On Optimizing Network Management and Traffic Transport in Software Defined Networks

by

Haythem YAHYAOUI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
Ph.D.

MONTREAL, JULY 25, 2022

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

M. Mohamed Faten Zhani, Thesis supervisor
Software and Information Technology Engineering Department at ÉTS

M. Amin Chaabane, Chair, Board of Examiners
Systems Engineering Department at ÉTS

M. Kim Khoa Nguyen, Member of the Jury
Electrical Engineering Department at ÉTS

Mrs. Noura Limam, External Independent Examiner
Computer Science department at University of Waterloo

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON "JULY 14, 2022"

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

Foremost, I would like to thank my advisor, Professor Mohamed Faten Zhani, for his assistance during my PhD studies. He has been extraordinarily patient with me, enabling me to develop my skills as a researcher. He has been an invaluable source of information and expertise during my studies.

In addition to my advisor, I would like to thank the other members of my thesis committee, Professors Amin Chaabane, Kim Khoa Nguyen and Noura Limam for accepting to evaluate this thesis. I really appreciate them for their insightful comments and their valuable feedback.

My heartfelt gratitude goes to my parents for everything they have done for me. I am immensely grateful for their spiritual support throughout my life. I also would like to thank my brothers for their unconditional support and confidence.

Sur l'optimisation de la gestion du réseau et du transport du trafic dans les réseaux définis par logiciel

Haythem YAHYAOUI

RÉSUMÉ

Les applications réseau futuristes telles que la téléchirurgie, la réalité virtuelle et la téléportation nécessitent des performances élevées en termes de latence, bande passante et taux de perte de paquets. Malheureusement, les infrastructures de réseau traditionnelles souffrent de nombreux défauts qui les rendent incapables de fournir de telles performances. Ces défauts vont de l'augmentation de la latence et de la consommation de bande passante à l'augmentation de taux de perte de paquets et de leur délai de transmission.

L'objectif de cette thèse est de concevoir des solutions pour l'optmisation de la gestion du réseau et du transport du trafic dans les réseaux définis par logiciel afin d'assurer les performances requises par les applications du futur et de minimiser la latence et la consommation de bande passante. Ainsi, la thèse comprend trois contributions principales. La première consiste à proposer un mécanisme de surveillance et de collecte de statistiques sur les flux permettant de minimiser les coût de surveillance dans les infrastructures à grande échelle. La deuxième contribution consiste à proposer un algorithme de routage qui utilise le data mining pour afin d'extraire le comportement des utilisateurs et optimiser le routage du trafic dans les réseaux définis par logiciel. La troisième contribution consiste à un nouveau protocole de transport permettant de réduire considérablement le délai de retransmission des paquets.

Mots-clés: Réseaux Définis par Logiciel, Surveillance des Flux, Routage, Protocole de Transport

On Optimizing Network Management and Traffic Transport in Software Defined Networks

Haythem YAHYAOUI

ABSTRACT

Futuristic network applications like telesurgery, virtual reality and teleportation require high infrastructure performance (i.e., latency, bandwidth, packet loss). Unfortunately, traditional network infrastructures are suffering from numerous shortcomings make them unable to provide the minimum required performance for such applications. These shortcomings range from increased latency and bandwidth consumption to increased packet loss and retransmission delay.

The objective of this thesis is to design solutions for the optimization of network management and traffic transport in software-defined networks in order to ensure the performance required by the applications of the future and to minimize latency. and bandwidth consumption. Thus, the thesis includes three main contributions. The first is to propose a mechanism for monitoring and collecting statistics on flows to minimize monitoring costs in large-scale infrastructures. The second contribution is to propose a routing algorithm that uses data mining to extract user behavior and optimize traffic routing in software-defined networks. The third contribution consists of a new transport protocol allowing to considerably reduce packet retransmission delays.

Keywords: Software Defined Networking, Flow Monitoring, Packet Routing, TCP, Packet Retransmission

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ALGORITHMS

Page

LIST OF ABBREVIATIONS

ACK	Acknowledgement
API	Application Programming Interface
CDF	Cumulative Distribution Function
ECMP	Equal-cost Multi-Path routing
FCT	Flow Completion Time
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
LCM	Low-Cost Monitoring for single controller infrastructures
LCM-M	Low-Cost Monitoring for Multiple controllers infrastructures'
NF	Network Function
QoS	Quality of Service
RTT	Round Trip Time
SDN	Software Defined Networking
SFC	Service Function Chain
TA	Transport Assistant
TCAM	Ternary Content-Addressable Memory
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VoIP	Voice over Internet Protocol
VR	Virtual Reality

LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

D	Destination node
h	Hop
S	source node
C	Controller

INTRODUCTION

With the emergence of a new breed of futuristic network applications like telesurgery, virtual reality and telepresence, today's networks and communication technologies are becoming obsolete and unable to cater the performance requirements needed by such applications. Indeed, today's networks are still suffering from potential congestion and unpredictable performance in terms of latency, bandwidth. However, for an application like telesurgery, a slight delay in the packet delivery could result in serious consequences. In addition, packet loss is another major problem leading to additional delay at the application level that may not be tolerated especially when the carried data is highly relevant for the application. As a result, accurately controlling parameters like packet loss, delay and bandwidth has become vital to be able to deploy and safely use future applications.

In this context, growing efforts are currently under way to redesign and rethink different aspects of network management including fault, configuration, accounting, performance and security in order to cater to the stringent requirements needed for future application and offer guaranteed performance. This thesis focuses on such aspects and advances the state of the art in order to address three network and service management challenges related mainly to network monitoring, data routing and transport.

In the following, we provide more details about these challenges, the thesis's objectives as well as the methodology that we adopted in order to achieve them.

Objectives and Methodology

Propelled by the need to overcome today's networks limitations and motivated by the emergence and maturity of Software Defined Networking (SDN) and cloud computing technology, there is a growing interest by the community to further optimize network management functionalities and traffic routing and transport protocols in order to minimize costs, reduce latency and minimize

bandwidth consumption. In this context, this dissertation presents three key contributions allowing to achieve these sought-after objectives. In particular, we propose the following contributions:

• *Design of an efficient traffic monitoring scheme:* to collect timely and accurately traffic statistics in software defined networks, the controller and the switches need to continuously exchange monitoring messages. These messages may consume significant amounts of bandwidth especially when there are several hops in the paths connecting the controller to the switches. Furthermore, monitoring messages could experience additional delays to reach the controller, which could make the reported statistics untimely and less relevant. Unlike prior monitoring mechanisms, we propose a monitoring scheme that carefully selects for each the switches that report the statistics so as to minimize bandwidth consumption and the reporting delay (i.e., the time needed for a monitoring message to reach the controller). In particular, we formulate the switch-to-flow assignment problem as an Integer Linear Program (ILP) and propose two heuristic algorithms to cope with large-scale instances of the problem. We consider two realistic scenarios where, in the first one, a single SDN controller is deployed to collect statistics, whereas in the second, multiple controllers are collecting statistics. Through extensive simulations, we that the proposed algorithms provide near-optimal solutions with minimal computation time and outperform existing monitoring strategies in terms of monitoring cost and reporting delay.

• *Design of a data-mining-based traffic routing scheme:* the routing strategy is crucial to efficiently deliver the traffic among the nodes and to carefully select and distribute the traffic within the network based on several criteria and objectives. For instance, selecting the path with the minimum number of hops could minimize the amount of bandwidth consumption as well as the latency. Unlike prior traffic routing strategies, in this thesis, we propose a novel routing strategy that leverages data mining techniques in order to identify the behaviors of the network users and

the size of their traffic flows (i.e., large, small) and efficiently compute the traffic routing rules. Specifically, we put forward a novel class-based routing strategy called LUNA. LUNA classifies the flows into mice and elephants based on their size. Afterwards, it leverages a data mining technique called association rules to generate the forwarding rules and route each flow based on its class (i.e., mouse or elephant). Experimental results show that LUNA has successfully identified the class of 80% of the flows. Furthermore, its class-based routing outperforms basic routing strategies in terms of flow completion time, throughput and packet loss by almost 47%, 41% and 23%, respectively.

• *Design of a novel transport protocol:* transport layer is typically implemented either by the User Datagram Protocol (UDP) or the Transport Control Protocol (TCP). While UDP is suffering from the lack of reliability, TCP provides a packet retransmission mechanism to detect and retransmit lost packets. That is why most network applications (e.g., HTTP, FTP) are relying on TCP to ensure a reliable delivery of the packets. However, TCP is an end-to-end protocol that stipulates that only lost packets are only detected and retransmitted from the source. This results in significant retransmission delays, especially when there are multiple hops between the TCP sender and receiver. In this context, we propose a novel transport protocol that is able to retransmit lost packets from intermediate nodes using a special network function, and thereby, it is possible to minimize the number of hops traveled by retransmitted packets and significantly reduce packet retransmission delays. In particular, we introduce a novel network function called Transport Assistant that could be deployed within the network in order to cache, detect and retransmit lost packets. Thanks to this function, there is no need to wait for the source to detect and retransmit lost packets as the TA ensures packet retransmission from the network and thereby minimize retransmission delays. Through extensive experiments, we show that using the transport assistant allows to outperform the standard TCP by minimizing the average packet transmission time, the flow completion time, the packet loss and the number of retransmitted packets from the source.

Thesis Organization

The remaining part of this dissertation is organized as follows. We start by presenting the background concept and the literature review in Chapter 1. We then introduce the proposed network traffic monitoring mechanism to minimize both latency and bandwidth consumption in chapter 2. We describe in chapter 3 the proposed data-mining-based traffic routing scheme able to minimize latency. In Chapter 4, we detail the novel transport protocol that we are proposing to reduce packet retransmission delays. Finally, we provide some concluding remarks and discuss potential key research directions and potential extension of our work.

CHAPTER 1

BACKGROUND MATERIAL AND LITERATURE REVIEW

This chapter starts first by presenting the basic concepts related to Software Defined Networking (SDN), network traffic monitoring and routing as well as transport protocols. We then review the main literature pertaining to the tackled research problems and we highlight the limitations of existing solutions while comparing them using several criteria.

1.1 Background Material

This sections presents the basic concepts related to the addressed research problems. We start by defining Software Defined Networking paradigm. We then present network traffic monitoring and routing mechanisms in SDN-based networks. Lastly, we provide an overview of the key transport protocols in traditional networks.

1.1.1 Software Defined Networking

Software Defined Networking is a relatively new technology that received a lot of attention as it offers a centralized control plane allowing a flexible and a fine-grained way to manage and configure the network as opposite to traditional networks where the control plane is decentralized and does not allow any flexibility in configuring and managing the flow routes. Indeed,the SDN technology separates the control plane (i.e., the decision) from the data plane (i.e., packet forwarding). The control plane is logically centralized in a component called "controller". The controller builds a global view with the detailed network state allowing to efficiently compute routing rules and it then communicates with the data plane switches in order to manage the infrastructure. Unlike traditional networks, SDN allows to dynamically change routing behavior on the fly without interruption at a fine-grained level. Furthermore, the global network view available at the SDN controller helps to more efficiently compute routing rules and to react to dynamic changes of the traffic or the network characteristics in order to achieve targeted

performance objectives (e.g., minimize delays, minimize packet loss, minimize bandwidth consumption).

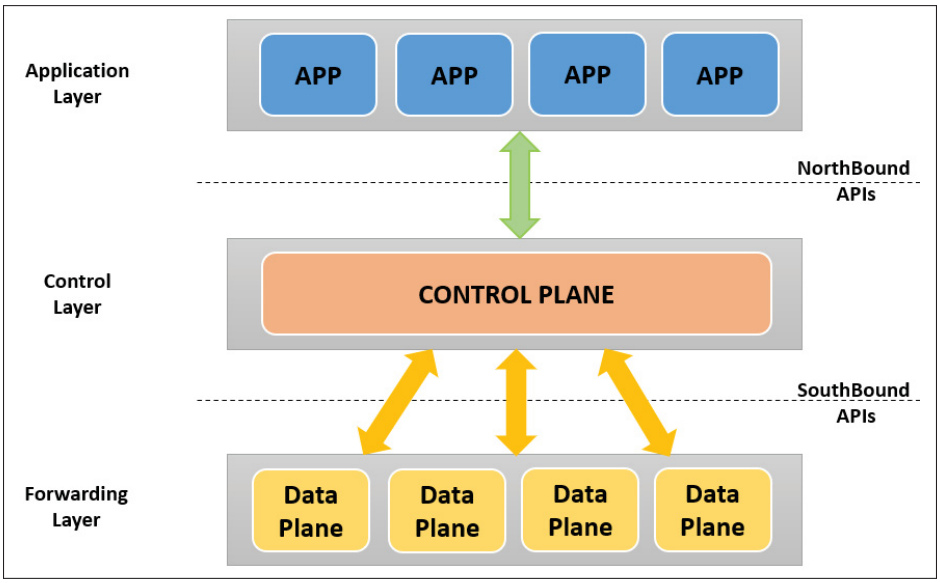

Figure 1.1 SDN Architecture

Figure 1.1 describes the typical SDN architecture composed of the control plane and the data plane. Both planes communicate together through the Southbound APIs. OpenFlow is the most known southbound protocol that provides three types of messages exchanged between the controller and the SDN switches: 1) controller-to-switch messages that are used to configure the switch and manage the flow tables, 2) symmetric messages that can be used in both direction and are useful to identify potential problems in the switch-controller connection, and 3) asynchronous messages that are sent from the switch to the controller to announce any change in the network state.

The main purpose of the controller in a typical SDN implementation is to manage the network traffic in order to achieve the objectives of the network operator (e.g., minimize latency, minimize bandwidth consumption, increase the throughput, minimize packet loss). In a huge infrastructure where a single controller is unable to manage thousands of flows (set of packets), the network operator could instantiate several controllers in order to distribute the control tasks among them (i.e., each controller could be responsible of a sub-network). In this case, these controllers

communicate among each other using Est and West APIs in order to share and synchronize the network state information and build the global view on the network state for each of them.

To manage the network and the offered services, the network operator could deploy SDN applications (e.g., routing, monitoring, congestion control) on the top of the controller. These applications communicate with the controller through an interface called the Northbound APIs (e.g., RESTful APIs). The Northbound APIs could be used by SDN applications to leverage the collected information from the controller to set up the routing rules. The controllers forward these packets according to the installed rules.

In the following, we detail and provide the potential challenges for network traffic monitoring, network traffic routing and network traffic transport protocol.

1.1.2 Network Traffic Monitoring

Network monitoring is the process of collecting the statistics about the network's components including network functions and the traversing traffic. Monitoring is crucial for all aspects of network and service management ranging from fault management, configuration and accounting to performance and security. This process helps to detect network infrastructure anomalies like traffic bursts, link failures, congestion and enables the network operator to quickly react to such anomalies in order to continuously ensure high network performance (i.e., latency, packet loss and throughput). Figure 1.2 illustrates the way that the controller and the switch communicate in order to collect flow statistics.

In order to collect flow statistics in a typical SDN implementation, the controller sends flow statistics' request messages to the switches that reply with flow statistics reply message. The exchanged messages could contain statistics of either a single flow or a set of flows. The reported statistics by the switch to the controller could include several measurements like packet count, flow duration, and amount of transmitted bytes.

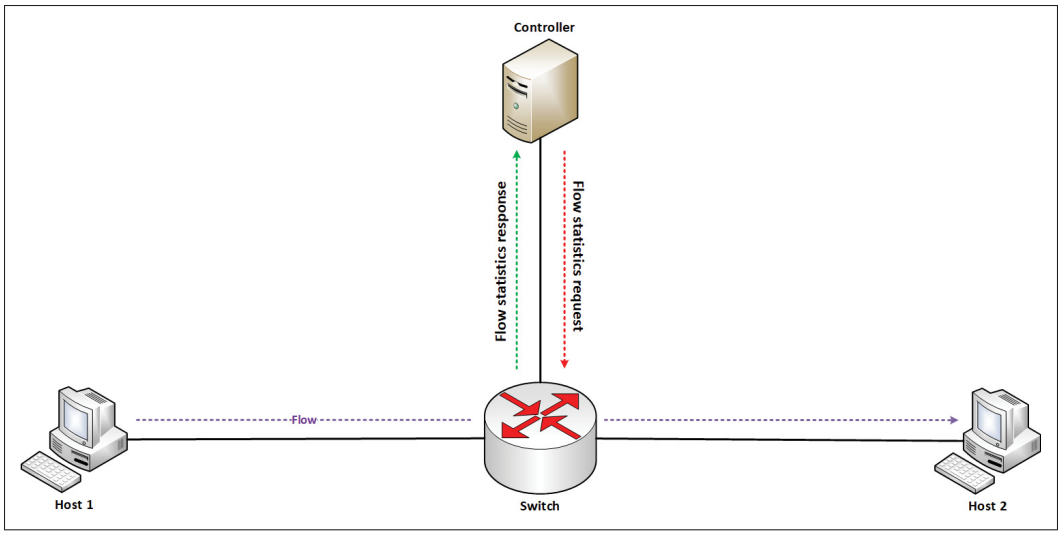

Figure 1.2 SDN flow monitoring

The main challenge for monitoring mechanisms is how to collect timely and accurately flow statistics while minimizing the costs of collecting statistics and transmitting monitoring messages to the controller. Ideally, the reporting switch should be physically close to the controller in order to ensure the lowest possible delay, and thereby ensure timely statistics. In addition, to ensure the accuracy of the measurements, monitoring messages should be sent frequently and continuously to the controller.

1.1.3 Traffic Routing Mechanisms

Network traffic routing is the process of selecting a route between two nodes while trying to satisfy performance requirements (e.g., low delay, high throughput, low bandwidth consumption). Routing is key for network operators as it has a significant impact on the network infrastructure performance. For instance, if delay is not carefully taken into consideration by the routing strategy, the impact on the performance of several network application running on the infrastructure could be significant.

Technically, the SDN controller computes the routing rules based on the statistics collected by the monitoring mechanism. Afterwards, the computed rules are installed in the switch in the

way that when the packets of a certain traffic arrive, they are forwarded accordingly. These rules are installed in the routing table which is stored in a limited memory in the switch called TCAM. Each rule has a timeout. If it expires, the rule is automatically deleted in order to effectively leverage the TCAM memory. When the TCAM does not contain a routing rule for a certain traffic, the switch requests a routing rules from the controller. This is why the first packet of a flow experiences additional delay compared to the remaining packets as these ones are directly forwarded thanks to the installed rule.

One of the main challenges for traffic routing strategies is how to select the shortest path that guaranties the lowest delay between two nodes. This is not trivial as the network state is changing frequently and the best path that guaranties the lowest delay for a certain traffic can become congested when other traffic arrive to the switch and need to be forwarded through the same path. Another interesting challenge is how to deliver the traffic while minimizing bandwidth consumption as bandwidth costs are high.

