An adaptive system for troubleshooting network issues in the context of encrypted data traffic

Van-Van Tong

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL VAL DE MARNE
Connaissance - Action

École doctorale Mathématiques et STIC

Mathématiques et Sciences et
Technologies de l'Information et de la Communication

Université Paris-Est Créteil

THÈSE

Pour l'obtention du grade de

Docteur de l'Université Paris-Est Créteil

Spécialité: "Informatique, Génie Informatique, Réseaux"

Présentée par

Van Van TONG

Un système adaptatif pour l'autodiagnostic de pannes dans les réseaux de communication dans un contexte de trafic chiffré de données

Directeur de thèse : Prof. **Abdelhamid MELLOUK**, Université Paris-Est Créteil (UPEC), France

Encadrant de thèse : **Sami SOUIHI**
Maître de conférences, Université Paris-Est Créteil (UPEC), France

Encadrant de thèse : **Hai Anh TRAN**
Maître de conférences, Institut Polytechnique de Hanoï (IPH), Vietnam

Soutenue publiquement

le 13 Décembre 2021

Devant le jury d'examen composé de:

Jean-philippe Georges,	Rapporteur	Professeur, Université de Lorraine, France
Farid Naït-Abdesselam,	Rapporteur	Professeur, Université Paris Descartes, France
Houda Labiod,	Examinatrice	Professeur, Télecom ParisTech, France
Abderrahim Benslimane,	Examinateur	Professeur, Université d'Avignon, France
Djamal Zaghlache,	Examinateur	Professeur, Télécom SudParis, France

LiSSiX **Laboratoire Images, Signaux et Systèmes Intelligents**

TINC Net **Équipe de contrôle intelligent dans les réseaux**

THESE

UPEC
UNIVERSITÉ PARIS-EST CRÉTEIL VAL DE MARNE
Connaissance - Action

MSTIC doctoral school

Mathématiques et Sciences et
Technologies de l'Information et de la Communication

University of Paris-Est Créteil

DISSERTATION

Submitted in partial fulfillment of the requirements for the Degree of

Doctor of Philosophy of Paris-Est University

in "Computer Science, Computer Networking" Specialization

Presented by

Van Van TONG

An adaptive system for troubleshooting network issues in the context of encrypted data traffic

PHD Thesis Advisor: **Prof. Abdelhamid MELLOUK**, University of Paris-Est Créteil (UPEC), France

PHD Thesis Supervisor: **Sami SOUIHI**
Associate Professor, University of Paris-Est Créteil (UPEC), France

PHD Thesis Supervisor: **Hai Anh TRAN**
Associate Professor, Hanoi University of Science and Technology (HUST), Vietnam

Defended publicly

on December 13, 2021

In front of a jury composed of:

Jean-philippe Georges,	Reviewer	Full Professor, University of Lorraine, France
Farid Naït-Abdesselam,	Reviewer	Full Professor, Paris Descartes University, France
Houda Labiod,	Examiner	Full Professor, Telecom Paris, France
Abderrahim Benslimane,	Examiner	Full Professor, Avignon University, France
Djamal Zaghlache,	Examiner	Full Professor, Telecom SudParis, France

LiSSiX **Laboratory of Images, Signals and Intelligent Systems**

TINC Net **Team of Intelligent Control for Networks**

THESIS

Résumé

Le réseau Internet devient de plus en plus complexe en raison du nombre croissant de périphériques réseau, des divers services multimédias et de la prévalence du trafic chiffré. Les solutions conventionnelles sont effectivement ingérables en raison des problèmes d'évolutivité, de la complexité temporelle élevée, du trafic chiffré, etc. Par conséquent, dans ce contexte, nous proposons une nouvelle architecture de dépannage efficace pour surmonter les limitations liées au trafic chiffré et à la complexité temporelle élevée. Cette architecture contient cinq modules principaux : une collecte de données, une détection d'anomalies, une remédiation temporaire, une analyse des causes profondes et une remédiation définitive. Dans la collecte de données, il y a deux sous-modules : la mesure des paramètres et la classification du trafic. Le sous-module de mesure des paramètres permet de collecter des paramètres de réseau en série temporelle pour les modules suivants. Selon ce sous-module, nous construisons et fournissons publiquement un ensemble de données de dépannage. En plus des paramètres du réseau, nous proposons une nouvelle approche de classification du trafic pour identifier les classes d'applications dans le contexte du trafic chiffré. Les paramètres du réseau sont analysés pour identifier automatiquement les anomalies du réseau et déclencher les modules de remédiation: i) Le module de remédiation temporaire qui vise à réduire les impacts négatifs des anomalies. Dans ce module, nous proposons un routage par segments basé sur la qualité d'expérience (QoE) des applications en utilisant l'apprentissage par renforcement pour sélectionner les chemins de routage appropriés correspondant à chaque application et répondre aux exigences strictes des accords de niveau de service (SLA), ii) Les modules d'analyse des causes profondes et de remédiation. Nous considérons la congestion comme un cas d'utilisation pour générer des anomalies et prendre en compte ses causes profondes, notamment la défaillance du lien, la défaillance du commutateur et la surcharge de la mémoire tampon. Cette architecture est mise en œuvre et validée dans des environnements SDN (Software-Defined Networking) en utilisant les contrôleurs d'ONOS.

Abstract

Nowadays, the Internet network is becoming more and more complex due to an ever-increasing number of network devices, various multimedia services and a prevalence of encrypted traffic. Conventional solutions are effectively unmanageable because of scalability issues, high time complexity, encrypted traffic and so on. Therefore, in this context, we propose a novel efficient troubleshooting architecture to overcome limitations related to encrypted traffic and high time complexity. This architecture contains five main modules: data collection, anomaly detection, temporary remediation, root cause analysis and definitive remediation. In data collection, there are two submodules: parameter measurement and traffic classification. The parameter measurement submodule is to collect time-series network parameters for other modules. According to this submodule, we build and publically provide a troubleshooting dataset for network troubleshooting. In addition to the network parameters, we propose a novel traffic classification approach to identify application classes in the context of encrypted traffic. The network parameters are analyzed to automatically identify the network anomalies and trigger remediation modules: i) the temporary remediation module which aims to reduce negative impacts of anomalies and guarantee the availability in the network. In this module, we propose an application-aware QoE (Quality of Experience)-based segment routing using reinforcement learning to select appropriate routing paths corresponding to each application and meet strict SLA (Service-level Agreement) requirements; ii) the root cause analysis and definitive remediation modules. We consider congestion as an use case to generate anomalies and take into account its root causes including link failure, switch failure and buffer overload. This architecture is implemented and validated in SDN (Software-Defined Networking) environments using ONOS controllers.

Acknowledgments

I am deeply indebted to Prof. Abdelhamid MELLOUK, Dr. Sami SOUIHI and Dr. Hai Anh TRAN for their guidance and advice from the beginning of my research at the University of Paris-Est Créteil until today. They were guiding me every single step of the PhD and supporting me to complete this thesis. They gave me opportunities to think outside the box, determine my research directions, and make sure that I was in the right direction. As English is my second language, I would like to thank them for their patience in publication and manuscript corrections. Furthermore, I feel honored for the great opportunity I had to work with them.

I would like to thank all colleagues in the TINCNET/LISSI/UPEC research team including Dr. Thiago Abreu, Dr. Said HOCEINI, Dr. Mohamed Aymen LABIOD and Dr. Brice AUGUSTIN. They are helping me in many administrative and teaching tasks and providing me with an environment where I can concentrate on my research work. Moreover, one of the valuable gifts of my life at UPEC is a group of friends: Lamine AMOUR, Fetia BANNOUR, Tran-Tuan CHU, Abhishek DJEACHANDRANE and the others. Their support over the years in all the ups and downs are immense comfort.

Last but not least, I would like to thank my parents for their endless love and encouragement. My journey would not be possible without their sacrifices. Their unconditional love and Skype call encouraged me to keep going when I could not see the light at the end of the tunnel.

Contents

List of Figures

List of Tables

List of Acronyms

5G Fifth Generation Technology Standard.

AdaBoost Adaptive Boosting.

AI Artificial Intelligence.

ANN Artificial Neural Network.

API Application Programming Interface.

ASs Autonomous Systems.

BDDP Broadcast Domain Discovery Protocol.

CNN Convolutional Neural Network.

DDoS Distributed Denial-of-Service.

DFS Depth-First Search.

DL Deep Learning.

DRL Deep Reinforcement Learning.

FTP File Transfer Protocol.

HoL blocking Head of Line blocking.

HTTP Hypertext Transfer Protocol.

HTTPS Hypertext Transfer Protocol Secure.

IANA Internet Assigned Numbers Authority.

ICMP Internet Control Message Protocol.

IETF Internet Engineering Task Force.

IoT Internet of Things.

IPsec Internet Protocol Security.

LLDP Link-Layer Discovery Protocol.

LSTM Long Short-Term Memory.

MAC Message Authentication Code.

MIB Management Information Base.

ML Machine Learning.

MLP Multi-Layer Perceptron.

MOS Mean Opinion Score.

MPTCP MultiPath Transmission Control Protocol.

NFV Network Function Virtualization.

NOs Network Operators.

OF OpenFlow.

P2P Peer-to-Peer.

QoE Quality of Experience.

QUIC Quick UDP Internet Connections.

RF Random Forest.

SDN Software-defined Networking.

SIP Session Initiation Protocol.

SLA Service-level Agreement.

SMTP Simple Mail Transfer Protocol.

SNMP Simple Network Management Protocol.

SR Segment Routing.

SSH Secure Shell Protocol.

SSL Secure Sockets Layer.

SVM Support-Vector Machine.

TCAM Ternary Content Addressable Memory.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

TLS/SSL Transport Layer Security / Secure Sockets Layer.

UDP User Datagram Protocol.

UN United Nations.

VPN Virtual Private Network.

List of publications

International journals

- **Van TONG**, Sami SOUIHI, Hai Anh TRAN and Abdelhamid MELLOUK. "SDN-based Application-aware Segment Routing for Large-scale Network." *IEEE Systems Journal* (Accepted in October 2021), 10 pages, doi: 10.1109/JSYST.2021.3123809.

International conferences

- **Van TONG**, Hai Anh TRAN, Sami SOUIHI and Abdelhamid MELLOUK. "Machine Learning based Root Cause Analysis for SDN Network." *2021 IEEE Global Communications Conference (GLOBECOM)*, 6 pages, Madrid, Spain, 7–11 December, 2021.

- **Van TONG**, Sami SOUIHI, Hai Anh TRAN and Abdelhamid MELLOUK. "Service-centric Segment Routing Mechanism using Reinforcement Learning for Encrypted Traffic." *2020 16th International Conference on Network and Service Management (CNSM)*, 2020, pp. 1-5, doi: 10.23919/CNSM50824.2020.9269070.

- Lamine Amour, **Van TONG**, Sami SOUIHI, Hai Anh TRAN and Abdelhamid MELLOUK. "Quality estimation framework for encrypted traffic (q2et)." *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1-6, doi: 10.1109/GLOBECOM38437.2019.9014234.

- **Van TONG**, Hai Anh TRAN, Sami SOUIHI and Abdelhamid MELLOUK. "Network troubleshooting: survey, taxonomy and challenges." *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, 2018, pp. 165-170, doi: 10.1109/SaCoNeT.2018.8585610.

- **Van TONG**, Hai Anh TRAN, Sami SOUIHI and Abdelhamid MELLOUK. "A novel QUIC traffic classifier based on convolutional neural networks." *2018 IEEE Global*

Communications Conference (GLOBECOM), 2018, pp. 1-6,
doi: 10.1109/GLOCOM.2018.8647128.

- **Van TONG**, Hai Anh TRAN, Sami SOUIHI and Abdelhamid MELLOUK. "Empirical study for dynamic adaptive video streaming service based on Google transport QUIC protocol." *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, 2018, pp. 343-350, doi: 10.1109/LCN.2018.8638062.

Introduction

« We are all now connected by the Internet, like neurons in a giant brain »

Stephen Hawking

1 General Context

Root causes analysis of anomalies plays an important in the network. Therefore, network troubleshooting which is a process of detecting anomalies, identifying its root causes and implementing remediation approaches to solve it definitively, is studied thoroughly by the research community [4, 5, 6]. The reason is that computer networks nowadays have rapidly evolved along with significant growth of IoT (Internet of Things) increase not only a network coverage but also a complexity in the computer network, bringing a risk of incurring problems in the network. For example, there are many problems in the network (e.g., server disruptions, cyberattacks, link failure, etc.).

Regarding the server disruptions, Tab. 1 illustrates the total downtime and corresponding money lost of several service providers [7]. For instance, Youtube and Paypal lost from $34,000 to over $6,700,000 related to a few hours of disruptions on their cloud servers due to failures.

Besides, many cloud services nowadays are disrupted by cyberattacks (e.g., DDoS

	Total Downtime (Hours)	Cost (USD)
Youtube	0.17	34,000
CloudFlare	1	168,000
Zoho	33.5	600,000
Cisco	5.33	1,066,000
eBay	6.25	1,406,250
Facebook	8.5	1,700,000
Paypal	30.2	6,795,000

Table 1: Downtime of service providers and their economic impacts.

(Distributed Denial of Service) attacks, etc.). DDoS attack is a kind of cyberattack designed to overload and disrupt network services by exhausting them with access requests. In February 2018, Github [8] was targeted by DDoS attacks with 1.3 Tbps of traffic that overwhelms their servers with 126.9 million packets per second. It was the biggest recorded DDoS attack at that time, but GitHub's systems only suffered from 20 minutes of downtime. The reason is that GitHub implemented a DDoS migration mechanism to detect and prevent attacks from exhausting their servers. Similarly, in February 2020, Amazon announced that their AWS Shield services mitigated the largest recorded DDoS attack with 2.3 Tbps of network traffic [8]. This attack which caused three days of "elevated threat" for AWS Shield services, was carried out using hijacked CLDAP (Connection-less Lightweight Directory Access Protocol) web servers. According to the latest report of Radware's Threat Research team during the first four month of 2021, the volume of DDoS attacks increases by 30 percent. DDoS attacks not only focus on cloud services but also financial institutes. For example, data centers of one of the top 15 banks in Europe, with over a trillion dollars in assets, were targeted due to three large bursts of traffic during the second week of June 2021 [9]. The first attack reached a peak at 80 Gbps within seconds while the second and last attacks reached a peak at 45 and 24 Gbps, respectively.

Before presenting the problem statements and objective of the thesis, we explain several notations in network troubleshooting. *Symptom* is an external manifestation of failures that leads to anomalies in the network. *Failure* happens when an error results in a malfunction of network systems. *Error* is a difference between observed and specified conditions. For example, there are errors related to packet transmission when packets arrive at a receiver. The receiver can incorrectly read bit value (e.g., reading bit 0 instead of bit 1, etc.). Error is an outcome of faults. In other words, *fault* (also referred to *problem*) is a root cause that leads the network systems to an error state. For example, a crashed program is a failure that occurs when the program enters a branch of code comprising a programming error. The root cause of programming error is a programmer who leads to the failure (crashed program). In general, network problems can be categorized according to its time duration [6, 10]:

- Permanent problem: It exists in the network until remediation is implemented (e.g., broken cable, malfunctioning interface card, etc.).
- Intermittent problem: It is a malfunction at a specific time interval in network sys-

tems that normally operate at other intervals. For example, Internet connections in a home network are not stable during rush hours because of many network devices. However, it is normal in other hours when the number of network devices accessing the Internet reduces.

- Transient problem: It is a temporary problem in the network for a short period and causes to slight performance degradation. After a given time, it disappears automatically without human intervention. For example, when a user accesses a multimedia service (e.g., video streaming, etc.), the service may not be available due to overload. After a few minutes, the user retries and can access the service.

2 Problem Statements and Objectives

In network troubleshooting, the processing time for root cause analysis and remediation can take from an hour to more than five hours depending on the status of anomalies in the network [11]. As a result, network systems can suffer from negative impacts (e.g., high latency, high loss, etc.). These impacts can result in frequent connection interruptions in the network. Depending on the anomaly's nature, there are two possible cases [12]. If the root cause of anomalies is identified and solved quickly, temporary remediation is not necessary. Otherwise, temporary remediation is required to guarantee the availability of the network. Therefore, it became inevitable to think about network troubleshooting frameworks that guarantee the network's availability during the root cause analysis and definitive remediation.

Although network troubleshooting is studied for the past two decades, there have been several concerns regarding its deployment in the context of encrypted traffic. The traditional troubleshooting mechanisms were not designed for encrypted traffic. However, many service providers today encrypted network traffic to prevent attackers from inspecting data packets for illegal activities. Concretely, 80 percent of web traffic was encrypted by 2019 compared to 40 percent by 2016 thanks to a recent Cisco report [13]. From the point of view of NOs (Network Operators), the information in the packets is hidden such as sequence number, acknowledgment number, payload signatures and so on. This brings several limitations related to network performance monitoring approaches (e.g., estimation of quality of experience, application identification, etc.) and intrusion detection systems [14, 15]. Therefore, encrypted traffic results in many obstacles for trou-

bleshooting, particularly in data collection (e.g., collecting performance metrics, etc.) and remediation approaches using a deep packet inspection (e.g., application-aware traffic engineering, signature-based intrusion detection systems, etc.).

The main objective of this work are the following:

- It is difficult to design an effective troubleshooting architecture for encrypted traffic where header and payload of packets can be encrypted to protect the data and user's privacy. Moreover, encrypted traffic results in obstacles for application-aware remediation mechanisms due to a lack of information about application classes. How can we design effective troubleshooting architecture for encrypted traffic?

- It is not easy to identify the real root cause of anomalies and solve it definitively. Therefore, we present a proof-of-concept of root cause analysis and remediation approaches to identify the root cause of anomalies and solve it completely.

- Identifying the root cause of anomalies and implementing remediation approaches require much time depending status of anomalies. Consequently, the network will be influenced negatively (e.g., high latency, high loss, etc.) during that period. How can we deal with this issue?

3 Main contributions

In this section, we outline the main contributions of this work. Concretely, we propose a novel troubleshooting framework for Network Operators in the context of encrypted traffic with five modules: data collection, anomaly detection, temporary remediation, root cause analysis and definitive remediation. In addition to the proposed troubleshooting framework, there are four main contributions as follows:

- In addition to the data collection module, we propose a novel traffic classification approach to classify encrypted traffic into different kinds of applications (e.g., video streaming, file transfer, etc.). The application class plays an important role in the remediation approaches (e.g., application-aware mechanisms, etc.) in network troubleshooting (refer to Chapter 3).

- We propose a temporary remediation approach to assure the availability of network as well as meet strict SLA requirements during the root cause analysis and definitive remediation (refer to Chapter 5).

- We present a proof-of-concept for the root cause analysis and definitive remediation in network troubleshooting which allows to automatically identify the root cause of anomalies and address it completely (refer to Chapter 6).

- To implement the network troubleshooting framework, it is necessary for troubleshooting datasets. Therefore, we build and contribute the troubleshooting datasets which contain a dataset for encrypted traffic classification approaches and two datasets for the root cause analysis in order to facilitate the network troubleshooting (refer to Chapter 2, Section 2.4).

4 Dissertation Organization

In this dissertation, each chapter is dedicated to one module of the proposed troubleshooting framework. The remainder of this dissertation is organized as follows:

Chapter 1 This chapter provides related work on several network problems (e.g., link failure, switch failure, etc.). In addition to explaining the fundamental parts of traditional troubleshooting architecture, we explain how network traffic is encrypted and provide an analysis on limitations of network troubleshooting for encrypted traffic.

Chapter 2 presents fundamental parts of novel troubleshooting architecture in the context of encrypted traffic and shows a proof-of-concept of this architecture in SDN (Software-defined Networking) environment. Besides, we present a parameter measurement module to collect data in order to build troubleshooting datasets. Moreover, the chapter thoroughly describes the troubleshooting datasets which are composed of datasets for the root cause analysis and for encrypted traffic classification.

Chapter 3 describes a novel encrypted traffic classification method to identify different kinds of applications. The purpose is to provide information about application classes for application-aware mechanisms in network troubleshooting.

Chapter 4 presents related work on anomaly detection. Moreover, this chapter takes into account congestion to generate anomalies and presents an anomaly detection approach using machine learning to detect these anomalies in the network.

Chapter 5 presents an application-aware segment routing mechanism in temporary remediation. This mechanism identifies application classes according to traffic classification. In a particular application, this mechanism implements a specific routing strategy based on a reinforcement learning algorithm to meet strict SLA requirements.

Chapter 6 considers congestion a use-case for the root cause analysis and definite remediation. This chapter presents a root cause analysis using machine learning to identify the root cause of congestion. Besides, the chapter presents an adaptive congestion control algorithm to solve it completely.

Conclusions and Perspectives The last chapter concludes this thesis and provides an insight into our future work and perspectives in the area of network troubleshooting.

Chapter 1

State-of-the-art on Network Troubleshooting

«A Protocol Approach to Troubleshooting»

Ed Wilson

Chapter 1 presents the state-of-the-art on network troubleshooting and a traditional troubleshooting architecture for non-encrypted traffic. Then, we discuss its limitations when traffic is encrypted.

Contents

1.1 Network Troubleshooting

In the early 19th century, technicians were dispatched to find problems in telegraph and phone line infrastructure to repair and solve the issues. Historically, a troubleshooter refers to a skilled worker who finds and solves technical problems. Nowadays, *troubleshooting* is a form of problem-solving which aims to repair failed processes in a machine or a system. According to related work [16, 17], there are several existing conceptions of the troubleshooting process. The basic concept of troubleshooting is finding the faulty components in a device to repair or replace it [18]. Schaafstal et al. [19] designed the troubleshooting process with four subtasks: formulating problem description, cause generation, test, and evaluation. Similarly, troubleshooting is considered as an iterative process with four subprocesses: problem space construction, problem space reduction, fault diagnosis, and solution verification [20].

Network troubleshooting is an iterative process with three subtasks: identifying, diagnosing and solving problems in the network. In the past, NOs (Network Operators) implemented manual troubleshooting tools such as *ping*, *traceroute* and so on. *ping* is a computer network administration utility designed to check a reachability between a source and a destination and round-trip time of packets in the network. *traceroute* is a computer network diagnostic utility used to display possible routes between a source and a destination and measure a transit delay of packets in the network. These troubleshooting tools are used to diagnose complex problems such as loops caused by undefined interaction between spanning tree protocols [21], etc. However, these approaches are not effective with a huge number of network devices. Besides, 24.6 percent of administrators reported that anomaly diagnosis takes over an hour on average to solve anomalies [22]. Therefore, it is necessary for an automated troubleshooting process that aims to detect an anomaly, locate its causes and solve it. Consequently, network troubleshooting is considered by the research community [4, 5, 6]. In the following subsection, we present state-of-the-art of network troubleshooting.

1.1.1 State-of-the-art

According to related work on network troubleshooting [4, 5, 23], problems can be classified into several categories thanks to locations where problems happen or factors that result in problems. Yu et al. [5] and Fonseca et al. [4] categorize problems into problems

in application, control and infrastructure layer. Similarly, problems can be classified into problems in application service providers (ASP) or Internet service providers (ISP) [23]. Besides, problems can be classified into problems caused by administrators (e.g., router misconfiguration, server misconfiguration, etc.) or problems that are not caused by administrators (e.g., link failure, switch failure, buffer overload, etc.). According to a survey of Network Operators (NOs) [11], in this thesis, we present several problems which are not caused by administrators in following subsections.

1.1.1.1 Rule Failure

Bu et al. [24] categorized failure rule in the network into missing fault and priority fault. The missing fault occurs when a rule is not executed as expected whereas the priority fault occurs when overlapping rules violate a priority order.

There are research studies concentrating on the missing fault including ATPG [22] and Monocle [25]. These approaches verify the rules by generating probe packets to exercise every rule. ATPG uses a header space analysis [26] to check the reachability between all test hosts. Then, the reachability result is transferred to a probe packet generator to compute a minimal set of probe packets via greedy algorithm [27]. Next, these probe packets are sent into the network systems to check the rule's corrections. If an error is detected, a fault localization algorithm is implemented to narrow down to identify the root cause. However, ATPG has a drawback when it generates the probe packets for all rules. It is not effective when there are only a few up-to-date rules. Consequently, Monocle is proposed to overcome this drawback. This approach only verifies recently-installed rules and reports misbehaviors. Besides, Monocle formulates knowledge from flow tables in the switches as constraints and applies an SAT-solver [28] to generate a set of probe packets.

Probing is an intrusive method which generates significant overhead and increases link utilization in the network. Consequently, it is necessary to minimize the number of probe packets. This is a minimum set cover problem, which is a NP-Complete problem [22]. Therefore, Bu et al. [24] proposed RuleScope, a framework for detecting rule failures in the network. RuleScope divides flow tables into solvable subsets of rules to minimize probe scale. Then, this approach creates a directed acyclic graph for each subset and generates a set of probe packets for each subset. As a result, this approach processes the probe packet generation more quickly due to a small scale of rule subsets.

Although RuleScope minimizes the number of probe packets, this approach suffers

from a drawback related to a separation in the flow tables. This lead to the priority fault in the switches. The separation in the flow tables into small subsets can result in pretermitting two overlapping rules in two different subsets of rules. Zhao et al. [29] proposed SERVE, a rule verification to identify rule failure in the switches automatically. First, SERVE extracts all rules for each device and builds a multi-rooted tree that considers rule connections. Next, SERVE analyses the multi-rooted tree to generate the minimum number of probe packets. The minimum set cover problem is a NP-Complete problem, so SERVE applies DFS (Depth-First Search) algorithm to generate the probe packets. In [30], Zhao et al. extended the previous study [29] to present a complete framework. After generating the probe packets, SERVE injects these packets into network systems using an out-band channel. Besides, SERVE also computes a desired network behavior using the multi-rooted trees. According to a comparison between the feedback from the out-band channel for every rule and the desired network behaviors, SERVE can detect faulty rules and send notifications to administrators. The SERVE's performance is evaluated to benchmarks in processing time, number of probe packets and overhead. Concerning the number of probe packets, SERVE decreases the number of probe packets by up to 75 percent in comparison with Monocle. Regarding the processing time, SERVE's figure is three times less than the figure for ATPG. As for the overhead, in-band bandwidth is not influenced according to using the out-band channel to inject the probe packets. Besides, the out-band bandwidth is far less than link capacity.

1.1.1.2 Link Failure

Link failure refers to unreachability between two switches. It can lead to a high packet loss and performance degradation in the network. Link failure can be detected according to probe packets in active monitoring approaches. *ping* is a simple troubleshooting tool that sends probe packets to check the reachability between two end-points. If probe packets are lost, it means that there is a faulty link between these end-points. Similarly, Cascone et al. [31] proposed a fast failure detection mechanism to detect the link failure based on the exchange of bidirectional "heartbeat" packets. When packet rate drops below a threshold, a node sends heartbeat packets to its neighbors. If there are no responses from its neighbors after a given time, the link failure happens in the network. However, this mechanism requires a strict consumption related to the backup solutions that cannot be utilized to guarantee the short failover delays (1 ms).

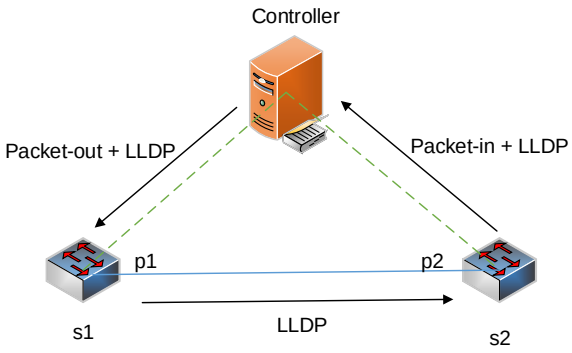


Figure 1.1: Unidirectional link discovery in LLDP.

Moreover, this problem can be detected by using LLDP (Link-Layer Discovery Protocol) in SDN (Software-defined Networking) [32, 33]. According to the topology discovery protocol, SDN controller can detect link failure and remove it from network topology. First, an OF (OpenFlow) switch connects to the controller so that the controller knows its active ports. Next, the controller generates a Packet-Out message to each active port in the switch to discover the topology. The LLDP between switch s_1 and s_2 is depicted in Fig. 1.1. First, the controller encapsulates an LLDP packet in a Packet-out message and sends it to the switch s_1. When switch s_1 receives the Packet-out message, it will forward the LLDP packet to switch s_2. After receiving the LLDP packet, switch s_2 encapsulates this packet in a Packet-in message and sends it back to the controller. The controller receives this message and creates a link from switch s_1 to s_2. The same process is performed to identify the link for an opposite direction. When link s_1-s_2 is faulty, the controller will not receive the Packet-in message from switch s_2. Then, the controller will remove this link from the network topology. In the network with S switches interconnected by a set of L links, the total number of Packet-out and Packet-in messages are described in Equation 1.1, 1.2, respectively. P_i is the number of the active port in the switch S_i.

$$\text{TOTAL}_{\text{PACKET-OUT}} = \sum_{i=1}^{S} P_i \tag{1.1}$$

$$\text{TOTAL}_{\text{PACKET-IN}} = 2L \tag{1.2}$$

Unlike the SDN environments, a hybrid SDN contains OF switches and traditional switches that LLDP cannot discover. Therefore, SDN controllers (e.g., Floodlight [34], OpenDayLight [35], etc.) utilize a combination of LLDP and BDDP (Broadcast Domain

Discovery Protocol) [32] for the topology discovery. There are two main differences between LLDP and BDDP. First, destination MAC address in BDDP is a broadcast address (FF:FF:FF:FF:FF:FF) in contrast to multicast addresses used by LLDP. This allows the traditional switch forwarding the BDDP packets to detect multi-hop links whereas LLDP is a mechanism for a single-hop link detection. Second, EtherType of LDDP is 0x88cc while EtherType of BDDP is 0x8999.

To keep up-to-date topology information, the controller needs to send $\text{TOTAL}_{\text{PACKET-OUT}}$ Packet-out messages and $\text{TOTAL}_{\text{PACKET-IN}}$ Packet-in messages periodically. This leads to a significant control overhead and an increase of link utilization. Therefore, many studies focus on optimizing the number of control packets for the topology discovery. Hasan et al. [36] proposed OpenFlow Discovery Protocol version 2 (OFDPv2) to reduce the number of Packet-out messages. First, OFDPv2 reduces the number of LLDP Packet-Out messages sent to each switch to one. Then, this protocol installs a new rule in each switch to forward LLDP packets received from the controller to all available ports. Finally, OFDPv2 customizes the event handler of the Packet-in message in the controller to parse the MAC address.

Besides, Xu et al. [37] proposed a monitoring link failure detection approach to reduce the number of Packet-out messages in LLDP to one. First, this approach calculates a monitoring tree path based on network topology. Then, this approach installs monitoring flow entries into each switch based on switch's locations in the monitoring tree path to match and apply actions to monitoring packets. Next, the controller sends a Packet-out message to a source node to instruct this node to send out the monitoring packets to other nodes following the monitoring tree. If a switch cannot receive the LLDP packet after a given time, this switch will send an alarm packet to the controller.

Unlike sending LLDP packets periodically to network systems to update the topology information, Azzouni et al. [38] proposed sOTDP, a secure and efficient OpenFlow Discovery Protocol that applies BFD (Bidirectional Forwarding Detection) [39] to detect link's events quickly. The controller listens to link's event notifications from switches to update the network topology instead of updating it periodically by sending LLDP packets. BFD is a detection protocol designed to provide a fast failure detection of forwarding paths. This protocol establishes a session between two endpoints over a particular link and sends control messages to detect active links.

1.1.1.3 Buffer Overload

Switch's buffer overload occurs when a data volume exceeds the storage capacity of buffer memory in switches. This can lead to congestion in the network. Phanishayee et al. [40] highlighted that an increase of buffer size can solve the buffer overload due to a massive amount of network traffic. Nevertheless, this approach has low scalability because replacement cost of conventional devices is extremely high. Therefore, many studies focus on congestion control and congestion avoidance mechanisms to prevent and avoid the buffer overload.

Concerning the congestion control mechanisms, there are two kinds of flows in the network: elephant flows and mice flows. Elephant flows are the large continuous flows that consume high bandwidth whereas mice flows are the small flows that consume low bandwidth. The elephant flows contain more than 90 percent of all bytes transmitted in the network [41, 42]. Thus, Kanagavelu et al. [43] proposed a local rerouting mechanism for elephant flows in the SDN environment to decrease negative impacts of buffer overload. If link utilization is higher than 75 percent of link capacity, an alarm is triggered. The controller re-routes the elephant flows into other paths with minimal link utilization to avoid the congested links. As for the mice flows, its lifetime is short, but the percentage of mice flows in network traffic is very high, approximately 90 percent [41, 42]. Trestian et al. [44] proposed MiceTrap, an OpenFlow-based traffic engineering approach for mice flows. First, the network traffic is classified into elephant flows and mice flows. The elephant flows are processed as in previous research studies (e.g., [43], etc.) whereas the mice flows are processed by MiceTrap. If a traffic volume exceeds a threshold, MiceTrap is triggered. Then, MiceTrap routes the mice flows using a weighted multi-path routing algorithm which assigns these flows into paths based on its weight. The higher the path's weight, the less the path is chosen.

