Available from: <https://www.openstack.org/>].





APPENDIX

จุฬาลงกรณ์มหาวิทยาลัย
CHULALONGKORN UNIVERSITY

APPENDIX A

Technical Details of Our Proposed Framework Being Applied to An Overlay Network

This section, we demonstrate the adoption of our proposed framework over the traditional Internet architecture. If one wants to implement this framework over the traditional Internet, basically he just needs to put OpenFlow switches between the data servers/clients and the gateway routers. Then, the *VXLAN* technology can be applied to each switch in order to create the *VXLAN* tunnels among the data servers/clients. These tunnels are managed and controlled as an overlay network by the controller. With this configuration, our framework can be deployed in practice. There is no need to replace all routers with SDN switches. Figure 50 show our proposed framework being applied to an overlay network.

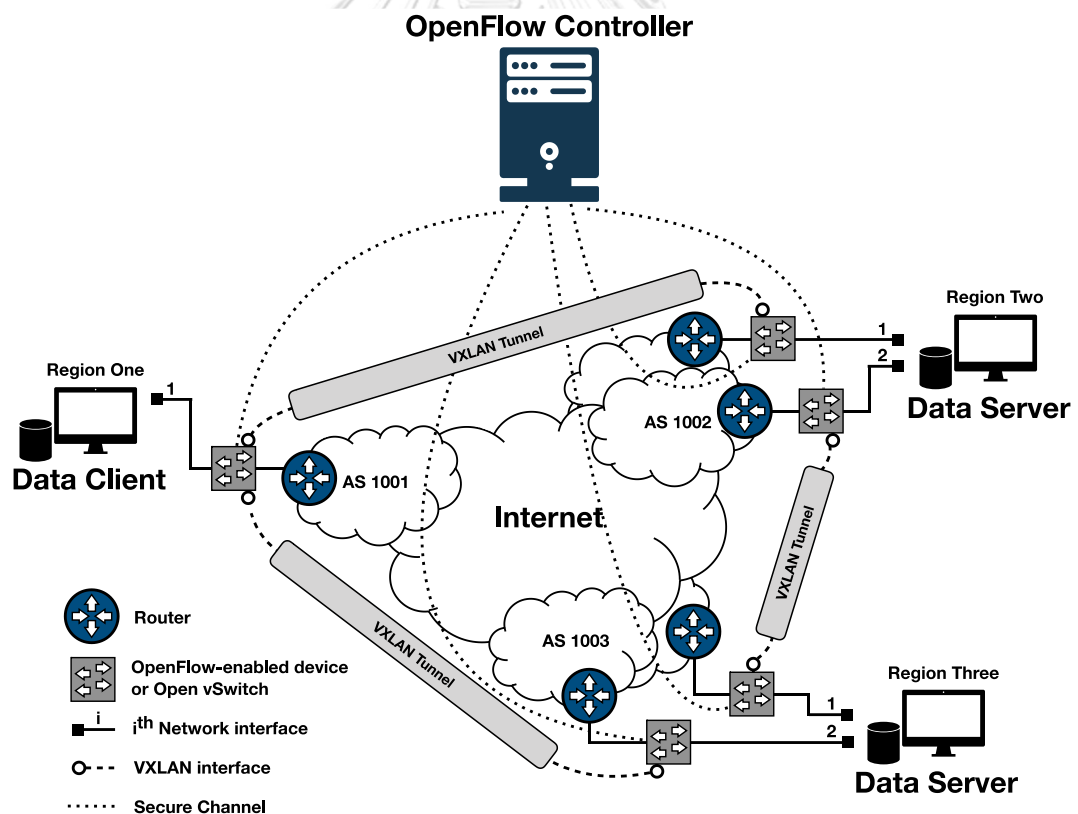


Figure 50 An example of our proposed framework being applied to an overlay network.

Figure 51 show examples of forwarding rules of OpenFlow-enabled switches that are in different regions. As can be seen in Figure 51, We can create simple forwarding rules with the lowest priority for handling other traffic flows (See *Switch A*). For the multipath decision, we can create complex forwarding rules with higher priority to manage your Big Data traffic flows (See *Switch A and Switch B*).