1.1.4 Traffic Transport Protocol

The transport layer of the OSI model is in charge of ensuring end-to-end process communication over the network. The main two protocols that are used today in the Internet are the Transport Control Protocol (TCP) and the User Datagram Protocol (UDP). While TCP ensures a reliable delivery for all the packets of the traffic flow thanks to its packet loss detection and retransmission mechanisms, UDP is a simple protocol that does not offer any reliability and is only limited to multiplexing and multiplexing services used to identify the source and the destination. That is why several network applications (e.g., HTTP, FTP) are relying on such a protocol to successfully deliver their traffic between end-hosts. Furthermore, TCP offers additional services like congestion and flow control and that are not offered by UDP.

In the following, we focus only on TCP as it is the most widely used protocol used today. Our goal is to further improve it in order to satisfy the requirements of futuristic applications.

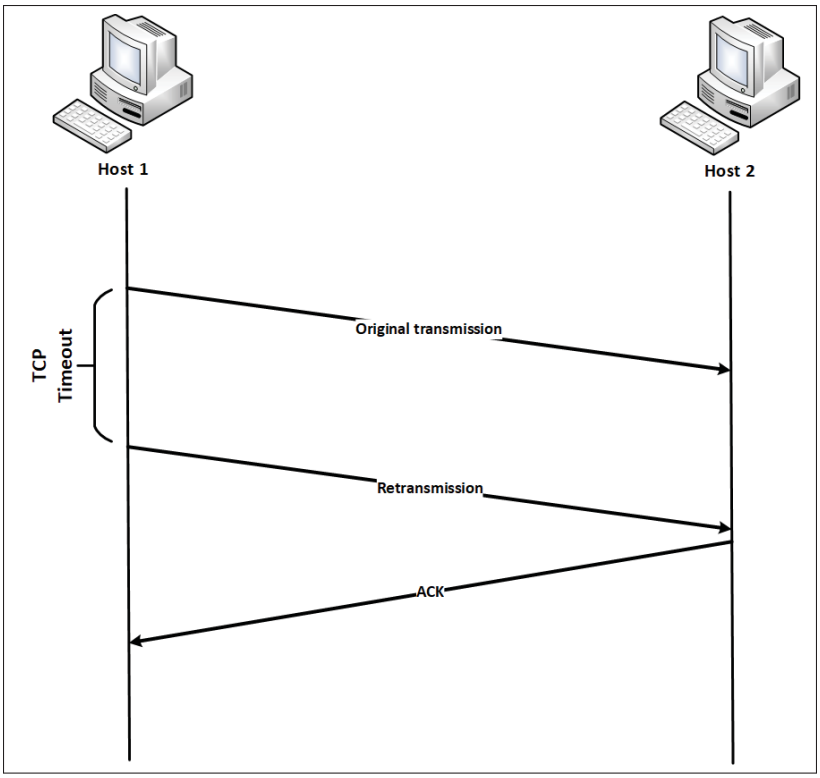

Figure 1.3 Transport protocol

In a typical TCP implementation, when a node transmits a segment, a timer called TCP Timeout is triggered. This timeout is computed during the connection establishment phase based on the delay between the two nodes (also called end-to-end delay) and generally it is almost equal to two end-to-end delays (called Round Trip Time - RTT). When the TCP timeout expires, the sender node assumes that the packet was lost and retransmits it. Another way, TCP uses to detect packet loss is based on duplicate acknowledgment. Indeed, when the sending nodes receives three duplicate acknowledgments, it assumes the first segment that was not acknowledged is lost and retransmits it.

There are several challenges with TCP to make it able to meet the futuristic network application requirements. For instance, TCP is not accurate in detecting packet loss as it tries to guess whether the packet is lost or not based on the timeout, which is usually variable and hence, not computed accurately. For instance, if the computed timeout is low, this would lead to

unnecessary retransmissions. This could increase the amount of consumed bandwidth as well as the transmission delay in the switches because unneeded packets will share the switches queue with the other packets.

1.2 Literature Review

In this section, we present and compare existing contributions that have aimed to improve network infrastructures performance in order to meet futuristic application requirements in terms of delay and bandwidth. First, we summarize literature focusing on minimizing flow monitoring cost. We then present existing solutions addressing the problem of improving network performance by proposing a routing strategy that carefully selects a path for the traffic. Finally, we present the existing works addressing the problem of minimizing TCP retransmission delays.

1.2.1 Flow Monitoring Solutions

The problem of minimizing monitoring cost has been widely addressed in the literature Yu *et al.* (2013) Su, Wang, Xia & Hamdi (2014) Tootoonchian, Ghobadi & Ganjali (2010) Chowdhury, Bari, Ahmed & Boutaba (2014) Van Adrichem, Doerr & Kuipers (2014) Yang & Yeung (2020) Henni, Hadjaj-Aoul & Ghomari (2016). Existing contributions can be classified into two categories: 1) solutions advocating to minimize the monitoring cost by minimizing the number of reporting switches and carefully selecting them among all the network's switches, and 2) solutions advocating to minimize the monitoring cost by varying or adapting the polling interval.

Among the solutions of the first category, Yu *et al.* (2013)'s work aims at measuring the utilization of the network links and propose a push-based approach called FlowSense. FlowSense collects only the byte count and duration of each flow thanks to the information available in the two OpenFlow messages `PacketIn` and `FlowRemoved` sent at the arrival of the flow and at the end of it, respectively. It then computes the link utilization by summing up the bandwidth usage of all the flows. The advantage of FlowSense is that it uses only two messages for each flow,

and hence, does not use any additional monitoring messages. However, this technique does not provide accurate statistics as the delay between the two message is unpredictable and could be high for long flows.

Su *et al.* (2014) propose FlowCover, a scheme that focuses on minimizing monitoring cost while ensuring high accuracy. FlowCover minimizes the number of monitoring messages by selecting only the least-loaded switches to report the flows' statistics to the controller. However, the selected switches can be far from the controller which lead to high reporting delay.

Tootoonchian *et al.* (2010) tackle the problem of measuring the traffic matrix with minimal number of monitoring messages. They propose OpenTM, a monitoring SDN application for OpenFlow controllers that reduces the number of messages by minimizing the number of reporting switches. It proposes five algorithms to select such switches as follows the last switch to the destination, uniformly random switch , non-uniformly random switch (switch selected with different probabilities), Round Robin and the least-loaded switch. However, OpenTM does not take into consideration the reporting delay, i.e., the time need to send the statistics from the switches to the controller.

Among the solutions of the second category which focus mainly on adapting the frequency at which statistics are reported to the controller. For instance, Chowdhury *et al.* (2014) address the problem of minimizing the number of monitoring messages and introduce PayLess, a low cost monitoring framework for SDN networks. PayLess measures the current byte count and compare it to the previous one. If the difference is greater than a threshold, the polling interval is reduced. Otherwise, the polling interval is increased. Similarly, Van Adrichem *et al.* (2014) introduce OpenNetMon which compares the current throughput measure with the previous measure and adjust the polling interval accordingly. However, both PayLess and OpenNetMon do not consider neither the bandwidth consumed by the monitoring messages, which may be high when the measured variables are very variable, nor the reporting delay as they do not discuss how the reporting switches are selected.

Furthermore, some contributions focused on finding a trade-off between improving the measurement accuracy and minimizing the monitoring cost. These two objectives are conflicting because to improve the accuracy, we need to increase the communication frequency between the switches and the controller, which results in an increase in the monitoring cost. For instance, Tang, Shojaee & Haque (2021) address the problem of minimizing both monitoring cost and measurement accuracy in SDN-based networks. They propose a solution called Accurate and Cost-Effective Measurement System in SDN that selects the reporting switches for the current flows and identifies the reporting frequency while respecting the switches capacities'. However, this solution does not consider the reporting delay when selecting the reporting switch, which prevents from having timely statistics.

1.2.2 Comparative Study of Existing Solutions

We summarize in Table 1.1 the aforementioned solutions based on their objectives i.e., minimize the number of monitoring messages, minimize the amount of bandwidth consumption or minimizing the reporting delay. It also shows the reporting frequency for each solution whether it is periodic (i.e., fixed) or adaptive (i.e., adjusted based on the variability measured statistics).

1.2.3 Traffic Routing Strategies

In this subsection, we present the related work pertaining to traffic classification and routing. These problems have been widely addressed in the literature Al-Fares *et al.* (2010) Curtis, Kim & Yalagandula (2011) Liu *et al.* (2014) Poupart *et al.* (2016) Chao, Lin & Chen (2016). Several objectives have been considered like minimizing the network congestion, the flow completion time and the workload of SDN controller and also maximizing the overall throughput in the network.

Liu *et al.* (2014) focus on minimizing network congestion and load imbalance in SDN networks. They introduce a routing algorithm which identifies elephant flows and routes them. A flow is identified as an elephant if the number of bytes in the TCP buffer exceeds a predefined threshold.

Table 1.1 Our work versus existing strategies

Paper	Objectives				Reporting frequency
	Minimize monitoring messages	Minimize bandwidth consumption	Respect reporting delay	Consider multiple controllers	
FlowSense Yu *et al.* (2013)	√	X	X	X	**Periodic**
FlowCover Su *et al.* (2014)	√	X	X	X	**Periodic**
OpenTM Tootoonchian *et al.* (2010)	√	X	X	X	**Periodic**
PayLess Chowdhury *et al.* (2014)	√	X	X	X	**Adaptive**
OpenNetMon Van Adrichem *et al.* (2014)	√	X	X	X	**Adaptive**
LSF Yang & Yeung (2020)	√	X	X	X	**Periodic**
Henni *et al.* (2016)	√	X	X	X	**Adaptive**
Tang *et al.* (2021)	√	√	X	X	**Adaptive**

All elephant flows are split into multiple sub-flows. To improve load balance and link utilization, each sub-flow is routed through different paths depending on link utilization. Experimental results show that the proposed routing algorithm outperforms single path routing and Equal Cost Multi-Path routing (ECMP) Hopps *et al.* (2000) in terms of network throughput and link utilization. However, the complexity of the proposed algorithm is greater than $O(n^3)$. As a result, it takes relatively long time to detect elephant flows and compute forwarding rules which can degrade the infrastructure's performance.

Al-Fares *et al.* (2010) address the problem of improving data center network utilization and propose Hedera, a dynamic flow scheduling system. Hedera considers that each flow is small until the number of packets exceeds a predefined threshold. To achieve optimal load balancing, small flows are routed using ECMP Hopps *et al.* (2000). For large flows, the link that has enough bandwidth is selected. Experimental results show that Hedera outperforms ECMP in terms of bandwidth utilization. However, Hedera increases the controller's workload because it requires to regularly update to the packet counters at the controller.

Curtis *et al.* (2011) focus on minimizing the workload of the network controller and present Mahout, a traffic management system that is able to identify the class of the flows and route them accordingly. Mahout detects elephant flows at the end host through a shim in the operating system. If the amount of data in TCP buffers exceeds a threshold, then this flow is identified as elephant. When an elephant flow is detected, the network controller is noticed using in-band signaling mechanism. The controller computes the forwarding rules based on link utilization and selects the one with the lowest utilization. Mice flows are routed using ECMP. Experimental results show that Mahout outperforms Hedera Al-Fares *et al.* (2010) in terms of the time needed to detect elephant flows which reduces controller workload. However, Mahout focuses only on minimizing the controller's workload.

Chao *et al.* (2016) tackle the problem of flow classification in software defined data centers. They devise FlowSeer, a fast elephant flow detection method at the switch level that uses data stream mining. FlowSeer collects some statistics from the first few packets of each flow to train

the stream classification models. These statistics include the IP address and the maximum and the minimum packet size. Moreover, FlowSeer enables the switch to identify elephant flows. Finally, elephant flows are routed through the least congested paths in order to improve the throughput. Experimental results show that FlowSeer outperforms ECMP and Hedera. However, FlowSeer focuses only on maximizing the throughput.

Poupart *et al.* (2016) address the problem of minimizing the flow completion time. They propose an online flow size prediction to improve network routing using several machine learning techniques including Neural Networks, Gaussian Process Regression and Online Bayesian Moment Matching. The prediction is based on some information collected from the first few packets including source IP, destination IP, source Port, destination Port, the protocol and the size of the first three packets. After predicting the flow size, large flows (elephant flows) are routed through the least congested paths. Experimental results show that the flow completion time has been significantly improved. However, this solution computes forwarding rules only when the flow arrives. As a result, the FCT is impacted by the time needed to compute and install forwarding rules.

Liu, Yang, Gong & Ren (2021) focus on minimizing the flow completion time by proposing a routing strategy that adapts the weight evaluation of the hops throughout the path as well as the link bandwidth. They define three kinds of flows: Time-Triggered, Audio-Video-Bridging and Best-Effort traffic. The proposed strategy selects a path for each flow based on the flow type. For instance, time-triggered flows are routed through the shortest path so that the additional network delays of the other paths are avoided.

Sun, Wang & Zhang (2021) tackle the problem of improving the quality of service (QoS) by proposing a routing strategy that selects a path for each flow based on its QoS requirements. To do so, a classification method called MACCA2-RF&RF is proposed. This method aims at identifying flow category to obtain QoS requirements. Then, they propose a path selection algorithm which selects QoS guaranteed routing path for different flows with distinct QoS requirements.

1.2.4 Comparison of Existing Strategies

Table 1.2 compares existing routing strategies with respect to their targeted performance objectives. It classifies them into reactive and proactive depending on the way they are computing and installing forwarding rules. Furthermore, it highlights the classification technique leveraged to classify the flows whether it is threshold-based or ML-based classification.

1.2.5 Solutions to Minimize TCP Retransmission Delay

In this subsection, we provide an overview of the literature focusing on minimizing the packet retransmission delay. Such a problem has been widely addressed recently Chen *et al.* (2019b) Wang, Chen, Chi & Lei (2017) Chen *et al.* (2019a) Wan, Campbell & Krishnamurthy (2002).

Existing contributions are mainly focusing on minimizing the retransmission delay by detecting and retransmitting lost packets from the switch's cache. In this way, the number of hops traveled by retransmitted packets is reduced and therefore the retransmission delay is minimized. For instance, Wang *et al.* (2017) address the problem of minimizing TCP retransmission delay and propose a scheme called SDUDP, where edges switches are enabled with a TCP-UDP conversation function in order to minimize TCP overload (handshaking, ACK), while middle switches are enabled with retransmission function to store, detect and retransmit lost packets. The retransmission function monitors the identification field of each packet. If it is out of order, the function assumes that a packet is lost and requests it from the previous switch in the path. However, delayed packets could be considered as lost which results in unneeded packet retransmissions.

Wan *et al.* (2002) which focus on the problem of minimizing packet retransmission delay in wireless sensor networks. They introduce a scheme called PSFQ which minimizes the retransmissions from the source by transmitting packets in a hop-by-hop manner. PSFQ detects lost packets by monitoring the sequences number. If it is out of order, then a lost packet is detected. Afterwards, the lost packet is requested from previous switch in the flow path. However, PSFQ may significantly increase flow completion time as when it detects a lost packet, it stops

Table 1.2 Our work versus existing strategies

Paper	**Objectives**				**Reactive**	**Proactive**	**Classification Technique**	
	Minimize Congestion	Maximize throughput	Minimize Flow Completion Time	Minimize workloads			Threshold based	ML based
Curtis *et al.* (2011)	X	X	X	√	√	X	√	X
Liu *et al.* (2014)	√	√	X	X	√	X	√	X
Al-Fares *et al.* (2010)	X	√	X	√	√	X	√	X
Liu *et al.* (2021)	X	√	X	√	√	X	√	X
Chao *et al.* (2016)	X	√	X	X	√	X	X	Data stream mining
Poupart *et al.* (2016)	√	X	√	X	√	X	X	Neural networks
Sun *et al.* (2021)	√	X	√	X	√	X	X	Variety of ML algorithms

forwarding next packets until the lost packet is recovered. Moreover, relying on the packet order to detect lost packets is not necessarily accurate as delayed packets could be considered lost.

Sugimoto & Ito (2021) address the problem of degradation of quality of service due to congestion in SDN networks and propose a method consisting of three control strategies: Selection Control, Redundancy Control and Single-path Control. The proposed method dynamically switches the three controls based on the network state. Selection control chooses the path with the largest amount of traffic as the main path for high priority packets and prioritizes packets based on their size. Redundancy Control duplicates the packets in the ingress switches and discards the duplicated packets before they arrive to their destinations. Finally, Single-path Control leverage a single path to transfer the packets. However, this method is suffering from several shortcomings. First, packet duplication causes high bandwidth consumption which results in high operational costs. Second, a packet loss detection mechanism is needed to carefully detect lost packets. lastly, critical packets are not considered.

Chen *et al.* (2019a) suggest a scheme that can be implemented over UDP. This scheme enables edge switches to store packets, detect lost packets and retransmit them thanks to a caching and retransmission function. This function monitors the duration between consecutive packets. If a packet transmission exceeds this duration, then the packet is considered lost. It is then retransmitted from the caching function. Closely, Chen *et al.* (2019b) address the problem of minimizing the retransmission delay and propose SDATP, an adaptive transmission protocol for critical-time services. SDATP aims at enabling the switches with additional functionalities like in-path packet caching and in-path packet retransmission. To do so, an algorithm is proposed which determines the number of caching switches and their placements in the way that minimizes the retransmission delay with minimum caching switches. it detects lost packets based on the time interval of consecutive packets and the number of disordered packets. However, both strategies in Chen *et al.* (2019b) and Chen *et al.* (2019a) are relying on the duration between consecutive packets to assume packet loss which is not accurate and could result in numerous unneeded packets retransmissions'.

1.2.6 Comparative Study

The aforementioned papers are suffering from several limitations. First, existing solutions do not inaccurate packet loss detection as they are relying either on the time interval between consecutive packets or on the packet order which can result in unneeded retransmissions for delayed packets. Second, they lack flexibility with respect to packet types as all packets are treated the same way. Finally, they are not scalable as the they rely on the switches resources' (i.e., processing, memory, storage), which are limited. Hence, for a large number of flows, the switches would not have enough resources to cache additional packets. Contrarily, the novelty of our work relies on the idea of proposing a network function implemented in software that accurately detects lost packets, and allows the the network operator to select which packets to prioritize and able also to scale-up when the number of flows increases. Table 1.3 compares the existing strategies in terms of addressed objective, flexibility, scalability and the packet loss detection metric.

1.3 Conclusion

In this chapter, we provided key concepts related to the three addressed topics in this dissertation, namely, traffic routing, flow monitoring and traffic transport protocol. We also presented the related work and highlighted the shortcomings of the existing solutions of each of the addressed problems.