Besides, there are studies focusing on both elephant and mice flows in the routing approach. Song et al. [45] proposed a routing algorithm which re-routes congested link when link utilization is higher than 70 percent of link capacity. However, the routing algorithms without considering available bandwidth can lead to congested links in the network. Thus, Attarha et al. [46] and Gholami et al. [47] proposed a routing algorithm that estimates link utilization and selects the links with high available bandwidth. Similarly, Sminesh et al.[48] proposed a routing algorithm using a Bayesian network. This approach contains three modules. The first one is to identify a bottleneck link when its link uti-

lization is higher than a threshold. The second one is to update topology and identify all available routing paths. The final one is to select the routing paths using the Bayesian network according to link utilization and residual bandwidth.

The above routing mechanisms consider a routing policy for different applications (e.g., video streaming, file transfer, etc.). However, it is ineffective because each application contains specific SLA requirements (e.g., low latency, high bandwidth, etc.). Consequently, many research work placed a special focus on application-aware routing algorithms. Li et al. [49] proposed an application-aware traffic control scheme that implements a simple flow classification to apply corresponding rules. Adami et al. [50] proposed a differentiated routing algorithm in the network. First, this approach applies a deep packet inspection to identify the SIP (Session Initiation Protocol) flows using its signatures (e.g., UDP packet with destination port 5060, content-Type is set to application/sdp, etc.). Next, this approach calculates the shortest paths for these flows using the Dijkstra algorithm with link utilization as a link cost. Similarly, Cheng et al. [51] proposed an application-aware routing algorithm to implement different routing policies corresponding to various applications. First, this method classifies the flows into three categories consisting of real-time applications (VoIP, video communication and gaming), streaming applications (streaming video, audio, IPTV and web browsing) and miscellaneous applications (file sharing and miscellaneous upload/download). Then, each category is processed with a specific routing policy using different network parameters such as link load, delay and delay variation. Pasca et al. [52] proposed AMPF, an application-aware multipath packet forwarding framework in SDN using a machine learning algorithm. First, AMPF classifies network flows into real-time traffic, buffered transfer, web browsing and restricted transfer. Next, AMPF calculates k available paths using Dijkstra's algorithm corresponding to each application. Finally, AMPF assigns network flows into these routing paths.

Regarding the congestion avoidance algorithms, Abdelmoniem et al. [53] proposed a switch-assisted TCP congestion control approach using a small modification to switches. When congestion is detected, this approach rewrites the receiver window field in TCP header to reduce sending rate. Similarly, Hwang et al. [54] proposed SCCP, a scalable congestion control protocol. SCCP limits sending rate of TCP senders by leveraging switches. Concretely, SCCP modifies the advertisement window field in the TCP header of packets traversing the switches. Besides, many studies are focusing on end-to-end congestion

avoidance algorithms which adjust sending rate at the sender's side [55, 56]. Xu et al. [57] proposed DRL-CC, a congestion control algorithm using DRL (Deep Reinforcement Learning) in MPTCP. DRL-CC monitors acknowledgment packets to extract network parameters (e.g., RTT, RTT deviation, goodput, etc.) and implements a DRL algorithm to adjust the congestion window (cwnd) at the sender's side. Similarly, Jay et al. [58] proposed Auro, an Internet congestion control algorithm using DRL. Auro monitors network states and adjusts sending rate using the DRL algorithm and network parameters including latency gradient, latency ratio and sending ratio.

1.1.2 Traditional Troubleshooting Architecture

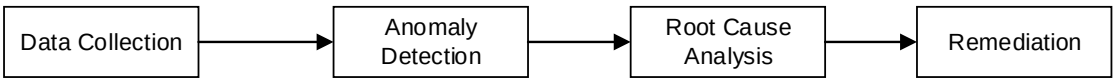


Figure 1.2: Overall Traditional Troubleshooting Architecture.

Following existing survey on network troubleshooting [4, 5, 6], the traditional troubleshooting architecture contains four main modules including data collection, anomaly detection, root cause analysis and remediation modules. The traditional troubleshooting architecture is depicted as in Fig. 1.2. First, the data collection module monitors the network systems and collects the network traffic using monitoring approaches (e.g., NetFlow [3], sFlow [59], OpenFlow-based monitoring approaches, etc.) to extract troubleshooting data (e.g., network parameters, etc.). Then, this data is analyzed in the anomaly detection module to detect anomalies (moments when network problems happen) in the network. After that, the root cause analysis analyzes thoroughly the troubleshooting data to identify the root cause of anomalies in the network. An anomaly can be caused by many root causes. For instance, congestion can result from link failure, DDoS attack, switch's buffer overload and so on. Finally, root cause is solved definitively in the remediation module to return normal states to the network. The purpose of the troubleshooting architecture is to build self-healing networks that automatically detect, diagnose, and remediate the network anomalies.

Facing a growing number of data stealing attacks, encrypted traffic is utilized by many service providers to make their data more secure and protect their user's privacy. This leads to many difficulties for network troubleshooting [14]. In the following sections, we present how network traffic is encrypted and drawbacks of network troubleshooting in

the context of encrypted traffic.

1.2 Background on encryption protocols

In the report of Thales e-Security in April 2017 [1], the percentage of encrypted flows ranges from 16 to 41 percent during the period 2005-2016. Fig. 1.3 shows the global growth of encrypted traffic from 2005 to 2016. Similarly, in a current Cisco report [13], 80 percent of web traffic is encrypted by 2019 in comparison with 40 percent in 2016.

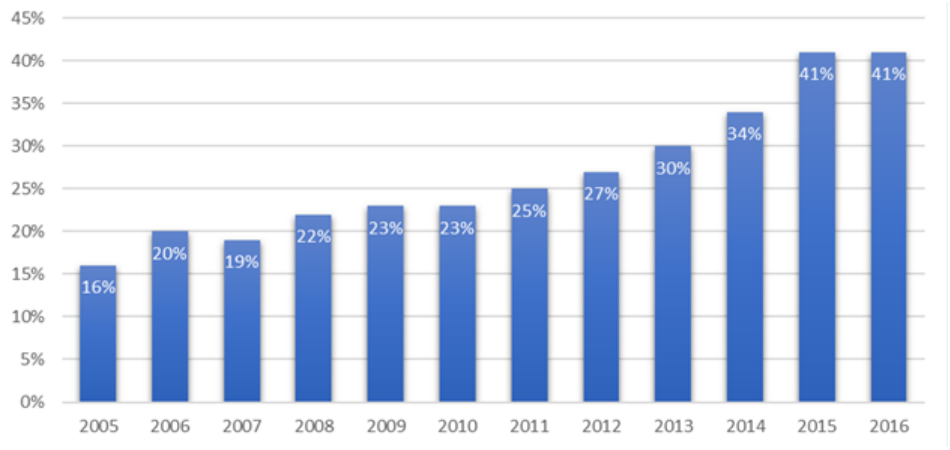


Figure 1.3: The global growth of the encrypted traffic [1].

According to related work [60], there are two kinds of encryption protocols in the network. The first one encrypts the whole packet (e.g., IPsec, etc.) while the second one only encrypts packets's payload (e.g., TLS, etc.). Besides, Google has recently developed QUIC (Quick UDP Internet Connections), a new transport layer network protocol from 2012, to provide secure connections and improve latency in the network [61]. Similar to TLS (Transport Layer Security), QUIC only encrypts the payload and a part of packet's header. QUIC is equivalent to a combination of TCP and TLS. Moreover, QUIC offers opportunities to overcome the limitations of TCP+TLS related to connection establishment time, TCP Head of Line blocking and connection migration [61]. Therefore, in this thesis, we consider QUIC as an encryption protocol in the network troubleshooting framework. The background about IPsec, TLS and QUIC are presented as follows.

1.2.1 QUIC

QUIC [61] is a novel transport protocol designed by Google from 2012. Its design is motivated by multiple existing protocols such as TCP, TLS and HTTP/2 (Hypertext Transfer Protocol version 2). In early 2021, QUIC is standardized by IETF (Internet Engineering

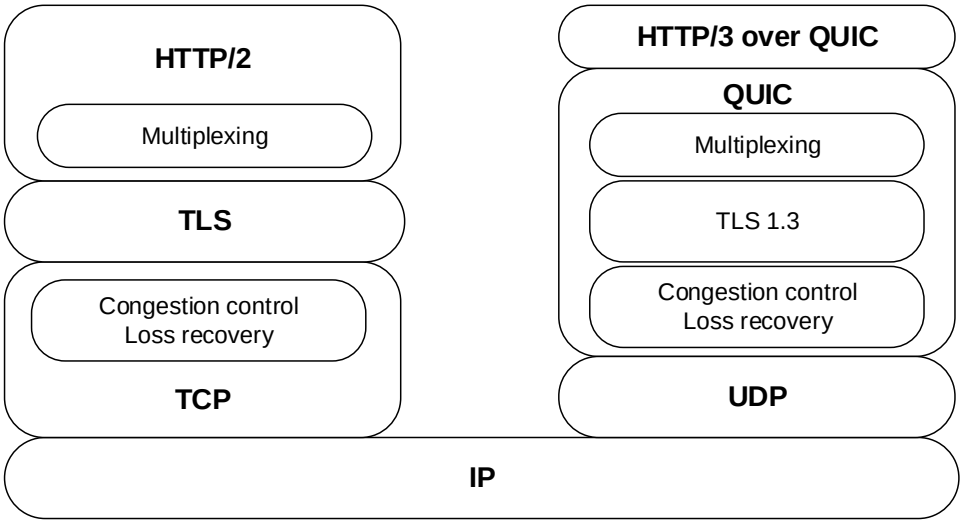


Figure 1.4: Difference between TCP+TLS and QUIC architecture.

Task Force) to solve urgent issues of TCP+TLS [62]. The difference between TCP+TLS and QUIC is depicted as in Fig. 1.4. QUIC is designed to improve latency in the network by sending data directly with 0-RTT in the best case. Moreover, it incorporates TLS1.3 to require all connections to be encrypted. QUIC also offers multiplexing features to send multiple data streams over a single connection. Furthermore, it offers opportunities to migrate connections without interrupting data communication. QUIC is implemented on the top of UDP (User Datagram Protocol), so it can be deployed in user space in contrast to TCP which is implemented in system kernels. This feature allows deploying QUIC quickly in an application update cycle. The deep analysis on QUIC's characteristics [2] is described as follows.

1.2.1.1 Encryption

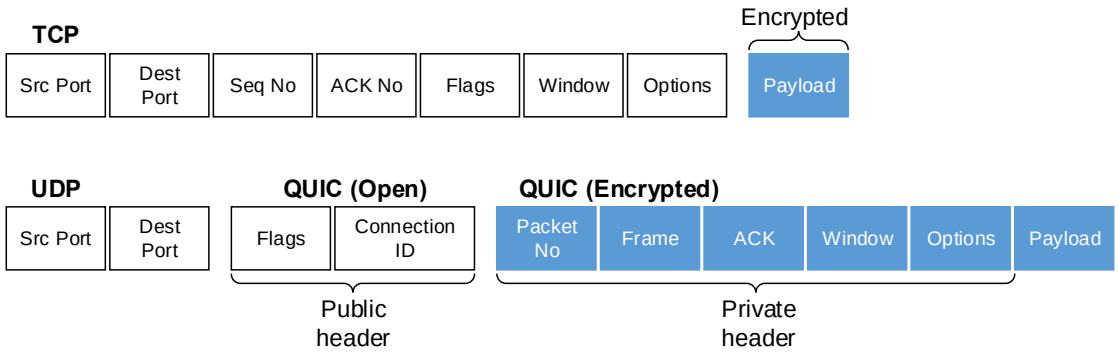


Figure 1.5: Comparison of QUIC packet format with TCP+TLS.

QUIC uses TLS 1.3 to provide end-to-end encryption. When the TLS handshake is complete, the encryption is performed on the UDP payload. In QUIC protocol, there are

two kinds of headers: public and private headers. The public header is not encrypted whereas QUIC encrypts its private header along with the payload. The comparison of QUIC packet format with TCP+TLS is depicted as in Fig. 1.5. QUIC public header contains connection ID and flags. The connection ID is an identifier used to identify a connection at an endpoint. The flags are used in the initial session establishment.

The remainder (private header and payload) is encrypted, so it is not visible to an eavesdropper. In TCP, the control of the protocol is explicit, so a third party (e.g., NOs, etc.) is able to inspect this information (e.g., sequence number, acknowledge number, etc.) for network troubleshooting. For example, NOs can infer network performance parameters (e.g., round-trip time, etc.) by inspecting TCP packets. In contrast, the private header and payload of QUIC are encrypted to protect from third-party inspection. This creates many obstacles for NOs in network troubleshooting. The detail is described thoroughly in section 1.3.

1.2.1.2 Connection Establishment

Nowadays, many services require a secure and reliable connection. The combination of TCP and TLS is a potential solution for this purpose. TCP+TLS requires at least two RTTs to establish a secure connection, bringing a significant latency. QUIC integrates TLS1.3 to set up a secure connection with 0-RTT for a connection establishment time. In other words, encrypted payload is sent in the first packet when a previous connection is resumed. Comparison of connection establishment between QUIC and TCP+TLS is described as in Fig. 1.6.

Concerning first-time connection establishment (Fig. 1.6a), QUIC only uses 1-RTT by combining transport and cryptographic handshake in contrast to 3-RTTs in TCP+TLS1.2 and 2-RTTs in TCP+TLS1.3. QUIC combines both transport and cryptographic handshake in the first packet of connection to reduce latency in connection establishment. When a client connects to a server for the first time, it sends a Client Hello message to the server for key negotiation, along with other information (e.g., connection ID, preferred version number, etc.). The client encodes the handshake using the version number which is proposed. If the server does not support this version, it redirects the client to a version negotiation process. Otherwise, the server replies with a Server Hello message containing necessary information (e.g., certificate, session information, etc.) for subsequent connections. Next, the client can send encrypted packets to the server.

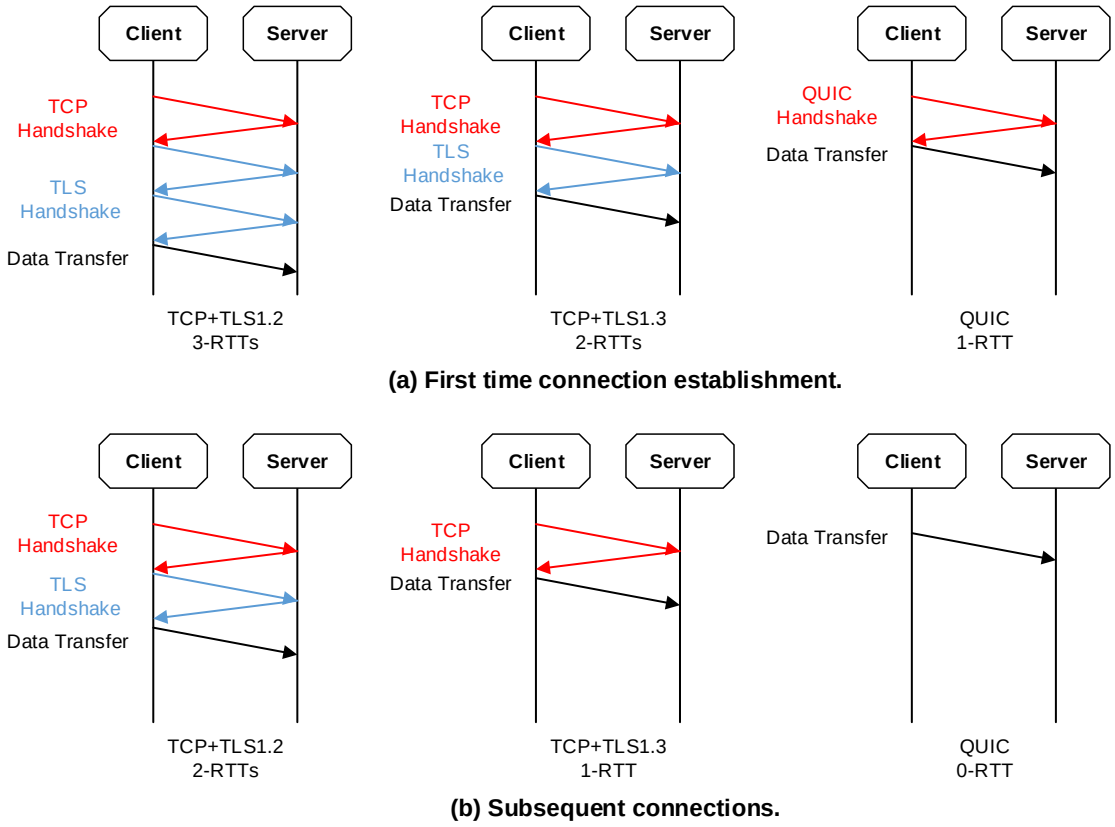


Figure 1.6: Comparison of connection establishment between QUIC and TCP+TLS.

The necessary information for the first-time connection establishment is stored in a cryptographic cookie at the client-side. It is used to authenticate the client in the subsequent connections. In the cookie, the server's Diffie-Hellman value is used to calculate an encryption key.

As for the subsequent connections (Fig. 1.6b), the client sends the Diffie-Hellman value and its cookie to the server, along with the encrypted payload. The server uses information in the cookie to authenticate the client. Then, it uses the Diffie-Hellman value to calculate the encryption key to decrypt the encrypted payload and sends back the encrypted responses to the client.

1.2.1.3 Multiplexing

Multiplexing is a method of sending multiple data streams over a single connection. In HTTP1.1, a client can only request one resource in a single connection as in Fig. 1.7a, so it needs to open multiple concurrent TCP connections. However, each client has a limited number of connections to a server, so sending a new request over one of these connections has to wait until a previous connection is complete. This leads to the HTTP HoL blocking (Head of Line blocking) which brings additional latency and complexity of

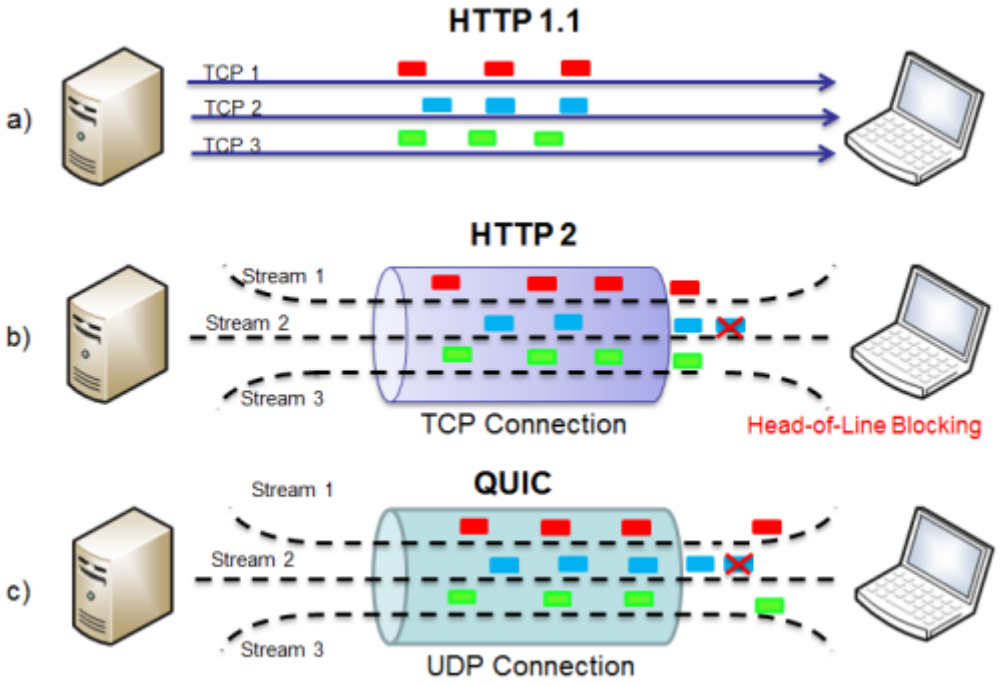


Figure 1.7: Multiplexing comparison between HTTP1.1 and HTTP/2 over TCP and QUIC [2].

managing multiple connections.

HTTP/2 (Fig. 1.7b) solves this problem by multiplexing multiple data streams over a single connection if there are multiple requests to the server. Data transmitted in the same connection is delivered to the client in order, leading to the TCP HoL blocking. If a TCP packet in a connection is lost, the server must wait until the TCP packet is retransmitted successfully before processing the following TCP packets.

QUIC (Fig. 1.7c) supports multiplexing multiple data streams over a single connection without requiring ordered delivery of all packets. Consequently, this can overcome the TCP HoL blocking. Data missing on a stream does not block the delivery of other streams.

1.2.1.4 Connection Migration

TCP uses a port and an IP address of both endpoints to identify a connection. All network connections will be interrupted when a client changes its IP address such as moving out of a Wi-Fi network range or switching from a wired to a cellular network. However, QUIC uses a connection ID (identifier) randomly generated by a QUIC client to identify a connection. When a QUIC client changes its IP address, it can continue to use the current connection ID from a new IP address without any connection interruptions.

1.2.2 Other Protocols

1.2.2.1 IPsec

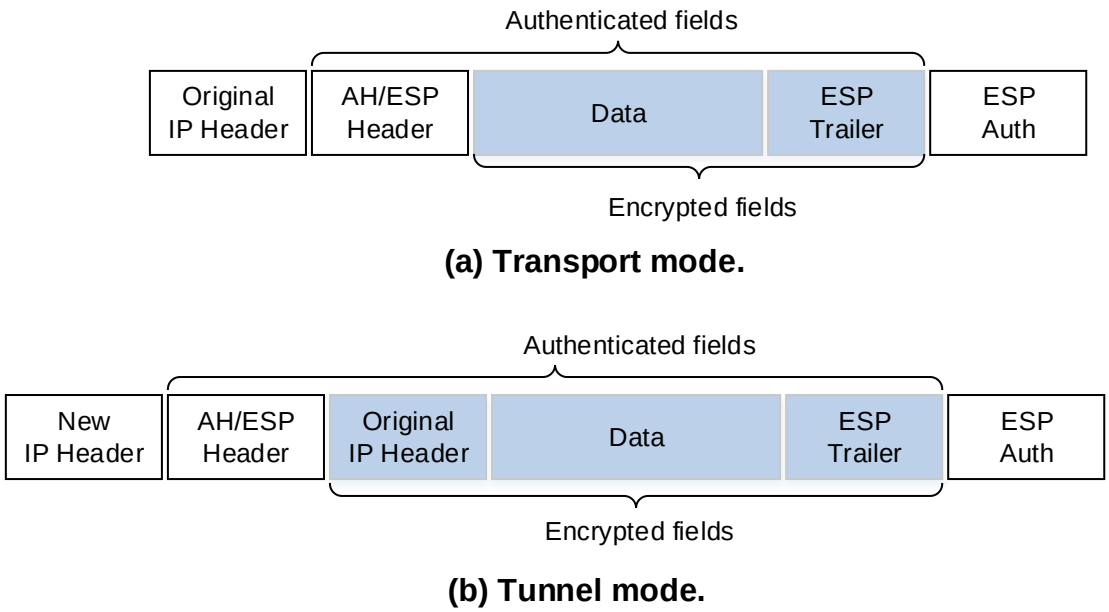


Figure 1.8: IPsec packet structure.

IPsec (Internet Protocol Security) [63] is a secure network protocol that encrypts and authenticates data packets to provide secure communication between two endpoints in the network. IPsec is used widely in VPN (Virtual Private Network). IPsec is a layer 3 end-to-end security scheme, so it not only protects payload but also IP header of data packets. IPsec uses UDP packets on port 500 for initial handshake, authentication and shared secret establishment. In this protocol, there are two main parts including Authentication Header (AH) and Encapsulating Security Payload (ESP). In the infancy of IPsec, the former is responsible for authentication while the latter provides data confidentiality. ESP adds a header and a trailer to each packet as in Fig. 1.8. IPsec can operate in two modes: transport and tunnel modes. Only data and ESP trailer are encrypted in the transport mode whereas an entire original packet and ESP trailer are encrypted in the tunnel mode.

1.2.2.2 TLS

TLS [64] is a successor of SSL (Secure Sockets Layer), a cryptographic protocol designed to provide a communication security in the network. It runs in the application layer to provide privacy and data integrity in data communication. TLS is considered as a secure layer in HTTPS (Hypertext Transfer Protocol Secure), a popular secure communication protocol on the Internet.

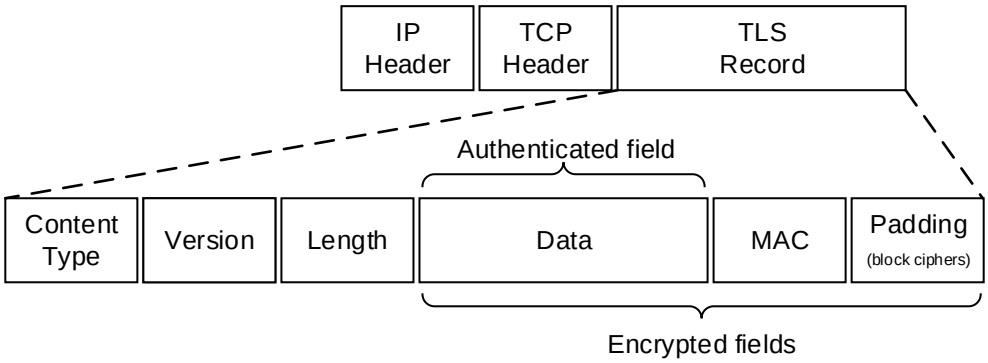


Figure 1.9: TLS record packet.

TLS contains two main layers including TLS record and TLS handshake. The former one aims to divide data into compressed fragments. In a TLS record, a fragment is followed by MAC (Message Authentication Code). The fragment and its MAC are encrypted together in a TLS record. The structure of TLS record packet is described as in Fig. 1.9. During the TLS handshake, clients and servers exchange messages to acknowledge and verify each other. Then, it establishes an encryption algorithm and agrees on session keys. At the beginning of TLS connections, clients and servers are authenticated according to X.509 certificates chain. In this phase, TLS messages exchanged are not encrypted until session keys are established and agreed. Next, these session keys are utilized by the TLS record to communicate data.

1.3 Drawbacks of Troubleshooting with Encrypted Traffic

Nowadays, many new protocols (e.g., TLS1.3, QUIC, etc.) are designed to provide encryption and authentication for data communication in the network. The authentication protects the data integrity and prevents unexpected modifications in data packets. The encryption provides the confidentiality of transport data and prevents a data inspection for illegal purposes (e.g., stealing information, etc.). This reduces the ability of NOs to obtain an insight into their networks and effectively manage their networks. In this subsection, we present two drawbacks of network troubleshooting in the context of encrypted traffic.

1.3.1 Network Performance Monitoring

Network performance monitoring is an essential part of the data collection module to collect network parameters for network troubleshooting. Parameter measurement helps

NOs to troubleshoot problems and identify whether the problems happen in their network or other sides (e.g., service providers, clients, etc.). Measurement approaches monitor network traffic and use information in packets (e.g., sequence number, acknowledgment number, etc.) to measure network parameters (e.g., Quality of Experience, etc.) or additional information (e.g., application class, etc.).

1.3.1.1 QoE Estimation

Concerning the Quality of Experience (QoE), many existing studies calculate QoE using a QoE model based on subjective data [65]. The QoE model is built according to application-level parameters (e.g., start-up delay, number of stalling events, bitrate, resolutions, etc.). These parameters can be extracted by analyzing client or server-side logs, but NOs do not have access to this information. Hence, NOs can implement network-side measurement approaches by analyzing network traffic to obtain these parameters. In fact, the application-level parameters can be measured thanks to the information in the HTTP packet headers [66, 67]. For example, Youtube QoE is determined by stalling events in the video playback. Therefore, Schatz et al. [66] proposed a deep packet inspection technique to measure the number of stalling events using IP network flows. The objective is to measure the playing time accumulated in the buffer of the Youtube player by comparing timestamp of packets and playback time of video frames. When the playback buffer is empty, the video starts to stall. The playback time can be obtained according to the HTTP packet headers. However, service providers (e.g., Youtube, etc.) nowadays encrypt network traffic (e.g., using TLS, QUIC, etc.) to protect user's privacy and data during data transmission. Consequently, QoE estimation approaches using the inspection of HTTP packet headers are no longer available. Therefore, Orsolic et al. [68, 69] proposed a QoE classification approaches for Youtube traffic using a Machine Learning (ML) algorithm. This approach monitors and analyzes network traffic to obtain 54 network features divided into six categories: packet length, transferred data size, packet count, inter-arrival time, throughput, and TCP flag count. Then, these parameters are analyzed in the ML algorithm to classify QoE into three QoE classes: low, medium and high QoE. There are many ML algorithms, so this approach evaluates performance of several ML algorithms (OneR, Naive Bayes, SMO, J48 and Random Forest) to select an appropriate one for the QoE classification.

1.3.1.2 Application Identification

Nowadays, with the development of 5G, there are many strict SLA requirements from end-users (e.g., high reliability, low latency, etc.) [70]. Consequently, differential treatment is expected by end-users. The emerging of application-aware mechanisms (e.g, network slicing, traffic engineering, etc.) which implement different policies corresponding to various applications, is a potential solution to overcome these drawbacks. In the past, NOs used port numbers or signatures in payload of packets to identify the application classes [71]. Libprotoident [72] is an open-source software library that implements the deep packet inspection for protocol identification. Libprotoident only uses port numbers and the first four bytes of payload in each direction. For example, Libprotoident identifies HTTP and SIP flows according to common strings (e.g., ET, PUT, HTTP, SIP, INVI, REGI, etc.) and registered ports (80, 8080, 8081, 443). Similarly, Adami et al. [50] implement a deep packet inspection approach to identify the VoIP flows using signatures of SIP protocol (e.g., content-type: "application/sdp", etc.). However, the packet's payload can be encrypted with encryption protocols (e.g., TLS, etc.). Therefore, Deri et al. [73] proposed nDPI, an open-source library for protocol identification using the information in both header and payload of packets. nDPI can handle TLS-encrypted traffic to identify known applications. In this case, only the network traffic in the initial key exchange can be decoded. nDPI can extract the hostname of servers to identify different applications. For instance, network flows towards the server name "api.facebook.com" are assigned as Facebook application. However, the server name can be encrypted according to DNS over TLS, HTTPS or QUIC [74]. Consequently, encrypted traffic classification approaches using packet inspection are not effective. This results in many obstacles for the encrypted traffic classification. Amaral et al. [75] proposed a new traffic classification method using flow-based features and a ML algorithm. This method monitors and analyzes network traffic to extract 12 flow-based features from the first five packets of flows (e.g., packet size, packet time-stamp, inter-arrival time, etc.). Then, these features are analyzed in the ML algorithm to identify the application classes. Lopez-Martin et al. [71] proposed a traffic classification approach using flow-based features and a DL (Deep Learning) algorithm to identify 108 application classes for IoT (Internet of Things) network.

1.3.2 Intrusion Detection System (IDS)

1.3.2.1 Firewall

In the network, middlebox is deployed to block malicious traffic and prevent illegal activities. It can classify network traffic into malicious or benign traffic to make appropriate policies [14]. In the TLS-encrypted traffic, Service Name Indicator (SNI) is used to identify malicious connections. If SNI in the client hello TLS message matches Subject Alternate Name (SAN) in the server certificate, the connection is considered as a benign connection. However, SNI can be encrypted to protect the information about destinations (e.g., domain name of websites, etc.) [74]. Nowadays, SNI is encrypted in the DNS (Domain Name System) over TLS, HTTPS and QUIC.

Besides, the middlebox can be used for content filtering [15]. The middlebox is to block content to meet requirements from law enforcement or regulatory authorities (e.g., enforcing content-based billing, etc.). They analyze data packets to obtain URL (Uniform Resource Locator) field. If it matches a blocked URL in a blacklist, a 404 error notification is returned. However, these approaches are unavailable when traffic is encrypted. DNS over TLS, HTTPS and QUIC hide the DNS information in the packets.

When network traffic is encrypted, it results in obstacles for malicious traffic prevention and content filtering. If the middlebox can not inspect network traffic for network management, it needs to be moved to the endpoints at a higher operational cost.

1.3.2.2 DDoS Protection and Migration

DDoS (Distributed Denial-of-Service) protection is necessary in network operations. It aims to detect DDoS traffic and redirect it to a migration system to filter out DDoS traffic from legitimate traffic.

DDoS can be implemented according to a botnet, a collection of compromised machines or bots [76]. Many bots nowadays use a domain generation algorithm (DGA) to generate domain names and connect to command and control (C&C) server. Therefore, many studies analyze the domain name of C&C server to detect DGA botnet and prevent DDoS attacks. Woodbridge et al. [77] proposed a DGA botnet detection system using a LSTM (Long Short-Term Memory) network. This system monitors and analyzes DNS packets to obtain the domain name of (C&C) server. Then, the domain name is analyzed in the LSTM network to detect the DGA botnet. If a domain name is classified as a malicious domain, the IP address of C&C server corresponding to this domain is blocked to

prevent DDoS attacks. Tran et al. [78] proposed LSTM.MI, a LSTM-based DGA botnet detection framework for handling multiclass imbalance. In LSTM.MI, original LSTM is adapted to be cost-sensitive to handle the multiclass imbalance. However, DNS information (e.g., domain name, IP address, etc.) is hidden with DNS over TLS, HTTPS and QUIC. This leads to many challenges for the DGA botnet detection systems as well as DDoS prevention.