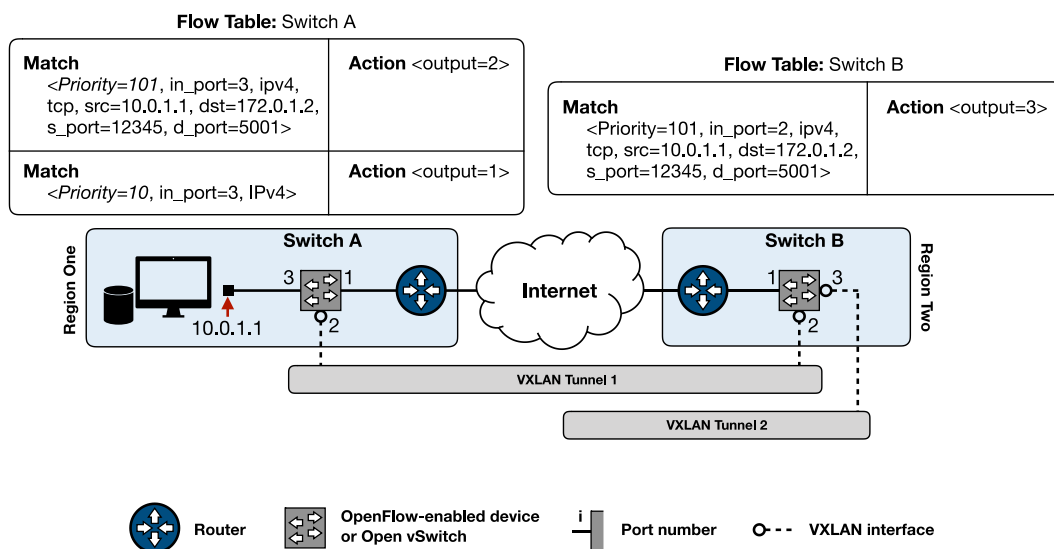


Figure 51 Examples of forwarding rules in the overlay-network environment.

VITA

NAME Kiattikun Kawila

DATE OF BIRTH 27 February 1989

PLACE OF BIRTH Phayao

INSTITUTIONS ATTENDED Received the B.Eng. degree in computer engineering from Kasetsart University Sriracha Campus, Thailand in 2010 and the M.Eng. degree in computer engineering from Chulalongkorn University, Thailand in 2013.

HOME ADDRESS 208 Phahonyothin Street, Mae Tam Sub-district, Mueang Phayao District, Phayao Province, 56000, Thailand

PUBLICATION International Journal Publication

K. Kawila, J. Kim and K. Rojviboonchai, "An SDN-coordinated Steering Framework for Multipath Big Data Transfer Application," IEEE Access, (Submitted).

N. Pramuanay, K. Na Nakorn, K. Kawila and K. Rojviboonchai, "LARB-Alpha: A Quantitative Study of Location-Aware Reliable Broadcasting Protocol in VANET," Journal of Internet Technology, vol. 18, no. 7, Dec. 2017, pp. 1669-1679.

International Conference Publication

A. Chanakitkarnchok, K. Kawila, G. Sato, Y. Owada and K. Rojviboonchai, "Disaster-Resilient Communication Framework for Heterogeneous Vehicular Networks," 2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC), 2019.

T. Banchuen, K. Kawila and K. Rojviboonchai, "An SDN framework for video conference in inter-domain

network," 2018 20th International Conference on Advanced Communication Technology (ICACT), 2018.

V. Deeying, K. Kawila, K. N. Nakorn and K. Rojviboonchai, "A study of vehicular desynchronization for platooning application," 2017 IEEE 17th International Conference on Communication Technology (ICCT), 2017.

P. Chumcharoen, K. Kawila, K. N. Nakorn and K. Rojviboonchai, "Queuing-aware Routing Algorithm in Software Defined Networks," 2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2017.

N. Pramuanay, K. N. Nakorn, K. Kawila and K. Rojviboonchai, "LARB: Location-aware reliable broadcasting protocol in VANET," 2016 13th International Joint Conference on Computer Science and Software Engineering (JCSSE), 2016.

National Journal Publication

J. Jullawat, K. Kawila, Kulit Na Nakorn and K. Rojviboonchai, "Quantitative Study of Path Selection Algorithm for Multipath TCP in Software-Defined Networking," Information Technology Journal, vol. 15, no. 1, Jun. 2019, 1-7.