In the next chapter, we address the first objective in this dissertation that consists in minimizing flow monitoring cost in large-scale SDN networks.

Table 1.3 Our work versus existing strategies

Paper	Objective		Flexibility	Scalability	Packet Loss Detection Metric
	Minimize Retransmission Delay	Consider critical flows			
Wang *et al.* (2017)	√	*X*	*X*	*X*	**Packet identification field**
Wan *et al.* (2002)	√	*X*	*X*	*X*	**Packet order**
Chen *et al.* (2019a)	√	*X*	*X*	*X*	**Time interval of consecutive packets**
Chen *et al.* (2019b)	√	√	*X*	*X*	**Time interval of consecutive packets**
Sugimoto & Ito (2021)	√	*X*	*X*	*X*	**Timeout, Duplicate acknowledgement**
TCP Postel *et al.* (1981)	*X*	*X*	*X*	*X*	**Timeout, Duplicate acknowledgement**

CHAPTER 2

MINIMIZE FLOW MONITORING COST IN LARGE-SCALE SDN NETWORKS

2.1 Introduction

Network monitoring services is central to all aspects of network and service management from fault management, configuration, accounting to performance and security. More interestingly, with the growing popularity of high-precision and critical applications (e.g., telesurgery, remote virtual reality, augmented reality), an accurate, timely and scalable monitoring of the traffic flows becomes necessary to carefully improve network services and quickly react to detect congestion, failures and adjust the network configuration and routing strategy Clemm, Zhani & Boutaba (2020); Zhani & ElBakoury (2020); Yahyaoui, Aidi & Zhani (2020) It is hence not surprising that a recent study Mon (2018) predicts that the network monitoring market will increase from 1.67 billion in 2018 to 2.93 billion in 2023.

In a typical software-defined network, switches could send periodically monitoring messages carrying flow statistics about the traversing traffic flows to the controller. The controller receives these statistics within a delay that depends mainly on *the polling interval*, i.e., the time between consecutive monitoring messages (reverse of the monitoring frequency), and *the reporting delay*, i.e., the time needed for a monitoring packet to leave the switch reporting the statistics (i.e., *the reporting switch*) and to cross the network to reach the controller.

To provide accurate and timely flow statistics, both the polling interval and the reporting delay should be minimized. While a small polling interval would improve the accuracy of the statistics and allow to report every and each small variation in the traffic, it may result in a large number of monitoring traffic between the switch and the controller. Scalability becomes a severe issue in a realistic case where millions of flows should be monitored. To make the matter worse, when the reporting switch is located several hops from the controller, monitoring messages consume large amounts of bandwidth throughout all the links composing the path towards the controller (referred to in the following as *the monitoring cost*). In addition, when the path between the

reporting switch and the controller has a high propagation delay, the reporting delay becomes significant and the statistics carried by the message might become obsolete.

To remediate to these problems, several research efforts have focused on minimizing the number of monitoring messages by either varying the polling interval or by minimizing the number of reporting switches Yu *et al.* (2013) Su *et al.* (2014) Tootoonchian *et al.* (2010) Chowdhury *et al.* (2014) Van Adrichem *et al.* (2014) Yang & Yeung (2020) Henni *et al.* (2016). However, they overlooked the reporting delay, which is a key parameter to consider in order to ensure a timely and accurate delivery of the monitored information in order to allow a quick reaction by the controller.

Unlike previous work, in this chapter, we aim at minimizing the monitoring cost while respecting the reporting delay constraint of each of the flows. Indeed, as applications have different requirements, their associated flows tolerate different monitoring reporting delays depending on how critical is the application and how fast should the reaction of the controller to any anomaly or change in the traffic pattern.

In our work, we consider two cases. The first one assumes that the monitoring traffic is forwarded to a single controller, whereas the second one, assumes monitoring traffic is distributed among multiple controllers within the network.

We hence formulate the problem of reporting switch-to-flow assignment for the two cases as an Integer Linear Program (ILP) that selects a reporting switch for each flow in order to minimize the overall monitoring cost and satisfy the flow reporting delay and while taking into account the switch monitoring capacity (i.e., the number of flows that can be monitored by the switch).

The addressed problem boils to the problem of resource allocation in virtualized networks which is known as NP-hard Zhani, Zhang, Simona & Boutaba (2013), we also propose two heuristic algorithms that carefully assign reporting switches for the flows taking into account the switch capacity, the requested flow reporting delay and the bandwidth consumption. The first

heuristic solution is the *Low-Cost Monitoring algorithm* (LCM) that assumes the monitoring traffic is forwarded to a single controller, whereas the second one, *Low-Cost Monitoring Multiple controllers* (LCM-M) assumes monitoring traffic is distributed among multiple controllers within the network.

Finally, we evaluate our algorithms for different scenarios and compare it to two existing solutions, OpenTM proposed by Tootoonchian *et al.* (2010) and FlowCover introduced by Su *et al.* (2014). The comparison shows that our solution provides near-optimal solution and outperforms OpenTM and FlowCover in terms of monitoring cost and reporting delay.

The remainder of this chapter is organized as follows. Section 2.2 describes the addressed problem and highlights the challenges. Section 2.3 details the proposed problem formulation. Section 2.4 presents the proposed heuristic. The experimental results are then presented in Section 2.5. Finally, Section 2.6 concludes the chapter.

2.2 Problem Description

In this Section, we further describe the addressed problem and we provide a deeper discussion about the impact of the polling interval and the reporting delay on the monitoring cost. Flow monitoring refers to collecting statistics about a flow using the switches crossed by this flow. Typically, the controller sends a *flow statistics* Request message to the switch that replies with a *flow statistics reply* message. According to OpenFlow specification Ope (2018) the minimum size of the request and reply messages are 122 bytes and 174 bytes, respectively. Therefore, a single flow measurement (e.g., flow statistics request and reply messages) woukd costs 296 bytes per message. To ensure high monitoring accuracy, the controller should request flow statistics from the switches at a high frequency, i.e., with a small polling interval.

Figure 2.1 shows the impact of varying the polling interval on the monitoring cost (bandwidth) for a single flow. It clearly shows that, for a small polling interval, the monitoring cost is high and drops when the polling interval decreases, however, at the expense of monitoring accuracy. Furthermore, when the reporting switch is several hops away from the controller, the amount of

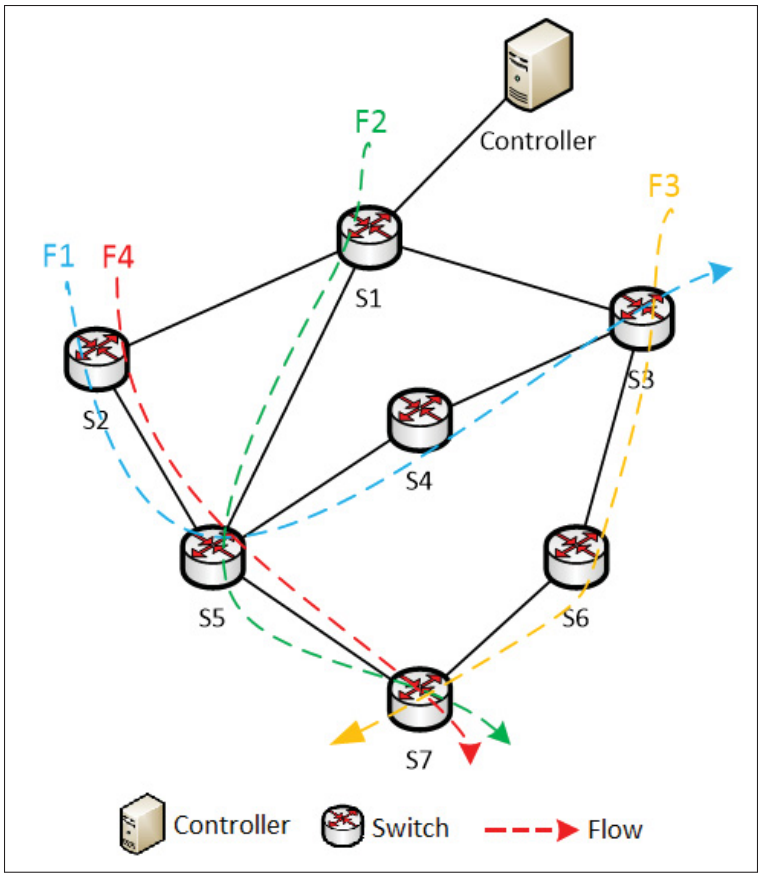

Figure 2.1 Per-flow monitoring cost

consumed bandwidth will be multiplied by the number of hops as it bandwidth is consumed throughout all the links of the path leading to the controller. It also leads to a high reporting delay due to the high propagation delay of the path.

In this work, we assume that the polling interval and the reporting delay of each flow are fixed by the network operator for each flow depending on the requirements of its associated application. For instance, these delays could be high for non-critical applications (e.g., web services, VoIP) where the monitoring could be delayed. However, they should be small to ensure a high-precision and fast monitoring for critical applications (e.g., telesurgery, real-time virtual reality).

The above discussion together with figure 2.2 clearly show that the selection of the monitoring switch for each flow has a direct impact on the monitoring cost and reporting delay, and hence

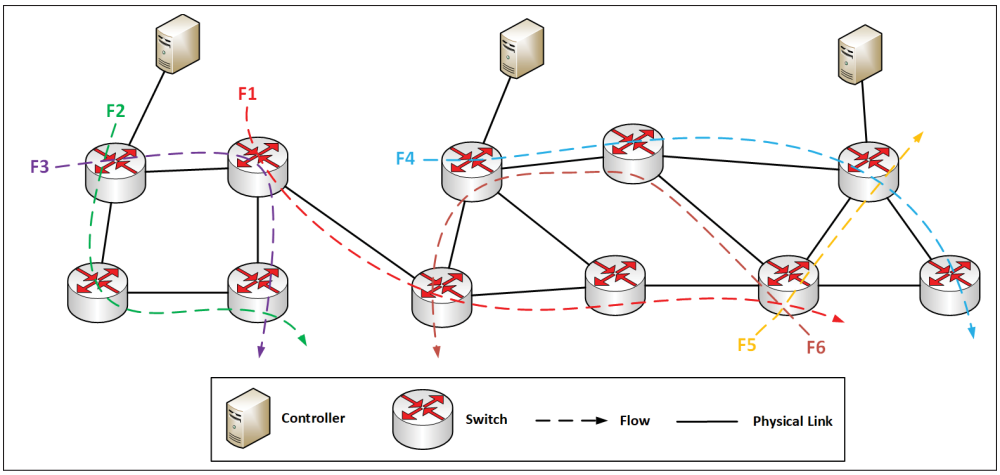

Figure 2.2 Flow to switch to controller assignment

the need for a flow-to-switch assignment scheme that ensures that a maximum number of flows are monitored with minimal monitoring cost and while satisfying the requested reporting delay.

In the following, we provide a brief overview of existing proposals to address this problem before presenting our proposed solution.

2.3 Problem Formulation

In this section, we mathematically formulate the flow-to-switch assignment as an Integer Linear Program (ILP) with the objective of minimizing the monitoring cost. We present, in the first subsection, the problem formulation by considering a single controller used for the whole network. In the second subsection, we present the formulation for the case of multiple controllers managing the network.

2.3.1 Single Controller Case

We model the network as an undirected graph $G = (S, E)$, where S is the set of switches and E is the set of edges. Let $S = \{s_1, s_2, ...s_{|S|}\}$ and the set of flows $F = \{f_1, f_2, ...f_{|F|}\}$. Table 2.1 summarises the used notations and their meaning.

Table 2.1 Notation and meaning

Notation	Meaning
S	Set of switches
F	Set of flows
K	Set of controllers S
j	a flow belonging to F
i	a switch belonging to S
k	a controller belonging to K
c_{ij}	The consumed bandwidth by the monitoring messages traveling from the switch i to the controller
c_{ijk}	The consumed bandwidth by the monitoring messages of the flow j traveling from the switch i to the controller k
m	The size of a monitoring message expressed in byte
n_i	The number of hops between the switch $i \in S$ and the controller
n_{ik}	The number of hops between the switch $i \in S$ and the controller $k \in K$
T_j	The requested reporting frequency for the flow $j \in F$ expressed in number of monitoring messages per second
σ_j	The maximum reporting delay for the flow j
p_{ij}	Boolean variable capturing the path of the flow j
p_{ijk}	Boolean variable capturing the path of the flow j by considering the controller k
x_{ij}	Decision variable to indicate if the switch i reports the statistics of the flow j
x_{ijk}	Decision variable to indicate if the switch i reports the statistics of the flow j to the controller k
d_i	The reporting delay of the switch i
d_{ik}	The reporting delay from the switch i to the controller k
γ_i	The maximum number of the monitored flows by switch i
δ_i	The maximum bandwidth consumption by monitoring messages in the switch i
α_k	The maximum number of the monitored flows by the controller k

The cost of monitoring the statistics of the flow $j \in F$ by the switch $i \in S$ is denoted by c_{ij} and represents the amount of bandwidth consumed by the monitoring messages that travels from the switch i to the controller. We define m as the size of a monitoring message expressed in byte, n_i the number of hops between the switch $i \in S$ and the controller. We also denote by

T_j the requested reporting frequency for the flow $j \in F$ expressed in number of monitoring messages per second. The maximum reporting delay for the flow j that can be tolerated is denoted by σ_j. The reporting frequency T_j and the reporting delay σ_j for each flow $j \in F$ can be fixed depending on the performance and monitoring delays requirements of the application associated with this flow and it can be adjusted by the network operator. Furthermore, we define p_{ij} as a Boolean variable capturing the path of the flow $j \in F$. Hence, p_{ij} is equal to 1 if the switch i belongs to the path of the flow j, and 0 otherwise.

The monitoring cost c_{ij} can be then expressed as follows:

$$c_{ij} = T_j n_i m \qquad \forall i \in S, \forall j \in F \tag{2.1}$$

• **Decision variables:** we define $x_{ij} \in \{0, 1\}$ to indicate if the switch i is reporting the statistics of the flow j.

$$x_{ij} = \begin{cases} 1, & \text{if switch } i \text{ reports the statistics of flow } j. \\ 0, & \text{otherwise.} \end{cases}$$

• **Objective function:** our main objective is to minimize the monitoring cost while satisfying the requested reporting delay for each flow. Technically, we aim at minimizing the number of hops crossed by the monitoring messages while taking into consideration the flow requirements in terms of reporting delay and the switch capacities. The objective function can be then expressed as follows:

$$Min \sum_{i \in S} \sum_{j \in F} x_{ij} c_{ij} \tag{2.2}$$

• **Problem Constraints:** in order to find a feasible solution, we should satisfy several constraints. For instance, we assume that the statistics are reported by a single switch for each flow. This

constraint can be expressed as follows:

$$\sum_{i \in S} x_{ij} = 1 \qquad \forall j \in F \tag{2.3}$$

We also need to ensure that the selected switch belongs to the path of the flow j. This constraint can be expressed as:

$$x_{ij} \leq p_{ij} \qquad \forall i \in S, \forall j \in F \tag{2.4}$$

The selected switch to report the statistics of the flow j should not exceed the requested reporting delay σ_j of the flow. We denote by d_i the reporting delay, i.e., the propagation delay, of the path from the switch i to the controller. This constraint can be written as follows:

$$x_{ij} d_i \leq \sigma_j \qquad \forall i \in S, \forall j \in F \tag{2.5}$$

When a switch is selected to report the statistics of several flows, its processing delay increases and thereby its reporting delay increases too. As a result, the selected switch may report untimely statistics. To avoid such a problem, we need to limit the number of monitored flows for each switch (e.g., each switch has a limited set of flows to report their statistics). To this end, we define γ_i as the maximum number of flows to be monitored by the switch i. This parameter can also be adjusted by the network operator. The switch capacity constraint can written as follows:

$$\sum_{j \in F} x_{ij} \leq \gamma_i \qquad \forall i \in S \tag{2.6}$$

We also define a threshold δ_i as the maximum bandwidth consumption by flow monitoring messages in each switch. For instance, the network operator can specify how much bandwidth is dedicated for flow monitoring messages in his network infrastructure. In the following, we

define the constraint of the maximum dedicated bandwidth for monitoring messages:

$$\sum_{j \in F} x_{ij} T_j m \leq \delta_i \qquad \forall i \in S \tag{2.7}$$

2.3.2 Multiple Controller Case

The undirected graph $G = (S, E)$ from the previous formulation is kept, where S denotes the set of switches and E denotes the set of edges. Let the set of controllers $K = \{k_1, k_2, ...k_{|K|}\}$, the set of switches $S = \{s_1, s_2, ...s_{|S|}\}$ and the set of flows $F = \{f_1, f_2, ...f_{|F|}\}$. Table 2.1 summarises the used notations and their meaning.

The monitoring cost of the flow j from the switch i to the controller k is denoted by c_{ijk} and represents the amount of bandwidth consumed by the monitoring messages when traveling from the switch i to the controller k. In order to simplify the formulation, we keep the same variables from the formulation of single controller case. These variables include the monitoring message size m, the reporting frequency T_j of the flow j, the maximum number of the monitored flows by each switch γ_i, the maximum reporting delay σ_j and the maximum bandwidth consumption by monitoring messages in each switch δ_i

The monitoring cost can therefore be written as follows

$$c_{ijk} = T_j n_{ik} m \qquad \forall i \in S, \forall j \in F, \forall k \in K \tag{2.8}$$

• **Decision variables:** we define $x_{ijk} \in \{0, 1\}$ to indicate if the switch i is reporting the statistics of the flow j to the controller k.

$$x_{ijk} = \begin{cases} 1, & \text{if switch } i \text{ reports the statistics} \\ & \text{of flow } j \text{ to the controller } k. \\ 0, & \text{otherwise.} \end{cases}$$

• **Objective function:** our main objective is to minimize the monitoring cost while satisfying the requested reporting delay for each flow. Unlike the formulation of single controller case, we aim at considering multiple controllers when selecting a reporting switch. Specifically, we aim at selecting a controller and a switch to monitor the incoming flow. The objective function can therefore be expressed as follows:

$$Min \sum_{k \in K} \sum_{i \in S} \sum_{j \in F} x_{ijk} c_{ijk} \tag{2.9}$$

• **Problem Constraints:** in order to find a feasible solution, we should satisfy several constraints. For instance, we assume that the statistics are reported by a single switch for each flow. This constraint can be expressed as follows:

$$\sum_{k \in K} \sum_{i \in S} x_{ijk} = 1 \qquad \forall j \in F \tag{2.10}$$

We also need to ensure that the switch i managed by the controller k belongs to the path of the flow j. This constraint can be written as follows:

$$x_{ijk} \leq p_{ijk} \qquad \forall i \in S, \forall j \in F, \forall k \in K \tag{2.11}$$

The reporting delay of the switch i reporting the statistics of the flow j to the controller k should not exceed the requested reporting delay σ_j by the network operator. We denote by d_{ik} the reporting delay from the switch i to the controller k. This constraint can be expressed as follows:

$$x_{ijk} d_{ik} \leq \sigma_j \qquad \forall i \in S, \forall j \in F, \forall k \in K \tag{2.12}$$

In order to avoid any kind of additional delay invoked by the increase of the switch processing delay due to a large number of flows. We need to limit the number of monitored flows by each switch. To do so, we present the following constraint:

$$\sum_{j \in F} x_{ijk} \leq \alpha_k \qquad \forall i \in S, \forall k \in K \tag{2.13}$$

We also define a constraint for the maximum bandwidth consumption by the monitoring messages in each switch. The network operator can change it depending on his infrastructure. This constraint can be expressed as follows:

$$\sum_{j \in F} x_{ijk} T_j m \leq \delta_i \qquad \forall i \in S, \forall k \in K \tag{2.14}$$

In order to avoid any impact on the controller performance and to minimize its workload, we need to limit the number of monitored flows for each controller. Therefore, we define α_k as the maximum number of flows per controller. This parameter can be adjusted by the network operator. The constraint to respect the maximum number of flows for each controller is defined as follows:

$$\sum_{j \in F} \sum_{i \in S} x_{ijk} \leq \alpha_k \qquad \forall k \in K \tag{2.15}$$

In order to minimize the number of monitoring messages in the infrastructure, we need to ensure that each flow j is monitored by a single switch i which reports the statistics to a single controller k.

$$\sum_{k \in K} x_{ijk} = 1 \qquad \forall i \in S, \forall j \in F \tag{2.16}$$

This formulation is proved to be NP-hard and hence, cannot scale for large instances of the problem. We therefore introduce in the following two heuristics to overcome this limitation.