Besides, DDoS can be detected according to a signature-based IDS [79, 80]. This approach inspects network traffic and checks anomalous signatures in a pre-defined list of signatures such as file hashes, malicious domains, known byte sequences, and so on. For example, Snort [81] is an open-source IDS for inspecting intruder signatures. Snort rules contain protocol, source IP address, destination IP address, source port, destination port and signatures in the payload for one or more patterns. When network traffic is encrypted, this approach is not effective. In this case, network features (e.g., packet length, inter-arrival time, etc.) can be used to detect DDoS traffic. Garcia et al. [82] proposed an anomaly detection system using AI (Artificial Intelligence) for detecting DDoS traffic over encrypted traffic. This approach monitors network traffic and extracts 57 network features (e.g., packet size, number of bytes, etc.). Then, these features are analyzed in a detection model to classify network traffic into legitimate or intrusive. In the detection model, network features are first analyzed in a Gaussian mixture model to calculate a probability density. If it is less than *threshold1*, network traffic is assigned as intrusive. If it is bigger than *threshold2*, it is assigned as legitimate. Otherwise, a reconstruction error is calculated according to Auto-Encoder. If the reconstruction error is less than *threshold3*, it is assigned as legitimate. Otherwise, it is assigned as intrusive. Similarly, Zolotukhin et al. [83] proposed an application-layer DDoS detection method for encrypted traffic. In this method, there are two main phases containing training and detection phases. In the training phase, this method monitors network traffic to collect network features of seven categories: duration of conversation, number of packet sent in one second, number of bytes sent in one second, average packet size, average TCP window size, average TTL and average of packets with different TCP flags. Then, legitimate traffic is clustered into clusters using a ML algorithm (e.g., K-mean, K-medoids, Fuzzy c-means, etc.). In the detection phase, network traffic is assigned as intrusive when it does not belong to any clusters.

1.4 Conclusion

With the ever-increasing of network devices, computer networks have become more and more complex. Therefore, network troubleshooting plays an important role in the network systems [4, 5, 6]. This section surveyed related work on several network problems: rule failure, link failure and switch's buffer overload. Facing a growing number of data theft attacks, many service providers encrypted network traffic to protect data and user's privacy. The network traffic can be encrypted by encryption protocols such as IPsec, TLS, and so on. This leads to many obstacles for network troubleshooting for encrypted traffic related to network monitoring approaches and intrusion detection systems.

When the packet's payload is encrypted (e.g., TLS, etc.) or even entire packet is encrypted (e.g., IPsec in the tunnel mode, etc.), it prevents NOs from inspecting network traffic to measure network parameters (e.g., QoE, etc.). Remarkably, the information about the application class is hidden in this case. Consequently, many encrypted traffic classification techniques are studied to identify the applications without decrypting the network traffic. Besides, encrypted traffic results in many difficulties for intrusion detection systems. In the past, IDS detects different kinds of attacks according to its signatures in the network traffic. However, these signatures are hidden due to encrypted traffic. Consequently, IDS today detects different kinds of attacks thanks to network parameters (e.g., packet length, inter-arrival time, etc.).

The traditional troubleshooting architecture has been studied by the research community over the past decades. Nevertheless, in the context of encrypted traffic, this architecture shows limitations related to network performance monitoring approaches and intrusion detection systems. In the next chapter, we present a novel troubleshooting architecture for encrypted traffic and the data collection module which aims to collect data for further modules in network troubleshooting.

Chapter 2

Novel Global Troubleshooting Framework for Encrypted Traffic

«The extension of the encryption boundary into the transport layer»

Mirja Kühlewind, MAMI
Management and Measurement
Summit 2018

In Chapter 2, we propose a novel troubleshooting architecture in the context of encrypted traffic and then present a proof-of-concept of this architecture in SDN (Software-defined Networking) environment. Then, we present the data collection module which helps to build datasets for further modules in network troubleshooting.

Contents

2.1 Novel Network Troubleshooting Architecture for Encrypted Traffic

As discussed in Chapter 1, the traditional troubleshooting architecture designed for non-encrypted traffic contains four essential modules: *Data Collection, Anomaly Detection, Root Cause Analysis* and *Remediation*. In this architecture, after detecting anomalies in the anomaly detection, its root cause will be identified in the root cause analysis module. Finally, the root causes will be addressed in the remediation module to return to a normal state in the network. During implementing the root cause analysis and remediation modules, the network will be negatively influenced (e.g., high latency, low reliability, etc.), resulting in negative economic impacts for NOs. According to a report of North American Network Operator Group [11], duration time for the root cause analysis and remediation approaches can be from an hour to more than five hours depending on the status of anomalies. In fact, Youtube lost $34,000 due to 0.17 hours of downtime while Cisco lost $1,066,000 because of 5.33 hours of downtime in their cloud servers [7].

Nowadays, many service providers (e.g., Youtube, Facebook, etc.) encrypt their data during transmission to protect data and user's privacy. However, the traditional troubleshooting architecture was not designed for encrypted traffic. According to related work [14, 15], encrypted traffic results in many obstacles related to network performance monitoring approaches in data collection (e.g., QoE estimation, application identification, etc.) and intrusion detection systems for anomaly detection. When traffic is encrypted, the information in packet header and payload are invisible. Hence, from the point of view of Network Operators (NOs), network performance monitoring approaches that implement a deep packet inspection face many obstacles to calculate QoE or to identify the application classes [66, 71, 73]. Moreover, intrusion detection systems which depend on signatures in the packet's payload (e.g., file hashes, known byte sequences, etc.) to identify different kinds of attacks are no longer available [79, 80].

Therefore, in this thesis, we propose a novel troubleshooting architecture for NOs in the context of encrypted traffic. In the proposed architecture, we propose two processes after detecting an anomaly in the network. The first process is the temporary remediation module that ensures the availability and reduces the affection of anomalies in the network until the root causes are addressed. In parallel with this process, we also implement the second process that is responsible for identifying and dealing with the root causes.

Implementing two processes aims to reduce the negative impacts of network problems during the root cause analysis and definitive remediation.

Concerning drawbacks of network troubleshooting, using network parameters is a potential solution for network performance monitoring approaches (e.g., QoE estimation, application identification, etc.) and intrusion detection systems in the context of encrypted traffic. In fact, there are DDoS detection methods for encrypted traffic using network parameters (e.g., packet length, inter-arrival time, etc.) [82, 83]. Similarly, Orsolic et al. [68, 69] proposed a QoE classification technique for Youtube traffic using 54 network parameters and ML algorithms. To identify the application classes, encrypted traffic classification approaches are using ML algorithms. These approaches focus on HTTPS [84] and VPN traffic [85]. Consequently, we propose an encrypted traffic classification approach to identify application classes for QUIC traffic. Application identification plays an important role in application-aware remediation mechanisms in network troubleshooting.

The proposed troubleshooting architecture for encrypted traffic is designed as in Fig. 2.1. The major modules are described as follows:

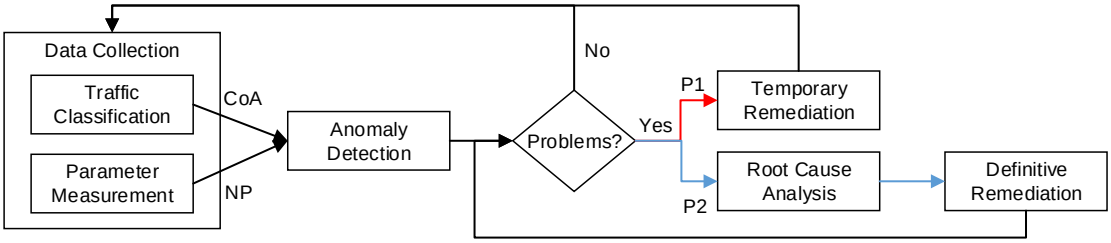


Figure 2.1: The novel troubleshooting architecture in the context of encrypted traffic.

- *Data Collection*: It is composed by two essential modules: traffic classification and parameter measurement. The former aims to identify the class of application (CoA) on network flows in the context of encrypted traffic while the latter aims to monitor and collect network parameters (NP) for further modules.

- *Anomaly Detection*: This module takes into account time-series network parameters from data collection to detect an anomaly (abnormal symptom of network problems) in the network. The purpose is to identify the moment when problems happen in the network by detecting its symptoms.

- *Temporary Remediation*: This module is to implement remediation approaches to

reduce negative impacts of anomaly (e.g., high latency, high loss, etc.) temporarily until its root causes are identified and solved.

- *Root Cause Analysis*: An anomaly can be caused by many root causes. For example, congestion can be caused by link failure, switch failure, buffer overload, etc. This module aims to identify its root causes (which problem leading to the anomaly) for further modules.

- *Definitive Remediation*: After identifying the root causes of anomaly, this module is to solve its root causes completely.

2.2 Proof-of-concept of Proposed Troubleshooting Architecture in SDN

To solve root causes (e.g., switch failure, misconfiguration, etc.) in the network, we sometimes need to update network devices. In traditional network architecture where control logic is distributed, updating policies in the network troubleshooting is implemented separately in each network device [86]. This can lead to inconsistency between network devices that results in other problems in the network. Fortunately, the emerging of SDN is a potential solution to overcome this obstacle. SDN with a separation between the control and infrastructure layers offers global visibility and better flexibility to troubleshoot the network. Besides, SDN offers monitoring tools (e.g., link layer discovery protocol, etc.) to facilitate network troubleshooting. Consequently, the proposed troubleshooting architecture is implemented in the SDN environment. In this context, we consider ONOS [87] as a SDN controller due to a popularity in existing studies [86, 88].

As a proof-of-concept of the proposed troubleshooting architecture for encrypted traffic, we present a troubleshooting framework for encrypted traffic in the SDN environment. This framework is described as in Fig. 2.2 with five main components: *Data Collection*, *Anomaly Detection*, *Temporary Remediation*, *Root Cause Analysis* and *Definitive Remediation*. During the transmission from servers to clients, network flow is transmitted from ingress switches to a sFlow collector [59] to obtain the class of application thanks to the traffic classification module. Besides, the measurement module monitors this flow to collect network parameters. After that, these parameters are analyzed in the anomaly detection module to detect anomalies (e.g., increase of latency, packet loss, etc.). When anomalies occur in the network, we need to implement the temporary remediation mod-

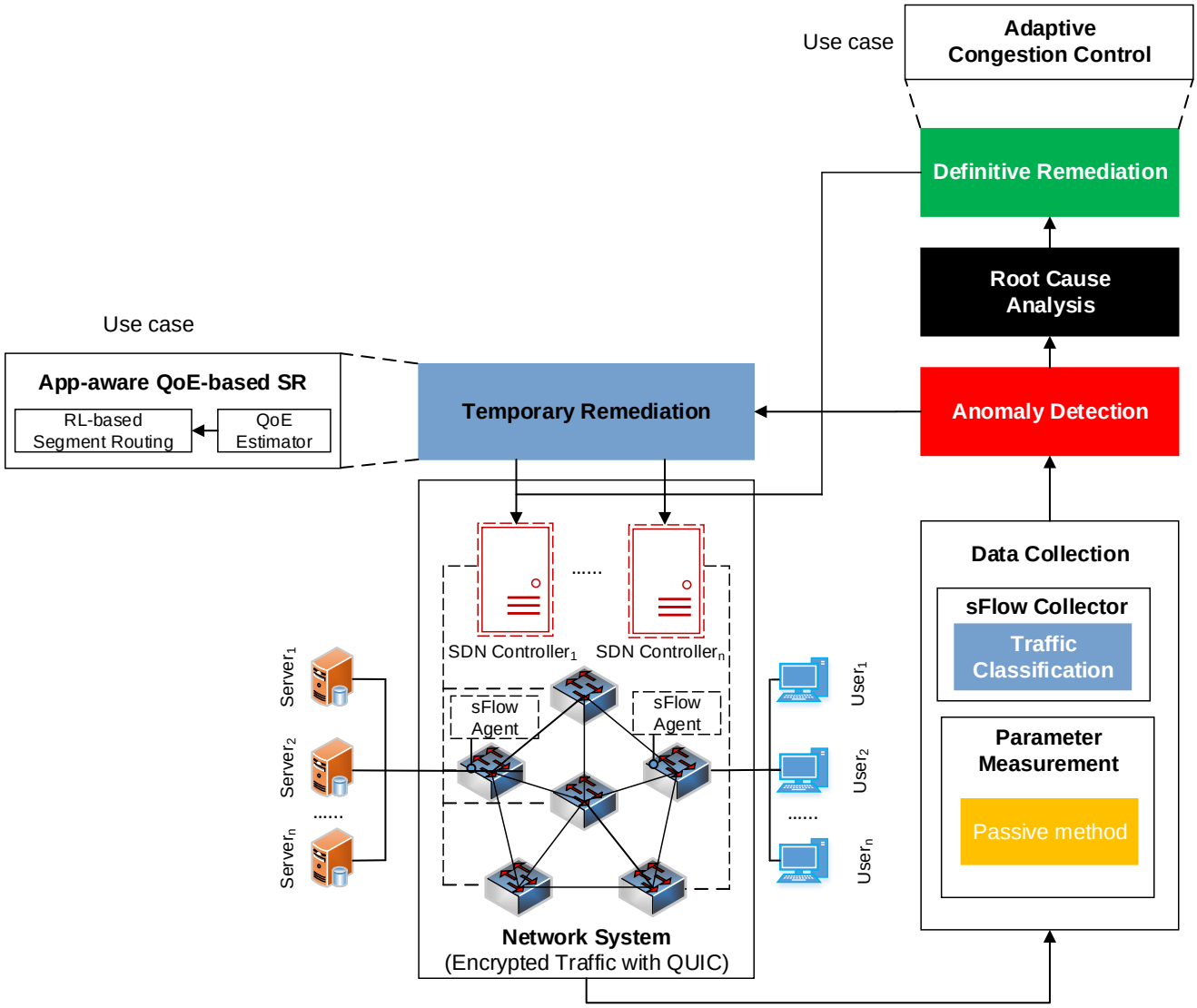


Figure 2.2: The novel troubleshooting framework in the SDN environment.

ule to guarantee the availability of the network and meet strict SLA requirements. There are many remediation approaches including load balancing, routing and so on. Many studies proved the performance of routing in problem remediation [89, 90]. In fact, Segment Routing (SR) is implemented by network operators (e.g., NTT [91], Vodafone [92], etc.). Therefore, we propose an application-aware QoE-based Segment Routing in the temporary remediation module. In parallel with the temporary remediation module, we also need to implement the root cause analysis module to identify the root cause of anomalies and solve it completely in the definitive remediation module. There are many problems in the network (e.g., misconfiguration, security, congestion, etc.), so we present an use-case related to the root cause analysis and definitive remediation approach in this thesis. We consider here congestion to generate anomalies because it is a widespread

issue from the point of view of NOs [11]. Concretely, the development of IoT and 5G increases the amount of Internet traffic, bringing much pressure for the network infrastructure and resulting in congestion in the network, particularly during the confinement of Covid-19. Congestion can be caused by link failure, switch failure or buffer overload. The objective of the root cause analysis is to identify the root causes of congestion. Concretely, the root cause analysis is to identify the type of problems (link failure, switch failure or buffer overload) leading to congestion in the network. If the link or switch failure happens in the network, we will send notifications to administrators to address it. Otherwise, we propose an adaptive congestion control mechanism in the definitive remediation to deal with buffer overload. The major components in this framework are described as follows:

- *Data Collection* consists of two essential modules: traffic classification and parameter measurement. In the parameter measurement module, we measure network parameters such as latency, packet loss, link utilization, number of packet sent, number of packet received, number of byte sent, number of byte received, number of flow entries in switch and QoE. These parameters are measured according to LLDP [93, 94] and *PortStatistics* API [95] in SDN controllers. The purpose is to build troubleshooting datasets in static and dynamic network for further modules. In the traffic classification module, there are two main stages of classification. The first classification stage is to identify chat and VoIP applications using flow-based features and a ML algorithm. The second classification stage aims to identify video streaming, file transfer and Google play music applications using packet-based features and Convolutional Neural Network. To implement this module, we need to collect raw network traffic. Collecting network traffic from Openflow switches is not effective with a huge amount of network traffic. Therefore, we implement sFlow [59], an industry standard supporting a packet sampling technique, to collect ten percent of network traffic at ingress switches. Concretely, sFlow agents sample network traffic traversing the ingress switches and forward it to the sFlow collector for the traffic classification.

- *Anomaly Detection* takes into account time-series network parameters to detect an abnormal symptom of network problems. This module aims to identify the moment when problems happen in the network by detecting its symptoms (anomalies). To generate network anomalies, we use a fault injection technique [96, 97]

to generate congestion to the network. We implement an anomaly detection approach using three time-series network parameters (latency, packet loss and link utilization) and a ML algorithm. To choose an appropriate ML algorithm, we evaluate the performance and processing time of ML algorithms such as Support-Vector Machine, Random Forest, Adaboost and Convolutional Neural Network.

- *Temporary Remediation* is the application-aware QoE-based segment routing mechanism that aims to consider various segment routing strategies corresponding to different kinds of applications to meet strict SLA requirements. In a specific routing strategy, this module monitors network states and selects appropriate paths using reinforcement learning. The proposed segment routing mechanism is based on the feedback of network environment (QoE) to adapt to unexpected network conditions. The segment routing mechanism is evaluated according to an emulation testbed with the SDN controller *ONOS* and the emulator *mininet*. Besides, we customize the *org.onosproject.segmentrouting* application [98] supported by *ONOS* to implement the proposed segment routing mechanism. In reinforcement learning, a trade-off between exploration and exploitation phases needs to be considered thoroughly. Therefore, it is formalized as a multi-armed bandit problem. In this thesis, we evaluate the performance of three selection algorithms (ϵ-*greedy, softmax* and *UCB1*) in four scenarios (perfect scenario, delay scenario, loss scenario and scenario with both delay and loss) to select an appropriate one for solving the multi-armed bandit problem. Besides, the proposed segment routing mechanism is evaluated with benchmarks (*Standard_SR* and *Max_QoE* mechanisms) related to MOS (Mean Opinion Score) against time, average optimal MOS, median MOS, 95% confidence interval of MOS, CPU usage and control overhead.

- *Root Cause Analysis* aims to identify the root cause of congestion in the network. The objective is to identify the type of problems resulting in congestion (link failure, switch failure or buffer overload). We implement a ML-based root cause analysis using nine time-series network features (e.g., latency, packet loss, number of byte sent, number of packets sent, etc.). Performance of ML algorithms (Support Vector Machine, Bagging , Random Forest, Adaboost, Gradient Boosting and Convolutional Neural Network) are evaluated to select an appropriate ML algorithm. We implement a feature selection method (wrapper method) to identify an appropriate

feature set to obtain a good performance. The performance of root cause analysis is evaluated with datasets in static and dynamic networks.

- *Definitive Remediation* is an adaptive congestion control mechanism that is in charge of dealing with root causes of congestion completely to return to normal states in the network. In this module, we propose an adaptive BBR (Bottleneck Bandwidth and Round-Trip Time) algorithm using reinforcement learning for QUIC traffic. This algorithm aims to adjust sending rate at sender sides corresponding to different network conditions. Adaptive BBR and benchmarks are evaluated in the context of HTTP/3 in contrast to existing studies evaluating congestion control algorithms in the context of HTTP/2. We customize the open-source *lsquic* [99] to implement the adaptive BBR algorithm and benchmarks. The performance of these algorithms is evaluated in an emulation testbed in dynamic network where RTT and loss of link change after a given time.

2.3 Data Collection

As discussed in the previous section, data collection plays a vital role in network troubleshooting. This is explained by a fact that the anomalies and its root causes result in a fluctuation of data collected in the network. In this section, we present several kinds of data classification, monitoring tools and the parameter measurement module which is to collect data for network troubleshooting.

2.3.1 Data Classification

According to related work [100, 101], Data can be classified in different methods as follows:

- Data types [100]: structured, unstructured and semi-structured data.
- Source [101]: application-level data, network-level data, user feedback, user profile, etc.
- Quantity: large data which need to be processed by big data processing techniques (e.g., Hadoop, etc.) before being taken into account and small data which can be taken into account without big data processing techniques.

In this manuscript, we consider the data classification according to data types because it is widely used in network troubleshooting [6, 102, 103].

Structured data refers to data that contains a definite format (e.g., network parameters, etc.). Therefore, it is easy to query and retrieve necessary information.

On the contrary, unstructured data contains no particular structure (e.g., troubleshooting ticket, etc.). The troubleshooting ticket is used to identify the anomaly and diagnose its root causes. When problems happen in the network, clients contact an IT helpdesk to report their problems. Then, the IT helpdesk collects information from the client's reports, generates troubleshooting ticket, and assigns it to the corresponding processing team to diagnose. However, troubleshooting ticket contains a large amount of unstructured text data, leading to many obstacles for data parsing and processing [103].

The semi-structured data is a combination of structured and unstructured data (e.g., system logs, etc.). System log records various system states and alarms to identify performance issues, failures, and its root causes. The information in the system log contains structure data including the IP address of the object generating an alarm, a timestamp, an alarm identifier, severity level of failures and so on [6, 102]. Besides, the system log also contains unstructured data due to an existence of network issues. The system log is available in the network systems, so it is a valuable resource for understanding network status. In fact, there are many anomaly detection approaches using system logs for identifying the anomaly in the network [102, 104]. However, these approaches contain several limitations including [6, 102]:

- Unstructured Data: Log data is unstructured, and it varies significantly between different network systems. Therefore, it is challenging to parse log data in the distributed network systems with different kinds of network devices.

- Processing massive data: Log data is used to identify the anomaly in the network, so it needs to be processed and analyzed in real-time. However, log data is usually recorded in a huge file (e.g., a few GB, etc.). Therefore, processing massive data in a short time is a challenge.

- Desynchronization: The clock desynchronization of systems logs in distributed systems can influence the order of received alarms and result in uncertain results.

In this thesis, we consider the network performance parameters as data for network troubleshooting. In fact, it is studied by the research community in network troubleshoot-

ing [4, 5, 6]. In the following subsection, we will present network monitoring techniques for data collection in network troubleshooting.

2.3.2 Monitoring Tools

To collect network parameters, there are two monitoring approaches including active and passive monitoring. Active monitoring approaches [105, 106] send probe packets into the network systems and analyze feedbacks to obtain network parameters. Passive approaches monitor the network traffic and analyze it to obtain the network parameters. The active approaches generate overhead in the network, making the status of problems more grave. Moreover, a loss of probe packets can lead to uncertain results. Therefore, we consider passive monitoring approaches to collect network parameters. In fact, there are popular standards for network monitoring including SNMP (Simple Network Management Protocol), NetFlow, sFlow, Openflow-based monitoring approaches as follows:

- SNMP is a popular protocol in network management, collecting information from different network devices (e.g., routers, switches, etc.) in the network [107]. SNMP offers massive statistical information about network devices through a database, called MIB (Management Information Base). MIB is a collection of parameters reflecting physical links (e.g., maximum packet size, transmission rate, etc.), routing (e.g., routing metrics, destination address, etc.), underlying protocols (e.g., TCP, ICMP, etc.) and so on. However, SNMP contains several drawbacks [108]. The first one is unreliability. SNMP uses the underlying UDP (User Datagram Protocol) protocol, so there is no guarantee of delivery, ordering, or duplicate protection for data transmission. The second one is related to a simple data structure. SNMP only supports a simple data structure, so it is inefficient to transfer data in a short time.

- NetFlow [3] is a flow sampling technique designed by Cisco, which is supported by many vendors including Juniper, HP, OpenVswitch and so on. NetFlow samples a flow with a specific probability, aggregates all packets of this flow into flow records and then exports towards NetFlow collectors. NetFlow architecture is depicted as in Fig. 2.3. NetFlow contains three main components including NetFlow exporter, NetFlow collector and analysis application. NetFlow exporter aggregates all packets of a flow into flow records (with a probability) and exports to one or more NetFlow collectors. NetFlow collector pre-processes data from NetFlow exporter and stores

in NetFlow storage. Analysis application collects data from NetFlow storage and analyzes for the network management.

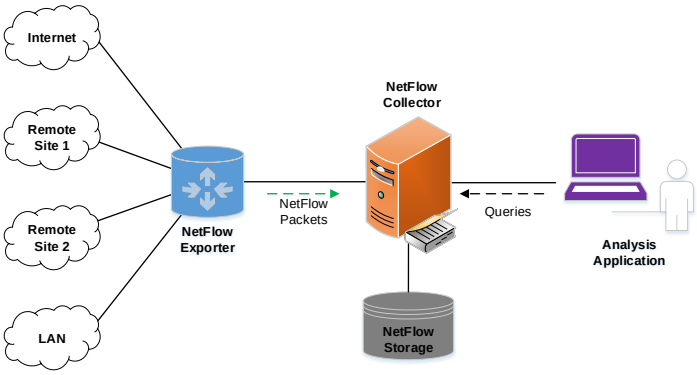


Figure 2.3: NetFlow Architecture [3].

- sFlow [59] is a standard for network monitoring with a packet sampling technique that is supported by vendors containing IBM, HP, OpenVswitch and so on. sFlow samples each packet with a specific probability and aggregates sampled packets into sFlow records and exports to sFlow collectors. sFlow architecture is shown as in Fig. 2.4.

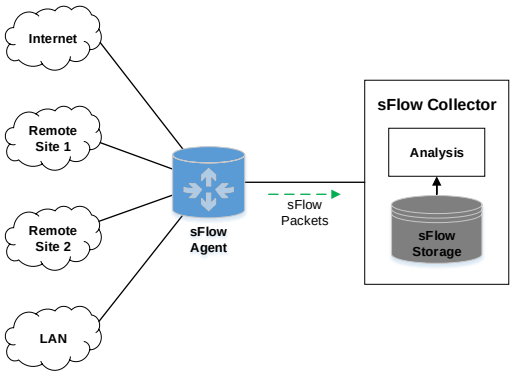


Figure 2.4: sFlow Architecture.

- Openflow-based monitoring approach is studied by the research community. Similar to NetFlow and sFlow, Openflow-based monitoring approaches collect network parameters based on information provided by Openflow agents in Openflow-supported devices. Openflow [109] is a communication protocol providing access to the forwarding plane of network devices (e.g., switch, router, etc.) in the network. Consequently, it offers opportunities to overcome the limitations of monitoring standards (e.g., sFlow, NetFlow, etc.).

In comparison with Openflow-based approaches, sFlow and NetFlow have several limitations related to scalability, flexibility, and reliability issues. Regarding the scalability issue, these approaches require configuring an agent for each network device. As for the flexibility issue, the agents send data from all network devices to the collectors periodically, which does not allow to select specific devices to be observed. Finally, concerning the reliability issues, these approaches use UDP protocol (a connectionless protocol) for data transmission, so lost UDP packets can lead to uncertain results. The difference between NetFlow, sFlow and Openflow-based monitoring approaches is described in Tab. 2.1.

Table 2.1: Comparison between NetFlow, sFlow and Openflow-based monitoring approaches.

	NetFlow/sFlow	Openflow-based approaches
Scalability	Require to configure an agent for each network device	Agent is configured automatically according to Openflow protocol
Flexibility	Send data from all devices to collectors periodically	Select which device to be observed at any time slot
Reliability	Unreliability due to using underlying UDP to send data to collectors	Using Southbound API

2.3.3 Parameter Measurement

In this thesis, we consider a dataset constituting by context-dependent parameters including:

- Latency
- Packet loss
- Link utilization
- Number of packet sent
- Number of packet received
- Number of byte sent
- Number of byte received
- Number of flow entries in switch
- QoE (Quality of Experience)

2.3.3.1 Latency

Latency is calculated according to LLDP (Link-Layer Discovery Protocol) in SDN controllers as in existing studies [93, 94]. In SDN controllers, LLDP is used to discover and update network topology. The detail is depicted in Fig. 2.5. To update the status of the link $s_1 - s_2$, the controller sends a Packet-out message containing a LLDP packet to instruct switch s_1 sending this LLDP packet to the neighboring switch s_2. The switch s_2 receives the LLDP packet, encapsulates this packet in a Packet-in message and sends it back to the controller. To estimate the latency, the controller encodes the *time-stamp* and puts it in the LLDP packet of Packet-out message. When the controller receives the Packet-in message, it will decode to obtain the *time-stamp*. The latency is a difference between current time and *time-stamp*.

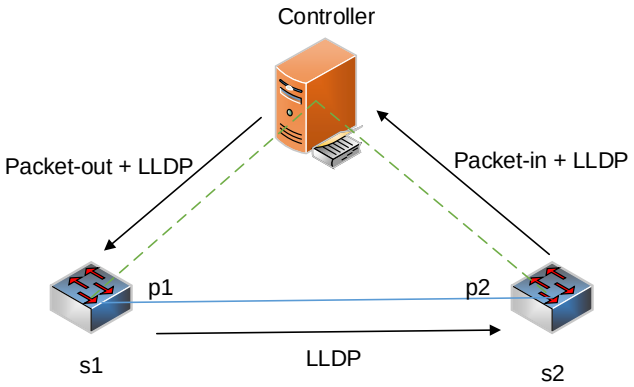


Figure 2.5: The link discovery in LLDP.

2.3.3.2 Packet Loss

Packet loss refers to the loss of LLDP packets on each link. When the controller sends the Packet-out message, we store the *time-stamp* of each LLDP packet in this message in a database. After the Packet-in message arriving the controller, we check the *time-stamp* and remove this from the database. After a given time, we increase the number of packet lost on a particular link by one if the difference between the current time and *time-stamp* in the database is bigger than a threshold. Then, we remove the *time-stamp* out of the database. The above parameters are calculated on each link and then aggregated to calculate parameters on the routing path of network flows.

2.3.3.3 Quality of Experience

QoE is a measure of user satisfaction of multimedia services (e.g., video streaming, file transfer, etc.). MOS (Mean Opinion Score) is used to estimate the QoE on user's side. Therefore, it is necessary to have a subjective dataset related to MOS. Consequently, we use a MOS dataset built in a collaboration between our laboratory (LISSI) and Orange (a Network Operator in France). This dataset was built in a subjective test campaign to obtain 1560 samples covering 23 parameters (e.g., application parameters, network parameters, etc.). Testbed for this campaign was implemented in the LISSI laboratory where 181 participants were asked to watch a short video and give their opinions. These participants are researchers and students from different disciplines between 19 and 38 years with little or no experience in video assessment experimentation. There are five levels of participants' opinions including 1 (Bad), 2 (Poor), 3 (Fair), 4 (Good) and 5 (Excellent).

According to the related research work, QoE estimation approaches are divided into three types: subjective, objective, and hybrid solutions. The first one requires participants to evaluate their perceptions about multimedia applications, so it is costly, time-consuming, and ineffective with real-time monitoring due to human intervention. The second one constructs an objective model estimating user perception using parameters (e.g., network parameter, application parameter, etc.), but identifying an effective model is sometimes complicated. The third one is a hybrid approach that uses a subjective dataset to learn QoE models based on ML algorithms. Consequently, it can combine advantages of both subjective and objective approaches.

Therefore, in this manuscript, we implemented the hybrid approach for QoE estimation, offering opportunities to estimate MOS in real-time. The MOS dataset and description about testbed building it (e.g., number of participants, understanding level of participants, etc.) are published to Github [110]. We have chosen the *random forest* algorithm to estimate the MOS score because it has less root mean square error than other ML approaches [111]. MOS is estimated according to network parameters (e.g., latency, packet loss, etc.). The detail of QoE estimation is described in [111].

2.3.3.4 Other Parameters

For other parameters, link utilization, the number of packet sent, the number of packet received, the number of byte sent, the number of byte received and the number of flow entries in switch are calculated according to the Openflow-based monitoring approach

via *Portstatistic* API [95] (Application Programming Interface) in SDN controllers.

2.3.3.5 Measurement Frequency

In parameter measurement, a frequency of polling for measurement is an essential factor. There are two methods for identifying the frequency of polling including periodic polling and event-triggered polling [112]. The former requests a measurement periodically after a given time while the latter requests the measurement when a pre-defined event occurs. The event-triggered polling requires building a database about pre-defined events, and it is not effective with unnoticed events. Periodic polling can overcome this obstacle and offer opportunities to monitor the network in real-time effectively. Therefore, we implement periodic polling in the parameter measurement module. The frequency is set to three seconds (standard window time) as in existing work [113, 114]. According to the parameter measurement module, we can collect data to build troubleshooting datasets for network troubleshooting. These datasets are described in the following subsections.

2.4 Troubleshooting Dataset

2.4.1 Datasets for root cause analysis

There are several troubleshooting datasets related to troubleshooting ticket, network parameters in NFV (Network Function Virtualization), etc. Tab. 2.2 describes the existing troubleshooting dataset. However, there is no troubleshooting dataset related network parameters for identifying root causes of anomalies (e.g., link failure, switch failure, etc.).

Therefore, in this manuscript, we build and contribute the troubleshooting dataset for network troubleshooting. Recently, during the Covid-19 pandemic, network infrastructures of NOs have to suffer a high pressure from a vast amount of Internet traffics. This results from many network devices accessing the Internet for education, remote working, entertainment and so on. Consequently, this leads to congestion in the network. Besides, according to a report of NOs, congestion is one of popular issues in the network [11]. Therefore, we take into account congestion to generate anomalies in the network and construct the troubleshooting dataset related to three root causes of congestion including link failure, switch failure and buffer overload. It is not easy to get a dataset related to these root causes because it happens unpredictably in the network. Therefore, many studies [96, 97] use a fault injection technique to generate root causes. Link failure is simulated by adding latency in links while switch failure is simulated by generating

Table 2.2: Existing troubleshooting dataset.

Data Types	Dataset	Class	Number samples
Troubleshooting Ticket	Servicenow [115]	Network, User Maintenance, Database, Application Workbench, and Security	3001
	Github [116]	Github Bug, Enhancement and Question	30000
Network Parameter	NFV [117]	Normal, CPU bottleneck, Memory bottleneck, and I/Obottleneck	3693

rule failures in the switches to obtain packet loss. Buffer overload is simulated by sending a huge amount of traffic exceeding link's capacity. These root causes are simulated in two scenarios including static and dynamic networks. In the static network, the delay and loss in the links are fixed with a specific value. In the dynamic network, the link state is changed between normal and error state according to Gilbert-Elliot (GE) model [118] following a probability value after a given time (400 seconds). In this model, the probability of changing from normal state to error state is p while the figure for changing from error state to normal state is r. When a link is in the error state, the loss and delay will be modified as described in the Tab. 2.3.

Table 2.3: Considered network conditions.

Root Causes	Static Network	Dynamic Network
Link failure	Delay: 125ms	Delay: [25, 50, 75, 100, 125ms]
Switch failure	Loss: 50%	Loss: [10, 20, 30, 40, 50%]
Buffer overload	Bandwidth: 10 mbps, Sending rate: 60-100 mbps	Bandwidth: 10 mbps, Sending rate: 60-100 mbps

In each network condition, we first extract nine features corresponding to network and standard parameters thanks to existing studies [94, 111] and API *PortStatistics* [95] in the controller. These parameters include latency, packet loss, link utilization, number of packet sent, number of packet received, number of byte sent, number of byte received, number of flow entries in switch and QoE (Quality of Experience). Then, these parameters

Table 2.4: Datasets for root cause analysis.

Root Causes	Static Network	Dynamic Network
Buffer overload	6900	6900
Link failure	9995	19710
Switch failure	7560	17565

on each link will be aggregated to calculate the paths' features to identify the root cause of anomalies. After that, these features are normalized into a value between 0 and 1 according to Max Normalization [119]. In RCA, we consider time-series features, and a sample aggregates features of ten consecutive time steps to identify the root causes. We built two datasets for network troubleshooting (one for each network condition). The number of network states corresponding to each root cause is depicted as in Tab. 2.4. These dataset are published to Github [120].

2.4.2 Dataset for traffic classification

Table 2.5: Classification dataset.

Application	Number of samples (flows)
Chat	2,783
VoIP	2,608
File transfer	4,451
Video streaming	5,844
Google play music	4,349

To validate the performance of the traffic classification module, we build a traffic classification dataset for QUIC traffic during three weeks (starting from March 2018). To capture the network traffic of video streaming, chat, VoIP and Google play music applications, we use Selenium WebDriver [121] in Google Chrome running on Ubuntu 16.04 OS. Besides, we also use quic-go [122] to transfer data between servers and clients using QUIC protocol. Then, network traffic of file transfer application is captured. We captured approximately 150 GB of network traffic including over 20,000 network flows of five QUIC-

based applications. The detail is described in Tab. 2.5. This dataset is published to Github [123].

In the following chapter, we present in detail a novel encrypted traffic classification approach for QUIC traffic. The approach is to identify application class in the context of encrypted traffic.

2.5 Conclusion

In this chapter, we present a novel troubleshooting architecture adapted to the context of encrypted traffic. Besides, we show the proof-of-concept of this architecture in SDN environments. After that, we present a parameter measurement module to collect network parameters composed of latency, packet loss, link utilization, number of packet sent, number of packet received, number of byte sent, number of byte received, number of flow entries in switch, QoE. The purpose is publicly provide troubleshooting datasets for the research community.

In addition to the network parameters, we identify the application class to facilitate the application-aware remediation approaches in network troubleshooting. The detail is described in the following chapter.

Chapter 3

Traffic Classification: Novel QUIC traffic Classifier based on Convolutional Neural Network

«Predicting the future isn't magic, it's artificial intelligence»

Dave Waters

In chapter 3, we present a traffic classification module to identify the application class for QUIC traffic. The application class also plays an important role in application-aware remediation approaches in network troubleshooting. This module is briefly described in the proposed troubleshooting framework (Fig. 2.2).

Contents

3.1 Introduction

Traffic classification [71] plays an essential role in network troubleshooting for NOs. The objective of traffic classification is to identify application classes in the network to implement application-aware mechanisms to address network problems and meet strict SLA requirements. Concretely, traffic classification is used widely in application-aware routing, billing policies, intrusion detection systems, identifying security threats, enforcing QoS requirements and so on [124]. Therefore, traffic classification is studied thoroughly by the research community.

In the past, many attempts to identify the application class relied on port-based approaches and payload-based approaches. The first approach uses port numbers in TCP/UDP packets to identify the application class. However, it is not effective because modern applications and protocols are not always tied to a specific port. Besides, many networks change the port number in the packets using NAT (Network Address Translation), leading to inaccurate results for traffic classification. The second approach [50] inspects a payload of packet to identify its signatures corresponding to a particular application, called Deep Packet Inspection. This approach can achieve high accuracy, but it is limited due to encrypted network traffic. Concretely, network traffic nowadays is encrypted to protect data and user's privacy, so the signature of application in the payload of packet is hidden.

To overcome drawbacks of such approaches, statistic-based approach using Machine Learning (ML) [71] is emerged. This approach analyzes high-level information obtained from network flows (e.g., TCP window size, packets inter-arrival times, etc.) using ML to identify the application class. Flow-based features can be extracted from all packets of a flow [125, 126] or the first few packets of a flow [71, 127]. However, these approaches require identifying pre-defined flow-based features with human intervention, so it requires a knowledge base about considered applications and a large analysis time. Consequently, many studies focus on a traffic classification approach using Deep Learning (DL). The objective of the DL algorithm is to combine feature extraction and classification modules into one module, so this approach does not require human intervention for the feature extractions. This method converts the payload of packet into bytes to create a vector representing an application class. Then, DL algorithms analyze this vector to automatically extract packet-based features (implicit features) and classify these features into the application classes for HTTPS [84] and VPN traffic [85, 128].

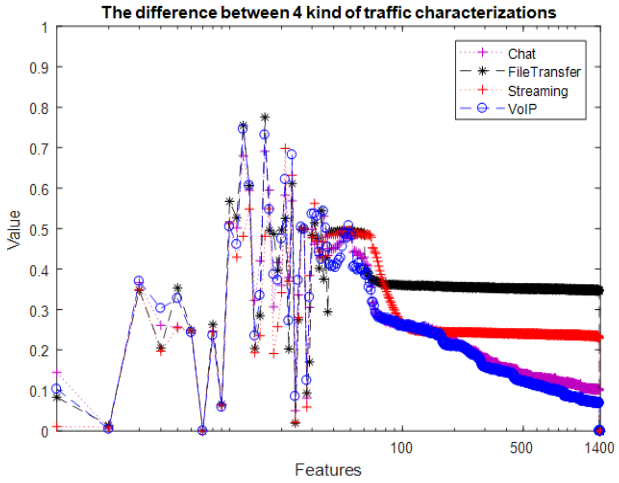


Figure 3.1: Byte in payload of QUIC packets for different applications.

Similar to HTTPS, Google has recently developed Quick UDP Internet Connection (QUIC) [61], a new transport layer network protocol on the top of UDP. QUIC contains many advantages related to connection establishment, congestion control, multiplexing without head of line blocking and connection migration. Besides, QUIC also provides a security protection equivalent to TLS/SSL (Transport Layer Security / Secure Sockets Layer).

Many existing studies focus on traffic classification solutions for HTTPS [84] and VPN traffic [85, 128]. In this thesis, we concentrate on a traffic classification solution for QUIC traffic. From the experiments, the investigation on bytes in payload of the QUIC packets (Fig. 3.1) illustrates that the DL-based traffic classification approaches cannot be effective for several QUIC-based applications including chat and VoIP applications. The reason is that there is less difference in the payload of packets for chat and VoIP applications.

Therefore, in this thesis, we propose a novel traffic classification method using Convolutional Neural Network (CNN). It is a hybrid between the statistical-based method and the DL-based method. The proposal contains two main stages. The first one uses flow-based features and a ML algorithm to classify the network traffic into chat, VoIP and other applications corresponding to elephant flows (large continuous flows in total bytes). The second one takes into account packet-based features and CNN to classify the elephant flows into video streaming, file transfer and Google play music. Chat, VoIP, video streaming, file transfer and Google play music are taken into account here because it is reported that these applications will comprise over 80 percent of global IP traffic by 2022 [129].

The remainder of this chapter is as follows. Section 3.2 presents background about

CNN and characteristics of QUIC-based applications. Related work on traffic classification is introduced in Section 3.3. Section 3.4 describes the novel traffic classification approach for QUIC traffic. Section 3.5 shows experimental results. Section 3.6 presents a conclusion which highlights future perspectives.

3.2 Background

3.2.1 Convolutional network

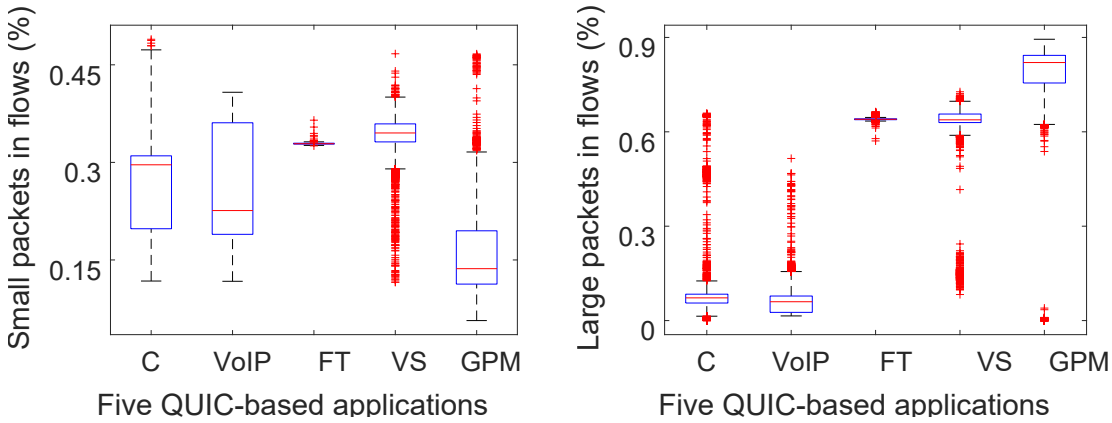
Convolutional network [130] [131] is known as Convolutional Neural Network (CNN), a specialized artificial neural network for data processing. It can be applied to pattern recognition, DGA botnet detection, traffic classification and so on [71]. Artificial Neural Network (ANN) [131] is a kind of machine learning algorithm inspired by the biological neural networks. However, ANN contains two main drawbacks [131] in the context with a large input size. ANN has connections between each neuron of a layer to each neuron of the following layers. It implements a matrix multiplication by a matrix of separate parameters. Besides, each element of weight matrix in ANN is utilized only once to compute the output of next layers.

CNN contains characteristics including spare connectivity, parameter sharing, and equivariant representations [131]. Consequently, it offers opportunities to overcome the disadvantages of ANN. Spare connectivity means that each neuron in a layer can interact indirectly with a large portion of the input to make a kernel smaller. With the parameter sharing characteristic, each kernel element is utilized at every position of the input to use the same parameters for more than one. The third characteristic indicates that a translation of input features results in an equivalent translation of outputs. These characteristics result in a reduction of memory requirements and processing time to obtain better performance compared to ANN.

3.2.2 Characteristics of QUIC-based applications

Fig. 3.2 show differences between QUIC-based applications related to flow-based features. In this paper, we use a collected dataset [123] with five applications containing VoIP (VC), chat (C), video streaming (VS), Google play music (GPM) and file transfer (FT).

In QUIC-based applications, QUIC is responsible for establishing, finishing connections, and transferring data between end-users. There are significant differences between



(a) Small packets.

(b) Large packets.

Figure 3.2: Percentage of small and large packets in flows.

these applications. In chat application of Google Hangout, all packets transferred between end-users are supported by QUIC, but the data transferred is insufficient. Therefore, most packets in flows are small packets whose packet lengths are less than 150 bytes. It is similar to VoIP of Google Hangout. In Fig. 3.2a, a range of small packets in the flows of chat and VoIP is larger than the figure for others.

In file transfer, video streaming and Google play music, servers and clients frequently transfer a large data, so there is a high number of large packets whose packet lengths are bigger than 1000 bytes in the network flows. In Fig. 3.2b, the percentage of large packets in the flows of these applications is over 60 percent. Moreover, the clients frequently reply small packets to the servers, so the percentage of small packets of these services is approximately 30 percent, except for Google play music (approximately 15 percent). However, the range of percentage of small packets in these applications is slightly tight.

3.3 Traffic Classification Approaches

In this section, we present related work on traffic classification including port-based approaches, payload-based approaches, statistic-based approaches and DL-based methods.

3.3.1 Port-based Approaches

Traditionally, the application class is identified according to a well-known port number in a packet. In the past, a particular application is registered to a specific port number in IANA (Internet Assigned Numbers Authority) [132]. The port-based approaches analyze

the packet to obtain the port number and identify the corresponding application class. Tab. 3.1 indicates ports registered by IANA and corresponding applications. For instance, SSH applications use port 22, and HTTP applications use port 80 for data transmission.

Table 3.1: Registered port numbers by IANA for several applications.

Registered ports	Applications
20	FTP Data
21	FTP Control
22	SSH
23	TELNET
25	SMTP
53	DNS
80	HTTP
161	SNMP
443	HTTPS

This approach is no longer available due to inaccuracy of classification results [71]. In fact, many applications use port numbers from a well-known application (e.g., port 443, etc.). Besides, many network administrators use NAT to change the port number of packets in the network.

3.3.2 Payload-based Approaches

An alternative for the port-based approach is the payload-based approach, called Deep Packet Inspection. This approach analyzes the payload of packets to obtain application's signature and identify a corresponding application. Tab. 3.2 shows the signatures for several P2P (Peer-to-Peer) applications [133].

Table 3.2: Signatures for several P2P applications.

Application	Signatures	Protocol
eDonkey 2000	0xe319010000, 0xe53f010000	TCP, UDP
BitTorrent	"0x13Bit"	TCP
Ares	GET hash:, Get sha1:	TCP

Adami et al. [50] identify the VoIP traffic according to the signatures of SIP protocol (e.g., content-type: "application/sdp", etc.). This approach can achieve good performance, but it contains three main drawbacks. First, searching the signatures in the payload of packets requires high resource consumption in the network devices. Second,

the packet's payload is hidden due to encrypted traffic nowadays, so this leads to many obstacles for signature extraction. Finally, this approach requires updating the signature frequently for a new version of applications or new applications.

3.3.3 Statistic-based Approaches

The emerging of statistic-based approaches using ML is a potential solution to overcome the limitations of port and payload-based approaches. This method extracts flow-based features corresponding to network flows and analyzes it using the ML technique to classify the network flows into different applications.

Williams et al. [134] proposed a traffic classification using statistical-based approach. This method extracts the flow-based features from all packets of flows and analyzes it using ML algorithms (e.g., Naive Bayes, C4.5, Bayesian Network, etc.) to classify network traffic into FTP, Telnet, SMTP, DNS, and HTTP. The flow-based features contain a protocol, flow duration, flow volume in bytes, packet length, and inter-arrival time in the flows. Similarly, Fathi-Kazerooni et al. [126] proposed a traffic classification approach using the random forest algorithm. This approach collects ten flow-based features from all packets of flows and analyzes it to identify six applications including Chrome, Google Driver, One Drive, One Note, Spotify and Whatapp. However, these approaches are not able to early detect applications because these approaches extract and analyze flow-based features after the flows terminate.

Many studies focused on statistical-based approaches for early application detection. Amaral et al. [75] proposed a novel traffic classification approach for the SDN environment. The objective is to collect 12 features from the first five packets of flows (e.g., packet size, packet time-stamp, inter-arrival time, etc.) and analyze it using ML algorithms (random forest, stochastic gradient boosting and extreme gradient boosting) to identify eight applications. Besides, Lopez-Martin et al. [71] proposed a traffic classification approach for IoT network. This approach extracts six time-series features of the first 20 packets of flows and analyzes it using DL algorithms (RNN, CNN and CNN+RNN) to classify network traffic into 108 applications. However, this approach uses the port number in traffic classification, so this can results in uncertain results when many applications use the same ports (e.g., port 445, etc.). Moreover, the statistic-based approaches require to identify pre-defined flow-based features with human intervention. Consequently, these approaches require a knowledge base about applications.

3.3.4 DL-based Approaches

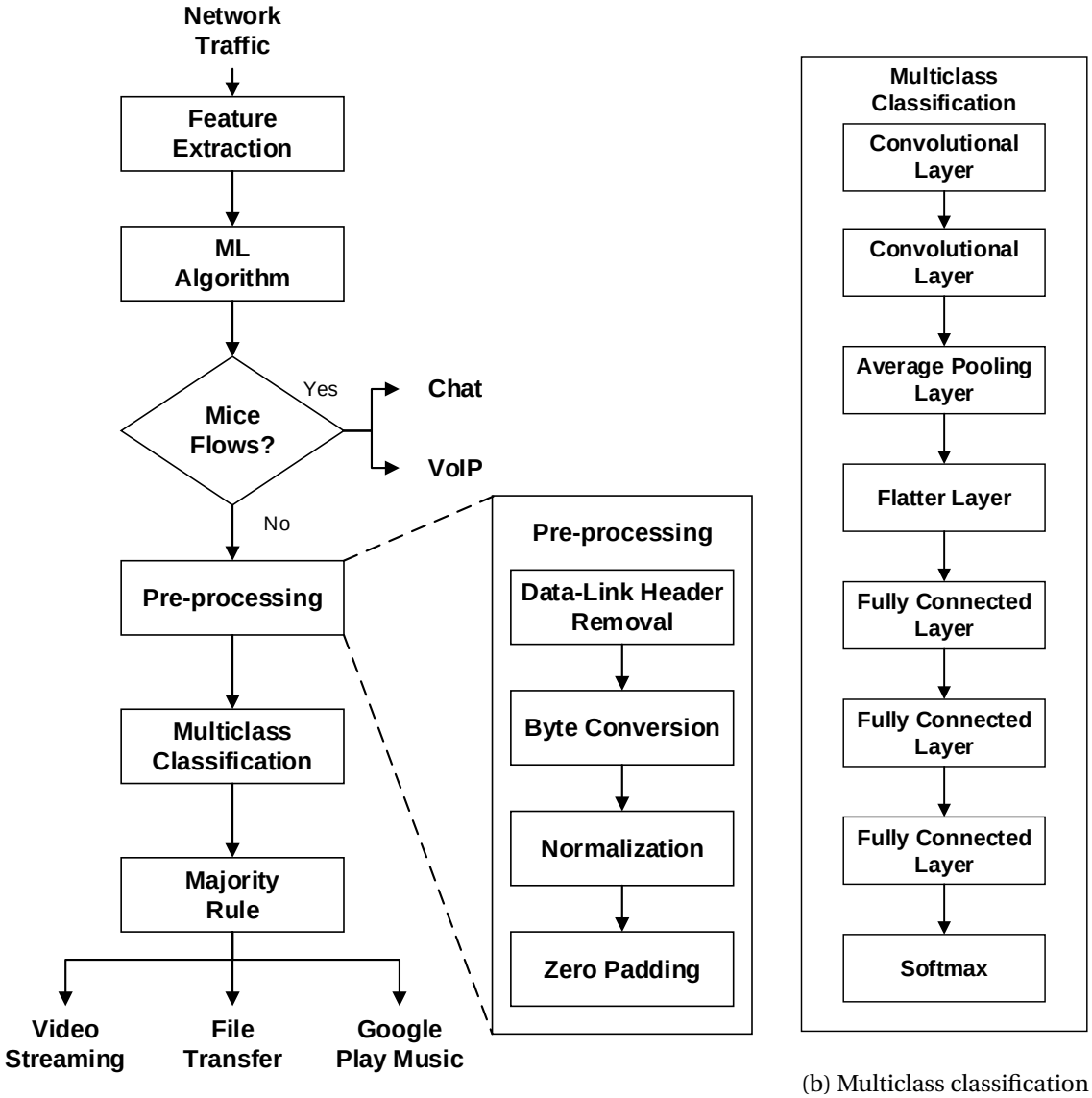
The DL-based approach is a variant of the statistic-based approach. The main goal of the DL algorithm is to combine the feature extraction and classification modules into one module. This approach extracts packet-based features (implicit features) according to analyzing the packet's payload using the DL algorithm, unlike the statistic-based approach which extracts the pre-defined flow-based features. Then, the packet-based features are analyzed according to a classifier in the DL algorithm to identify the application class.

Lotfollahi et al. [128] proposed "Deep Packet", a novel traffic classification using DL algorithms. Deep Packet processes 1500 bytes in the payload of packet to obtain a vector. Next, this vector is analyzed in the DL algorithms to classify network traffic into email, chat, FTP, Skype, torrent, etc. In this approach, the authors evaluate the performance of two DL algorithms including stacked autoencoder and CNN. However, using all bytes in the packet's header (e.g., port number, etc.) can lead to uncertain results because many applications use the same ports. Similarly, Wang et al. [85] analyze the first 784 bytes in each packet using CNN to identify six VPN and six non-VPN applications. Moreover, Wang et al. [84] proposed an encrypted traffic approach using the DL algorithm in the home gateway. This method processes the first 1480 bytes of each packet and analyzes it in ML algorithms to identify 15 applications in the context of encrypted traffic (e.g., Facebook, Gmail, SFTP, etc.). To select an appropriate algorithm, the authors evaluate the performance of multi-layer perceptron (MLP), stacked autoencoder and CNN. However, these DL-based approaches only take into account the payload of a packet in network flows to identify the application class. To improve the performance of traffic classification, Lim et al. [135] propose a traffic classification using multi-layer LSTM (Long Short-Term Memory) which considers payload of many packets in a network flow. This approach analyzes the payload of the first 30 packets in a flow to create a 3-dimensional vector and analyzes it using multi-layer LSTM to identify eight applications.

The existing studies focus on traffic classification approaches for non-encrypted traffic or encrypted traffic including HTTPS and VPN traffic. Therefore, in this thesis, we focus on a traffic classification approach for QUIC traffic. After investigating the characteristics of QUIC-based applications, we realized that DL-based approaches are not effective for several applications (e.g., chat and VoIP, etc.). Consequently, we propose a hybrid QUIC traffic classification approach using both flow-based features, packet-based features and DL algorithms.

3.4 Proposal: Novel Traffic Classification Method for QUIC Traffic

In this thesis, we propose a new traffic classification method using flow-based features, packet-based features and CNN. The detail is described as in Fig. 3.3a. There are two kinds of network flows: mice flows (small continuous flows in total bytes) and elephant flows (large continuous flows in total bytes). Therefore, the proposal comprises two main stages of classification. In the first stage, flow-based features are analyzed in ML algorithms (e.g., random forest, SVM, etc.) to identify chat and VoIP applications (mice flows). After that, packet-based features of elephant flows are extracted and analyzed according to CNN to classify elephant flows into file transfer, video streaming, or Google play music.



(a) Overall structure.

(b) Multiclass classification module using CNN.

Figure 3.3: Novel traffic classification approach for QUIC traffic.

3.4.1 Traffic Collection

In traffic classification, we need to collect network traffic and analyze it to identify the application classes. Network traffic can be collected directly from Openflow switches in the SDN environment, but it is not effective with a tremendous amount of network traffic. Consequently, it is necessary for a network monitoring standard supporting sampling technique to reduce traffic volume. There are two sampling techniques including flow sampling and packet sampling. Flow sampling (e.g., NetFlow, etc.) samples a flow with a specific probability, aggregates all packets of this flow into flow records, and exports towards collectors. Packet sampling (e.g., sFlow, etc.) samples each packet with a specific probability and aggregates sampled packets into sFlow records and exports to collectors. The flow sampling technique can collect too many packets of several flows and no packets for other flows. Therefore, it is ineffective for traffic classification. However, we can partly collect packets of all flows in the network with the packet sampling technique. Therefore, sFlow [59] is used here in the traffic classification approach. sFlow was widely used by vendors including IBM, HP, OpenVswitch and so on.

sFlow contains two main elements including sFlow agents and sFlow collectors. sFlow agents are implemented at ingress nodes (edge nodes) in the network. When there is network traffic traversing the sFlow agents, it will sample these packets, aggregate them and export to the sFlow collectors for the traffic classification.

3.4.2 Flow-based Features

In the first classification stage, flow-based features are used to identify chat and VoIP applications. According to subsection 3.2.2, there are differences between five applications, particularly packet length. Therefore, we take into account eight flow-based features in the first classification stage: average payload length, percentage of small, medium, and large packets in the network flows in both directions between clients and servers. The length of small, medium and large packets range from 0 to 150, 150 to 1000 and over 1000 bytes, respectively.

3.4.3 Pre-processing

The pro-processing module processes QUIC packets of elephant flows, converts its payload into bytes and normalizes into decimal values for further modules. This module

contains four main steps consisting of data-link header removal, byte conversion, normalization and zero padding.

The data-link header contains information related to the physical layer, which plays an important role in frame forwarding in the network. However, this information is useless for traffic classification, so the data-link header will be filtered in the data-link header removal step. Besides, we only use the payload of packets because the remaining information in the packets (e.g., port number, etc.) can result in uncertain results in the traffic classification. After that, the payload of packets is converted from bit to byte to reduce an input size. Then, these bytes are normalized by dividing by 255, the maximum value for a byte. CNN requires the same input length while the length of QUIC packets varies from over 50 to approximately 1400 bytes. Therefore, the packets are inserted zero values in the zero padding step to have a similar length for each packet. If the packet length is less than 1400, these packets are padded with zero values at the end of packets. Finally, each packet is represented by a vector containing 1400 decimal values.

3.4.4 Proposed Traffic Classification Method

Fig. 3.3a describes the overall of the proposed traffic classification method. First, network traffic is collected, and then the first few packets of each flow are processed in the feature extraction module to extract eight flow-based features representing the network flow. The more packets in a flow we use, the higher accuracy we can obtain. However, it results in a high processing time. Therefore, we consider the first twenty packets of each flow to extract flow-based features as in existing study [71]. Then, these features are analyzed in ML algorithms which classify the network flows into chat, VoIP and elephant flows. ML algorithm is considered as a classifier for identifying the application class. There are many ML algorithms (e.g., random forest, SVM, MLP, etc.), but the random forest algorithm outperforms the others [136] (refer to experimental results). Consequently, random forest is considered as a ML algorithm in the first classification stage. After identifying chat and VoIP applications, the remainder of network traffic (elephant flows) is processed in the second classification stage to classify elephant flows into video streaming, file transfer and Google play music application.

First, the packet of elephant flows is processed in the pre-processing module to extract the vector of 1400 decimal values representing each packet. Then, this vector is analyzed in the multiclass classification module to identify the application class. The structure of

this module is described in Fig. 3.3b. The size of input vector is large (1400 values), so ANN as well as traditional ML algorithms (e.g., SVM, MLP, etc.) are not effective in data processing [137]. Besides, CNN contains characteristics including spare connectivity, parameter sharing, and equivariant representations. This offers opportunities to learn more effective representations in comparison with traditional machine learning algorithms [137]. Therefore, CNN is taken into account in the multiclass classification module.

The structure of the multiclass classification module contains five essential layers including convolutional layer, average pooling layer, flatter layer, fully connected layer and softmax. The convolutional layer is considered as a feature learning to extract packet-based features representing a packet in elephant flows. The average pooling layer replaces the input vector at a certain location with an average value of nearby values while the flatter layer flattens an output obtained from the average pooling layer into a one-dimensional vector. In the fully connected layer, each neuron has complete connections to each neuron in the following layers. The softmax layer is a generalization of a logistic function that squashes an n-dimensional vector into an n-dimensional vector with element values between 0 and 1. The combination of the fully connected layer and softmax is similar to a classifier that indicates a relationship between the input vectors and the application classes to classify the elephant flows into different kinds of applications.

In each elephant flow, we extract packet-based features of the first few packets, classify these packets and aggregate classification results to identify application class for the elephant flow according to the majority rule. The majority rule is a decision rule that selects an alternative with high votes. If most packets among these packets are classified as an application, the flow will be assigned to that application. If the vote of applications is equal, a classification probability of these packets is used to identify the application class. The higher number of packets in each flow classified, the higher accuracy we can obtain. Nevertheless, this can lead to a high processing time. Therefore, we consider the first ten packets of each elephant flow to identify the application class. The detail of the majority rule is described in Algorithm 1. $label(id)$ is an application class of flow after the classification and $prob_{ij}$ is a probability of packet $packet_i$ classified as application app_j in the flows. argmax is a function that returns an index of the highest value in the input. m and n are a number of application classes and a number of packets required for the classification in each flow, respectively.

The proposal contains two main algorithms including random forest and CNN. There-

Algorithm 1 Majority rule

Require: id, FlowId, $label$, Flow$Label$, app, C$ount$, $prob$, Prob, a

while id in FlowId **do**
  for j = 1 to m **do**
    for i = 1 to n **do**
      if $label(id)$ is app_j **then**
        $a_{ij} = 1$
      else
        $a_{ij} = 0$
      end if
    end for

$$Count_j = \sum_{i=1}^{n} a_{ij}$$

$$Prob_j = \sum_{i=1}^{n} prob_{ij}$$

  end for
  if all C$ount_j$ unique in C$ount$ **then**
    Flow$Label(id)$ = argmax(C$ount$)
  else
    Flow$Label(id)$ = argmax(Prob)
  end if
  Return application class Flow$Label(id)$
end while

fore, time complexity of the proposal is related to time complexity of these algorithms. Regarding the time complexity of the random forest algorithm, its time complexity in the training phase is O($n_0 * log(n_0) * d * k$) where n_0 is the number of input sample, d is dimensional of data and k is the number of trees in the forest while the time complexity in testing phase is O($k * d^{'}$) where $d^{'}$ is the depth of tree [138].

As for the time complexity of CNN, total time complexity of all convolutional layers [139] is estimated by Equ. 5.1:

$$O(\sum_{l=1}^{d} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2). \tag{3.1}$$

where d is the number of convolutional layers, n_{l-1} is the number of input channels of the l-th layer, n_l is the number of filters in the l-th layer, s_l is the spatial size of the filter and m_l is the spatial size of the output feature map.

This time complexity is applied in both training and testing phase, and training time is approximately three times of testing time.

3.5 Experimental results

3.5.1 Dataset specification

In the experiment, we implement a testbed to collect network traffic and publically provide a real traffic classification dataset from March to April 2018 [123]. Network traffic of video streaming, chat, VoIP and Google play music applications are captured by watching videos on Youtube, accessing Google Hangout and Google play music automatically according to Selenium WebDriver [121] in Google Chrome running on Ubuntu 16.04 OS. Network traffic of file transfer application is collected by transferring data between servers and clients using *quic-go* [122], an open-source library supporting QUIC protocol. As a result, approximately 150 GB of network traffic was collected, including over 20,000 network flows of five QUIC-based applications. The detail is described in Tab. 3.3. In the first classification stage using the random forest algorithm, the dataset is divided into 5-fold. Concretely, the training and testing phase comprise 20 and 80 percent of the dataset, respectively. In the multiclass classification, the dataset is also divided into 5-fold. The training, validation, and testing comprise 45, 5, 50 percent of the dataset, respectively.

Table 3.3: Dataset specification.

Application	Number of samples (flows)
Chat	2,783
VoIP	2,608
File transfer	4,451
Video streaming	5,844
Google play music	4,349

The proposal is written in Python using Keras library [140] and scikit-learn tools [141]. Besides, all experiments are implemented in a workstation (Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.6 GHz and 16 GB of RAM) running Ubuntu 16.04.

3.5.2 Performance metrics

In this section, we present measures [142] to evaluate the performance of the proposal consisting of precision, recall, and F1-score (F-measure). Precision is a percentage of relevant flows that are retrieved while recall is a percentage of retrieved flows that are relevant. F1-score represents a harmonic mean between precision and recall. The detail of

these parameters are described in equation 3.2, 3.3, 3.4.

$$Precision = \frac{TP}{TP + FP}. \tag{3.2}$$

$$Recall = \frac{TP}{TP + FN}. \tag{3.3}$$

$$F1 - score = \frac{2}{1/Recall + 1/Precision}. \tag{3.4}$$

Quality of overall classification can be evaluated in two ways [142]. In macro-averaging, a metric is averaged over all classes that are treated equally. Micro-averaging is based on cumulative True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) of the dataset.

3.5.3 Performance analysis

Table 3.4: Performance metrics of ML algorithms in the first stage of classification.

Class	Precision			Recall			F1-score			Nb Samples
	MLP	SVM	RF	MLP	SVM	RF	MLP	SVM	RF	
Chat	0.9800	0.9759	0.9671	0.0072	0.8448	0.9454	0.0143	0.9057	0.9562	2227
VoIP	0.6298	0.9460	0.9671	0.9335	0.9608	0.9680	0.7529	0.9534	0.9675	2086
Elephant flows	0.8690	0.9582	0.9757	0.9628	0.9800	0.9797	0.9135	0.9690	0.9777	11351
Micro	0.8234	0.9587	0.9724	0.8234	0.9587	0.9744	0.8234	0.9587	0.9734	15664
Macro	0.8263	0.9600	0.9679	0.6278	0.9286	0.9634	0.5536	0.9427	0.9656	

Table 3.5: Time complexity of ML algorithms in the first stage of classification.