2.4 Proposed Heuristic

In this section, we present our heuristic solutions to address the problem of minimizing the monitoring cost while respecting the reporting delay. Specifically, we propose two heuristics Low-Cost Monitoring (LCM) and Low-Cost Monitoring Multiple controllers (LCM-M) that consider the single controller case and the multiple controllers case, respectively.

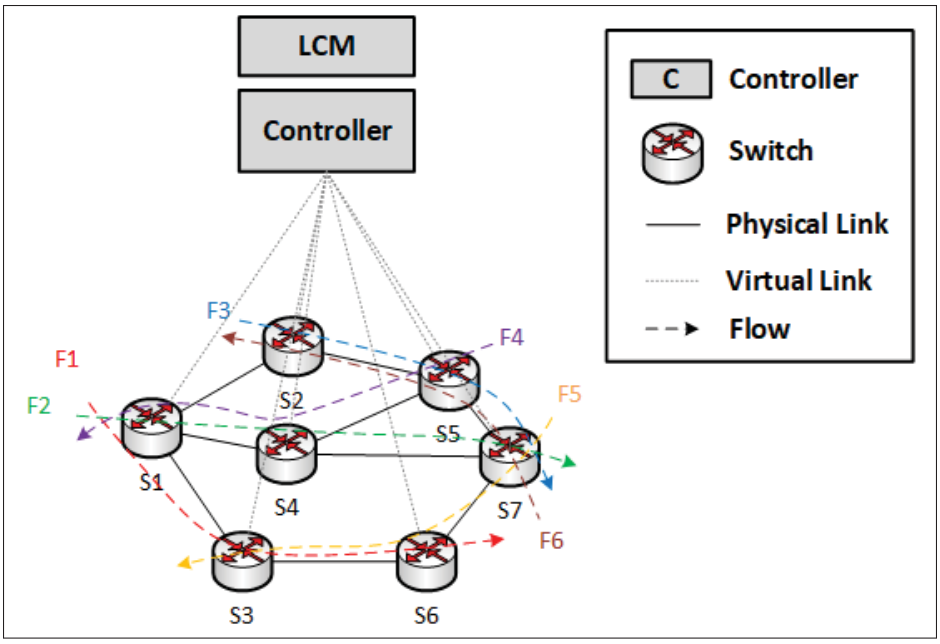

Figure 2.3 Low cost monitoring

Figures 2.3 and 2.4 present how our solution could be implemented in realistic infrastructures. Unlike figure 2.3 which describes LCM where a single controller is considered, 2.4 describes LCM-M which consider multiple controllers and highlights how LCM-M instances could exchange information (e.g., the number of hops between the switches and each controller) between each other in order to easily select the reporting switch and receiving controller. In the following, we provide the details of the two heuristics.

2.4.1 Low Cost Monitoring Solution (single controller)

The proposed algorithm is called *Low Cost Monitoring* (LCM) and aims at selecting the switch to report the statistics of each the flows of the network. It considers three criteria: the monitoring cost, the maximum number of flows assigned for each switch (i.e., switch capacity) and the reporting delay. In the following, we provide more details about these criteria:

• Monitoring cost: we consider the monitoring cost as the amount of bandwidth consumed by the monitoring messages. This amount depends on the number of monitoring messages and

the number of hops crossed throughout the switch-to-controller path. For instance, a single monitoring message sent to the controller through a path consisting of 3 hops will generate 3 messages. To estimate the bandwidth consumed by the monitoring messages, we multiply their number by their size and the number of hops in their paths towards the controllers.

• Switch capacity: a single switch could be report the statistics of several flows as it may belong to the path of multiple flows, When a switch reports the statistics of a significant number of flows simultaneously, it may be overloaded and its processing delay may increase which impacts it main task (i.e., packet forwarding). To deal with such an issue, LCM sets a capacity threshold for each switch (i.e., the maximum number of flows) so as to balance the monitoring load among all switches. In other words, if the capacity of a switch is reached, it cannot be considered by LCM as a candidate to monitor more flows.

The proposed algorithm LCM can be implemented as an SDN application running on the top of an SDN controller. LCM computes the shortest paths from the controller to the reporting switches and then merges them to build *the shortest path tree*.

The switch selection is made by considering the number of hops in each switch-to-controller path and the number of assigned flows for each switch. The closest switch with number of assigned flows smaller than its capacity is selected to report the flow statistics.

Algorithm 1 takes as input the set of flows and their paths and generate a tree consisting of the reporting switch of each flow. For switch selection, LCM considers not only the number of hops to the controller, but also the reporting delay and the number of assigned flows for each switch. The reporting delay and number of assigned flow should not exceed σ and γ, respectively. These thresholds can be adjusted by the network operator depending on the infrastructure's characteristics.

The reporting delay depends on the number of hops in the switch-to-controller path (e.g., the propagation delay) and the processing delay of the switches belonging to the switch-to-controller

Algorithm 2.1 LCM

```
Input: The topology, active flows F, σ, γ and δ
Output: Reporting_switches[]

for all f ∈ F do
    flow_path[] ← get_flow_path(f);
    switch_list[] ← sort_switches_by_hops(flow_path[]);
    for s ∈ switch_list[] do
        candidate ← get_the_first_switch();
        sDelay ← reporting_delay(candidate);
        sFlows ← assigned_flows(candidate);
        sBW ← bandwidth_consumption(candidate);
        if ((sDelay ≤ σ) & (sFlows ≤ γ) & (sBW ≤ δ)) then
            Reporting_Switches[] ← candidate;
        end if
        else
            Seek the next switch in the list;
        end if
    end for
end for
```

path. Therefore, the more we minimize the number of hops and switches in this path, the more the reporting delay is decreased.

2.4.2 Low Cost Monitoring Solution (multiple controllers)

The second proposed algorithm is called Low Cost Monitoring for Multiple controllers (LCM-M). It aims at minimizing the monitoring cost while respecting the reporting delay in large scale networks. More specifically, algorithm 2.2 selects not only the reporting switch but also the receiving controller in a way that minimizes the monitoring cost and satisfies the requested reporting delay which is represented by σ, the switch and controller capacities and the dedicated bandwidth for monitoring messages which are represented by σ, γ, δ, α, respectively.

In large-scale networks where there are millions of flows, selecting the reporting switch and receiving controller is highly challenging. Indeed, the capacity of of the switches and controllers

in terms of the maximum number of monitored flows should be respected. Technically, when the number of flows to be monitored increases, the controller's processing delay may increase and hence the controller decisions may experience additional delays. To solve this problem, LCM-M takes into account the maximum number of monitored flows in each controller provided by the network operator. Afterwards, it selects the reporting switch and receiving controller that better minimize the monitoring cost.

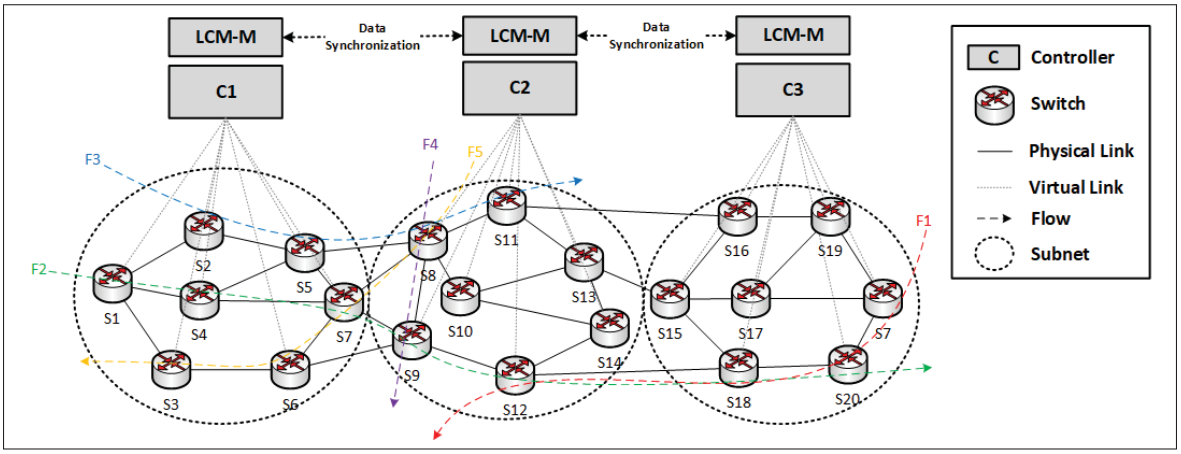

Figure 2.4 (Low Cost Monitoring solution - multiple controllers)

2.5 Evaluation

2.5.1 Single Controller Case

We evaluate our algorithm through extensive simulations using different evaluation scenarios to clearly show the improvements of our algorithm compared to the monitoring strategies proposed in the literature. For each strategy, we measure the monitoring cost, the reporting delay and the execution time. We also investigate the impact of varying the polling interval on the monitoring cost. For the simulation environment, all our simulations were conducted using Mininet (2018) where we implement the GEANT topology Geant (2018). The capacities and delays of the links are randomly varied between 5Mbps to 20Mbps and between 10ms to 100ms, respectively. We considered 10 scenarios where the number of flows is linearly varying from 100 to 1000 flows.

Algorithm 2.2 LCM-M

Input: The set of flows and their paths, The set of controllers, σ,α, γ and δ
Output: Reporting switch and receiving controller []

```
1  for all f ∈ F do
2      flow_path[] ← get_flow_path(f);
3      controller_and_switch_list[] ← sort_controllers
         and_the_switches_by_hops(flow_path[]);
4      for s ∈ controller_and_switch_list[] do
5          candidate ← get_the_first_switch_and_controller();
6          sDelay ← reporting_delay(candidate);
7          sFlows ← assigned_flows_per_switch(candidate);
8          cFlows ← assigned_flows_per_controller(candidate);
9          sBW ← bandwidth_consumption(candidate);
10         if ((sDelay ≤ σ) & (sFlows ≤ γ)
             & (cFlows ≤ α) & (sBW ≤ δ)) then
11             Reporting_Switches_and_receiving
                 controller[] ← candidate;
12         end if
13         else
14             Seek the next switch in the list;
15         end if
16     end for
17 end for
```

In addition to the proposed monitoring strategy LCM, we implemented two monitoring strategies from the literature, namely OpenTM Tootoonchian *et al.* (2010) and FlowCover Su *et al.* (2014). OpenTM selects for each flow the last switch to the destination to report the statistics whereas FlowCover selects the switches with highest number of flows the report the statistics. For LCM, we fixed the dedicated bandwidth for monitoring messages δ at 30% of the total bandwidth capacity and we varied the switch capacity γ (i.e., maximum number of monitored flows per switch) from 10 to 100 depending on the scenario. Finally, we compute the reporting delay of the switches based on the number of hops between the switch and the controller.

Fig. 2.5 depicts the monitoring cost obtained with the three compared strategies and the optimal one provided by CPLEX for the 10 different scenarios. It clearly shows that LCM outperforms

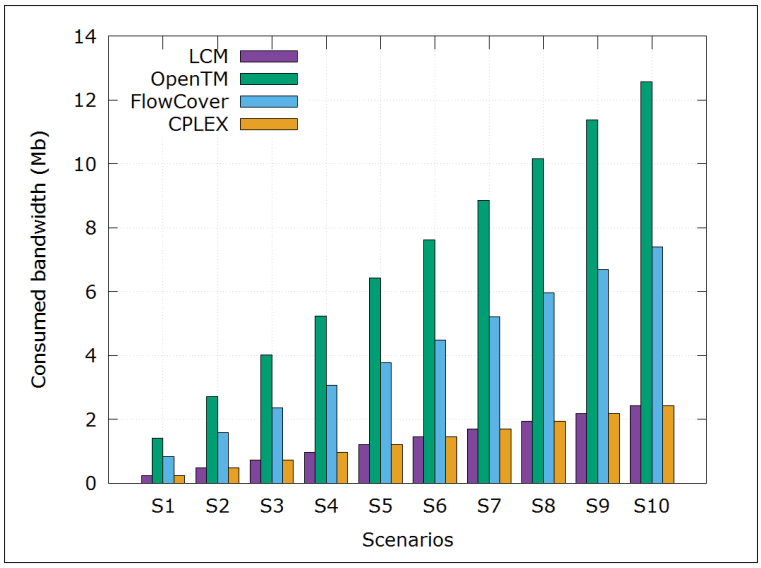

Figure 2.5 LCM monitoring cost

OpenTM and FlowCover and provides near optimal results. In fact, OpenTM selects the last switch to the destination to report the statistics whereas FlowCover selects the highest loaded switch. Both strategies do not take into account the number of hops crossed by a monitoring message to reach the controller. Unlike these strategies, LCM selects the closest switch to the controller satisfying the bandwidth, delay and capacity constraints. Therefore, the monitoring messages are traveling through less hops to reach the controller.

Fig. 2.7 illustrates the reporting delay for the implemented strategies. It clearly shows that LCM outperforms OpenTM and FlowCover and provides near optimal results. Indeed, FlowCover selects the highest loaded switch and does not consider neither the processing delay nor the number of hops in the switch-to-controller path. Similarly, OpenTM does not take into account the number of hops and the processing delay as it selects the last switch to the destination. On the other hand, LCM does not only select the closest switch to the destination but also takes into account the number of assigned flows for each switch to avoid overloading switches.

Fig. 2.8 shows the execution time of the all strategies. The execution time is the time needed to generate the set of reporting switches for all flows. As shown in this figure, LCM takes more time to generate the set of reporting switches compared to OpenTM and FlowCover. This is

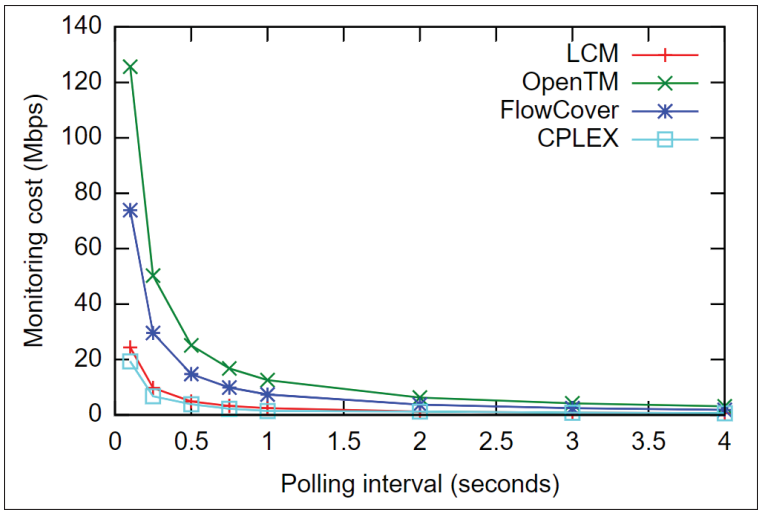

Figure 2.6 Polling interval

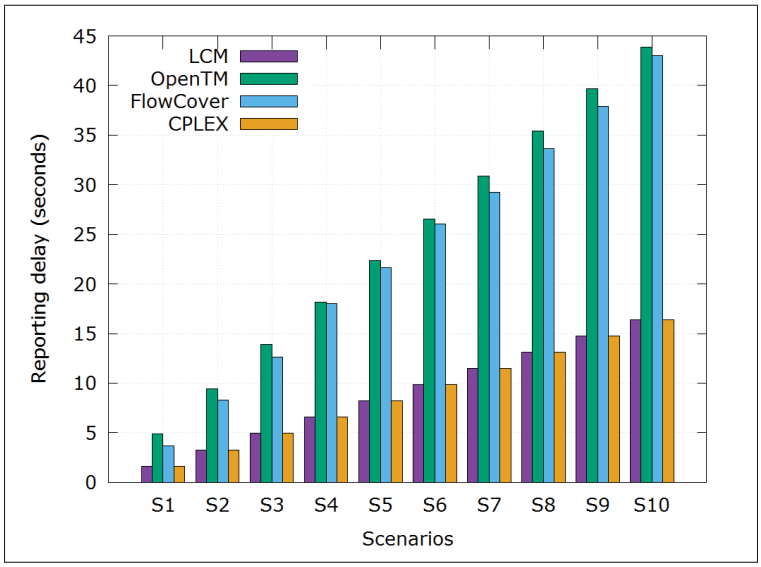

Figure 2.7 LCM reporting time

because LCM searches for the solution considering more criteria like the number of hops in the switch-to-controller paths, the number of assigned flows, the reporting delay and the bandwidth. However, the advantage of LCM is that it generates a near optimal result in much less time than CPLEX Cpl (2015).