Algorithm	Average Training Time (ms)	Average Testing Time (ms)
MLP	0.25505	0.00042
SVM	7.76140	0.01472
RF	0.11787	0.03165

This section is dedicated to evaluating the performance of the proposed traffic classification method which contains two stages of classification. In the first stage, we evaluate the performance and time complexity of ML algorithms (e.g., SVM, random forest, etc.) to select an appropriate ML algorithm. In the second stage, we evaluate its performance over different scenarios including different subsets of the input vector and various loss

functions. Besides, we evaluate the running time of the proposed traffic classification approach.

3.5.3.1 First Stage of Classification

The performance of MLP (Multilayer Perceptron) [143], SVM (Support-Vector Machine) [144] and RF (Random Forest) [145] are described in Tab. 3.4. The micro and macro precision, recall and F1-score of MLP are the lowest in three ML algorithms with below 83 percent while the figures for RF are the highest with approximately 97 percent. Tab. 3.5 indicates the time complexity of ML algorithms in the first stage of classification. We calculate the training time and testing time for the training and testing dataset and then generate the average training and testing time for a sample in these datasets. Although MLP does not have good performance, its traing and testing time are the lowest in three algorithms with 0.25505 and 0.00042 ms, respectively. The average testing time of SVM is lower than the figure for RF with 0.01472 and 0.03165 ms, respectively. However, the difference between macro recall of SVM and RF is approximately 4 percent. Therefore, RF is selected as a ML algorithm in the first stage of classification.

With RF algorithm, the precision, recall and F1-score of chat and VoIP applications are impressive, with approximately 96 percent, except for the recall of chat application which is 94.54 percent. Chat and VoIP applications have similar value regions in flow-based features, so several flows of VoIP application are classified incorrectly to chat application. This leads to lower results for chat application. In particular, the performance of elephant flows (file transfer, video streaming, and Google play music) are over 97 percent. The reason for it is that there is a difference in flow-based features between chat, VoIP and elephant flows. As a result, the micro and macro-averaging precision, recall and F1-score of the dataset achieve over 96 percent.

3.5.3.2 Second Stage of Classification

After classifying the network traffic into chat, VoIP or elephant flows, each packet of elephant flows is pre-processed in the pre-processing module to obtains a vector of 1400 decimal values, and this vector is analyzed in the second stage of classification to classify into video streaming, file transfer or Google play music.

In the first scenario, we investigate the influence of different subsets of input vectors in the second stage of classification. We implement the experiments on five subsets with

the sparse_categorical_crossentropy loss function to select the subset with good performance. Fig. 3.4 indicates the macro-averaging precision, recall and F1-score of five subsets including the first 300, 600, 900, 1200 and 1400 decimal values. There is an upward trend in the macro-averaging precision, recall and F1-score in five subsets. The performance for the first 300 and 900 decimal values are not good, with over 70 percent for the macro-averaging F1-score. The reason for it is that the precision of video streaming application with the first 300 values is 64.34 percent. Besides, flows of video streaming are classified incorrectly to the flows of file transfer, so this reduces the recall of file transfer in the experiment of the first 300 values. It is similar to the subset of the first 900 values. The discriminant power with the subset of 1400 values is larger than the figure for others, so the macro-averaging precision, recall and F1-score are the highest, with approximately 99 percent.

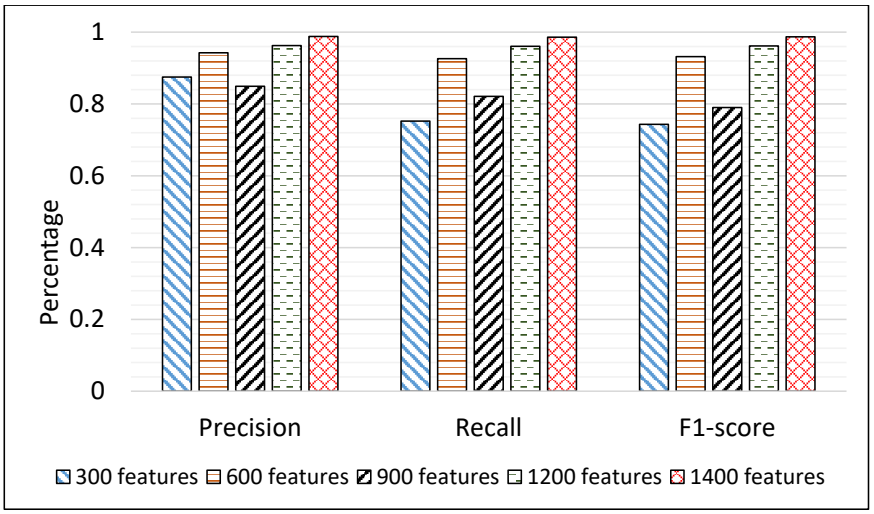


Figure 3.4: Macro-averaging precision, macro-averaging recall and macro-averaging F1-score in different subsets.

In the second scenario, we investigate the influence of loss functions [146] on the convolutional neural network (Fig. 3.5). As the above discussion, the performance of the dataset with 1400 values is the highest in five subsets, so we use this dataset in the second scenario. In this scenario, we evaluate the performance of three kinds of the loss function (categorical_hinge (Hinge), mean_squared_error (MSE) and sparse_categorical_crossentropy (SCCE)) to select the appropriate loss function. Hinge loss is the loss function that is notably used in Support Vector Machine for "maximum-margin" classification. In Fig. 3.5, the performance of hinge loss is not good because the macro-averaging F1-score is over 20 percent. The reason is that many flows of file transfer are classified incorrectly.

Mean squared error is the loss function that measures an average square of error. The performance of MSE loss is higher than the figure for hinge loss. A noticeable feature is that the performance of SCCE is the highest in three loss functions. The macro-averaging of SCCE is the largest, with approximately 99 percent.

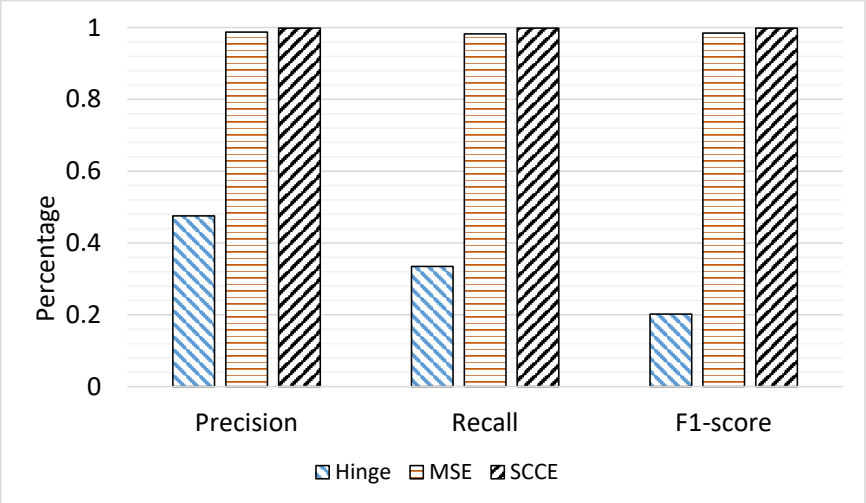


Figure 3.5: Macro-averaging precision, macro-averaging recall and macro-averaging f1-score in different loss functions.

Briefly, we consider the dataset with 1400 values corresponding to 1400 bytes in the payload of packets and the SCCE loss function in the second stage of classification. Fig. 3.6 indicates the overall results of the proposed traffic classification approach with these selections. The micro and macro-averaging of precision, recall, and F1-score are over 96 percent. These results are summarized according to the results of the first and second stage of classification.

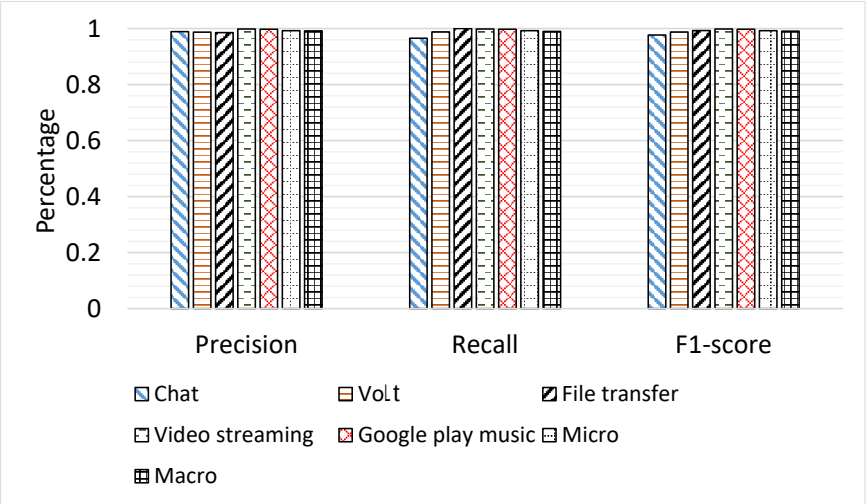


Figure 3.6: Precision, recall and F1-score of the proposed method on five QUIC-based applications.

3.5.3.3 Time Analysis

Regarding the time complexity, the average training and testing time of RF in the first stage of classification are 0.11787 and 0.03165 ms, so this stage can process up to over 33,000 network flows per second. The average training and testing time of CNN in the second stage of classification are 11.1842 and 3.4 ms, so this stage can process approximately 300 network flows per second. Moreover, it is reported that Openflow switches can handle from 150 to 750 new flows per second [147]. Similarly, Aliyu et al. [148] reported that HP ProCurve 5406zl with K.15.10.0009 firmware could handle 275 new flows per second. Therefore, the proposed traffic classification approach can operate effectively with SDN switches.

3.6 Conclusion

In the chapter, we present the novel traffic classification method using the convolutional neural network, flow-based features, and packet-based features to identify various QUIC-based applications. There are two main classification stages in the proposed method. The first one is to classify the network traffic into chat, VoIP or elephant flows, whereas the second one aims to classify the elephant flows into video streaming, file transfer or Google play music. The experimental results demonstrate that the proposed method can detect five QUIC-based applications with high accuracy (over 96 percent).

Traffic classification aims to identify application classes for network management (e.g., application-aware traffic engineering, network slicing, etc.). Therefore, time complexity is an essential factor in traffic classification. In the future, we will develop the proposed traffic classification approach with more suitable time complexity. Besides, we will construct a larger dataset, deeply investigate the flow-based features, and compare the performance of the proposed method with existing traffic classification approaches to make the proposed method more reliable.

After collecting network parameters from parameter measurement and application class from the traffic classification module, we need to analyze it to identify network anomalies. The objective is to identify when problems happen in the network and make appropriate decisions to optimize network performance. Consequently, in the next chapter, we present a proposed anomaly detection approach using machine learning algorithm.

Chapter 4

Anomaly Detection

«The computer was born to solve problems that did not exist before.»

Bill Gates

Chapter 4 presents a proposed anomaly detection approach to identify network anomalies. This approach is implemented in the anomaly detection module which is depicted as in Fig. 2.2.

Contents

4.1 Introduction

An anomaly is an observation that deviates from other observations to raise a suspicion that it is generated by a different mechanism [149]. According to its nature, the anomaly can be categorized into point, collective, and contextual anomalies [150]. The point anomaly is a deviation of a data sample from usual behaviors. This kind of anomaly is simplest and widely studied by the research community. For example, packet loss in the network every day is less than 1 percent, but it increases to more than 20 percent on a specific day. This situation is considered as a point anomaly. The collective anomaly happens when a collection of data samples behaves anomalously with an entire dataset. Anomalous behavior is not considered as a collective anomaly, but point anomaly happening continuously in duration is considered as a collective anomaly. The contextual anomaly is an event or behavior considered as an anomaly depending on the context. In contextual anomaly, there are two features including contextual (e.g., geographic coordinates in spatial data, time in time-series data, etc.) and non-contextual features (e.g., an indicator determining the context of anomaly, etc.). For example, the throughput of network systems increases significantly at 0 AM every day due to regular backup. This activity creates an outlier related to traffic volume, but it is not considered as an anomaly. However, an outlier at 1 AM can be considered as a contextual anomaly.

Anomaly detection plays an important role in the network. It helps NO to detect problems in the network early to reduce negative impacts of problems quickly (e.g., high loss, low latency, etc.). Therefore, there are many anomaly detection techniques over the past ten years including knowledge-based mechanisms, rule inductions, ML-based mechanisms and so on. Despite many available techniques, following are obstacles in the anomaly detection [149]:

- Boundary between normality and anomaly is not clear. There is no universal rule to decide whether a behavior is normal or anomalous, except a subjective judgment. Consequently, data can contain noise, which leads to low performance.
- There is a lack of public labeled dataset in the anomaly detection.
- Designing a general anomaly detection technique is not accessible because there are specific requirements for each context. For instance, an anomaly detection technique for wired networks may have limited usage in the wireless network.

4.2 Anomaly Detection Approaches

Maha et al. [151] grouped the studies on anomaly detection into four main mechanisms including knowledge-based mechanisms, rule inductions, information theory and ML-based mechanisms. In the following section, we present a brief explanation of these mechanisms and highlight its advantages and drawbacks.

4.2.1 Knowledge-based Mechanisms

This mechanism (called expert system) requires human knowledge about anomalies to obtain its patterns and encodes these patterns in a method that is realized by the machine. Consequently, the knowledge-based mechanism contains two main steps. First, network experts analyze a large amount of data manually to identify anomaly patterns. Then, these patterns are deployed in the network systems to detect anomalies in the future. The knowledge expert can be structured in different ways:

- Rule-based systems: It contains *if-then* rules related to events or alarms. It requires an assumption that the same network conditions will result in the same consequences.

- Statistic-based systems: It is an advanced version of the rule-based system. The objective is to create rules based on statistical calculation. Casas et al. [152] proposed mPlane, a distributed measurement platform for ISP networks. mPlane allows detecting anomalies based on similarity level between distribution of network parameters and baseline. The baseline value is the value of network parameters when there is no performance problem in the network. The similarity level between distributions is calculated on Kullback-Leibler divergence. Alvarez Cid-Fuentes et al. [96] proposed an anomaly detection approach in distributed systems. This approach detects anomalies using a statistical model. If the radial basis function of an instance is bigger than a threshold, this instance will be considered as an anomaly. Similarly, Chen et al. [153] proposed a matrix differential decomposition-based anomaly detection and localization in NFV (Network Function Virtualization). This method identifies the anomalies in NFV environments according to the deviation between measured and reference RTT matrix.

The knowledge-based mechanisms suffer from low false positives because it only detects anomalies specified by network experts. However, it contains two main problems.

First, it is costly and time-consuming because list of anomaly patterns should be exhaustive. Second, it is not effective with new kinds of anomalies that are not defined in the rules. Therefore, the research community moves toward other anomaly detection approaches.

4.2.2 Rule Inductions

This approach uses a learning algorithm to learn rules in a deterministic way for anomaly detection. In this approach, there are two main phases including training and testing. In the training phase, this method learns the rules from a large amount of labeled data. Then, the testing phase detects the anomalies according to these rules. There are many learning algorithms for rule induction including decision tree, association rules and so on. The decision tree is a learning algorithm that uses a tree-like decision model for anomaly detection. Association rules generate the rules based on a support and a confidence. Bohmer et al. [154] proposed a novel anomaly detection approach based on association rule mining to reduce false positives in anomaly detection. However, it is not effective with high-dimensional data due to high processing time and accuracy issues.

4.2.3 Information Theory

This approach is related to an assumption that anomalies in a dataset vary its information content. This approach is based on measuring an information indicator of data flows to build an appropriate anomaly detection model. According to related work [149], there are many measuring functions including entropy, conditional entropy, relative entropy, relative conditional entropy and information gain. For example, an anomaly detection model needs to look for features with high information gain. If all features have low information gain, performance of anomaly detection is not good. In other words, the higher the information gain of features, the better the performance. This approach is to obtain an appropriate classifier for anomaly detection, but it is sensitive to noise.

4.2.4 ML-based Mechanisms

This approach analyzes data points using ML algorithms to classify it into anomaly or normality. In this approach, there are three main steps. First, we identify features representing for a data point. Then, we analyze the training data using ML algorithms to learn a model for the classification. Finally, we use this model to classify new data. There are

many ML algorithms including decision tree, SVM, neural network, and so on.

In fact, many studies have used decision tree in anomaly detection [155, 156]. This approach detects anomalies using a tree-like decision model. The time complexity of this approach depends on the time complexity of algorithms used to construct the trees (e.g., ID3, C4.5, etc.). Although this approach can detect anomaly effectively, it contains two main drawbacks: high dimensional data and over-fitting problem. Consequently, Random Forest (RF) algorithm is studied in the anomaly detection by the research community [157]. Random forest builds multiple decision trees on a different subset of input dataset, and aggregates the output of these decision trees to make the final decision. In this case, it can overcome the over-fitting problem. In the random forest algorithm, a balance between accuracy and computational resources needs to be considered thoroughly. The higher the number of decision trees in the random forest algorithm, the more accurate the classification results and the higher computational resources.

Besides, Bayesian networks are studied in the anomaly detection [158, 159]. This approach can be easily and quickly implemented, but it requires an independent assumption between data features. This property is not always guaranteed in the high dimensional data where data features have a correlation. When the independence assumption is not guaranteed, it can lead to low performance.

Support-Vector Machine (SVM) is another algorithm which is taken into account in the anomaly detection [160, 161, 162]. Its objective is to find a hyperplane in data space to detect the anomaly based on a kernel function. However, with non-kernel functions, SVM can suffer from the over-fitting problem, leading to low performance in anomaly detection.

Despite good performance, the ML-based mechanisms contain two main disadvantages. First, it requires a labeled dataset to train an anomaly detection model. Second, it cannot effectively detect unknown problems which are not in the training dataset.

4.3 Proposed Anomaly Detection Approach using Machine Learning

Although there are many anomaly detection approaches, we implement an anomaly detection approach using machine learning in this thesis. The reason is that it offers opportunities to expand to detect anomalies caused by unknown problems easily. This can be achieved by increasing the number of features and expanding the training dataset.

Regarding the first drawback of this approach, we simulate root causes of anomalies to collect the training dataset. As for the second drawback, we can frequently update the anomaly detection model when there are new problems.

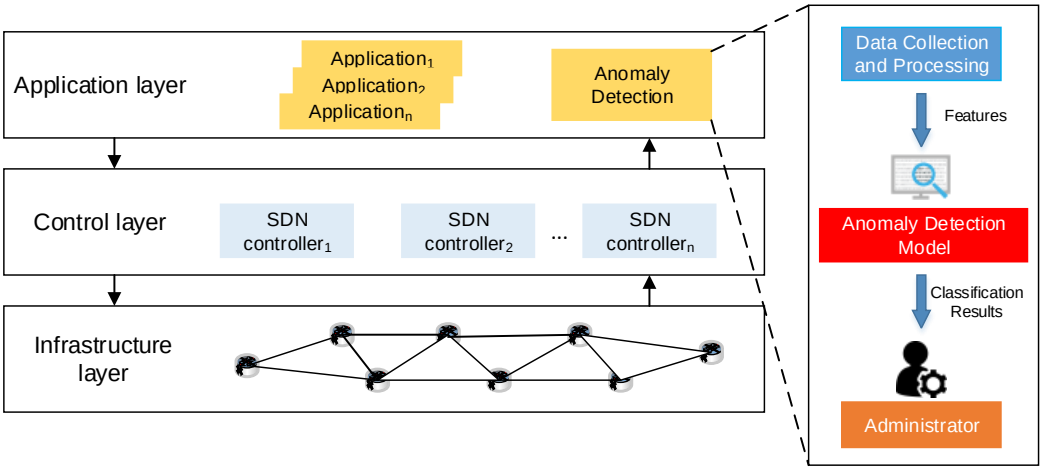


Figure 4.1: Overall architecture of ML-based anomaly detection mechanism in the SDN environment.

In this thesis, we take into account the anomaly detection mechanism in SDN environment because it offers global visibility and monitoring tools to facilitate anomaly detection (as discussed in Chapter 2). The overall architecture of the ML-based anomaly detection mechanism in the SDN environment is depicted as in Fig. 4.1. First, the network traffic from the infrastructure layer is collected and analyzed in the data collection and processing module to extract the network features representing network status. Then, these features are analyzed in the anomaly detection model which is built according to ML algorithms, to classify the network status into anomaly or normality.

4.3.1 ML-based Anomaly Detection Method

After collecting and extracting network features from data collection and processing module, we analyze these features using the ML-based anomaly detection method to identify the anomalies in the network. Network problems can lead to performance degradation and a fluctuation of network features. ML algorithms aim to learn this fluctuation to detect anomalies in the network. The detail of this approach is described as in Fig. 4.2.

There are two main phases in this method including training and testing phases. In the training phase, the network features will be analyzed according to a ML algorithm to obtain a pre-trained anomaly detection model for the testing phase. The objective of the anomaly detection model is to detect the anomalies in the network. There are different

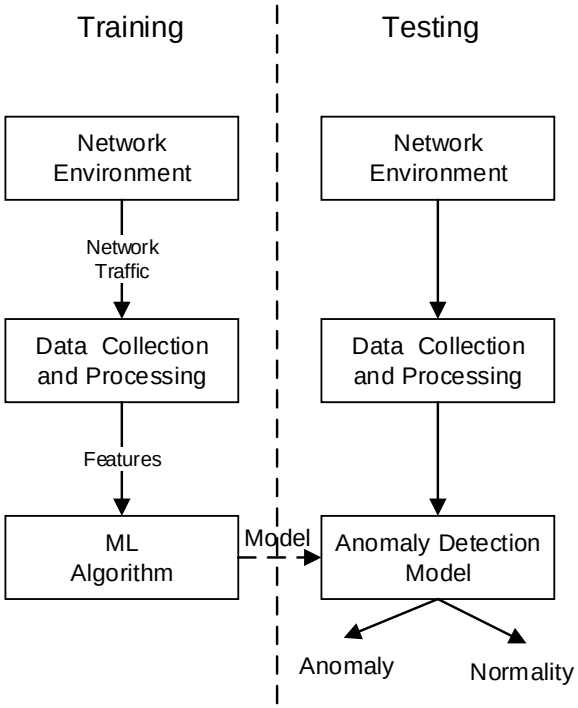


Figure 4.2: The ML-based Anomaly Detection Method.

kinds of ML algorithms for anomaly detection. Consequently, we evaluate several algorithms to select the appropriate one and balance between accuracy and time complexity. The considered ML algorithms includes Multi-Layer Perceptron (MLP) [163], Support-Vector Machine (SVM) [164], Random Forest (RF) [164], Adaptive Boosting (AdaBoost) [165] and Convolutional Neural Network (CNN) [166]. Concerning the CNN model, we use two convolutional 1D (1 dimensional) layer, a max Pooling 1D, a flatter layer and three fully connected layers. For the others algorithms, we adjust hyper-parameters such as depth of tree, number of estimators and so on. These configurations are chosen as in existing study [97].

4.3.2 Data Collection and Processing

This module collects and extracts network features representing network status (anomaly and normality) in the network. This module refers to the parameter measurement module in the proposed troubleshooting framework (Fig. 2.2). In this thesis, we consider congestion and simulate three root causes (link failure, switch failure and buffer overload) to obtain anomalous network status. We simulate link failure by generating a delay in links. Besides, switch failure is simulated by generating rule failure leading to a packet loss in the switches. Buffer overload is simulated by generating a massive amount of network traf-

Table 4.1: Considered network conditions.

Status		Static Network	Dynamic Network
Anomalous status	Link failure	Delay: 125ms	Delay: [25, 50, 75, 100, 125ms]
	Switch failure	Loss: 50%	Loss: [10, 20, 30, 40, 50%]
	Buffer overload	Bandwidth: 10 mbps, Sending rate: 60-100 mbps	Bandwidth: 10 mbps, Sending rate: 60-100 mbps
Normal status		Sending rate: 1-3 mbps	

fic exceeding the link's capacity. The anomalous network statuses are simulated in two scenarios including static and dynamic networks. In the static network, delay and loss parameters in the links are tied to a particular value. In contrast, in the dynamic network, link status is changed according to Gilbert-Elliot (GE) model [118] following a probability value after a given time (400 seconds). The probability of changing from normal to anomalous status and anomalous to normal status are p and r, respectively. If link status is anomalous, delay and loss are changed as in Tab. 4.1.

In each link, we extract three network parameters including latency, packet loss and link utilization. These parameters are discussed in the parameter measurement module (Chapter 2, Section 2.3.3). This choice is explained by a fact that these parameters are taken into account in many SLA (Service-level Agreement) of service providers [167]. The parameters on each link will be aggregated to calculate the parameter of a routing path (network features) which is used for anomaly detection. After that, these features are normalized into a value between 0 and 1 according to Max Normalization [119]. In anomaly detection, we consider time-series features, and a sample aggregates features of ten consecutive time steps to identify the anomaly. An input sample is a matrix 10×3 (number of time steps × number of features). The anomaly detection datasets are depicted as in Tab. 4.2.

Table 4.2: Anomaly Detection Datasets.

Status	Static Network	Dynamic Network
Anomalous	24452	44172
Normal	11075	11075

4.4 Experimental Results

4.4.1 Experimental Setup

In the experiments, *mininet* [168] is used to generate network topology with five spine nodes and two leaf nodes. The delay and loss parameters on each link is configured in *mininet* based on *TCLink* argument. In GE model, p and r are set to 0.81 and 0.07 as in [118]. The ML algorithms are implemented according to a library *scikit-learn* [141] in Python. The training, validation and testing phase comprise 76, 4, and 20 percent of the anonaly detection dataset.

Performance of ML algorithms is evaluated according to performance metrics: precision, recall, and F1-score (F-measure) [142]. Precision is the percentage of relevant flows retrieved, while recall is the percentage of retrieved flows that are relevant. F1-score (F-measure) represents a harmonic mean between precision and recall. The detail of these parameters are depicted in Equations 4.1, 4.2, 4.3. There are two ways to evaluate the quality of the overall classification including micro-averaging and macro-averaging value. In macro-averaging, a metric is averaged over all classes that are treated equally whereas micro-averaging is based on the cumulative True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) of the dataset.

$$Precision = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}. \tag{4.1}$$

$$Recall = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}. \tag{4.2}$$

$$\mathrm{F1}-score = \frac{2}{1/Recall + 1/Precision}. \tag{4.3}$$

4.4.2 Performance Analysis

There are two anomaly detection datasets including datasets in static and dynamic networks. We consider a balance between the performance and time complexity of ML algorithms to select an appropriate one. Tab. 4.3 shows the performance metrics of ML algorithms for the dataset in static network. In the dataset in static network, the range of network features between classes is sometimes overlapping, so this leads to noises in the dataset. The performance metrics of SVM are the lowest in four algorithms. The mi-

cro and macro precision, recall and F1-score are approximately 83.3 percent. The reason is that SVM does not perform well when the dataset contains more noises (e.g., classes are overlapping, etc.) [169]. RF builds multiple decision trees on different subset of the dataset and selects the class selected by most trees. The number of decision trees in RF is set to 100 as default in library *scikit-learn*. Consequently, RF can achieve good classification results. The precision, recall and F1-score of RF algorithm are close to 98.8 percent. AdaBoost is a ML algorithm using boosting ensemble method and decision tree as the individual model. The boosting ensemble method is to learn the individual model from the mistakes of previous models. Despite learning from the mistakes by increasing weight of misclassified samples, it can lead to uncertain results when dataset contains noises. Therefore, the precision, recall and F1-score of AdaBoost reduce from 1 to 1.5 percent in comparison with RF algorithm. Besides, The performance of CNN is the highest in four algorithms, with approximately 99 percent.

Tab. 4.4 indicates running time of ML algorithm for the dataset in static network. These values are the average training and testing time for a sample in the training and testing dataset. SVM has the largest training and testing time, with 0.9183 and 0.3940 ms. The testing time of RF and AdaBoost are nearly equal to 0.009 ms. Although CNN has the highest performance, its training and testing time are higher than the figures for RF and AdaBoost. In a fold, the training time of an epoch of CNN is 0.0548 ms. This value increase against the number of folds and epochs. In the thesis, we configure the number of folds and epochs to 20. Therefore, the total training time for a sample is 21.92 ms (0.0548·20·20). The testing time of CNN nearly doubles the figures for RF and AdaBoost.

Tab. 4.5 and 4.6 illustrate the performance metrics and time complexity of ML algorithms for the dataset in dynamic network. There is an imbalance between the number of normal and anomalous network status, so the normal network status is classified as anomalous network status in SVM algorithm. Therefore, the precision, recall and F1-score of normal network status are 0 in SVM algorithm. Moreover, CNN has the highest performance metrics in four algorithms, but its time complexity nearly doubles the figure for RF and AdaBoost. Regarding the balance between the performance metrics and the running time of ML algorithms, we consider RF as a ML algorithm in the ML-based anomaly detection method.

Table 4.3: Performance metrics of ML algorithms in anomaly detection for the dataset in static network.

Class	Precision				Nb
	SVM	RF	Adaboost	CNN	samples
Normal	0.7240	0.9881	0.9587	0.9830	2215
Anomalous	0.8950	0.9880	0.9834	0.9965	4891
Micro	0.8379	0.9880	0.9757	0.9923	7106
Macro	0.8095	0.9881	0.9710	0.9898	7106
Class	Recall				Nb
	SVM	RF	Adaboost	CNN	samples
Normal	0.7756	0.9734	0.9634	0.9923	2215
Anomalous	0.8661	0.9947	0.9812	0.9922	4891
Micro	0.8379	0.9880	0.9757	0.9923	7106
Macro	0.8209	0.9840	0.9723	0.9923	7106
Class	F1-score				Nb
	SVM	RF	Adaboost	CNN	samples
Normal	0.7489	0.9807	0.9610	0.9876	2215
Anomalous	0.8803	0.9913	0.9823	0.9944	4891
Micro	0.8379	0.9880	0.9757	0.9923	7106
Macro	0.8146	0.9860	0.9717	0.9910	7106

Table 4.4: Time complexity of ML algorithms in anomaly detection for the dataset in static network.

Algorithms	Training Time (ms)	Testing Time (ms)
SVM	0.9183	0.3940
RF	0.2040	0.0098
AdaBoost	0.1548	0.0094
CNN	0.0548	0.0163

Table 4.5: Performance metrics of ML algorithms in anomaly detection for the dataset in dynamic network.

Class	Precision				Nb samples
	SVM	RF	Adaboost	CNN	
Normal	0	0.9738	0.9060	0.9774	2215
Anomalous	0.7995	0.9850	0.9773	0.9946	8834
Micro	0.7995	0.9828	0.9630	0.9911	11049
Macro	0.3998	0.9794	0.9417	0.9860	11049
Class	Recall				Nb samples
	SVM	RF	Adaboost	CNN	
Normal	0	0.9395	0.9097	0.9783	2215
Anomalous	1	0.9937	0.9763	0.9943	8834
Micro	0.7995	0.9828	0.9630	0.9911	11049
Macro	0.5000	0.9666	0.9430	0.9863	11049
Class	F1-score				Nb samples
	SVM	RF	Adaboost	CNN	
Normal	0	0.9563	0.9079	0.9779	2215
Anomalous	0.8886	0.9893	0.9768	0.9945	8834
Micro	0.7995	0.9828	0.9630	0.9911	11049
Macro	0.4443	0.9728	0.9424	0.9862	11049

Table 4.6: Time complexity of ML algorithms in anomaly detection for the dataset in dynamic network.

Algorithms	Training Time (ms)	Testing Time (ms)
SVM	0.8439	0.4344
RF	0.1810	0.0103
AdaBoost	0.1615	0.0092
CNN	0.0403	0.0165

4.5 Conclusion

This chapter presents an anomaly detection approach using a ML algorithm to detect the anomalies in the network. Our contribution consists of an anomaly detection approach contains two main modules: i) data collection and processing and ii) anomaly detection model. The first module collects and extracts network features representing network status. Then, these features are analyzed in the anomaly detection model which is learned according to a ML algorithm. There are many ML algorithms, so we evaluate the performance and time complexity of these algorithms to select an appropriate ML algorithm. As a result, RF algorithm is selected as a ML algorithm in anomaly detection. The experimental results show that the micro and macro F1-score can achieve up to approximately 99 percent in the considered datasets.

After identifying the anomalies in the network, we need to implement a basic remediation approach to reduce the negative impacts of anomalies and guarantee the availability of the network until the root cause of anomalies is identified and solved. Therefore, in the following chapter, we present a QoE-based application-aware segment routing to make appropriate routing strategies corresponding to different applications and meet strict user's requirements for encrypted traffic.

Chapter 5

Temporary Remediation: SDN-based Application-aware Segment Routing for Large-scale Network

«The Internet is not just one thing»

James H. Clark

Chapter 5 presents a QoE-based application-aware segment routing in the SDN environment for encrypted traffic. This approach is implemented in the temporary remediation module which is depicted briefly as in Fig. 2.2.

Contents

5.1 Introduction

According to the related work [4, 5], there are many remediation approaches including load balancing, routing and so on. Moreover, many studies proved the performance of routing in problem remediation [89, 90]. Therefore, we consider an application-aware routing in the temporary remediation module. The application-aware routing which implements different kinds of routing policies corresponding to differentiated applications, can overcome this drawback. To facilitate the application-aware routing, it needs two requirements related to a routing technique supporting Traffic Engineering and a network architecture that can obtain a global view of network.

For the first requirement, Multi-Protocol Label Switching (MPLS) is a potential solution which is deployed by network operators to improve their IP networks. However, MPLS contains three main drawbacks [170, 171]. First, it requests an IP network to maintain an explicit state at network nodes along an MPLS path, bringing a scalability problem in both control and data plane. Second, MPLS can not benefit from the load balancing given by Equal-cost Multi-path routing (ECMP). Finally, MPLS with the support of IGPs (Interior Gateway Protocols) for routing protocol can not be easily implemented on multiple Autonomous Systems (ASs). Therefore, many network operators (e.g., NTT, Vodafone, etc.) implement Segment Routing (SR) in their network infrastructure as a solution for these issues [91, 92]. The core idea of SR architecture is based on the notion of source routing [172] and tunneling to guarantee the scalability property in decreasing the amount of state information to be processed in the core network. Besides, SR's main benefit is to fix the scalability issues and limitations of the MPLS approach. Concretely, SR does not require any state maintenance in core network nodes. Moreover, it takes advantage of the ECMP routing and the implementation on multiple ASs [173].

For the second requirement of the application-aware routing, Software-defined Networking (SDN) is a promising solution. It decouples the control layer from the infrastructure layer, offering an opportunity to obtain a global network view. Therefore, application-aware SDN-based SR is concerned by the research community [173, 174]. However, this SR mechanism contains two main disadvantages:

- First, this routing mechanism considers the application class to meet different SLA (Service-level Agreement) requirements, but application identification is sometimes complicated due to encrypted traffic. In the past, the application class can be ob-

tained by DSCP (Differentiated Services Code Point) in packet's header or deep packet inspection. However, DSCP field can be changed during packet's transmission [175] while the deep packet inspection is not effective with encrypted traffic [176]. Indeed, many service providers encrypt their data during the transmission to protect the user's privacy. According to a recent Cisco report [13], 80 percent of web traffic was encrypted by 2019 in comparison with 40 percent by 2016. Consequently, there is a necessity for a novel traffic classification approach to identify the application class in this case. Much existing research work focus on the classification approaches for VPN [85] and TLS/SSL [84] traffic. However, a novel solution is studied here to obtain the application class for the traffic of QUIC (Quick UDP Internet Connection) [61] which is a new transport layer network protocol developed by Google from 2012. The amount of QUIC traffic comprises 35 percent of Google's egress traffic (approximately 7 percent of global Internet traffic) and continues to increase in the future [177].

- Second, routing paths are identified according to the human intervention, so it is not adaptive with the unprecedented change of network environments. For the past few decades, QoS (Quality of Service) played an essential role in the network systems, so many studies [178] concentrated on QoS-aware SR mechanisms. The main objective is to optimize the network resource utilization as well as meet network requirements related to QoS. Nevertheless, selecting appropriate QoS parameters is sometimes complicated. Therefore, QoE (Quality of Experience), which network operators need to assure, is considered in SR in this work. QoE is a metric correlated to QoS metrics, but it is a perception of end-user. Consequently it facilitates network policies to guarantee the user's SLA requirements.

In this chapter, a novel SDN-based SR mechanism for encrypted traffic is proposed for network operators. The objective is to implement the corresponding routing policies for various SLA requirements because each application has a specific SLA requirement. This mechanism selects the appropriate paths in a particular routing policy using Reinforcement Learning (RL) and the network environment's feedback (QoE) to adapt to dynamic network conditions. There are traditional algorithms in the QoE-aware SR scheme (e.g., selecting a path with maximal QoE value, etc.). However, it is not effective with dynamic networks. With the development of 5G, the networks have become more and more com-

plex. Using the RL algorithm in the QoE-aware SR mechanism offers opportunities to select appropriate paths to adapt to the dynamic changes of network environments and improve long-term performance.

Outline: The remainder of the chapter is structured as follows. Section 5.2 introduces the related work in SR. In section 5.3, the chapter presents the proposed SR mechanism. Section 5.4 describes the experimental results of the proposed mechanism. Finally, the chapter concludes with section 5.5 which highlights our future work.

5.2 Application-aware Routing Mechanisms

This section presents related work on application-aware routing, application-aware MPLS, application-aware SR mechanisms and highlights its pros and cons.

5.2.1 Application-aware routing

Li et al. [179] emphasized that network operators have encountered a challenge of providing better services with the rapid growth of 5G and multimedia services which requires diverse network requirements (e.g., low latency, high reliability, etc.). However, network operators are unaware of which applications in their networks, so they cannot obtain a global view to manage the networks effectively. Therefore, many studies focus on application-aware routing techniques in network management. Application-aware routing is a routing technique that selects optimal routes corresponding to an individual application thanks to a corresponding routing strategy [180].

Adami et al. [50] proposed an application-aware routing technique for SIP (Session Initiation Protocol) traffic. First, this approach uses a deep packet inspection to identify SIP flows using its signatures (e.g., UDP packet, port 5060, content-Type: application/sdp, etc.). Then, this approach selects the shortest paths for these SIP flows using the Dijkstra algorithm with link utilization as a constraint.

Cheng et al. [51] proposed an application-aware routing algorithm to deploy various routing policies corresponding to different applications. This algorithm classifies network traffic into three categories: real-time applications (e.g., VoIP, gaming, etc.), streaming applications (e.g., video streaming, web browsing, etc.) and miscellaneous applications (e.g., file sharing, etc.). Then, each category is processed with a specific routing policy using different network parameters such as link load, delay and delay variation.

U-chupala et al. [180] proposed Overseer, an application-aware routing in SDN. Overseer analyzes network traffic using a deep packet inspection to classify into three kinds of applications including bandwidth-oriented (e.g., HTTP, FTP, media streaming, etc.), latency-oriented (e.g., SSH, online game, etc.) and default applications. Then, Overseer implements different routing policies corresponding to different kinds of applications containing maximal-bandwidth, minimal-latency and minimal-path-length strategies. The first one is to forward packets of bandwidth-oriented applications through the paths with the largest bandwidth while the second one aims to forward packets of latency-oriented applications through the paths with the smallest latency. The final one is to implement a default strategy that selects the shortest paths to forward packets of default applications.

Jeong et al. [181] proposed an application-aware traffic engineering system in SDN to implement differentiated treatments on various applications. This approach uses a deep packet inspection to identify application classes and sends application information to a Traffic Scheduler. In this module, each application is assigned to a corresponding priority queue. According to application-specific priority queues and current network states, Traffic Scheduler calculates optimal routes corresponding to different applications. In this approach, the authors take into account two kinds of traffic including iPerf and Youtube.

Rego et al. [182] proposed an improvement of Open Shortest Path First (OSPF) routing protocol in SDN environments. This approach changes the metric calculation (e.g., bandwidth, delay, etc.), adapting to different applications to select appropriate routing paths. However, identifying a practical metric calculation is sometimes complicated.

The above approaches identify the application class using the deep packet inspection, but it is not effective with encrypted traffic. Moreover, these approaches use an IP routing mechanism that identifies next-hops based on a destination IP address. It requires much processing time because each switch needs to analyze the packet's header to obtain the IP address. Besides, there is a large number of rules to be installed in the switches due to the rapid growth of Internet traffic, so these mechanisms require more TCAM (Ternary Content Addressable Memory) resource consumption [183].

5.2.2 Application-aware MPLS

The emerging of MPLS is a promising solution to overcome the drawbacks of IP routing mechanisms. MPLS identifies the next-hops based on the label added to the packet, so

this requires less processing time and TCAM resource consumption.

Bahnasse et al. [184] proposed an application-aware MPLS in SDN to optimize network resources. This approach identifies VoIP, video, HTTP, and ICMP traffic based on DSCP field in the packet's header. Then, a specific routing policy is applied for each application to meet bandwidth constraints. However, identifying the application class using DSCP sometimes is inaccurate because it can be changed during data transmission [175].

Google [185] proposed Espresso, a SDN-based Internet peering edge routing infrastructure that offers an opportunity for the application-aware MPLS mechanism at the Internet-peering scale. According to integrating application-aware MPLS mechanism, Espresso delivers 13 percent more network traffic on their infrastructures and improves link utilization and user perception compared to BGP (Border Gateway Protocol)-based routing. Nevertheless, MPLS suffers from several hindrances related to scalability problem and ECMP routing.

5.2.3 Application-aware SR

Many other proposals on SR are proposed to address the hindrances of MPLS. Kukreja et al. [173] presented a demonstration of SDN-based SR for multi-domain networks. In this demonstration, an orchestrator finds suitable routing paths and encodes to packet's header corresponding to diverse applications. The objective is to meet different resource requirements of these applications (e.g., bandwidth, delay constraints, etc.). Nevertheless, it is assumed that the orchestrator knows the class application.

Peng et al. [174] proposed an application-aware network framework that takes advantage of SR to meet their SLA requirements. This framework identifies the application characteristics according to the deep packet inspection mechanism and then forwards packets into corresponding paths (policy or traffic engineering tunnel). Nevertheless, these SR mechanisms identify the routing paths thanks to human intervention, which is ineffective with the unprecedented change of network environments. In contrast, the proposed SR mechanism identifies the routing paths using RL to adapt to dynamic networks. Moreover, classifying the network traffic using deep packet inspection is not practical due to encrypted traffic nowadays [176]. Therefore, the proposed SR mechanism implements a novel traffic classification approach to classify the encrypted network traffic and identify the application classes.

5.3 Proposed Adaptive Segment Routing Mechanism for Encrypted Traffic

5.3.1 Overview of SDN-based Adaptive Segment Routing Framework

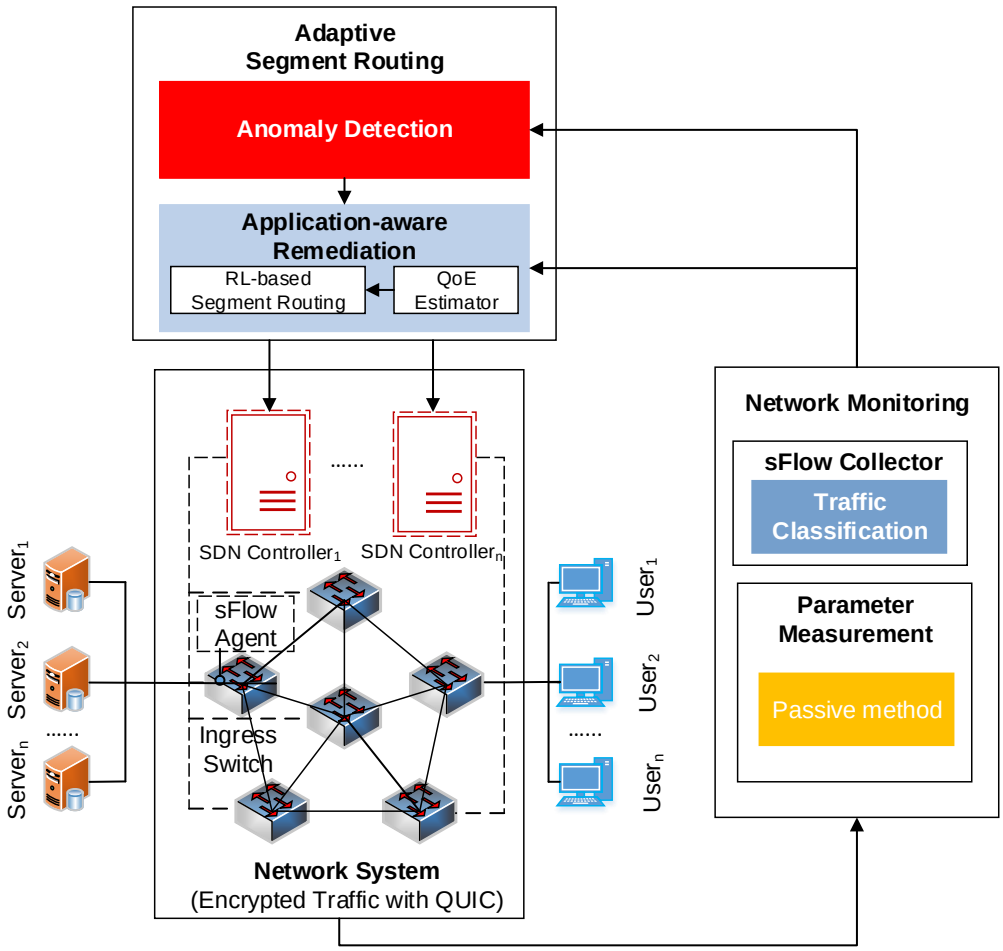


Figure 5.1: The SDN-based adaptive SR framework issued from the global troubleshooting framework.

The proposed multi-modular system is depicted in Fig. 5.1. During the transmission from servers to clients, a network flow is transmitted from ingress switches to a sFlow collector [59] to obtain the class of application thanks to traffic classification module. In the beginning, the network flow is forwarded using the shortest paths from the standard SR algorithm. After that, network parameters from parameter measurement module are analyzed in anomaly detection module to detect abnormal symptom of problems (e.g., increase of latency, packet loss, etc.). When the problems occur in the network, the network flow is forwarded using an adaptive SR mechanism that selects the appropriate routing paths to meet strict user requirements related to the QoE corresponding to each network

application. When the problems are solved, the network flow will be forwarded using the standard SR algorithm. The major components in this framework are described as follows:

- *Network monitoring* includes two essential modules containing traffic classification and parameter measurement modules. The former aims to identify the class of application on network flows (video streaming, file transfer and VoIP) in the context of encrypted traffic while the latter aims to monitor and collect network parameters for further modules.
- *Anomaly Detection* takes into account time-series network parameters to detect an abnormal symptom of network problems.
- *Application-aware Remediation* is used to consider various routing strategies corresponding to different kinds of applications to optimize the QoE in the network. This module is considered as a temporary remediation module to decrease negative impacts of anomalies (e.g., high loss, high latency, etc.) until its root causes are solved completely.

The proposed SR mechanism considers the class of application to implement appropriate routing policies corresponding to each kind of application. When network traffic is encrypted, this information is hidden. Therefore, there is a necessity of a traffic classification module for identifying this information. The traffic classification is presented in the following section.

5.3.2 Network Monitoring

5.3.2.1 Traffic Classification

A novel traffic classification approach is presented in this section to identify different kinds of applications for encrypted traffic. In [129], it is reported that video streaming, file sharing, and VoIP will comprise over 80 percent of global IP traffic by 2022. Consequently, these applications are considered in the traffic classification module. Regarding the network traffic for traffic classification, it can be collected directly from Openflow switches, but it is not effective with a huge amount of network traffic. Therefore, sFlow [59], a standard for network monitoring supporting packet sampling technique, is deployed to reduce the collected traffic volume and offer opportunities to implement the

traffic classification module in real-time. When the network traffic traverses the ingress switches (edge switches), the sFlow agents send the network traffic to the sFlow collector after sampling. At the sFlow collector, the traffic classification module collects the network traffic to identify the class of application. This module is described as follows (Fig. 5.2).

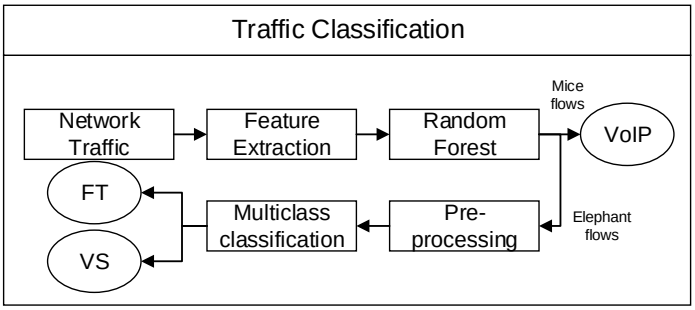


Figure 5.2: The novel traffic classification approach for encrypted traffic.

There are two kinds of flows in the network traffic including mice flows (small continuous flows in total bytes) and elephant flows (huge continuous flows in total bytes). After investigating their characteristics, flow-based features (handcrafted features) of the first few packets in each flow [71] are collected and analyzed using the random forest algorithm [136] to classify the network traffic into the mice flows (VoIP) or the elephant-flows (video streaming and file transfer). This approach considers the random forest algorithm in mice-flow identification over several traditional machine learning algorithms (e.g., SVM, MLP, etc.) based on the research in [136]. After that, the elephant flows are classified into video streaming (VS) or file transfer (FT) using packet-based features (implicit features) and the convolutional neural network (CNN). The latter learning algorithm is taken into account in this approach because it contains characteristics including spare connectivity, parameter sharing, and equivariant representations. This helps to learn more effective representations in comparison with traditional machine learning algorithms [131]. The detail of the traffic classification module is described in our previous work [186] (Chapter 3).

Regarding time complexity of this module, it is based on CNN, and the total time complexity of all convolutional layers [139] is estimated by Equ. 5.1:

$$O(\sum_{l=1}^{d} n_{l-1} \times s_l^2 \times n_l \times m_l^2). \tag{5.1}$$

where d is the number of convolutional layers, n_{l-1} is the number of input channels of the l-th layer, n_l is the number of filters in the l-th layer, s_l is the spatial size of the filter and m_l is the spatial size of the output feature map.

This time complexity is applied in both training and testing phase, and training time is approximately three times of testing time.

5.3.2.2 Parameter Measurement

Many SLAs of service providers depend on several performance metrics such as latency, packet loss, and link utilization [171, 184]. Consequently, a parameter measurement module is implemented to measure these parameters on each link in the network. Latency is measured according to an existing work [94] while packet loss and link utilization are measured thanks to *PortStatistics* API [95] in the controller.

5.3.3 Anomaly Detection

In this section, an anomaly detection approach is presented to identify abnormal symptoms of network problems. Unlike the existing rule-based approaches that trigger the alarms when network parameters (e.g., packet loss, delay, etc.) exceed a threshold, its fluctuations are monitored to early detect the unusual symptoms in this approach [187, 188]. The time-series parameters on network flows such as latency, packet loss, and link utilization are taken into account. These parameters are concatenated into a 1-dimensional vector. Then this vector is analyzed according to the random forest algorithm as in existing work [97, 189] to classify the network states into normal or abnormal states to early detect the anomaly of network problems.

The time complexity of random forest algorithm in training phase is O($n_0 \times log(n_0) \times d \times k$) where n_0 is the number of input sample, d is dimensional of data and k is the number of trees in the forest while the time complexity in testing phase is O($k \times d^{'}$) where $d^{'}$ is the depth of tree [138].

5.3.4 Application-aware Remediation

5.3.4.1 QoE Estimator

The rapid growth of the Internet leads to the diversity of multimedia applications. Consequently, deploying a general QoE model for different applications is ineffective due to its various QoS requirements. Therefore, we implement a novel QoE estimator (Fig. 5.3)

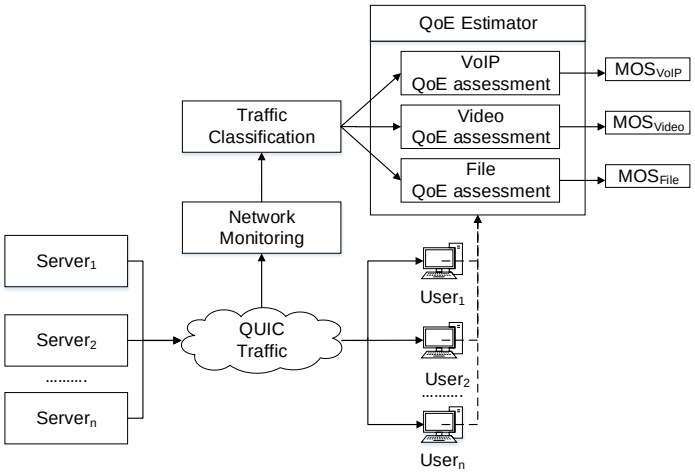


Figure 5.3: The QoE estimator for encrypted traffic.

to calculate QoE of different applications. The application class is identified in the traffic classification module. Then, a specific QoE model is used to calculate QoE for a particular application.

QoE can be calculated according to subjective and objective approaches. The former requires participants to evaluate their MOS (Mean Opinion Score) about multimedia applications while the latter builds an objective model calculating MOS using parameters (e.g., network parameter, application parameter, etc.). However, the subjective approach is costly and ineffective with real-time monitoring, whereas identifying an effective model in the objective approach is sometimes complicated. Therefore, we implement a hybrid approach that uses a subjective dataset to learn QoE models based on ML algorithms. The hybrid approach offers opportunities to estimate MOS in real-time. We use a MOS dataset which is built in a collaboration between our laboratory (LISSI) and Orange (a Network Operator in France) [110]. There are many ML algorithms (e.g., MLP, SVM, random forest, etc.). The random forest has less root mean square error in comparison with others [111], so it is considered as a ML algorithm in QoE estimation. MOS is calculated thanks to network parameters (e.g., latency, packet loss, etc.). There are five levels of MOS including 1 (Bad), 2 (Poor), 3 (Fair), 4 (Good) and 5 (Excellent). The detail of this module is described in [111].

5.3.4.2 RL-based Segment Routing

The SR algorithm is formalized as a RL task that contains *agent, state, action, reward,* and *policy* (Fig. 5.4). The detail is described as follows:

Agent: An entity in the network system applies a learning algorithm to perform its

tasks. In a routing problem, an agent selects appropriate paths to optimize a reward.

State s: A snapshot of the network environment which is observed by the agent.

Action a: An action illustrates how an agent replies to the network environment. In the routing problem, the action is a routing path between a server and a client in the network. All routing paths in the network can be obtained in the SDN controller.

Policy π: A policy is a map from an observed state to action in the network environment.

Reward r: A reward is a feedback of the network environment corresponding to the agent. In the routing problem, the agent monitors a network state s and performs an action a from the routing policy. Then, the agent moves to next state s' and receives a reward r. The reward is the MOS score of a chosen path which is calculated via the pre-trained QoE estimator (section 5.3.4.1).

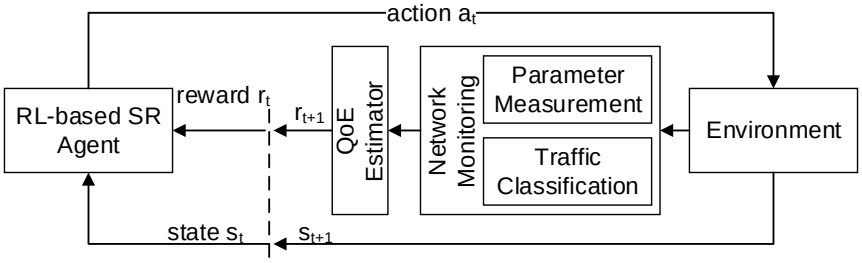


Figure 5.4: The RL-based SR mechanism.

The main goal of RL is to optimize an objective function O_f (Eq. 6.4):

$$O_f = Max\, E[\sum_{t=0}^{\infty} \gamma_t \times r_t]. \tag{5.2}$$

where $\gamma_t \in [0, 1]$ is a discount factor.

In RL, there are two kinds of approaches including model-based and model-free approaches. In the first approach, the agents learn the environment model and enhance its policies to obtain optimality while the agents optimize its policies without prior information about the network environment in the second approach. The first one can learn faster than the second one. It is still less popular because of a large storage cost and dependence on accuracy of an initial information [190]. Therefore, the model-free approach is used in this work. In this approach, Q-value estimates how good it is to execute a given action in a given state. $Q(s, a)$ is the expected return starting from state s and taking action a following policy π. At a time step, the agent is in state s, performs action a, receives

reward r and moves to next sate s'. The Q-value is updated as in Eq. 6.5:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')]. \tag{5.3}$$

where α is a learning rate and γ is a discount factor.

Time complexity of reinforcement learning algorithm is $O(m \times N_0)$ where N_0 is the number of action, and m is the size of state space [191, 192].

5.3.4.3 Exploration-Exploitation Trade-off

An exploration and exploitation phase in RL needs to be balanced to obtain an optimal cumulative MOS score (cumulative reward). The exploitation phase which selects the routing path (action) with maximal Q-value, can not be implemented systematically because each routing path needs to be evaluated frequently to achieve the optimal MOS score. In this chapter, the trade-off between the exploration and exploitation phase is formalized as a MAB problem (Multi-Armed Bandit). MAB problem is a formalization of sequential decision-making tasks. At a time step, a decision-maker selects an action and receives a reward from an unknown distribution corresponding to this action. The main objective is to maximize the total reward received through a sequence of actions. In this chapter, three selection algorithms are presented to resolve the MAB problem: ϵ-*greedy* [193], *softmax* [194], and *UCB1* (Upper Confidence Bounds) [195].

First, ϵ-*greedy* is the simplest algorithm to resolve the bandit problem. Concretely, the agent selects the routing path with the highest Q-value with a probability of (1-ϵ). Otherwise, the agent selects the routing path randomly. Then, the ϵ value reduce against the time so that the agent can learn more about the network environment and become more confident.

Second, the *softmax* algorithm selects the routing paths according to a probability function of Q-value. Each routing path a_i is assigned to a probability p_i as in Eq. 5.4:

$$p_i = \frac{e^{\frac{Q_{a_i}}{\tau}}}{\sum_{j=1}^{N_0} e^{\frac{Q_{a_j}}{\tau}}} \tag{5.4}$$

where τ is a temperature parameter, N_0 is a number of routing paths and Q_{a_j} is a Q-value of routing path a_j.

When the temperature parameter τ is reduced, the routing paths are exploited more frequently. In that way, the temperature parameter τ is reduced each episode (forwarding time). Therefore, *softmax* algorithm not only explores the less-used routing paths but also selects the best routing path in terms of expectation gains.

Finally, the *UCB1* algorithm is related to an index-based algorithm. UCB-index is defined as a sum of a current Q-value and a confidence bound. The UCB-index is described as in Eq. 5.5:

$$\text{UCB} - index_{a_i} = \text{Q}_{a_i} + \sqrt{\frac{2ln(\text{N})}{n_{a_i}}}. \tag{5.5}$$

where Q_{a_i} is a Q-value of routing path a_i, n_{a_i} is a number of chosen time of routing path a_i and N is an episode number (forwarding time).

After calculating the UCB-index for each routing path, *UCB1* algorithm selects the path with maximal UCB-index. As shown in Eq. 5.5, the UCB-index comprises two parts including Q-value Q_{a_i} and confidence bound $\sqrt{\frac{2ln(\text{N})}{n_{a_i}}}$. A routing path is chosen when the Q-value is large or the confidence bound is high. When the routing path with the large Q-value is chosen, this choice is an exploitation trial. When the confidence bound is high, this choice is an exploration trial. The confidence bound is higher when the number of chosen times of the routing path is smaller in comparison with other paths. In other words, the less routing path is selected, the more it has the opportunity to be selected.

5.4 Experimental results

5.4.1 Experiment Setup

Table 5.1: Configuration of the PC used in the testbed.

Operation System	Ubuntu 16.04.6 LTS
Processor	Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
Memory	2133MHz DDR4 8GB

The performance of the proposed SR mechanism is evaluated via an emulation testbed with *mininet* v2.2 [196] and *ONOS* controller v2.4 [197]. *mininet* is a popular emulator in the research community to emulate the network topology. ONOS only supports leaf-spine topology, so we use this topology from a simple scene (five spine nodes and two

leaf nodes) to a complex one (more than 15 nodes). The number of nodes in the topology is set to 7, 17, 32 and 47. Moreover, *ONOS* also supports segment routing through *org.onosproject.segmentrouting* application, and we customize it to implement the proposed SR mechanism. To generate QUIC traffic of video streaming in the network, we replay a pcap file of QUIC traffic from servers to clients using *tcpreplay* application, which is collected according to watching videos in Youtube with Google Chrome. The testbed is implemented in a PC described in Tab. 5.1. For the exploration-exploitation trade-off, ϵ and τ are set to 1 and 2, respectively. For the RL algorithm, α and γ are set to 0.7 and 1, respectively. These parameters are selected according to an existing work [198]. The source code of the proposed framework is available at [120].

Table 5.2: Scenarios.

Scenarios		Descriptions
perfect scenario		No delay and loss
Scenario with faults	*delay*	25, 50, 75, 100 and 125ms
	loss	5, 10, 15, 20 and 25%
	delay and loss	(25ms, 5% loss), (50ms, 10%), (75ms, 15%), (100ms, 20%) and (125ms, 25%)

The experiments are considered in four scenarios (Tab. 5.2). The network parameters are chosen to cover the maximum QoE range. The first scenario is to evaluate the performance of the proposed approach in the network condition without delay and loss, and other scenarios are used to consider the proposed mechanism in the context with faults. Loss and delay parameters in the routing paths are generated according to *mininet*. In the last three scenarios, each routing path is randomly set to a specific delay, loss or both delay and loss. These delay and loss parameters are changed every 50 episodes which are chosen according to the experiments to generate dynamic network states. An uniform link capacity is set to 10 Mbps, and a sending rate is set to 2.5 Mbps as in an existing work [183].

The proposed application-aware SR mechanism is designed for three applications including video streaming, file transfer and VoIP. Among these applications, video streaming comprises the highest global IP traffic [129], so we consider video streaming as a proof-of-concept to thoroughly validate the performance of the proposed SR mechanism.

5.4.2 Benchmark

To validate the performance of the proposed SR mechanism, our proposal is compared with the benchmarks including:

- *Standard SDN-based SR (Standard_SR)*: This algorithm uses the Dijkstra algorithm to determine the shortest paths between servers and clients.
- *SDN-based SR with maximal QoE (Max_QoE)* [199]: This approach calculates the MOS score of all routing paths and selects the paths with maximal MOS score. In contrast, the MOS score of a chosen path is calculated in the proposed mechanism which helps to reduce resource consumption.

These approaches are evaluated via the following performance metrics:

- *MOS*: the perceived quality at the user's side.
- *CPU Usage*: the percentage of the CPU's capacity which is calculated via *ps* command (process status) in the Unix-like operation systems.
- *Control Overhead*: To discovery network topology and update status of links (e.g., latency, loss, etc.), a routing algorithm needs to generate the control packets (e.g., LLDP packets, etc.). Control overhead refers to the ratio of the control packet number to the total number of the sent packets.

5.4.3 Performance Analysis

The selection algorithms for MAB formalization are presented including *UCB1, softmax and* ϵ*-greedy*. Their performances are first evaluated to select the appropriate selection algorithm. Then, the performance of our proposal is compared to the benchmarks including *Standard_SR* and *Max_QoE* mechanisms related to MOS score, CPU usage, and control overhead. Besides, the running time of each module in the proposed SR mechanism is thoroughly evaluated.