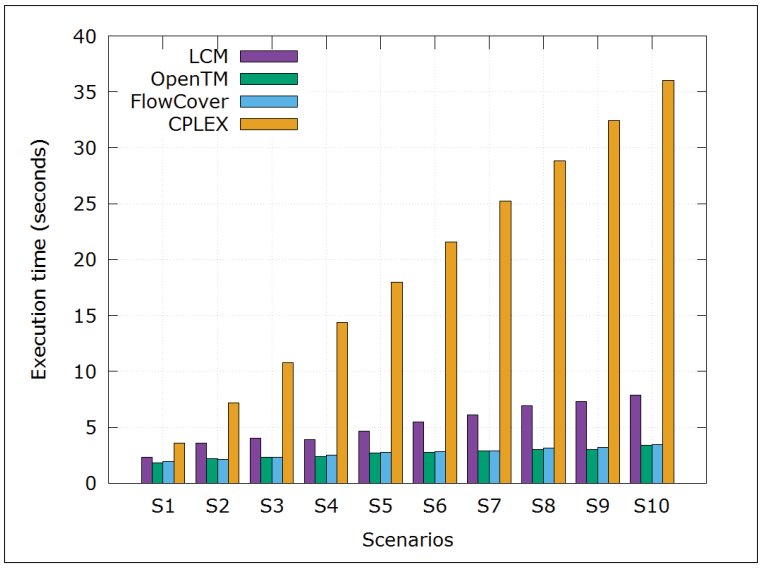

Figure 2.8 Execution time

We next study the impact of the polling interval (i.e., the time between two consecutive monitoring messages) on the monitoring cost. Indeed, reducing the polling interval leads to higher accuracy but at the expense of higher monitoring cost (i.e., more monitoring messages).

Fig. 2.6 shows the monitoring cost (computed using Eq. 2.1) for different polling intervals T_j and for all the studied strategies. It shows that LCM outperforms OpenTM and FlowCover and provides results close to the optimal ones found by CPLEX. For instance, when the polling interval is equal to 100 ms, the monitoring cost of LCM is almost 24.36 Mbps compared to when it comes to 125.68 Mbps and 73.88 Mbps for OpenTM and FowCover, respectively. This shows that LCM provides higher monitoring accuracy with minimal monitoring cost.

2.5.2 Multiple Controller Case

In the following, we evaluate our solution to minimize the monitoring cost considering multiple controllers. We use 10 scenario where we vary the numbers of flows, switches and controllers. We also vary the bandwidth capacity and the propagation delay of each link. Table 2.2 provides more details about the implemented scenarios. We compare our work to the results provided by the solver and the results of the baseline solution.

Table 2.2 Evaluation scenarios

Scenario	Number of flows	Number of switches	Number of links	Number of controllers	Links delays' (ms) (randomly)	Links capacities' (Mbps) (randomly)
1	50 000	1000	3000	25	[10, 100]	[10, 70]
2	70 000	1400	4200	35	[10, 100]	[10, 70]
3	120 000	2400	7200	60	[10, 80]	[10, 70]
4	170 000	3400	10 200	85	[10, 80]	[20, 80]
5	200 000	4000	12 000	100	[10, 70]	[20, 80]
6	250 000	5000	15 000	125	[5, 70]	[20, 100]
7	500 000	10 000	30 000	250	[5, 50]	[20, 150]
8	700 000	14 000	42 000	350	[5, 50]	[20, 150]
9	1 Million	20 000	60 000	500	[5, 40]	[50, 170]
10	2 Million	40 000	120 000	1000	[5, 30]	[50, 200]

In the baseline solution, we assume that each controller monitors a set of switches and each last switch to the destination in a flow path, reports the statistics to its managing controller.

We measure the monitoring cost which is the amount of consumed bandwidth when reporting the statistics to the controller, the reporting delay which is the time needed to report the statistics from the switch to the controller and the computation time which is the time spent to select a reporting switch for each flow.

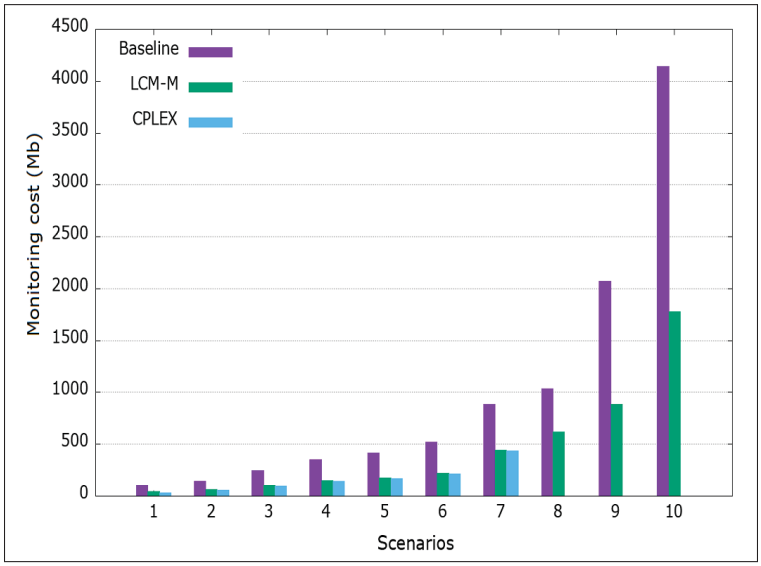

Figure 2.9 Monitoring cost

Figure 2.9 describes the monitoring cost for the three solutions. Its is clearly shown that LCM-M outperforms our baseline and provides a near optimal solution. While the baseline solution assumes that each set of switches is managed by a single controller and select the last switch to the destination to report the flow statistics, LCM-M selects the reporting switch and the receiving controller in the flow path that better minimize the bandwidth consumption and satisfy the selection criteria (i.e., the switch and controller capacities, the reporting delay, the number of hops between the switch and the controller). As a result, the monitoring messages are traveling the minimum number of hops to reach the controller. Starting from scenario 8, the ILP solver (i.e., CPLEX) is not able to find the optimal solution due to the high number of inputs.

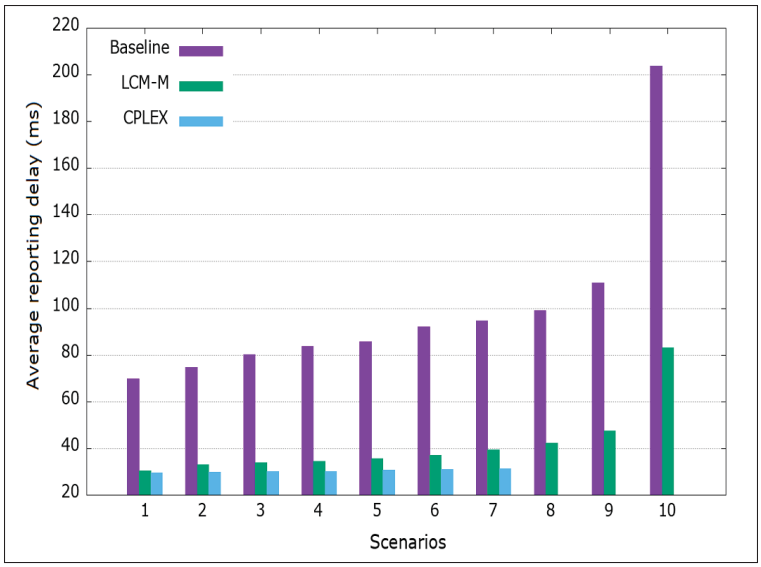

Figure 2.10 LCM-M reporting time

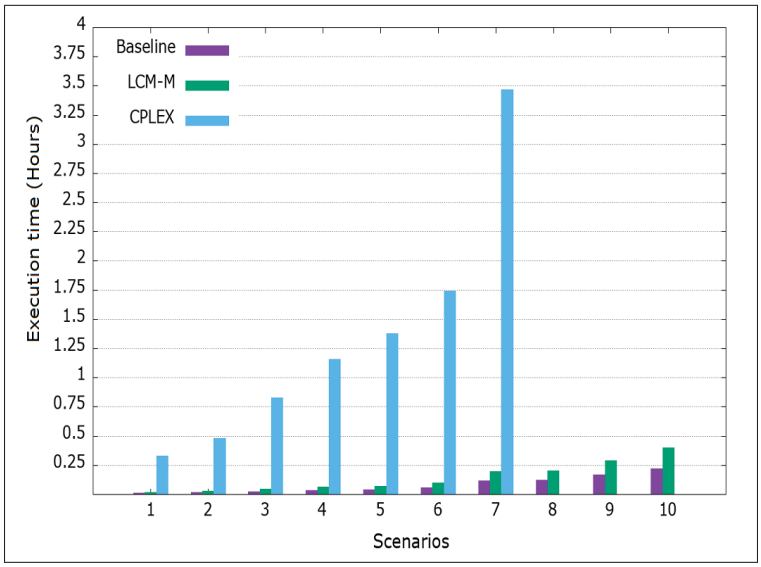

Figure 2.11 Execution time

Figure 2.10 depicts the average reporting delay for the three solutions. As presented in this figure, LCM-M outperforms the baseline and provides a near optimal solution. We compute the average reporting delay by summing up the reporting delay of all the flows then dividing by the number of flows in each scenario. Unlike the baseline, LCM-M takes into account the delay in

each switch-to-controller path before selecting the reporting switch and the receiving controller. That is why LCM-M provides significant results compared to the baseline. Similarly like the monitoring cost metric, our solver is not able to find the optimal solution starting from scenario 8.

Figure 2.11 illustrates the computation time for the three solutions. Its is clearly shown that the solver provides the optimal results but with unacceptable computation time. As a matter of fact, the flow can start and finish transferring before selecting the reporting switch. In this way, some flows could not be monitored and the controller could miss important information that can help to further improve the network performance and survivability. In contrast to LCM-M, the baseline provides the results faster than LCM-M because it does not take into account any criteria when selecting the reporting switch and receiving controller.

2.6 Conclusion

In this chapter, we addressed the problem of minimizing the flow monitoring cost while respecting the reporting delay. We hence proposed two heuristic algorithms LCM and LCM-M that ensure the minimum monitoring cost. The performed simulations show that both heuristics outperform existing monitoring strategies and provides near optimal solution in minimal computation time compared to CPLEX.

As a future work, we plan to propose a network function to minimize the controller's workload by minimizing the amount of monitoring messages between the switch and the controller.

CHAPTER 3

ON OPTIMIZING TRAFFIC ROUTING IN SDN NETWORKS

3.1 Introduction

Cloud storage is a promising service that allows customers to store and retrieve files while benefiting from the scalability and performance provided by cloud infrastructures Dieye, Zhani & Elbiaze (2017). According to a recent study, the worldwide storage industry would grow by 25.8 percent per year by 2021 Markets & Markets (2018). As a result, minimizing file transfer time through large-scale networks has become of a paramount importance for storage service providers as it may impact users' satisfaction. According to Tlili, Yahyaoui, Zhani & Elbiaze (2019), a slight increase in the file transfer time can reduce the user's satisfaction and thereby lead to a loss of revenue. For instance, according to Amazon an increase of 100ms in the file transfer time can decrease sales by 1% Linden (2006).

Typically, the time needed to transfer a file is the time needed to transfer all the packets of its corresponding flow. This time is referred to as the Flow Completion Time (FCT). In this context, Software Defined Networking (SDN) is a promising technology providing the programmability which can improve network infrastructure's performance and in particular the flow completion time Nunes, Mendonca, Nguyen, Obraczka & Turletti (2014). SDN enables the implementation of customized routing strategies which can significantly minimize the FCT Poupart *et al.* (2016).

Recently, several network operators are gradually embracing SDN technology. This raises several challenges as how to improve the performance of SDN infrastructures. In this context, one of the main issues is how to minimize the file transfer time. To solve such issue, the flow completion time of each file should be minimized. This requires to carefully route the flow of packets from the source to the destination.

Several studies have recently addressed this particular issue Poupart *et al.* (2016) Liu *et al.* (2014) Al-Fares *et al.* (2010) Chao *et al.* (2016). Typically, they classify the flows into mice and

elephants and they compute forwarding rules accordingly. For instance, elephant flows can be routed through least congested paths and thereby minimizing their FCTs Poupart *et al.* (2016). However, previous work address this issue in a reactive way. In other words, they compute forwarding rules and install them in the switches only when the flow arrives. As a result, the flow completion time is impacted by the time needed to compute and install forwarding rules. Unlike existing work, we are the first to leverage association rules Agrawal, Imieliński & Swami (1993) to compute routing paths in SDN networks. Furthermore, we computes forwarding rules and install them in the switches proactively. In other words, forwarding rules are computed and installed before the flow arrival. As a result, the packets are forwarded as soon as they arrive and the flow completion time is not impacted.

In this chapter, we address the problem of minimizing flow completion time and we propose a class-based routing strategy called LUNA operating as follows:

- LUNA classifies the flows into mice and elephants based on their size using K-means algorithm Jain (2010).
- LUNA analyzes users' behavior by computing their corresponding association rules. An association rule is an **if-then** statement describing the relation between a user and the class of his typically requested flows.
- LUNA routes each flow based on its class.

LUNA computes and installs forwarding rules proactively which means before the flow arrival. Unlike reactive strategies, when the packets of a flow arrive, they are directly forwarded without any additional delays. Furthermore, LUNA improves not only the flow completion time, but also the throughput and the network congestion.

The remainder of this chapter is organized as follows. The proposed routing strategy is detailed in Section 3.2. The experimental results are presented in Section 3.3. Finally, Section 3.4 concludes the chapter.

3.2 System Design

In this section, we propose a new class-based routing strategy called LUNA. It aims at improving network performance by minimizing flow completion time and packet loss and maximizing the network throughput. In the following, we detail how LUNA achieves this objective.

LUNA focuses only on the user IP and the flow to be routed from the server to the user. The different steps of LUNA are presented as follows:

- **Collect historical users' requests:** LUNA first collects all the information related to the historical users' requests including: the user IP, the request, the time when the user sends the request and the number of bytes in the reply. The number of bytes in the reply represents the size of the flow to be routed to the user.
- **Classify the flows into mice and elephant:** it focuses on the flow size (i.e., the number of bytes in the reply) to classify the flows into mice and elephants. To avoid any classification thresholds which impact the flexibility of LUNA, an unsupervised classification algorithm k-means is leveraged. Thresholds based classification impacts the flexibility of routing solutions because when deploying such solutions in different network infrastructures, the threshold from an infrastructure to another should be carefully chosen, hence the need to avoid them.
- **Learn association rules:** an association rule is a data mining technique describing if-then statements. It is computed using a statistic metric called confidence index Agrawal *et al.* (1993). In this work an association rule is written as:

  $$Bob \rightarrow elephant$$

  In other words, **if** the switch receives a packet to be routed to Bob then this packet belongs to an elephant flow.
- **Compute forwarding rules:** forwarding rules are computed for each user by considering his association rule. If there is an association rule for such user, then the forwarding rule is computed based on the class of the flow. In particular, if the association rule is

$Bob \rightarrow elephant$ then the forwarding rules route all the packets with IP destination: Bob through the least congested path.

As presented in Fig. 3.1, LUNA is an SDN application running on top of the SDN controller. It consists of three modules: a traffic monitoring module, an association rules generator and a routing rules generator.

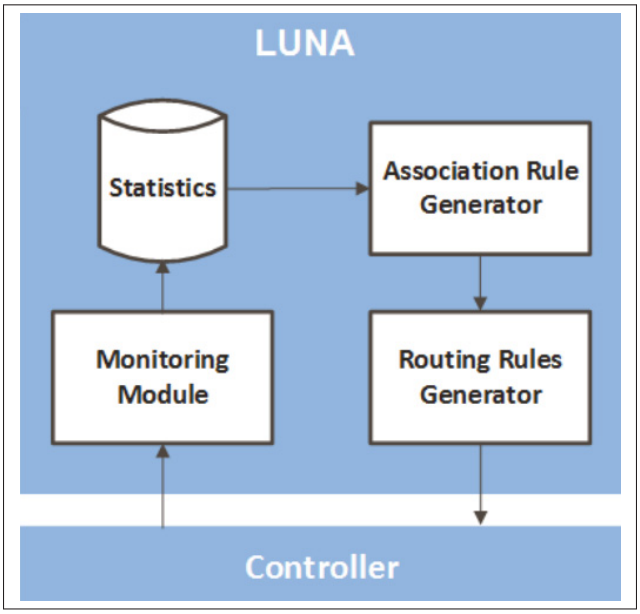

Figure 3.1 LUNA Architecture

3.2.1 Traffic Monitoring Module

The traffic monitoring module collects the traffic information from the network infrastructure. These information are the user IP, the request, the request type and the number of bytes in the reply. The request can be a specific web page or a file. The number of bytes in the reply represents the size of the flow to be routed to the user.

3.2.2 Association Rules Generator

The association rules generator classifies the flows into mice and elephants based on their size using K-means algorithm (K=2). Afterwards, it generates association rules between all users and the class of their flows. Table 3.1 presents a sample of the dataset after classification.

Table 3.1 Dataset sample

User	Flow class
Bob	elephant
Bob	elephant
Alice	elephant
Bob	elephant
Alice	mouse
Alice	elephant
Eve	mouse
Eve	mouse

In the dataset, each column consists of set of items and each line is called item set. Association rules learning tries to find frequent item sets among large datasets. Afterwards, it extracts frequent patterns. Such a technique helps to understand users' behavior. In particular, an association rule can be written as $X \rightarrow Y$, where X and Y are the user and the class of his typically requested flows. In other words, an association rule in this work is written as $Bob \rightarrow elephant$ This means that Bob is frequently asking for elephant flows. There are two essential metrics to compute association rules: support and confidence index. The support of an item set is the percentage of its occurrence in the dataset. The confidence index of an association rule is the percentage that presents the occurrence frequency of a rule head (i.e., Bob) among all item sets containing the rule body (i.e., elephant) in the dataset. It presents how reliable an association rule is. The higher the confidence index is, the more likely the head item (i.e., Bob) occurs among item sets containing the rule body (i.e., elephant).

The confidence index (CI) is presented by equation 3.1:

$$CI(user \rightarrow class) = \frac{freq(user, class)}{freq(user)} \tag{3.1}$$

Where freq(user,class) represents how frequently the user asks for a flow class in the dataset whereas freq(user) represents how frequently the user occurs in the dataset. If $CI(user \rightarrow class) = 1$, then the user asked for 100% of all his occurrences in the dataset for that flow class which is the case of $Bob \rightarrow elephant$ in table 3.1. LUNA leverages this association rule to predict the future flow class to be requested by the user. In other words, if a user occurs in the dataset for n times and he asked for elephant flows for n times. Then, we can assume that he will ask for elephant flow for his n+1 time.

Algorithm 3.1 Association Rules Generator

Input: DataSet consisting of historical user requests
Output: Association Rules []

```
1  i = 0;
2  for all network users do
3      if (CI(user → elephant) = 1) then
4          // The class of the flow for this user is elephant
             Association_Rules[i] ← "user → elephant";
5          i++;
6      end if
7      else if (CI(user → mouse) = 1) then
8          // The class of the flow for this user is mouse
             Association_Rules[i] ← "user → mouse";
9          i++;
10     end if
11     else
12         // The class of the flow is unknown
             Association_Rules[i] ← "user → ∅";
13         i++;
14     end if
15 end for
```

3.2.3 Routing Rules Generator

The routing rules generator represents the core component of LUNA. It selects of path for each flow based on the generated association rules. In particular, if the flow class is an elephant, then it selects the least congested path. As a results, the congestion is avoided and the flow completion is improved. If the flow class is unknown, then the routing path is randomly selected. A flow class is reported unknown only if there is no association rule for his user.

3.3 Implementation and Evaluation

In this section we evaluate the solution using Mininet (2018) and a real dataset.

3.3.1 Dataset Presentation

The used dataset for the experimentation consists of a set of HTTP requests collected from ClarkNet (2018) during a month from August 1^{st} to August 31^{st}, 2016. The dataset contains more than one million requests and almost 78000 users. For each request, we have the user IP, access time, reply code and number of bytes in the reply. In this work, we are focusing on the number of bytes of the reply as this amount represents the size of the flow to be routed from the server to the user.

Fig. 3.2 illustrates the cumulative distribution function of the size of the flows to be routed to the users. Almost 90% of the flows are smaller than 32Kb.

3.3.2 Experimentation

To show the performance of LUNA, We implement our solution and two different routing strategies. An ideal routing where we assume that all the classes of the flows are known and therefore all elephant flows are routed through least congested paths. And a random strategy where all flows are randomly routed.

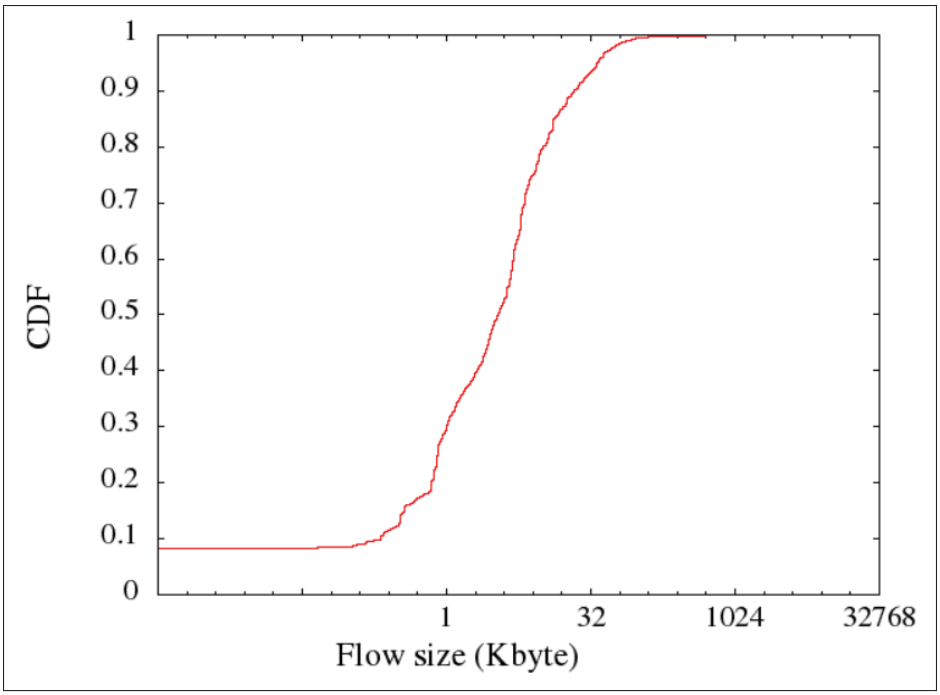

Figure 3.2 Flow size distribution

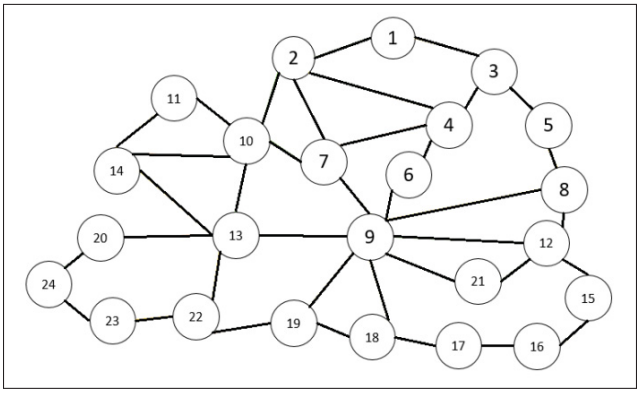

Figure 3.3 Network infrastructure

We implement LUNA as an SDN application on top of the ONOS controller Berde *et al.* (2014). To generate association rules, we run algorithm 3.1. We implement the GEANT topology Geant (2018) illustrated by Fig. 3.3 and we randomly set links capacity from 10Mbps to 20Mbps. All the flows from the collected dataset (i.e., HTTP reply) should be routed through the network

infrastructure. We connected the source to node number 1 and the destination to node number 22.

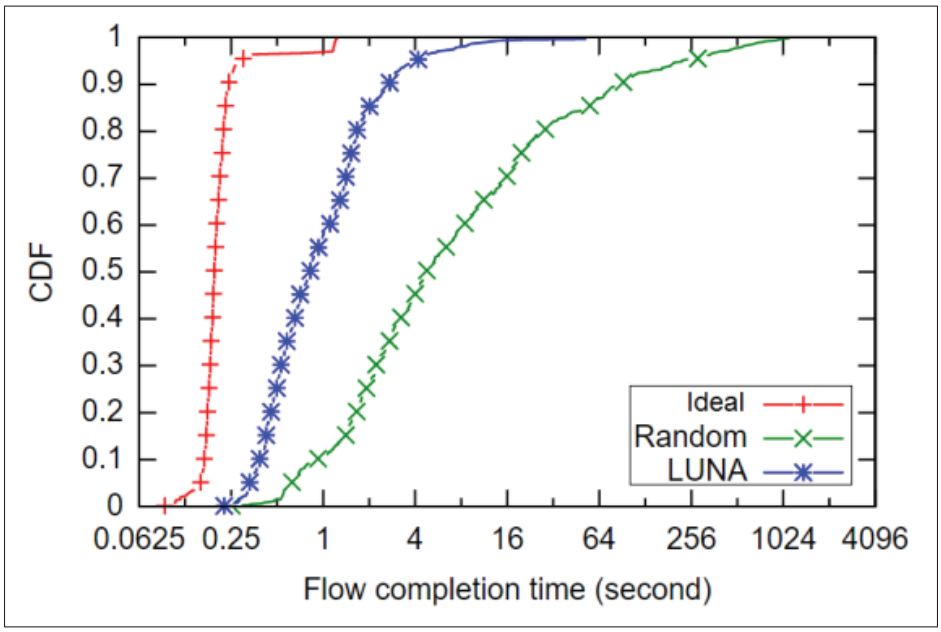

Figure 3.4 Flow completion time

Fig. 3.4 illustrates the cumulative distributed function of the flow completion time for each routing strategy. It is clear that LUNA can significantly improve the FCT compared to Random strategy. Almost 90% of flows are transferred with FCT less than 4 seconds. But for random strategy, almost 90% of flows are transferred with FCT less than 256 seconds. To explain this result, LUNA is routing network flows based on their classes but when the class is not provided (i.e., unknown user behavior), the flow is routed randomly. Thus, in its worst case (i.e., the case where it is unable to predict all flows' classes) LUNA deals like random routing strategy.

Fig. 3.5 depicts the throughput for the three routing strategies. The throughput is the amount of transferred data in a given time. It is clearly shows that LUNA can also improve the throughput. Almost 90% of the flows are transferred with a throughput greater than 81 kb/s. However, for random strategy, almost 90% of the flows are transferred with a throughput greater than 32 kb/s. To explain this results, contrary to random routing, LUNA is avoiding congestion by selecting least congested paths. The queuing delay is therefore minimized and the throughput is improved.

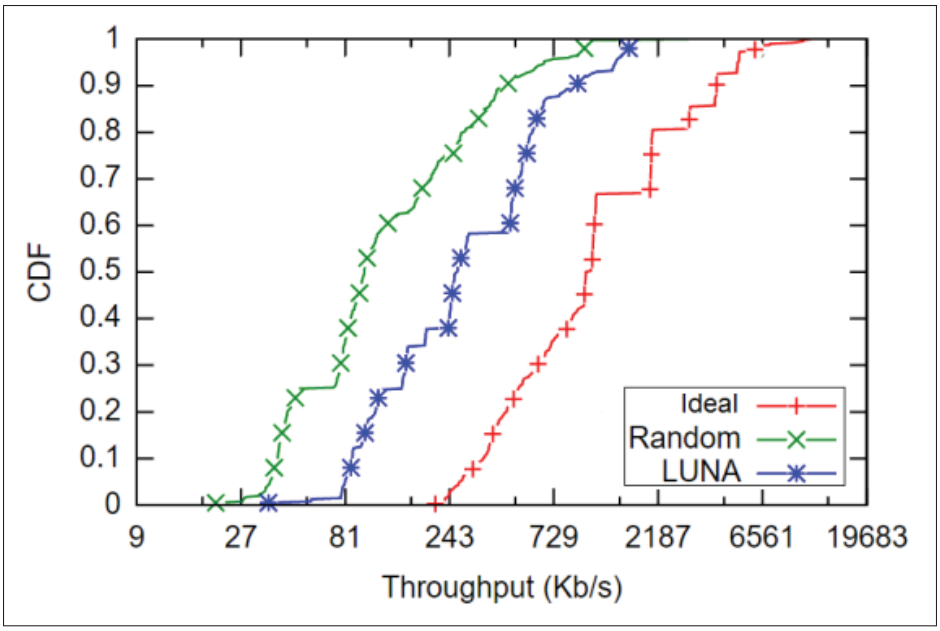

Figure 3.5 Throughput

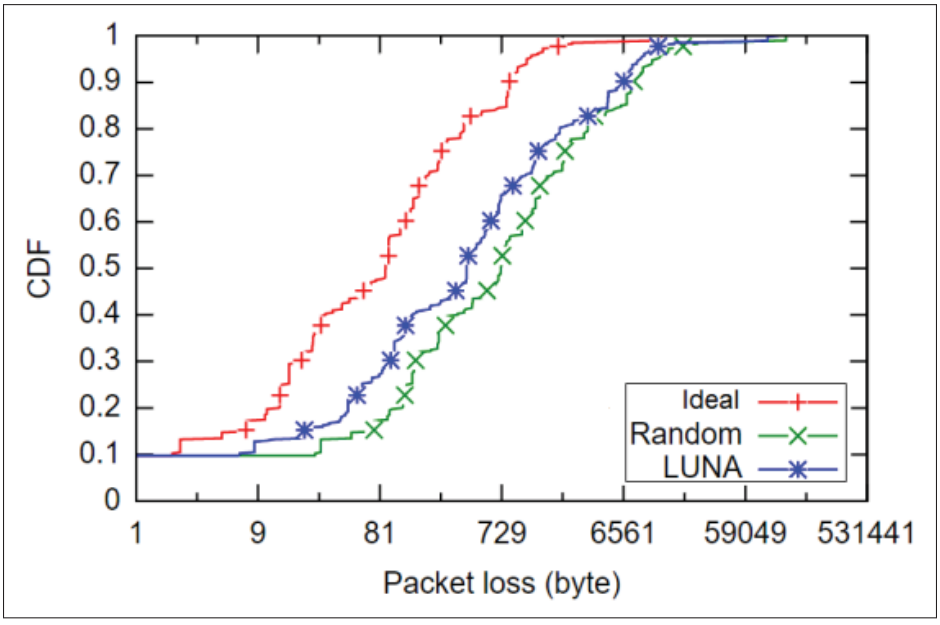

Figure 3.6 Packet loss

Fig. 3.6 compares the three strategies based on the amount of data which are lost. A packet loss occurs when a router is overloaded and cannot receive additional packets. This figure highlights

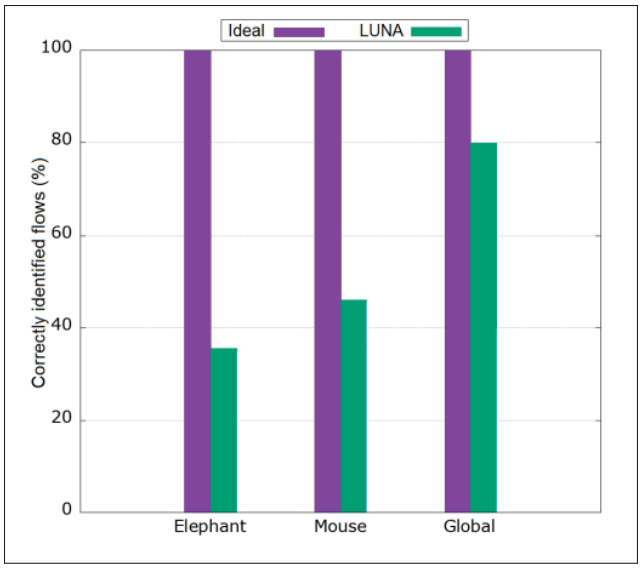

Figure 3.7 Percentage of correctly identified flows

the improvements of LUNA in term of packet loss compared to random. For instance, almost half of flows are transferred with amount of packet loss less than 421 bytes. However, for random strategy, almost half of flows are transferred with amount of packet loss less than 736 bytes.

In this work, we define the accuracy as the number of correctly identified flows (mouse or elephant) over the total number of flows which can be represented as follows:

$$Accuracy = \frac{Numberofcorrectlyidentifiedflows}{Totalnumberofflows} \tag{3.2}$$

Fig. 3.7 shows the percentage of correctly identified flows for the two routing strategies. LUNA has succeeded to identify almost 34% of elephant flows and almost 46% of mouse flows. The reason behind incorrectly identifying a user's flow class is due to the fact that there is no association rule for this user.

3.4 Conclusion

In this chapter, we focused on the problem of minimizing flow completion time in SDN networks. We hence proposed a predictive routing strategy called LUNA based on association rules mining. Unlike the existing solutions, LUNA is able to improve network performance by avoiding congestion. In particular, elephant flows are routed through least congested paths. On average, LUNA demonstrates high classification accuracy exceeding 80% for all flows. In particular, 34% of elephant flows and 46% of mouse flows. Experimental results show that LUNA outperforms the random solution and converges to the ideal routing in terms of the FCT, throughput and packet loss.

As future work we aim at optimizing the number of routing rules in large scale to save TCAM memory.

CHAPTER 4

ON MINIMIZING SEGMENT RETRANSMISSION DELAY

4.1 Introduction

With the emergence of new network applications like telesurgery, telepresence and virtual/augmented reality, minimizing the network delays becomes a pressing requirement Zhani & ElBakoury (2020). For instance, a telesurgery application would require stringent bounded end-to-end delays to ensure realistic, reliable and smooth operation Sedaghat & Jahangir (2021). Increased network delays and high transfer times could also lead to significant loss of revenue for several network applications. For instance, according to a study from Amazon, an increase of $100ms$ in file transfer time (e.g., web pages) could decrease the sales by up to 1% Yahyaoui *et al.* (2020).

One of the major reasons of high network delays is congestion which occurs when a network link receives more packets than it can handle Yahyaoui & Zhani (2020). This naturally leads to increased queuing delays and high rates of dropped packets Xu, Zhao & Muntean (2021). In today's Internet where the Transport Control Protocol (TCP) is the most used protocol to ensure the transport of packets between two end-points, lost packets are detected and retransmitted by the source Stevens *et al.*. More precisely, when the source transmits a packet, it launches a timer called the TCP timeout. When this timer expires before receiving an ACKnowledgement (ACK) from the destination, the source assumes that the packet is lost and retransmits it. This retransmission mechanism is suffering from several shortcomings. First, TCP is not able to accurately detect packet loss as the TCP timeout may expires before the acknowledgement is received.

However, this mechanism suffers from several shortcomings. First, the detection of packet loss is not accurate as the TCP timeout may expire before the acknowledgement is received. Indeed, the packet or its acknowledgement may experience some additional delays (for instance, because of congestion or routing through different paths). In this case, the packet is not lost but the timeout expires, leading to unneeded retransmissions.

Another major shortcoming of TCP is that, when the packet is really lost, the retransmission is only triggered when the TCP timeout expires and the retransmission is only performed by the source. This means that the total time needed to deliver the packet, including the retransmission time, is estimated to be three times the end-to-end delay (two end-to-end delays for the timeout and one end-to-end delay for the retransmitted packet). Of course, this time could be much higher when the same packet experiences several losses.

In this chapter, we address the problem of minimizing TCP retransmission delay and we propose a novel network function called Transport Assistant (TA) which can be deployed within the path connecting the source to the destination. The role of the TA is to cache (i.e., keep a copy of the transmitted packets), detect packet loss and retransmit lost packets so as to minimize the total packet delivery time (which includes packet retransmission time). Several instances of the TA could be created between the source and the destination and could cooperate to achieve the sought-after objectives.

As advocated in Zhani & ElBakoury (2020); Bueno *et al.* (2022), future networks are expected to be softwarized and fully programmable, which makes it possible to implement and dynamically instantiate advanced network functions like the Transport Assistant function proposed in this work.

In this chapter, our contribution aims at designing and evaluating the performance of the TA that should be able to perform the following operations:

- *Packet caching*: the TA should be able to smartly cache packets in order to use this cache to retransmit the packets in case of failure. As there might be several instances of the TA within the path, the caching strategy should optimize the usage of the available caches and avoid caching the same packets multiple times.
- *Packet loss detection and retransmission*: the TA should be able to detect lost packets and locate as accurate as possible the location where the loss has occurred and then request to retransmit the lost packets from the nearest TA in order to minimize retransmission time.

The remainder of this chapter is organized as follows. Section 4.2 details the proposed solution. Section 4.3 presents the experimental results. Finally, Section 4.4 concludes the chapter.

4.2 Proposed Solution

In this section, we introduce the proposed network function called Transport Assistant (TA). The TA aims at improving TCP retransmission mechanism by accurately detecting lost packets and minimizing their retransmission delay. The TA encompasses 1) a packet caching mechanism which caches packets based on the type of the flow (i.e., critical, non-critical) and ensures that packets are not cached multiple times in different TA instances, 2) a packet loss detection mechanism which monitors the traffic and detects lost packets, and 3) a retransmission mechanism which enables the TA instances throughout the path to communicate with each other in order to locate and retransmit a copy of the lost packet.

Several challenges pertaining to the deployment of TA need to be addressed. For instance, packet caching, loss detection and retransmission strategies should be carefully designed in order to minimize the size of the cache and reduce the number of unneeded retransmissions. In the following, we detail how these challenges are addressed in this work.