5.4.3.1 Time Analysis

In section 5.3, we analyzed the mathematical time complexity of algorithms in the proposed SR mechanism. According to this time complexity, the traffic classification requires

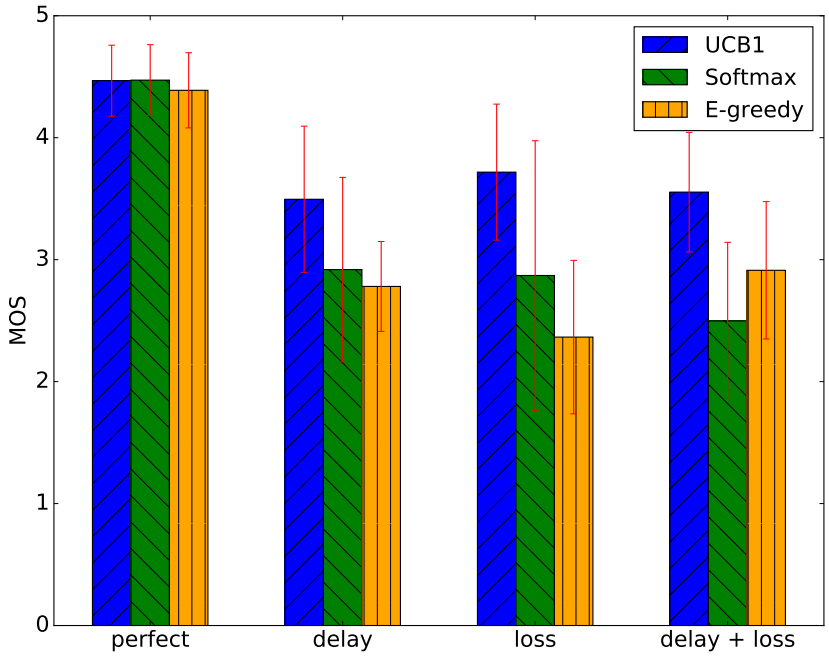
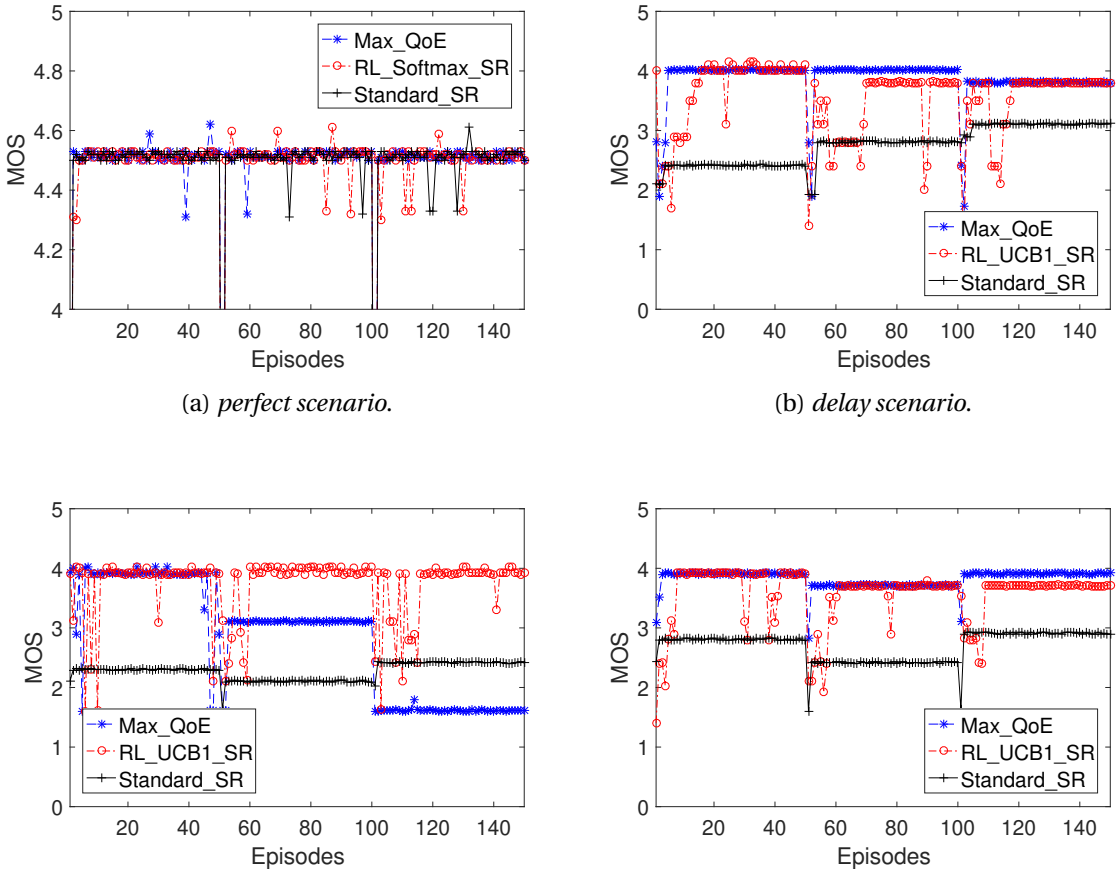


Figure 5.5: Average MOS score and standard deviation of three selection algorithms in the proposed SR mechanism.

higher processing time in comparison with other modules. It requires 0.0316 ms to identify the network flows of VoIP and 3.4 ms to identify the network flows of video streaming and file transfer. To implement this module in real-time, we use sFlow, a standard for network monitoring, to reduce the amount of network traffic collected for the traffic classification module when there is a huge amount of traffic in the networks. In the testbed, we configure sFlow agents to collect 10 percent of network traffic going through sFlow agents. Regarding the running time of the anomaly detection module, it requires 0.0127 ms to process a network flow. As for the RL-based Segment Routing module, it searches in the Q-table which representing routing policies, to select an appropriate path, so it can make instant decisions.

5.4.3.2 Selection algorithms

Fig. 5.5 illustrates the average MOS score and standard deviation of *UCB1, softmax* and ϵ-*greedy* in the proposed SR mechanism with four scenarios. In the *perfect scenario*, there is no delay and loss in the network. The sending rate is 2.5 Mbps while the link capacity is 10 Mbps. Consequently, changing the routing paths does not lead to the fluctuation of the MOS score in three selection algorithms. Although there is no significant difference between these algorithms, the MOS score of *softmax* is slightly better than the others.



(a) *perfect scenario.*

(b) *delay scenario.*

(c) *loss scenario.*

(d) *scenario with loss and delay.*

Figure 5.6: The MOS score of three SR mechanisms.

The average MOS score of *softmax, UCB1* and ϵ-*greedy* reach to 4.47, 4.46 and 4.39, respectively. Besides, their standard deviations are 0.29, 0.291 and 0.31, respectively. In this scenario, the MOS score varies from 4.3 to 4.5. The MOS score of ϵ-*greedy* converges to 4.3 while the MOS score of the others converges to 4.5. Therefore, the average MOS score of ϵ-*greedy* is lower than two other selection algorithms.

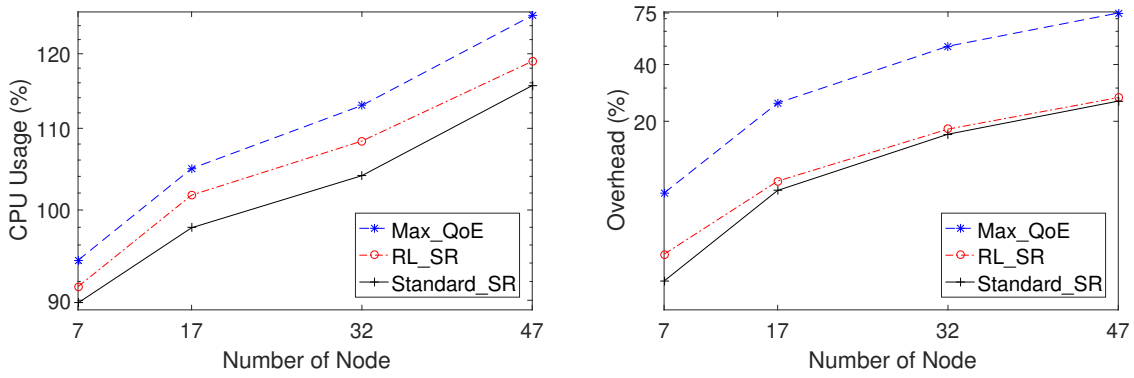
The average MOS score of *UCB1* is the highest, and the standard deviation is the lowest in three selection algorithms for the other scenarios. The average MOS score of *UCB1* in *delay scenario*, *loss scenario* and the final scenario are 3.55, 3.7 and 3.5, respectively. *UCB1* first explores the routing path during the first few episodes and then converges to optimal MOS score. When the network states are changed, *UCB1* can explore the less-used routing paths. Consequently, it continues to reach an optimal value quickly. *softmax* chooses the routing paths according to probability function, so it takes much time to converge to an optimal value. As a result, the MOS score of *softmax* is lower than the figure for *UCB1*. For ϵ-*greedy*, it first explores the routing paths with a high ϵ value. When ϵ value reduces to approximately 0, it can not explore the routing paths frequently. Therefore, the MOS

score of ϵ-*greedy* is lower than two other selection algorithms.

According to Fig. 5.5, our proposal will implement different selection algorithms corresponding to various scenarios. *softmax* will be implemented for *perfect scenario* while *UCB1* will be implemented for other scenarios in our proposal in the following experiments. Our proposal refers to *RL_Softmax_SR* in *perfect scenario* and *RL_UCB1_SR* for other scenarios.

5.4.3.3 Segment routing mechanisms

Fig. 5.6 illustrates the MOS score against the episodes (forwarding time) of three SR mechanisms in four scenarios. In *perfect scenario* (Fig. 5.6a), the role of the routing paths is the same due to no delay and loss in the network. As a result, the MOS score of three SR mechanisms is nearly equal. The average MOS score of our proposal (*RL_Softmax_SR*), *Max_QoE* and *Standard_SR* are 4.47, 4.47 and 4.46, respectively. Besides, their standard deviations are 0.29, 0.29 and 0.34, respectively.



(a) Average CPU usage.

(b) Average overhead.

Figure 5.7: The average CPU usage and overhead of three SR mechanisms.

In *delay scenario* (Fig. 5.6b), each routing path is set to a specific delay parameter which leads to the differences in the MOS score between the routing paths. Therefore, there is a significant difference between three SR mechanisms. The MOS score of *Standard_SR* is the lowest in three SR mechanisms because it selects the shortest paths to forward the packets. The MOS score of this mechanism depends on the delay parameter which is set to the shortest path. The network states are changed every 50 episodes, so the MOS score of this mechanism changes periodically. The MOS score of *Standard_SR* in the first, second and last 50 episodes are approximately 2.4, 2.8 and 3.1, respectively. *Max_QoE* monitors the network topology, calculates the MOS score of all routing paths and

selects the path with the best MOS score. Therefore, it achieves a high MOS score (approximately 4) in the majority of episodes. In this scenario, our proposal (*RL_UCB1_SR*) implements *UCB1* to determine the routing paths to forward the packets. *UCB1* explores the routing paths in the first 16 episodes, so the MOS score varies between 1.7 and 4. After that, the MOS score of our proposal converges to an optimal value (approximately 4). At 50th and 100th episodes, the network states are changed to create dynamic network environments. UCB-index of the less-used routing path becomes larger when a routing path is not chosen frequently. Consequently, *UCB1* can explore the less-used routing paths to select the appropriate path. As a result, the MOS score of our proposal can converge to optimal value quickly after a few episodes. Besides, our proposal can choose another routing path at 24th and 89th episodes after converging to an optimal value. In other words, our proposal does not always select the routing path with the best UCB-index to guarantee the exploration-exploitation trade-off. In Fig. 5.6b, the MOS score of our proposal is slightly lower or equal to the MOS score of *Max_QoE*.

In *loss scenario* (Fig. 5.6c), the MOS score of *Standard_SR* is lower than the figure for our proposal. The MOS score of this mechanism in the first, second and last 50 episodes are 2.3, 2.1 and 2.4, respectively. Similar to previous scenarios, our proposal using UCB1 selection algorithm (*RL_UCB1_SR*) first explores the routing paths during the first few episodes and then converges quickly to optimal value (nearly 4). A remarkable feature from Fig. 5.6c is that the MOS score of *Max_QoE* is lower than the MOS score of our proposal from the 50th episode. ONOS controller sends the control packets (e.g., LLDP packets, etc.) to discover the network topology every 3 seconds, so the delay on the routing paths is measured according to control packets as in an existing work [93]. Therefore, *Max_QoE* can monitor the network and select the path with the best MOS score in *delay scenario*. The loss parameter is measured via *PortStatistics* API in ONOS controller after data is transmitted in the network. Consequently, the loss on a routing path is not updated when this path is not chosen. At the beginning of the first 50 episodes in *loss scenario*, the loss of all routing paths is initiated to 0. Therefore, *Max_QoE* can select the path with the best MOS score. At the 50th episode, the network states are changed. The routing paths with low MOS score in previous network state becomes the paths with high MOS score in current network state, but the loss on these paths are not updated. Therefore, the MOS score of *Max_QoE* is lower than the figure for our proposal.

In *scenario with delay and loss* (Fig. 5.6d), the MOS score of *Standard_SR* is the lowest

in three SR mechanisms. Similar to *delay scenario*, the MOS score of our proposal (*RL_UCB1_SR*) is equal or slightly smaller than the MOS score of *Max_QoE*. The MOS score of *Max_QoE* in the first, second and last 50 episodes are 3.9, 3.7 and 3.9, respectively. Besides, the optimal MOS score of our proposal are 3.9, 3.7 and 3.7, respectively. Although the MOS score of *Max_QoE* is higher than the others, it requires much resource consumption related to CPU usage and overhead.

There is less difference between the CPU usage and overhead in four scenarios, so the average CPU usage and overhead of these mechanisms in four scenarios are depicted in Fig. 5.7. The number of leaf nodes is increased while the number of spine nodes is not changed to obtain a larger network topology in these experiments. Fig. 5.7a shows the CPU usage of three SR mechanisms. The testbed is implemented in a computer with 4 cores described in Tab. 5.1, so the CPU usage can reach 400 percent in maximum. The CPU usage of *Max_QoE* is the highest in three SR mechanisms because it monitors the entire network topology in order to obtain the routing path with the best MOS score. The CPU usage of *Max_QoE* for 7 nodes is 94.3 percent, and it increases to 125.48 percent for 47 nodes. The CPU of our proposal is lower than the figure for *Max_QoE*. It raises from 91.45 to 118.95 percent when the number of nodes increases from 7 to 47. Besides, there is an increase in the CPU usage of *Standard_SR* from 89.73 to 115.63 percent when the number of nodes raises from 7 to 47.

Fig. 5.7b indicates the control overhead of three SR mechanisms. The overhead of *Max_QoE* is higher than two other mechanisms because *Max_QoE* sends the control packets to monitor the entire network topology. It increases rapidly from 8.33 to 75 percent when the number of nodes increases from 7 to 47. Our proposal only monitors the chosen paths, so the overhead of our proposal and *Standard_SR* are nearly equal. The overhead of our proposal with 7 and 47 nodes are 3.95 and 26.71 percent, respectively. The overhead of *Max_QoE* is nearly three times the overhead of our proposal for 47 nodes. In other words, our proposal reduces up to 64.39 percent of overhead in comparison with *Max_QoE*. Although *Max_QoE* can achieve a high MOS score in the majority of episodes, it requires much resource consumption in terms of CPU usage and overhead. Consequently, it is not appropriate for a large-scale network. In contrast, the MOS score of our proposal is nearly equal or higher than the figure for *Max_QoE* in considered scenarios, but it requires less resource consumption in comparison with *Max_QoE*. Some important results are summarized as in Tab. 5.3 and 5.4. Tab. 5.3 indicates important results related to the median

Table 5.3: Summarization of average optimal MOS, median and 95 % confidence interval of MOS in the SR mechanisms.

Scenarios	Mechanisms	Average	Improvement of proposal (%)	Median	95% Confidence Interval
Perfect	Proposal	4.47	-	4.53	4.47±0.047
	Standard_SR	4.46	Nearly equal	4.52	4.46±0.056
	Max_QoE	4.47	Equal	4.53	4.47±0.047
Delay	Proposal	3.86	-	3.82	3.86±0.1
	Standard_SR	2.76	139.8	2.80	2.76±0.05
	Max_QoE	3.93	Nearly equal	4.01	3.93±0.062
Loss	Proposal	3.9	-	3.93	3.9±0.089
	Standard_SR	2.26	172.6	2.11	2.26±0.023
	Max_QoE	2.86	136.4	3.11	2.86±0.154
Delay+ Loss	Proposal	3.76	-	3.70	3.76±0.078
	Standard_SR	2.7	139.3	2.40	2.7±0.04
	Max_QoE	3.83	Nearly equal	3.72	3.83±0.024

MOS, 95 % confidence interval of MOS and the average optimal MOS score which is the average value of the optimal MOS score in each 50 episode of three SR mechanisms. Tab. 5.4 illustrates the essential results in term of the average overhead against the number of nodes of our proposal and *Max_QoE*. Confidence interval (CI) gives an estimated interval for an unknown population parameter. It is associated with a confidence level, representing a probability which the estimated interval includes a true value of the parameter. 95% confidence interval is computed at the 95% confidence level containing the parameter. It is calculated by Equ. 5.6.

$$\mathrm{CI} = \overline{x} + z^{\star} \cdot \frac{\sigma}{\sqrt{n}} \tag{5.6}$$

where $\overline{x}$ is mean of MOS, σ is its standard deviation, n is number of samples and $z^{\star}$ is 1.96 for 95% of confidence level.

In Tab. 5.3, 95% confidence interval of the proposed SR mechanism is wider than the figure for *Max_QoE* in four scenarios. The reason is that the proposal needs to explore the routing paths to select the appropriate one, so its MOS fluctuates more frequently than the figure for *Max_QoE*.

Table 5.4: Summarization of average overhead in the SR mechanisms.

Mechanisms	Average overhead (%)			
	7	17	32	47
Max_QoE	8.33	24.99	50	75
Proposal	3.95	9.64	18.18	26.71
Improvement of proposal (%)	**52.58**	**61.43**	**63.64**	**64.39**

5.5 Conclusion

Segment Routing needs to be performed more adaptively to avoid network problems (e.g., congested links, etc.) and meet different service-level agreement requirements. To cope with these demands, we propose a novel SDN-based adaptive segment routing framework for network operators in the context of encrypted traffic. Our proposal is developed on the SDN controller which can be integrated into networks supporting virtualized architectures related to SDN. The proposed segment routing mechanism implements different routing policies corresponding to various applications and meets strict service-level agreement requirements. Moreover, the appropriate routing path is selected according to reinforcement learning policy and the feedback of the network environment (QoE). The experimental results show that the proposed SR mechanism using reinforcement learning outperforms the standard SR mechanism in terms of QoE and reduces up to 64.39 percent of overhead in comparison with *Max_QoE* mechanism.

Segment list is one of the important factors of the segment routing mechanism, so path encoding algorithm needs to be investigated thoroughly to optimize the performance of the segment routing mechanism. After detecting the network problems and implementing the adaptive segment routing mechanism to reduce its influences, the root causes of the issues needs to be considered to deal with it definitely. Therefore, the root cause analysis mechanism will be investigated in our future work.

After reducing the negative impacts of anomalies in the network (e.g., high latency, high loss, etc.) with the temporary remediation module, we need to identify the root causes of anomalies and solve it completely. In the following chapter, we present an use-case for the root cause analysis and definitive remediation to identify the root cause of congestion and address it definitively.

Chapter 6

Root Cause Analysis and Definitive Remediation

«The spread of computers and the Internet»

Marc Andreessen

Chapter 6 presents an use-case for the root cause analysis and definitive remediation. Concretely, we present a root cause analysis mechanism using machine learning to identify the root causes of congestion, and then we implement an adaptive congestion control algorithm in the definitive module to solve it completely. These modules are described in detail in Fig. 2.2.

Contents

6.1 Root Cause Analysis: Machine Learning based Root Cause Analysis for SDN Network

6.1.1 Introduction

Root cause analysis refers to a process of identifying and delimiting elements leading to anomalies. There are three main objectives in the root cause analysis [151]:

- The first one is to identify which network problems resulting in the anomalies (e.g., link failure, switch failure, etc.). If a network problem happens, it will be classified to return the type of the problem [200].
- The second one is to identify events that lead to the anomalies. Event logs of network elements can be used to study their causalities and identify anomalous events [201].
- The final one is to localize network elements that result in the anomalies. Performance evaluation at network elements can be used to identify the anomalous elements [202].

The root cause analysis can return the type of network problems, anomalous events and anomalous network elements. In this manuscript, we focus on identifying the type of problems that results in network anomalies. We assume that one problem results in anomalies in the network. When many problems lead to anomalies simultaneously in the network, network administrators need to troubleshoot problems manually to solve them completely.

In the past, administrators are able to troubleshoot network problems (e.g., using *ping*, *traceroute*, etc.) and solve it manually. However, it is not effective because of the huge number of network devices and human intervention. Consequently, the root cause analysis is studied by the research community. Conventional root cause analysis (e.g., knowledge-based mechanism, etc.) [153, 203] identifies the root causes using rules and policies with a specific threshold. However, an effective threshold identification is sometimes complicated. The emerging of Machine Learning (ML) and Deep Learning (DL) is a potential solution to solve this drawback. In fact, root cause analysis using machine learning and deep learning is studied by the research community to troubleshoot the network anomalies [96, 97]. Hong et al. [97] proposed a root cause analysis method using

ML to identify the problems related to SLA (service-level agreement) violations for virtual network management. Similarly, Kawasaki et al. [204] proposed a ML-based fault classification to identify three kinds of problems including node-down, interface-down and CPU overload. However, these studies have not considered a balance between accuracy of machine learning algorithms and its processing time to select the appropriate algorithm yet.

In this section, we implement a root cause analysis (RCA) mechanism using ML and time-series network parameters to identify network problems leading to anomalies in the SDN environment. In the root cause analysis, the balance between accuracy and processing time of different ML algorithms (e.g., Random Forest, Gradient Boosting, Convolution Neural Network, etc.) is considered to select the appropriate algorithm. Unlike the existing studies that select the ML algorithm with the best performance, we select the machine learning algorithm with good performance and low time complexity to improve processing time when there is vast network traffic.

The remainder of the section is organized as follows. Subsection 6.1.2 presents related work on root cause analysis. The ML-based RCA mechanism is discussed in subsection 6.1.3. Subsection 6.1.4 describes the experimental results. The section concludes with subsection 6.1.5 which highlights our future work.

6.1.2 Root Cause Analysis Mechanisms

This subsection presents related work on root cause analysis using knowledge-based mechanisms, causality/dependency graphs and ML-based mechanisms.

6.1.2.1 Knowledge-based Mechanism

This mechanism (called expert system) requires prior knowledge about anomalies to build rules to identify the types of problems and localize its locations. A rule is presented in a form as follows: *if* <symptoms> *then* <root cause>. Zhou et al. [205] proposed BigRoots, a rule-based root cause analysis mechanism in big data systems. BigRoots uses four kinds of features including discrete, numerical, resource and time features. The objective is to identify the problems related to CPU, I/O (Input/Output) and network. Hochenbaum et al. [203] proposed a root cause analysis in cloud infrastructure using statistical rules. This approach uses system, application and core driver metrics to localize the location of anomalies.

Despite a high accuracy, the knowledge-based mechanism contains two mains disadvantages. First, it can not identify unknown problems which are not in the rule datasets. Consequently, this approach requires updating rule datasets frequently when there are new network problems. This results in a high maintenance cost. Second, searching in a large rule dataset is costly and time-consuming.

6.1.2.2 Causality/Dependency Graph

A causality graph is a graph that shows a relationship between network problems and its symptoms. The objective of the causality graph is to represent initial problems (root causes) and its symptoms and model complex chains of intermediate problems related to these symptoms. For example, Bayesian network is an instance of the causality graph. The Bayesian network is a directed acyclic graph (DAG) representing cause-and-effect relationships between random variables. A node in the DAG is the random variable (e.g., network elements, problems, etc.) whereas an edge is a causality between two connected nodes. Conditional probabilities express the strength of these relationships. This approach requires a deep knowledge of cause-and-effect relationships between network problems and its symptoms.

Bennacer et al. [206] proposed a root cause analysis approach using a Bayesian network to localize locations of anomalous elements. Benayas et al. [207] proposed a root cause analysis using a Bayesian network in Big Data infrastructures. The objective is to infer ten kinds of problems (e.g., changing flow priorities, modifying in-port rules, etc.) thanks to Bayesian network and network parameters (e.g., change in a number of hosts, change in time-out, etc.)

The Bayesian network contains two main drawbacks. First, it depends on reliable prior knowledge to build DAG models, so it requires deep knowledge about problems and its symptoms. Second, the complexity of inference in the Bayesian network increases significantly with a huge number of nodes, so it is not effective with large-scale networks [206].

6.1.2.3 ML-based Mechanism

Each network problem has a specific behavior that leads to particular range of parameters (e.g., network parameters, resource parameters, etc.). ML-based mechanism analyzes these parameters to identify the root cause of anomalies based on classifications. In fact,

many studies considered ML-based root cause analysis to detect which problems leading to anomalies in the network.

Qiu et al. [208] proposed a novel method using ML for identifying the root cause of anomalies in NFV infrastructure. This approach collects and analyzes the features of CPU consumption, disk I/O, and memory consumption using ML algorithms including Neural Network, Neural Network+SVM, K-Nearest Neighbors, Linear SVM, Radial Basis Function SVM, Decision Tree and Random Forest. This approach aims to identify the problems related to CPU, memory and I/O.

Kawasaki et al. [204] proposed a ML-based fault classification to analyze the root cause of failures in the NFV environment. This approach collects 41 features from the network environment and analyzes these features using ML algorithms (e.g., Random Forest, Support Vector Machine, etc.) to identify three kinds of problems including node-down, interface-down and CPU overload.

Similarly, Hong et al. [97] proposed an anomaly detection method using machine learning to identify the problems related to resource usage and SLA violations. This method identifies the problems (e.g., high CPU utilization, lack of memory, etc.) based on 25 features and ML algorithms including Distributed Random Forest, Gradient Boosting, Extreme Gradient Boost and Deep Learning.

These studies have not considered the balance between the accuracy and the time complexity of ML algorithms yet. Besides, the ML-based mechanisms have two main drawbacks. First, it requires a large labeled dataset to train a root cause analysis model. Second, it cannot effectively identify unknown problems which are not in training datasets.

6.1.3 Proposed ML-based RCA Mechanism

This subsection presents the proposed root cause analysis using machine learning to identify root cause of anomalies in the network.

The overall architecture of ML-based RCA in the SDN environment is depicted as in Fig. 6.1. First, network traffic is collected from the infrastructure layer and processed in Data Collection and Processing module to extract network features. Then, these features are analyzed in the Root Cause Analysis model to identify the type of problems resulting in anomalies in the network. Finally, classification results are notified to the administrators to operate the network effectively.

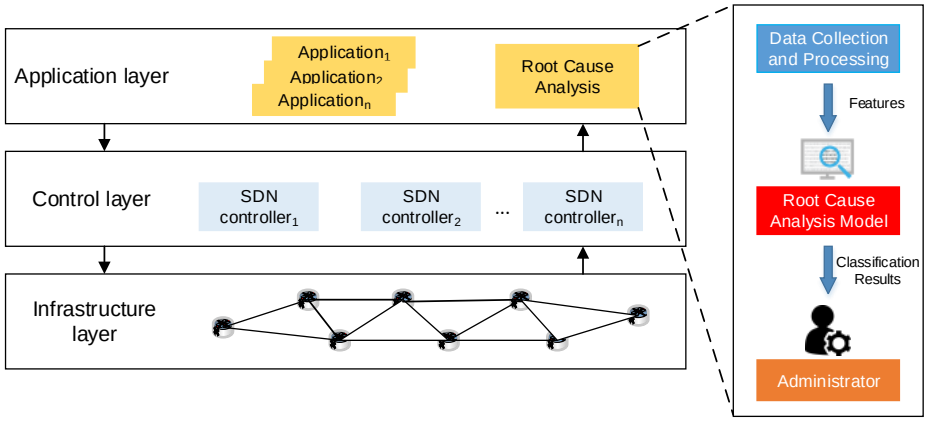


Figure 6.1: Overall architecture of ML-based RCA in SDN environment.

6.1.3.1 Data Collection and Processing

Data Collection and Processing module aims to collect and extract network features corresponding to the network problems for the RCA model. According to a report of network operators [11], we consider three kinds of problems including link failure, switch failure and buffer overload. These problems occur unpredictably in the network, so building troubleshooting datasets related to these problems is not accessible. Therefore, we use a fault injection technique to generate network problems to the network as in existing studies [96, 97]. Link failure is simulated by generating latency in links while switch failure is simulated by generating rule failures in the switches to bring packet loss in links. Buffer overload is simulated by generating a vast amount of traffic surpassing the capacity of links.

These network problems are simulated in two scenarios including static and dynamic networks. In static network, the delay and loss in the links are tied to a specific value. In dynamic network, the link state is changed between normal and error state according to Gilbert-Elliot (GE) model [118] following a probability value after a given time (400 seconds). In this model, the probability of changing from normal to error state and from error to normal state are p and r, respectively. When a link is in the error state, the loss and delay change as in Tab. 6.1.

In each network condition, we collect and extract nine features related to network and standard parameters according to existing studies [94, 111] and API *PortStatistics* [95]. These parameters contain latency, packet loss, link utilization, number of packet sent, number of packet received, number of byte sent, number of byte received, number of flow entries in switch and QoE (Quality of Experience). Then, these parameters on each

Table 6.1: Considered network conditions.

Root causes	Static Network	Dynamic Network
Link failure	Delay: 125ms	Delay: [25, 50, 75, 100, 125ms]
Switch failure	Loss: 50%	Loss: [10, 20, 30, 40, 50%]
Buffer overload	Bandwidth: 10 mbps, Sending rate: 60-100 mbps	Bandwidth: 10 mbps, Sending rate: 60-100 mbps

link are aggregated to calculate the path's features to identify different kinds of network problems. After that, these features are normalized into a value between 0 and 1 according to Max Normalization [119]. In RCA, we consider time-series features, and a sample aggregates features of ten consecutive time steps to identify the network problems. An input sample is a matrix 10×9 (number of time steps × number of features).

Table 6.2: Troubleshooting Datasets.

Root causes	Static Network	Dynamic Network
Buffer overload	6900	6900
Link failure	9995	19710
Switch failure	7560	17565

We built troubleshooting datasets in static and dynamic networks. The number of network states corresponding to each problem is depicted as in Tab. 6.2.

6.1.3.2 ML-based RCA Method

The collected features from the Data Collection and Processing module are analyzed in a ML-based RCA method to identify different kinds of problems. The existence of problems leads to a fluctuation of these features. According to ML algorithms, this fluctuation can be detected to identify the root causes of anomalies. The detail of this approach is described as in Fig. 6.2.

There are two main phases in this method including training and testing. In the training phase, the network features will be analyzed according to a ML algorithm to obtain a pre-trained RCA model for the testing phase. The RCA model aims to infer problems in

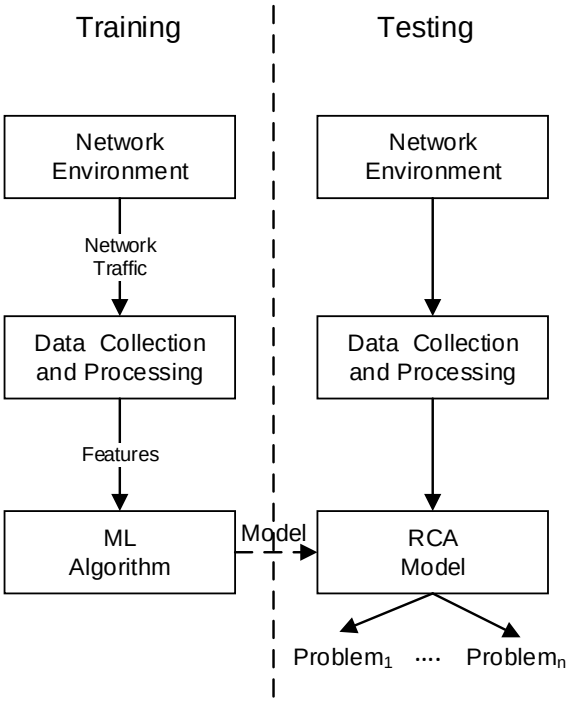


Figure 6.2: The ML-based RCA Method.

the network. There are different kinds of ML algorithms for the RCA. Therefore, we evaluate several algorithms to select the appropriate one and balance between accuracy and time complexity. The considered ML algorithms includes Support Vector Machine (SVM) [164], Bagging [209], Random Forest (RF) [164], Adaboost [165], Gradient Boosting [210] and Convolutional Neural Network (CNN) [166]. Regarding the CNN model, we use two convolutional 1D (1 dimensional) layer, a max Pooling 1D, a flatter layer and three fully connected layers. For the others algorithms, we adjust hyper-parameters such as number of estimators, depth of tree and so on. These configurations are chosen as in existing studies [97] to optimize the performance of ML algorithms.

6.1.4 Experimental Results

6.1.4.1 Experimental Setup

In the experiments, *mininet* [168] is used to generate network topology which contains five spine nodes and two leaf nodes. The delay and loss on each link is configured thanks to *mininet*. In GE model, p and r are set to 0.81 and 0.07 as in [118]. The ML algorithms are implemented according to a library *scikit-learn* [141] in Python. The training, validation and testing phases comprise 76, 4 and 20 percent of the troubleshooting dataset, respectively. The datasets and source code of the ML-based RCA mechanism are published in

[211].

Performance of ML algorithms is evaluated thanks to performance metrics including precision, recall, and F1-score [142]. Precision is the percentage of relevant flows that are retrieved, while recall is the percentage of retrieved flows that are relevant. F1-score (F-measure) represents a harmonic mean between precision and recall. The detail of these parameters are depicted in Equations 6.1, 6.2, 6.3. There are two ways to evaluate the quality of the overall classification including micro-averaging and macro-averaging value. In macro-averaging, a metric is averaged over all classes that are treated equally whereas micro-averaging is based on the cumulative True Positive (TP), False Positive (FP), True Negative (TN) and False Negative (FN) of the dataset.

$$Precision = \frac{TP}{TP + FP}. \tag{6.1}$$

$$Recall = \frac{TP}{TP + FN}. \tag{6.2}$$

$$F1 - score = \frac{2}{1/Recall + 1/Precision}. \tag{6.3}$$

6.1.4.2 Performance Analysis

6.1.4.2.1 Dataset in Static Network There are two datasets including datasets in static and dynamic networks. First, we evaluate the balance between the accuracy and the time complexity of ML algorithms to select the appropriate one with the dataset in static network. The performance metrics of these algorithms are described in Tab. 6.3. Ensemble learning is a model that makes predictions according to different models. There are two popular ensemble methods including bagging and boosting. Bagging trains individual models on a different subset of datasets in parallel and aggregate to improve the performance. In contrast, Boosting trains individual models in a sequence, and each model learns mistakes from the previous one to enhance the performance. In this section, Bagging uses SVM as a base classifier, so its F1-score is higher than the figure for SVM. RF is the ML algorithm using the bagging ensemble method and decision tree as the individual model, so its F1-score (approximately 96.3 percent) is higher than the figure for SVM and Bagging.

Adaboost and Gradient Boosting are the ML algorithms using boosting ensemble method

Table 6.3: Performance metrics of the considered ML algorithms for the dataset in static network.