4.2.1 Packet caching

One of the main limitations of packet caching mechanisms in the literature is that packets are cached blindly, without considering their types (i.e., critical, non-critical). In addition, caching all packets could quickly overload the cache and hence, many lost packets would not have any cached copies. To solve these limitations, the proposed TA considers two types of flows and caches first packets of critical flows. Packets of non-critical flows are only cached when there is room and could be dropped anytime if needed to make room for packets of critical flows. Furthermore, when a TA instance decides to cache a packet, it marks it with a special flag (1-bit) so that next TA instances do not cache the packet again. This allows to efficiently use the resources allocated to the caches of the different TA by avoiding caching the same packets in multiple TAs.

4.2.2 Packet loss detection

Most of the existing packet loss detection strategies are relying either on the time interval of consecutive packets or on the packet order to assume a lost packet. The major shortcoming of these strategies is that delayed packets could be considered as lost which results in increased unneeded packet retransmissions. To solve this issue, we first force the TCP packets of the same connection to follow the same path in order to avoid packet disorder. A TA instance detects then the existence of missing packets when the sequence number (the one used by TCP) of the received packet is not the expected sequence number. In this case, the TA sends an explicit request to the prior TA instance asking for the missing packets. When such a request is received by a TA instance, it retransmits the requested packets if they are available in its cache otherwise forwards the request backwards to the prior TA. This is repeated until one of the prior TAs retransmits the missing packets or the request reaches the source.

It is worth noting that, when the TA is deployed, the traditional TCP packet loss detection and retransmission mechanism is disabled in order to avoid unneeded retransmissions.

4.3 Evaluation

In this Section, we evaluate the advantage of using the TA through extensive experiments using different scenarios to clearly highlight the benefit provided by the TA compared to the baseline (i.e., standard TCP).

4.3.1 Experimental Environment

To show the benefits of using the TA, we used the Mininet Emulator Mininet (2018) in which we integrated the Transport Assistant function that we developed in Python.

In our experiments, we considered four scenarios as described in Fig. 4.1a, 4.1b, 4.1c and 4.1d. The first scenario is the Baseline (Fig. 4.1a) as there are no TAs deployed and standard TCP implementation is used. The considered topology is shown in the figure with two sources

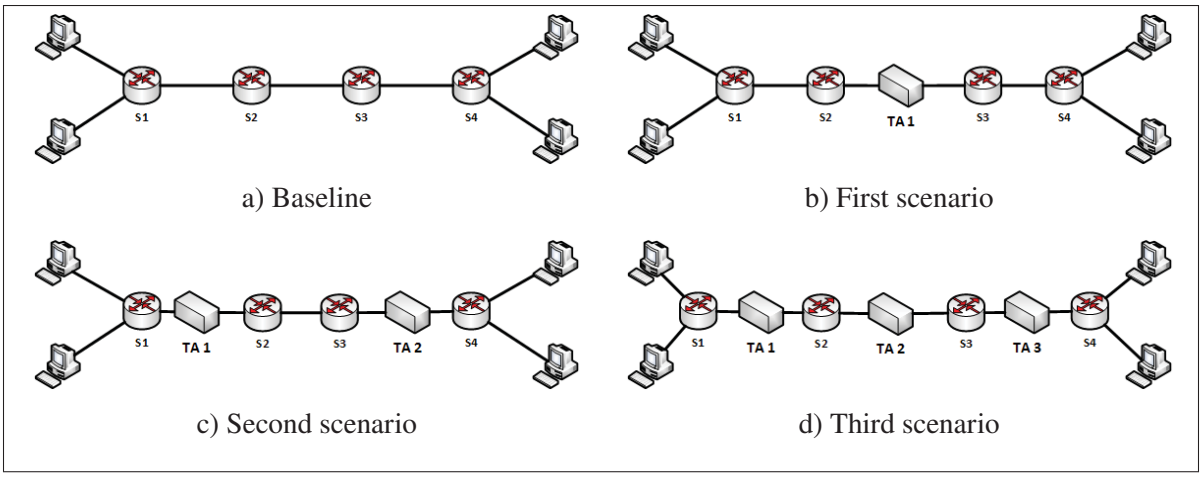

Figure 4.1 Experimental infrastructure

of traffic flows (i.e., the two hosts on the left side of the figure) sending traffic through the same path towards two destinations (i.e., the two hosts on the right side of the figure). This path is composed of four switches connected using links with capacities and delays randomly generated between 10Mbps to 70Mbps and between 10ms to 100ms, respectively. The sources are sending 100 traffic flows where 30% of them are critical and 70% are non-critical. The size of the flows was randomly generated between 1Mb and 5Gb.

In the other three scenarios (Fig. 4.1b, 4.1c and 4.1d), we considered the same topology used in the Baseline (i.e., with the same propagation delays and bandwidth between switches); However, we incorporated 1, 2 or 3 TA instances within the path towards the destinations as shown in the figures. In these scenarios, the sources are sending the same flows considered in the Baseline with exactly the same characteristics (i.e., starting time, size, source, destination, type). Furthermore, as the TA is used in these three scenarios, the standard TCP packet loss detection and retransmission mechanisms are disabled as the TAs ensure packet retransmission as described above.

We ran the experiments for the four considered scenarios and we evaluated, for each of them, the obtained performance using several metrics including the flow completion time which is the time needed to transfer all the packets of the flow, the average segment transmission time (which

includes retransmission time), the number of lost packets, and the number of retransmitted packets from the source.

4.3.2 Experimental Results

Fig. 4.2 depicts the flow completion time obtained with the four compared scenarios. It clearly shows that all the scenarios containing at least one TA outperform standard TCP. As a matter of fact, TCP retransmits lost packets from the source after a timeout which significantly increases the flow completion time. Unlike TCP, the Transport Assistant retransmits lost packets from its cache or request them from another TA instance. As a result, the number of hops crossed by a retransmitted packet from a TA is smaller than the number of hops crossed by a retransmitted packet from the source (the case of TCP). In addition, the figure show that flow completion time is further reduced when increasing the number of TA instances between the source and the destination. This is expected as, the more TA instances there are, the more cache is available, which means there is less probability to resort to the source to retransmit the lost packet and packets will be retransmitted from one of the TAs.

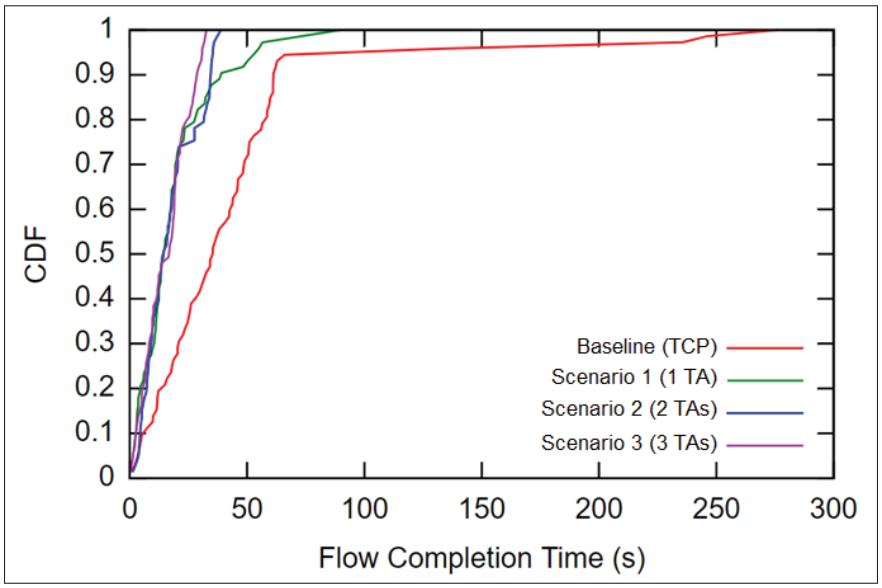

Figure 4.2 Global Flow completion time

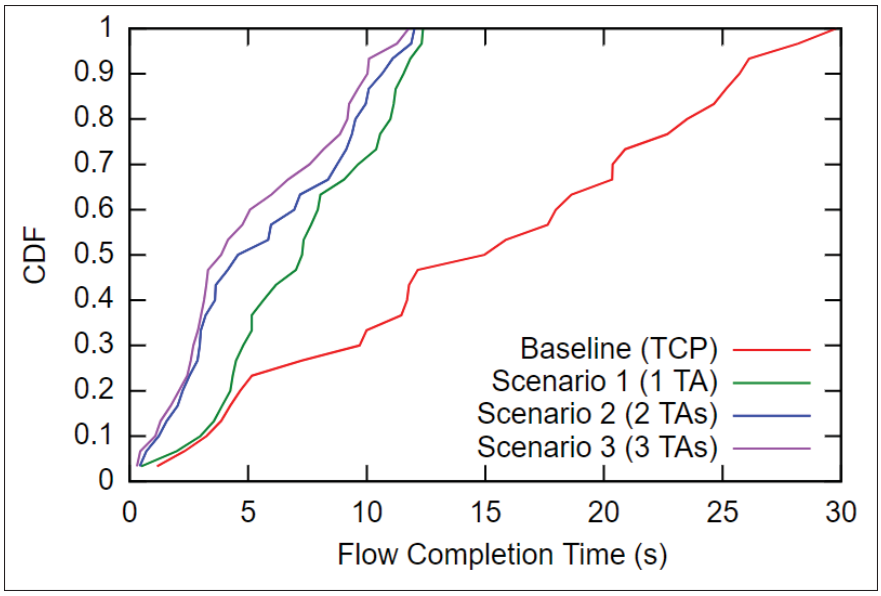

Figure 4.3 Flow completion time for critical flows

Fig. 4.4 illustrates the average segment transmission time (including retransmission time) for all the considered scenarios. As shown in the figure, all the scenarios using the TA provide better results than the baseline (i.e., standard TCP). Unlike standard TCP, lost segments are retransmitted from the TA instances rather than the source and this significantly reduces the average segment transmission time compared to TCP. The segment transmission time is naturally further reduced when more TAs are deployed.

Figures 4.3 and 4.5 show respectively the flow completion time and the average segment transmission time for critical flows. Unlike standard TCP, which retranmits the packets blindly without considering their types (i.e., critical or non-critical), the TA prioritizes critical packets. This clearly shows that using the TA helps to prioritize critical flows and significantly minimizes their retransmission time compared to that obtained with TCP.

Fig. 4.6 shows the amount of lost packets in all the studied scenarios. It clearly shows that, even with a single TA, the use of the TA provides better results than standard TCP and reduces the number of lost packets.

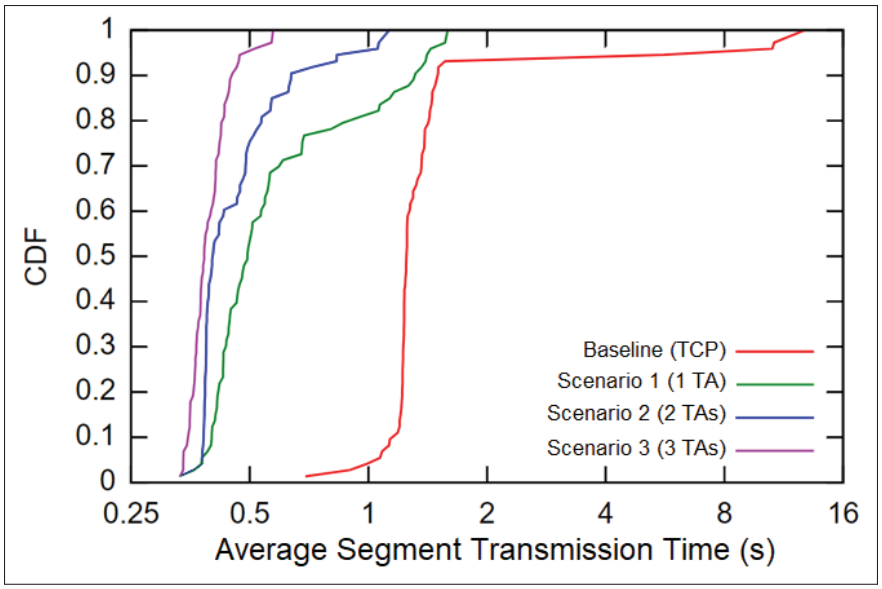

Figure 4.4 Global average packet transmission time

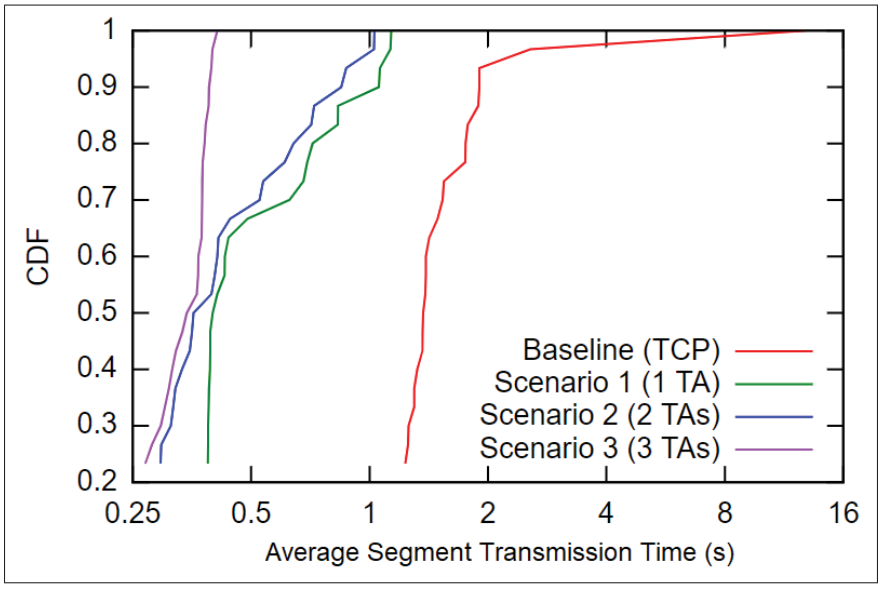

Figure 4.5 Average packet transmission time for critical flows

Fig. 4.8 depicts the amount of retransmitted packets from the source. As expected, the TA significantly reduces the number of packets retransmitted from the source compared to TCP. Moreover, by increasing the number of TA instances, the amount of retransmitted packets from

the source is further minimized as packet loss detection and retransmission are handled by more TA instances.

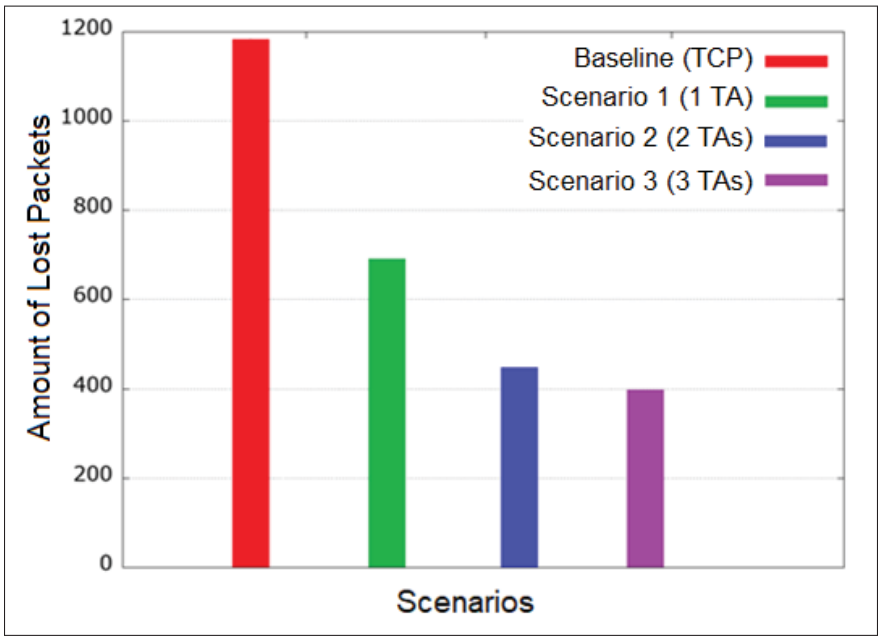

Figure 4.6 Global lost packets

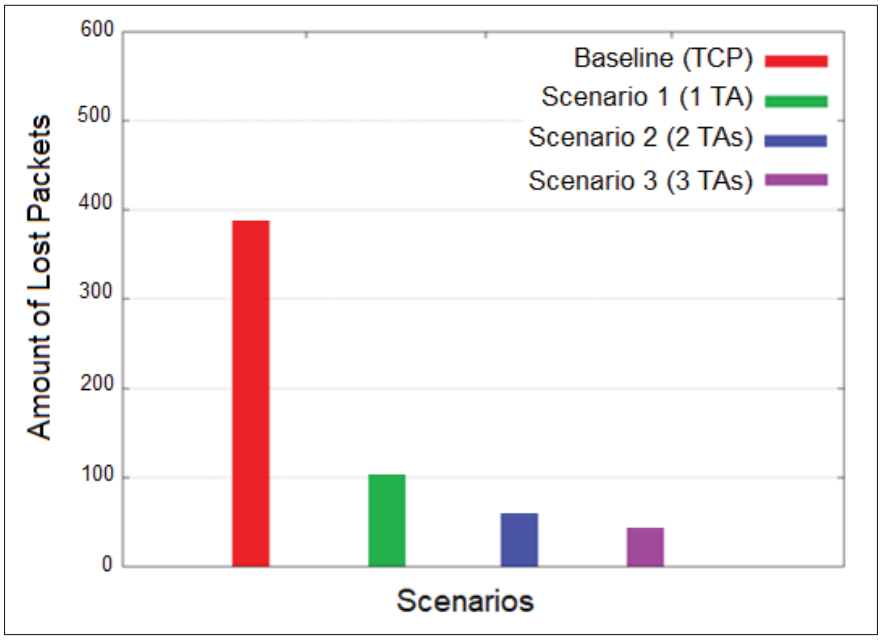

Figure 4.7 Lost packets of critical flows

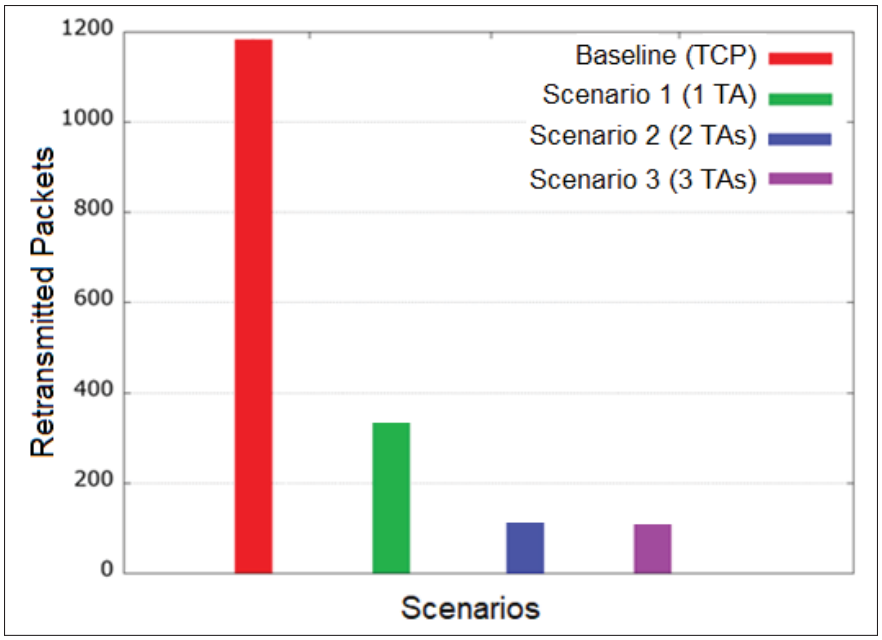

Figure 4.8 Retransmitted packets from the source of all the flows

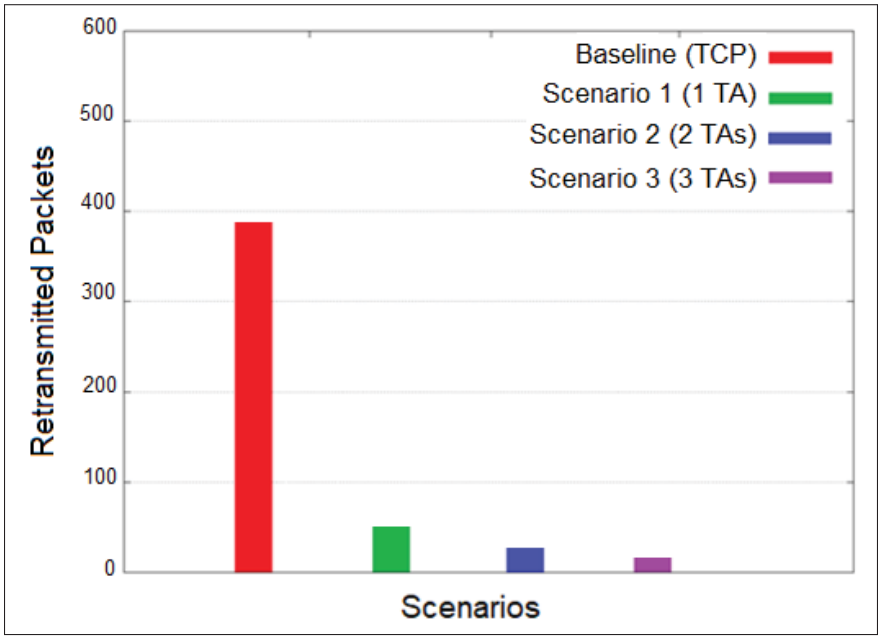

Figure 4.9 Retransmitted packets from the source of critical flows

Figures 4.7 and 4.9 show respectively the amounts of lost packets and the amount of retransmitted packets from the source for only the critical flows. It is clearly shown that the TA significantly reduces both metrics compared to standard TCP as critical packets are given more priority in the cache of the TA instances than non-critical packets and hence they are served first.

4.4 Conclusion

In this chapter, we addressed the problem of minimizing the TCP retransmission delay and we proposed a novel network function called Transport Assistant (TA) that could be integrated within the network with the goal of minimizing packet retransmission delays compared to TCP and allowing the possibility to ensure more priority to critical flows over the non-critical ones. The proposed solution considers the deployment of several TA instances that can smartly cooperate in order to ensure packet caching as well as efficiently detecting and retransmitting lost packets so as to achieve the sought-after objectives.

The performed experiments using Mininet show that, compared to the standard TCP, the use of the Transport Assistant significantly reduces average packet transmission time (which includes retransmission time), flow completion time, the number of lost packets and the number of packets retransmitted from the source.

As a future work, we aim at further refining the TA caching, packet detection and retransmission mechanisms in order to further optimize the use of the cache and reduce packet transmission time.

CONCLUSION AND RECOMMENDATIONS

5.1 Thesis Summary

In this thesis, we have presented three objectives aiming at enabling today's network infrastructures to provide the required performance (i.e., latency, bandwidth and packet loss) of futuristic network applications.

In chapter 1, we minimized the monitoring cost (expressed in bandwidth consumption) while ensuring a minimal reporting delay by proposing two heuristic algorithms: LCM and LCM-M for single and multiple controller infrastructures, respectively. Extensive simulations show that both heuristics significantly minimize the monitoring cost. They provide near optimal solution with minimal computation time compared to CPLEX.

In chapter 2, we improved the network performance (i.e., delay, throughput and packet loss) by introducing LUNA, a novel routing strategy that understands the user behavior and extracts the size of frequently requested flows thanks to the association rules mining technique to compute routing rules. Extensive simulations show that LUNA significantly minimizes both delay and packet loss as well as maximises the network throughput.

In chapter 3, we solved the problem of increased retransmission delay in TCP. We proposed a novel network function called Transport Assistant that is able to cache, detect and retransmit lost packets. As a result, instead of retransmitting lost packets from the source which in fact costs too much delay, the transport assistant that is placed in the service function chain will take in charge the packet retransmission. Extensive simulations show that the more we add transport assistant instances to the service function chain, the more the network performance are improved.

5.2 Future Research Directions

Minimize monitoring cost: the number of monitored flows per controller is crucial for high performance networks. When the controller monitors multiple flows, its processing delay could increase due to the huge number of monitoring messages frequently exchanged. Therefore, all the traffic management decisions taken by the controller could be untimely, which could result in poor management decisions. That is why we plan to introduce a novel network function which minimizes the monitoring messages transmitted from the switch to the controller. It collects them and take decisions without interacting with the controller.

Improve network performance with a novel routing strategy: LUNA routing strategy computes the routing rules based on the user behavior then install them in the switches (exactely in TCAM memory). However, TCAM memory is limited and efficiently leveraging this memory is a must to achieve high network performance. That's why the routing strategy should consider the number of generated rules, their timeouts and priorities. We plan to extend the idea of routing the traffic based on the user behavior in order to effectively leverage the available resources.

Minimize TCP retransmission delay: selecting which kind of packets to prioritize when caching and retransmitting lost packets without any interaction with the network operator is challenging. For instance, the Transport Assistant is not able to identify packets of critical applications which represents an issue that should be solved. Therefore, we plan to further extend the idea of Transport Assistant in order to make it able to take efficient decisions independently.

APPENDIX

List of publications:

- Yahyaoui, Haythem, Mohamed Faten Zhani, Ouns Bouachir and Moayad Aloqaily. "On Minimizing Flow Monitoring Costs in Large-Scale SDN Networks." In *the International Journal of Network Management*. **Under review**.
- Yahyaoui, Haythem, Melek Majdoub, Mohamed Faten Zhani, and Moayad Aloqaily. "On Minimizing TCP Retransmission Delay in Softwarized Networks." In *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pp. 1-6. IEEE, 2022.
- Yahyaoui, Haythem, and Mohamed Faten Zhani. "On providing low-cost flow monitoring for SDN networks." In *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, pp. 1-6. IEEE, 2020.
- Yahyaoui, Haythem, Saifeddine Aidi, and Mohamed Faten Zhani. "On using flow classification to optimize traffic routing in SDN networks." *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2020.

BIBLIOGRAPHY

(2015). The CPLEX Optimizer. Retrieved on 2020-02-02 from: https://www.ibm.com/ca-fr/analytics.

(2018). The expected growth in the network Monitoring market. Retrieved on 2020-03-02 from: https://www.marketsandmarkets.com/Market-Reports/network-monitoring-market-51888593.html.

(2018). OpenFlow protocol. Retrieved on 2020-02-19 from: https://www.opennetworking.org/wp-content/uploads/2013/04/openflow-spec-v1.0.0.pdf.

Agrawal, R., Imieliński, T. & Swami, A. (1993). Mining association rules between sets of items in large databases. *ACM SIGMOD international conference on Management of data*, pp. 207-216.

Akyildiz, I. F., Lee, A., Wang, P., Luo, M. & Chou, W. (2014). A roadmap for traffic engineering in SDN-OpenFlow networks. *Elsevier computer Networks*, 71, 1-30.

Al-Fares, M., Radhakrishnan, S., Raghavan, B., Huang, N., Vahdat, A. et al. (2010). Hedera: dynamic flow scheduling for data center networks. *Nsdi*, 10(8), 89-92.

Baik, S., Lim, Y., Kim, J. & Lee, Y. (2015). Adaptive flow monitoring in SDN architecture. *Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pp. 468-470.

Berde, P., Gerola, M., Hart, J., Higuchi, Y., Kobayashi, M., Koide, T., Lantz, B., O'Connor, B., Radoslavov, P., Snow, W. et al. (2014). ONOS: towards an open, distributed SDN OS. *Proceedings of the third workshop on Hot topics in software defined networking*, pp. 1-6.

Bueno, G., Saquetti, M., Rodrigues, P., Lamb, I., Gaspary, L., Luizelli, M. C., Zhani, M. F., Azambuja, J. R. & Cordeiro, W. (2022). Managing Virtual Programmable Switches: Principles, Requirements, and Design Directions. *IEEE Communications Magazine*, 60(2), 53-59.

Chao, S.-C., Lin, K. C.-J. & Chen, M.-S. (2016). Flow classification for software-defined data centers using stream mining. *IEEE Transactions on Services Computing*, 12(1), 105-116.

Chen, J., Yan, S., Ye, Q., Quan, W., Do, P. T., Zhuang, W., Shen, X. S., Li, X. & Rao, J. (2019a). An SDN-based transmission protocol with in-path packet caching and retransmission. *International Conference on Communications (ICC)*, pp. 1-6.

Chen, J., Ye, Q., Quan, W., Yan, S., Do, P. T., Zhuang, W., Shen, X. S., Li, X. & Rao, J. (2019b). SDATP: An SDN-based adaptive transmission protocol for time-critical services. *IEEE Network*, 34(3), 154-162.

Chowdhury, S. R., Bari, M. F., Ahmed, R. & Boutaba, R. (2014). Payless: A low cost network monitoring framework for software defined networks. *IEEE Network Operations and Management Symposium (NOMS)*, pp. 1-9.

ClarkNet. (2018). ClarkNet HTTP requests. Retrieved on 2019-10-29 from: http://ita.ee.lbl.gov/html/contrib/ClarkNet-HTTP.html.

Clemm, A., Zhani, M. F. & Boutaba, R. (2020). Network management 2030: Operations and control of network 2030 services. *Journal of Network and Systems Management*, 1-30.

Curtis, A. R., Kim, W. & Yalagandula, P. (2011). Mahout: Low-overhead datacenter traffic management using end-host-based elephant detection. *IEEE International Conference on Computer Communications (INFOCOM)*, pp. 1629-1637.

Dieye, M., Zhani, M. F. & Elbiaze, H. (2017). On achieving high data availability in heterogeneous cloud storage systems. *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 326-334.

Dridi, L. & Zhani, M. F. (2016). SDN-guard: DoS attacks mitigation in SDN networks. *IEEE International Conference on Cloud Networking (Cloudnet)*, pp. 212-217.

Geant. (2018). Geant Topology. Retrieved on 2019-10-29 from: https://geant3plus.archive.geant.net/home.aspx.

Grover, N., Agarwal, N. & Kataoka, K. (2015). liteflow: Lightweight and distributed flow monitoring platform for SDN. *Proceedings of the IEEE Conference on Network Softwarization (NetSoft)*, pp. 1-9.

Hark, R., Tounsi, K., Rizk, A. & Steinmetz, R. (2019). Decentralized Collaborative Flow Monitoring in Distributed SDN Control-Planes. *International Conference on Networked Systems (NetSys)*, pp. 1-8.

He, Q., Wang, X. & Huang, M. (2018). OpenFlow-based low-overhead and high-accuracy SDN measurement framework. *Transactions on Emerging Telecommunications Technologies*, 29(2), e3263.

Henni, D.-E., Hadjaj-Aoul, Y. & Ghomari, A. (2016). Probe-SDN: A smart monitoring framework for SDN-based networks. *Global Information Infrastructure and Networking Symposium (GIIS)*, pp. 1-6.

Hopps, C. et al. (2000). *Analysis of an equal-cost multi-path algorithm*. RFC 2992.

Jain, A. K. (2010). Data clustering: 50 years beyond K-means. *Elsevier Pattern recognition letters*, 31(8), 651-666.

Kreutz, D., Ramos, F. M., Verissimo, P. E., Rothenberg, C. E., Azodolmolky, S. & Uhlig, S. (2014). Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1), 14-76.

Li, Q., Zou, X., Huang, Q., Zheng, J. & Lee, P. P. (2018). Dynamic packet forwarding verification in SDN. *IEEE Transactions on Dependable and Secure Computing*, 16(6), 915-929.

Linden, G. (2006). Make data useful. Stanford CS345 Talk.

Liu, J., Li, J., Shou, G., Hu, Y., Guo, Z. & Dai, W. (2014). SDN based load balancing mechanism for elephant flow in data center networks. *IEEE International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pp. 486-490.

Liu, Y., Yang, D., Gong, K. & Ren, J. (2021). A routing strategy with adaptive weight between 'hop'and 'bandwidth'. *IEEE 5th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC)*, pp. 1-6.

Markets & Markets. (2018). Cloud storage market. Retrieved on 2019-10-01 from: https://www.marketsandmarkets.com/cloud-storage.asp.

Mininet. (2018). Mininet website. Retrieved on 2019-10-01 from: http://mininet.org/.

Nunes, B. A. A., Mendonca, M., Nguyen, X.-N., Obraczka, K. & Turletti, T. (2014). A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications surveys & tutorials*, 16(3), 1617-1634.

Postel, J. et al. (1981). Transmission control protocol. *STD 7, RFC 793*.

Poupart, P., Chen, Z., Jaini, P., Fung, F., Susanto, H., Geng, Y., Chen, L., Chen, K. & Jin, H. (2016). Online flow size prediction for improved network routing. *IEEE International Conference on Network Protocols (ICNP)*, pp. 1-6.

Sedaghat, S. & Jahangir, A. H. (2021). RT-TelSurg: Real Time Telesurgery Using SDN, Fog, and Cloud as Infrastructures. *IEEE Access*, 9, 52238-52251.

Stevens, W. et al. TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms. *rfc 2001, January*.

Su, Z., Wang, T., Xia, Y. & Hamdi, M. (2014). FlowCover: Low-cost flow monitoring scheme in software defined networks. *IEEE Global Communications Conference (GLOBECOM)*, pp. 1956-1961.

Su, Z., Wang, T., Xia, Y. & Hamdi, M. (2015). CeMon: A cost-effective flow monitoring system in software defined networks. *Elsevier computer Networks*, 92, 101-115.

Sugimoto, S. & Ito, Y. (2021). Proposal of Adaptive TCP Multi-Pathization Method with SDN. *IEEE 10th Global Conference on Consumer Electronics (GCCE)*, pp. 635-636.

Sun, W., Wang, Z. & Zhang, G. (2021). A QoS-guaranteed intelligent routing mechanism in software-defined networks. *Elsevier Computer Networks*, 185(2), 107709.

Tang, F., Shojaee, M. & Haque, I. (2021). ACE: an Accurate and Cost-Effective Measurement System in SDN. *arXiv preprint arXiv:2108.12849*, pp. 1-6.

Tlili, G., Yahyaoui, H., Zhani, M. F. & Elbiaze, H. (2019). Daresch: deadline-aware request scheduling for cloud storage services. *Annals of Telecommunications*, 74(9), 545-557.

Tootoonchian, A., Ghobadi, M. & Ganjali, Y. (2010). OpenTM: traffic matrix estimator for OpenFlow networks. *International Conference on Passive and Active Network Measurement*, pp. 201-210.

Van Adrichem, N. L., Doerr, C. & Kuipers, F. A. (2014). Opennetmon: Network monitoring in openflow software-defined networks. *IEEE Network Operations and Management Symposium (NOMS)*, pp. 1-8.

Vazirani, V. V. (2001). Approximation algorithms. *Springer*.

Wan, C.-Y., Campbell, A. T. & Krishnamurthy, L. (2002). PSFQ: a reliable transport protocol for wireless sensor networks. *ACM international workshop on Wireless sensor networks and applications*, pp. 1-11.

Wang, M.-H., Chen, L.-W., Chi, P.-W. & Lei, C.-L. (2017). SDUDP: A reliable UDP-Based transmission protocol over SDN. *IEEE Access*, 5, 5904-5916.

Xia, W., Wen, Y., Foh, C. H., Niyato, D. & Xie, H. (2014). A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1), 27-51.

Xu, C., Zhao, J. & Muntean, G.-M. (2021). Congestion control design for multipath transport protocols: A survey. *IEEE communications surveys & tutorials*, 18(4), 2948-2969.

Xu, H., Yu, Z., Qian, C., Li, X.-Y. & Liu, Z. (2017). Minimizing flow statistics collection cost of SDN using wildcard requests. *IEEE Conference on Computer Communications (INFOCOM)*, pp. 1-9.

Yahyaoui, H. & Zhani, M. F. (2020). On providing low-cost flow monitoring for SDN networks. *IEEE International Conference on Cloud Networking (CloudNet)*, pp. 1-6.

Yahyaoui, H., Aidi, S. & Zhani, M. F. (2020). On using flow classification to optimize traffic routing in SDN networks. *IEEE Annual Consumer Communications & Networking Conference (CCNC)*, pp. 1-6.

Yang, X., Han, B., Sun, Z. & Huang, J. (2017). SDN-based DDoS attack detection with cross-plane collaboration and lightweight flow monitoring. *IEEE Global Communications Conference (GLOBCOM)*, pp. 1-6.

Yang, Z. & Yeung, K. L. (2017). An efficient flow monitoring scheme for SDN networks. *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 1-4.

Yang, Z. & Yeung, K. L. (2020). Flow monitoring scheme design in SDN. *Computer Networks*, 167, 107007.

Yu, C., Lumezanu, C., Zhang, Y., Singh, V., Jiang, G. & Madhyastha, H. V. (2013). Flowsense: Monitoring network utilization with zero measurement cost. *International Conference on Passive and Active Network Measurement*, pp. 31-41.

Zhani, M. F. & ElBakoury, H. (2020). FlexNGIA: A flexible Internet architecture for the next-generation tactile Internet. *Journal of Network and Systems Management*, 1-45.

Zhani, M. F., Zhang, Q., Simona, G. & Boutaba, R. (2013). VDC planner: Dynamic migration-aware virtual data center embedding for clouds. *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pp. 18-25.