Class	Precision						Nb Samples
	SVM	Bagging	RF	Adaboost	CNN	Gradient Boosting	
Buffer Overload	0.8040	0.8525	0.9380	0.9637	0.9593	0.9675	1380
Link Failure	0.9007	0.9802	0.9715	0.9670	0.9658	0.9764	1999
Switch Failure	0.9739	0.9875	0.9776	0.9617	0.9834	0.9791	1512
Micro	**0.8920**	0.9403	**0.9634**	0.9644	**0.9691**	**0.9746**	4891
Macro	**0.8928**	0.9401	**0.9624**	0.9641	**0.9695**	**0.9743**	4891
Class	Recall						Nb Samples
	SVM	Bagging	RF	Adaboost	CNN	Gradient Boosting	
Buffer Overload	0.8797	0.9928	0.9870	0.9819	0.9899	0.9928	1380
Link Failure	0.8844	0.9415	0.9560	0.9540	0.9745	0.9735	1999
Switch Failure	0.9134	0.8909	0.9517	0.9623	0.9431	0.9597	1512
Micro	**0.8920**	0.9403	**0.9634**	0.9644	**0.9691**	**0.9746**	4891
Macro	**0.8925**	0.9417	**0.9649**	0.9661	**0.9692**	**0.9753**	4891
Class	F1-score						Nb Samples
	SVM	Bagging	RF	Adaboost	CNN	Gradient Boosting	
Buffer Overload	0.8401	0.9173	0.9619	0.9727	0.9743	0.9928	1380
Link Failure	0.8925	0.9604	0.9637	0.9605	0.9701	0.9735	1999
Switch Failure	0.9427	0.9367	0.9645	0.9620	0.9629	0.9597	1512
Micro	**0.8920**	0.9403	**0.9634**	0.9644	**0.9691**	**0.9746**	4891
Macro	**0.8918**	0.9382	**0.9633**	0.9651	**0.9691**	**0.9753**	4891

Table 6.4: Time complexity of ML algorithms in RCA for the dataset in static network.

Algorithms	Training Time (ms)	Testing Time (ms)
SVM	**1.21472**	**0.49883**
Bagging	37.9581	35.8988
RF	**0.03084**	**0.00143**
Adaboost	0.36427	0.01378
CNN	**26.044**	**0.02636**
Gradient Boosting	**1.48182**	**0.00555**

and decision tree as the individual model. The objective of boosting ensemble method is to learn from the mistakes. Adaboost learns from the mistakes by increasing weight of misclassified samples while Gradient Boosting uses the gradient instead of adjusting the weight. Hence, Gradient Boosting is more flexible than Adaboost. As a result, the micro and macro F1-score of Gradient Boosting is slightly higher than the figure for Adaboost. Moreover, F1-score of CNN is nearly similar to Gradient Boosting with 97 percent.

Despite the good performance, Adaboost, CNN and Gradient Boosting require much processing time. The time complexity of the considered ML algorithms is depicted as in Tab. 6.4. This table describes the average processing time for a sample in the training and testing phase. The training and testing time of RF are the lowest in the considered ML algorithms with 0.03084 and 0.00143 ms, respectively. The testing time of Adaboost, CNN and Gradient Boosting are nearly 10, 18 and 4 times the testing time of RF while the difference in F1-score between these algorithms is less than 1 percent. Therefore, we consider RF as a ML algorithm for RCA.

We use nine features to identify the root cause of anomalies, but some of these features are ineffective for the RCA. Consequently, we implement a feature selection method (wrapper method) to identify the appropriate feature set. The wrapper method [212] evaluates all possible feature sets based on a specific machine learning algorithm and selects the feature set with the highest accuracy. RF is considered as a learning algorithm in the wrapper method due to the balance between its accuracy and time complexity. Fig. 6.3 illustrates the accuracy against the number of features in the feature selection method. The feature set with seven features indicates better results than the others, so this feature set is considered in the RCA. This feature set contains latency, link utilization, number of packet received, number of byte sent, number of byte received, number of flow entries in switch and QoE. Tab. 6.5 shows F1-score of the selected feature set (Feature Set 2) in

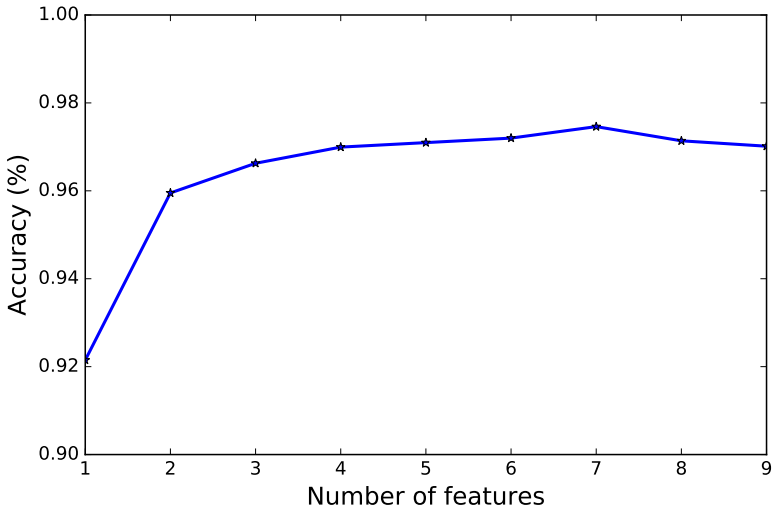


Figure 6.3: The accuracy against the number of features in the feature selection method.

Table 6.5: F1-score of two feature sets in the RCA for the dataset in static network.

Class	F1-score		Nb Samples
	Feature Set 1	Feature Set 2	
Buffer Overload	0.9619	0.9748	1380
Link Failure	0.9637	0.9710	1999
Switch Failure	0.9645	0.9674	1512
Micro	**0.9634**	**0.9710**	4891
Macro	**0.9633**	**0.9710**	4891

comparison with the feature set with all nine features (Feature Set 1).

6.1.4.2.2 Dataset in Dynamic Network Tab. 6.6 and 6.7 illustrate the performance metrics and the time complexity of ML algorithms with the selected feature set in the RCA for dynamic network. Similar to the dataset in static network, micro and macro F1-score of Gradient Boosting are the highest (approximately 95 percent) while the testing time of RF is the lowest in the considered ML algorithms for the dataset in dynamic network. The difference of F1-score between RF and Gradient Boosting is approximately 1 percent, so RF is considered as a ML algorithm in RCA for the dataset in dynamic network. The RCA can achieve good results in the dynamic network with over 93 percent of the micro and macro precision, recall and F1-score. The micro and macro F1-score in the dynamic network are approximately 94 percent, and reduce about 3 percent compared to the figure

Table 6.6: Performance metrics of the considered ML algorithms for the dataset in dynamic network.

Class	Precision						Nb Samples
	SVM	Bagging	RF	Adaboost	CNN	Gradient Boosting	
Buffer Overload	0.6710	0.7813	0.9191	0.9201	0.9040	0.9535	1380
Link Failure	0.7797	0.8669	0.9471	0.9244	0.9483	0.9487	3942
Switch Failure	0.8492	0.8554	0.9451	0.9326	0.9292	0.9578	3513
Micro	**0.7914**	0.8491	**0.9418**	0.9269	**0.9333**	**0.9530**	8835
Macro	**0.7666**	0.8345	**0.9371**	0.9257	**0.9272**	**0.9534**	8835
Class	Recall						Nb Samples
	SVM	Bagging	RF	Adaboost	CNN	Gradient Boosting	
Buffer Overload	0.5659	0.7739	0.9464	0.9341	0.9688	0.9667	1380
Link Failure	0.8630	0.8757	0.9500	0.9434	0.9302	0.9619	3942
Switch Failure	0.7996	0.8488	0.9308	0.9055	0.9229	0.9377	3513
Micro	**0.7914**	0.8491	**0.9418**	0.9269	**0.9333**	**0.9530**	8835
Macro	**0.7428**	0.8328	**0.9424**	0.9277	**0.9406**	**0.9554**	8835
Class	F1-score						Nb Samples
	SVM	Bagging	RF	Adaboost	CNN	Gradient Boosting	
Buffer Overload	0.6140	0.7776	0.9325	0.9270	0.9353	0.9601	1380
Link Failure	0.8192	0.8712	0.9486	0.9338	0.9392	0.9553	3942
Switch Failure	0.8236	0.8521	0.9379	0.9188	0.926	0.9476	3513
Micro	**0.7914**	0.8491	**0.9418**	0.9269	**0.9333**	**0.9530**	8835
Macro	**0.7523**	0.8336	**0.9397**	0.9266	**0.9335**	**0.9543**	8835

Table 6.7: Time complexity of ML algorithms in RCA for the dataset in dynamic network.

Algorithms	Training Time (ms)	Testing Time (ms)
SVM	**3.70088**	**1.43646**
Bagging	81.4241	67.4686
RF	**0.38460**	**0.00587**
Adaboost	0.35830	0.01368
CNN	**26.1040**	**0.02455**
Gradient Boosting	**1.40897**	**0.01087**

for the static network. The reason is that the error status as well as the delay and loss value change after a given time in the dynamic network. This leads to less fluctuation in the network features, and therefore reduces these performance metrics.

6.1.5 Conclusion

In this section, we present a root cause analysis approach using machine learning and time-series network parameters to identify the root cause of anomalies (type of problems leading to the anomalies) in the SDN environment. There are various machine learning algorithms, so we consider the balance between accuracy and time complexity to select the appropriate algorithm. The experimental results illustrate that the precision, recall and F1-score of the root cause analysis approach are approximately 97 percent in the static network.

In the future, we will extend the troubleshooting datasets to consider more network problems. Besides, there is a massive amount of useful resources (e.g., system log, etc.) in addition to the network parameters to enhance the performance of root cause analysis. Therefore, system logs as well as data-processing applications (e.g., Hadoop, Splunk, etc.), will be considered to improve the performance of root cause analysis.

6.2 Definitive Remediation: Adaptive QUIC BBR Algorithm using Reinforcement Learning for Dynamic Networks

6.2.1 Introduction

Nowadays, the Internet grows rapidly, leading to many hindrances for service providers such as latency optimization, congestion and so on. Growth in latency-sensitive web service leads to various strict latency requirements from end-user's side. Moreover, the Internet is changing from insecure to secure traffic rapidly, which leads to more latency in data transmission (e.g., additional latency due to TLS handshake, etc.). The emerging of QUIC (Quick UDP Internet Connections) [61], a new transport layer network protocol developed by Google, is a potential solution to improve latency in the network. In early 2021, QUIC was standardized by Internet Engineering Task Force (IETF) to offer opportunities to reduce latency in connection establishment and prevent HoL Blocking (Head-of-Line Blocking) in comparison with TCP+TLS. QUIC accounts for over 30 percent of Google's total egress traffic and approximately 7 percent of global Internet traffic [61]. According to a recent statistic of Statista, the global Internet traffic is expected to reach 333 exabytes (EB) per month by 2022 compared with 100 exabytes per month in 2017 [213]. The rapid growth in the global Internet traffic creates much pressure for network infrastructure and results in congestion in the network, particularly in the Covid-19 pandemic. Therefore, congestion control plays an essential role in the network. In fact, congestion control is considered by much existing work to prevent congestion in the network [214, 215]. However, Google implemented a static congestion control mechanism for QUIC in its infancy.

In the early stage, QUIC used Cubic for the congestion control like in TCP. Cubic is a loss-based approach that changes a cwnd (congestion window) parameter to adjust an amount of inflight data (data sent but not yet acknowledged) in a buffer of network devices based on a loss signal. Loss happens in the network when the buffer is overloaded. Consequently, a large buffer size leads to a bufferbloat, influencing on interactive applications (e.g., multiplayer online games, etc.). On the other hand, a small buffer size results in high packet loss and low throughput, impacting multimedia applications (e.g., video streaming, etc.). Moreover, using loss signal in congestion control can lead to uncertain results because non-congestion packet loss is a widespread phenomenon in the network (e.g., port flap in routers, lossy wireless links, etc.) [214].

To overcome these disadvantages, Google has developed BBR (Bottleneck Bandwidth

and Round-Trip Time), a new congestion control algorithm [216]. BBR is a rate-based approach that monitors delivery rate and minimal RTT (Round-Trip Time) to adjust sending rate. However, a static congestion control algorithm cannot be effective across different network conditions [217]. After evaluating the performance of BBR and its variants across network conditions (section 6.2.4.2), we conclude a necessity of an adaptive congestion control algorithm for dynamic networks. Therefore, in this section, we propose an **A**daptive **BBR** algorithm (A-BBR) using Reinforcement Learning (RL) for QUIC protocol. The experimental results (section 6.2.4.2) show that A-BBR has a higher overall average reward by 18.6, 30.76, 13.3 and 27.5 percent compared to origin BBR and three other BBR's variants. Besides, the fairness index of A-BBR is similar to the figure for the others (approximately 1), showing a fair bandwidth sharing between flows of these congestion control algorithms. The contributions of this work are listed as follows:

- A-BBR aims to select the appropriate policies to adaptively change the sending rate to unprecedented changes of network environments.
- A-BBR with RL allows implementing a real-time solution in congestion control, unlike existing studies using Deep Reinforcement Learning (DRL) [217] which requires high resource consumption and time complexity.
- Performance of A-BBR and benchmarks are evaluated in the context of HTTP/3 (Hypertext Transfer Protocol version 3) in contrast to existing studies evaluating congestion control algorithms in the context of HTTP/2 [218]. HTTP/3 offers opportunities to solve the drawbacks of HTTP/2 in terms of HoL Blocking and latency reduction in connection establishment [61]. To the best of our knowledge, this is the first study comparing the performance of congestion control algorithms for QUIC in the context of HTTP/3.

Outline: The remainder of this section is organized as follows. Subsection 6.2.2 presents related work on loss-based congestion control algorithms, rate-based congestion control algorithms and improvement of these algorithms. The adaptive BBR algorithm (A-BBR) is discussed in subsection 6.2.3. Subsection 6.2.4 describes the experimental results of A-BBR. The section concludes with subsection 6.2.5 which highlights our future work.

6.2.2 Congestion Control Mechanisms

In this subsection, we present related work on congestion control algorithms such as loss-based congestion control algorithms (New Reno and Cubic), rate-based congestion control algorithm (BBR) and improvement of rate-based congestion control algorithm. These algorithms are supported in open-source libraries of QUIC protocol (e.g., *lsquic, picoquic, quic-go, quiche*, etc.) [219].

6.2.2.1 Loss-based Congestion Control

According to related work [220, 221], there are many congestion control algorithms in the network such as Tahoe, New Reno, Westwood, BIC, Cubic and so on. New Reno is implemented in Window XP while Cubic is deployed in Linux kernels version 2.6.19 and above from 2006 [221]. These algorithms are loss-based congestion control algorithms that adjust sending rate based on the feedback of receivers (e.g., acknowledgment packets, etc.).

The loss-based congestion control algorithms contain three main phases including *Slow Start, Recovery* and *Congestion Avoidance*. A sender enters the *Slow Start* phase when its congestion window is less than the *ssthresh* threshold which is initialized to an infinite value in the beginning. Then, cwnd increases double every RTT to increase sending rate quickly in the early stage. The sender moves to the *Recovery* phase when there is a loss in the network. The loss happens when one of two following conditions is met. Firstly, the elapsed time from when a packet is sent is equal or bigger than a threshold ($9 \times RTT/8$). Secondly, the difference between the highest packet number of acknowledged packets and the packet number of the unacknowledged packet is equal or bigger than three. In the *Recovery* phase, the sender re-transmits lost packets and reduces *ssthresh* threshold to the half value of current cwnd in New Reno (or reduces ssthresh by 30 percent in Cubic). Besides, it sets current cwnd to current *ssthresh* threshold before moving to the *Congestion Avoidance* phase. The sender ends the *Recovery* phase and enters the *Congestion Avoidance* phase when packets sent during the *Recovery* phase are acknowledged. In the *Congestion Avoidance* phase, the sender increases cwnd by one every RTT in New Reno (or increases cwnd following a cubic function every RTT in Cubic) to avoid congestion.

6.2.2.2 Rate-based Congestion Control

BBR [216] is a rate-based approach that adjusts the sending rate according to a pacing_-gain and a delivery rate estimated from acknowledged packets. The objective is to de-

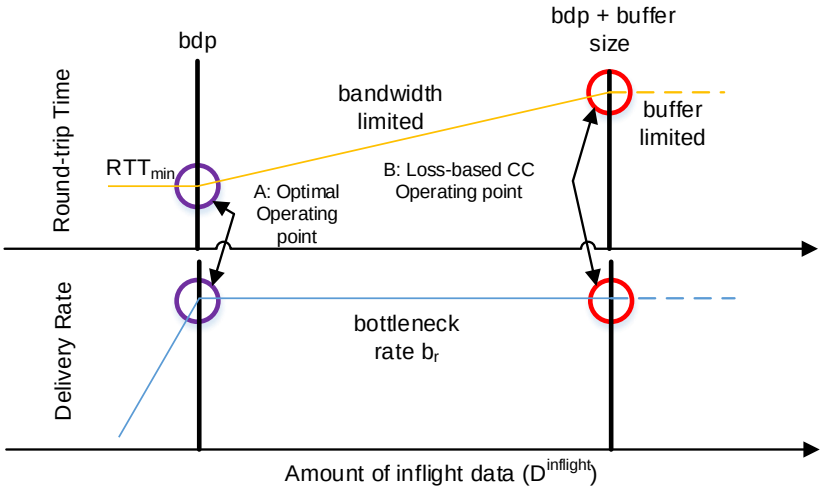


Figure 6.4: Congestion control operating point: delivery rate and RTT against the amount of inflight data.

termine the amount of inflight data ($D^{inflight}$) in order to utilize bandwidth-delay product bdp (available bottleneck bandwidth) effectively. This value is calculated according to delivery rate and RTT as follows: $bdp = b_r \times RTT_{min}$, where b_r is the delivery rate and RTT_{min} is minimal RTT.

Fig. 6.4 shows the delivery rate and RTT against $D^{inflight}$. Operating point *A* shows that $D^{inflight}$ is equal to the bandwidth-delay product bdp. After reaching point *A*, increasing in $D^{inflight}$ does not increase the delivery rate. If $D^{inflight}$ is continued to increase, the buffer starts to be filled with excess data. When the buffer is fully filled, the operating point shifts to *B* where the inflight packets start to be dropped. BBR tries to shift the operating point toward A while loss-based approaches (e.g. Cubic, etc.) move the operating point to B which results in unexpected delay in data transmission.

There are four main phases in BBR including *Startup, Drain, ProbeBW and ProbeRTT*. In the *Startup* phase, BBR increases its sending rate by using a pacing_gain of $2/ln(2)$. If there are three consecutive rounds where increasing the sending rate only leads to a small growth in the delivery rate (less than 25 percent), BBR exits this phase and enters the *Drain* phase. In this phase, BBR decreases its sending rate using a pacing_gain of $ln(2)/2$ until the amount of inflight data matches bdp. After that, BBR enters the *ProbeBW* phase to probe more bandwidth using a pacing_gain cycle of [1.25, 0.75, 1, 1, 1, 1, 1, 1]. BBR increases the sending rate using each pacing_gain in a duration of RTT_{min} which is updated every 10 seconds in the *ProbeRTT* phase. When an increase in the sending rate leads to a growth in the delivery rate, BBR sets the delivery rate to a new sending rate. Otherwise, BBR maintains the current sending rate. BBR contains a drawback related to

a static congestion policy. Concretely, it uses a fixed pacing_gain sequence to adjust the sending rate which is not effective across network conditions.

6.2.2.3 Improvement of Rate-based Congestion Control

In BBR, the sending rate is calculated according to the pacing_gain and the delivery rate. The pacing_gain is changed every period of time (RTT_{min}) following the sequence of [1.25, 0.75, 1, 1, 1, 1, 1, 1]. This fixed pacing_gain sequence is suboptimal in several network environments. For example, in the wireless network, many packets will be aggregated in a single large frame to reduce overhead and increase efficiency in data transmission [222]. This leads to a higher throughput and airtime consumption. Consequently, the amount of inflight data $D^{inflight}$ needs to be increased in order to reach bdp. Therefore, there is a need for a higher pacing_gain in order to probe more bandwidth. Wang et al. [222] proposed BBR+ with a novel pacing_gain sequence of [1.5, 0.5, 1.5, 0.5, 1.5, 0.5, 1.5, 0.5]. BBR+ increases the pacing_gain to 1.5 to probe more bandwidth and then reduces it to 0.5 to decrease the excess data in the buffer. Similarly, Zhang et al. [223] presented BBR-Tsunami with a pacing_gain sequence of [1.5, 0.75, 1.25, 1.25, 1.25, 1.25, 1.25, 1.25]. According to BBR+ and BBR-Tsunami, we propose another pacing_gain sequence of [2, 0.5, 1.5, 0.5, 2, 0.5, 1.5, 0.5] to probe more bandwidth in the context of high packet loss. This algorithm refers to M-BBR (Modified BBR).

A static congestion control algorithm is ineffective for various network conditions, so Nie et al. [217] proposed TCP-RL, an adaptive congestion control schema in TCP using Deep Reinforcement Learning. TCP-RL takes into account three network parameters (throughput, RTT and loss) using Deep Reinforcement Learning to select an appropriate congestion control algorithm. In contrast, we propose an adaptive BBR algorithm in QUIC protocol for dynamic networks.

6.2.3 Proposal: Adaptive BBR Algorithm

In this section, an adaptive BBR algorithm (A-BBR) is proposed to perform effectively across network conditions. This algorithm is inspired by a reinforcement learning task that selects the appropriate set of pacing_gain thanks to feedbacks from the network environment to optimize an objective function. Determining the appropriate policies to solve this task is formalized in this section.

Selecting the appropriate pacing_gain sequence is formalized as a reinforcement learn-

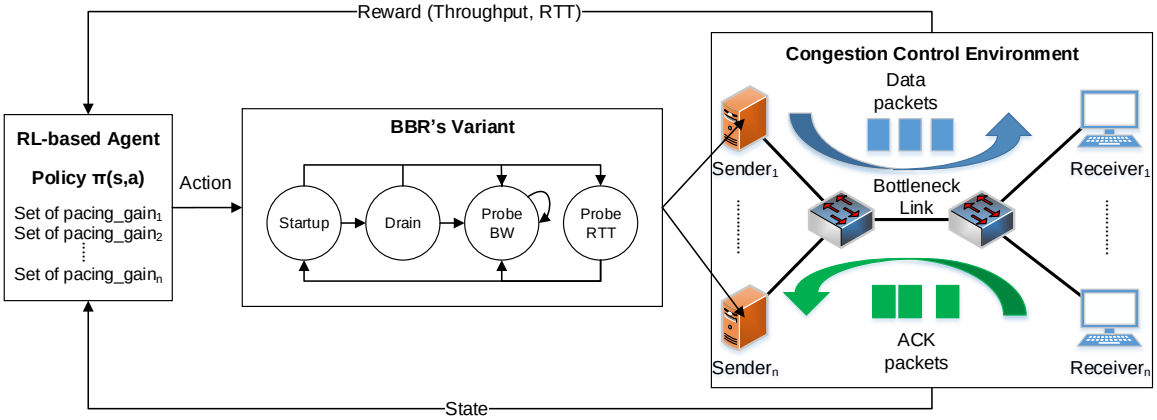


Figure 6.5: Adaptive BBR alggorithm.

ing task (Fig. 6.5). The detail is described as follows:

- *Agent*: An agent is an entity in the network system executing its tasks according to a learning algorithm. In A-BBR algorithm, an agent selects the appropriate pacing_gain sequence to optimize an objective function.

- *State*: An instance of the network state that is monitored by the agent.

- *Action*: An action describes how an agent reacts to the network environment. In A-BBR algorithm, the action is a pacing_gain sequence corresponding to each BBR variant. The considered pacing_gain sequences contain [1.25, 0.75, 1, 1, 1, 1, 1, 1] (Origin-BBR), [1.5, 0.5, 1.5, 0.5, 1.5, 0.5, 1.5, 0.5] (BBR+), [1.5, 0.75, 1.25, 1.25, 1.25, 1.25, 1.25, 1.25] (BBR-Tsunami) and [2, 0.5, 1.5, 0.5, 2, 0.5, 1.5, 0.5] (M-BBR). Changing the pacing_gain leads to the change in sending rate corresponding to various network conditions. These pacing_gain sequences are selected to probe more available bandwidth across network conditions (e.g., lossy network, etc.).

- *Policy*: A policy is a mapping between a state and an action in the network environment.

- *Reward*: A reward is a feedback from the network environment. At step t, the agent monitors a network state s and executes an action a. After that, the network state moves to s', and the agent receives a corresponding reward r. In several studies [217, 224], reward is defined as follows: $r = \log \frac{Throughput}{\text{RTT}}$. However, this reward function leads to a small difference in the performance of BBR with considered pacing_gain. Therefore, reward is defined as follows: $r = \frac{Throughput}{\text{RTT}}$. The reward is normalized

into a value between 0 and 1 according to min-max normalization function [119] to facilitate a comparison between the congestion control algorithms.

The objective of RL is to optimize the accumulative reward O_f (Equ. 6.4):

$$O_f = Max\, E[\sum_{t=0}^{\infty} \gamma_t \times r_t]. \tag{6.4}$$

where $\gamma_t \in [0, 1]$ is a discount factor.

There are two approaches in RL: model-based and model-free. The first approach learns an environment model to optimize the optimality while the second one learns its policies without prior information in the network environment. In this work, the model-free approach is considered because it requires less storage cost and dependence on accuracy of initial information than the model-based approach [190]. The quality of action a is evaluated via Q-value $Q(s, a)$ which is updated every step as in Eq. 6.5:

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a')]. \tag{6.5}$$

where α is a learning rate and γ is a discount factor.

In this approach, $\epsilon - greedy$ [225] is considered as a selection algorithm to balance between exploration and exploitation phases. Concretely, the agent chooses the pacing_gain sequence with the highest Q-value with a probability of (1-ϵ). Otherwise, the agent chooses the pacing_gain sequence randomly. Then, the ϵ value is reduced every step according to ϵ_decay until it drops to ϵ_min.

The time complexity of the reinforcement learning algorithm is $O(m \times N)$ where N is the number of actions, and m is the size of state space [192].

6.2.4 Experimental Results

6.2.4.1 Experimental Setup

The testbed is implemented with dumbbell topology (in Fig. 6.5) similar in [216]. *mininet* [168] is used to emulate network topology and change network states in the bottleneck link. This link is set with a bandwidth of 10 mbps, a RTT of [50, 200, 100 ms] and a loss of [30, 10, 0, 20, 1, 5 %] to create a variety of network conditions. Therefore, there are 18 considered network conditions corresponding to the change of loss and RTT. The number of senders and receivers is set from one to three. In the experiments, a file with a size of 40 MB is sent from servers to clients. In sender's side, the proposal (A-BBR) and benchmarks (Cubic, Origin-BBR, BBR+, BBR-Tsunami and M-BBR) are implemented according to customizing the library *lsquic* [99] to provide QUIC over HTTP/3. In receiver's side, Google

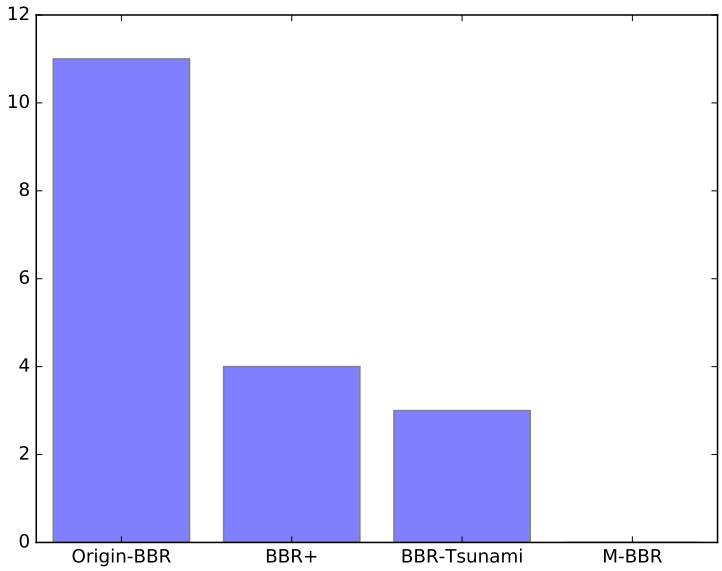


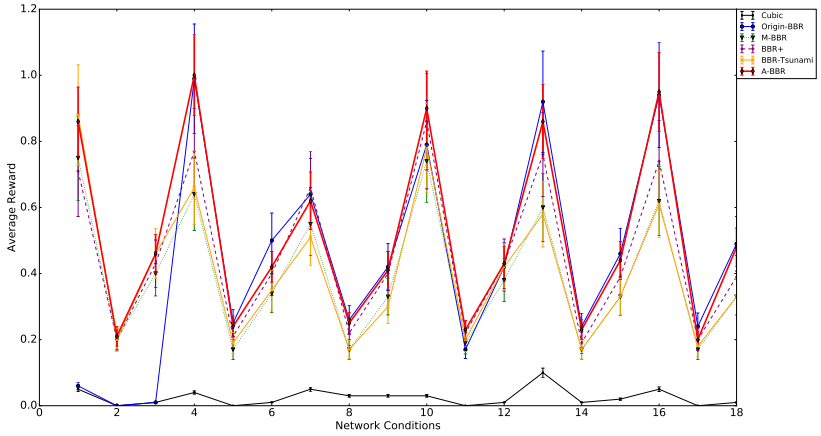
Figure 6.6: Number of network conditions in which each congestion control algorithm obtains the best performance.

Chrome is used to communicate with senders via QUIC over HTTP/3. In reinforcement learning, ϵ is initiated to 1. ϵ_decay, ϵ_min, α and γ are set to 0.95, 0.3, 0.7 and 1, respectively. These parameters are chosen according to the experiments.

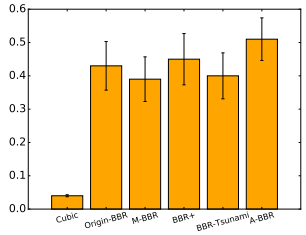
6.2.4.2 Performance Analysis

Fig. 6.6 illustrates the number of network conditions in which each congestion control algorithm obtains the best performance (largest reward). There are 4 BBR variants corresponding to 4 pacing_gain sequences (described in section 6.2.3). This figure indicates that a specific congestion control algorithm cannot perform well across various network conditions. In the considered network conditions, Origin-BBR performs well overall, but it is effective in only 11 network conditions. In contrast, M-BBR performs best in none of these network conditions. This confirms the necessity of the adaptive BBR algorithm for QUIC in dynamic networks.

In reality, network conditions are changed over time. Consequently, we evaluate the performance of A-BBR and benchmarks in the considered network conditions. Fig. 6.7a shows their standard deviations and average rewards in these network conditions. A-BBR can achieve good results in almost network conditions that will be changed every 100 time steps. The change in these network conditions leads to the change of RTT and loss in the network. Consequently, this leads to the change of throughput, resulting in the fluctu-



(a) Average reward and standard deviation in each considered network condition.



(b) Overall average reward and standard deviation in all considered network conditions.

Figure 6.7: Average reward and standard deviation of A-BBR and benchmarks.

ation of reward in these network conditions. According to RL, A-BBR can learn to keep the same pacing_gain sequence or select other sequences to obtain good results and optimize the accumulative reward. Loss is set to 30 percent, and RTT varies from 50 to 200 in the first three network conditions. In these network conditions, the average reward of Origin-BBR ranges from 0.01 to 0.06. The reason is that Origin-BBR increases the sending rate with a pacing_gain of 1.25 for a duration (RTT_{min}), but there is 30 percent of loss in this scenario. This leads to a decrease in throughput and reward against time. When Origin-BBR tries with lower pacing_gain value, its throughput and reward continue to decrease against time. Moreover, the average reward of Cubic is the lowest in the considered congestion control algorithms. In these network conditions, there is a loss in the network. Cubic reduces its cwnd according to loss signal, so there is a decrease in its throughput and average reward. A noticeable feature from Fig. 6.7a that the average reward of the considered congestion control algorithms reaches a peak every three network conditions. In every three network conditions, RTT is set to 50 ms in the first network condition, then increases to 200 ms in the second one and finally reduces to 100 ms in the last one. This leads to the fluctuation of average reward in these algorithms. The overall average reward and standard deviation of these congestion control algorithms obtained in all considered network conditions are illustrated as in Fig. 6.7b. The overall average reward of the proposal is the highest with 0.51 while the figure for the Origin-BBR, M-BBR, BBR+, BBR-Tsunami and Cubic are 0.43, 0.39, 0.45, 0.4 and 0.04, respectively. The overall average reward of our proposal shows an improvement of 18.6, 30.76, 13.3 and 27.5 percent in comparison with the figure for Origin-BBR and three other BBR's variants. Be-

sides, the overall standard deviation of A-BBR is nearly equal to the figure for Origin-BBR, M-BBR, BBR+ and BBR-Tsunami. Their overall standard deviations range from 0.064 to 0.077. Some important results related to overall average reward and overall standard deviation are summarized in Tab. 6.8.

Table 6.8: Some important results of the considered congestion control algorithms.

Algorithms	Origin-BBR	M-BBR	BBR+
Overall average reward	0.43	0.39	0.45
Algorithms	BBR-Tsunami	Cubic	A-BBR
Overall average reward	0.4	0.04	0.51
Algorithms	Origin-BBR	M-BBR	BBR+
Overall standard deviation	0.073	0.067	0.077
Algorithms	BBR-Tsunami	Cubic	A-BBR
Overall standard deviation	0.069	0.003	0.064

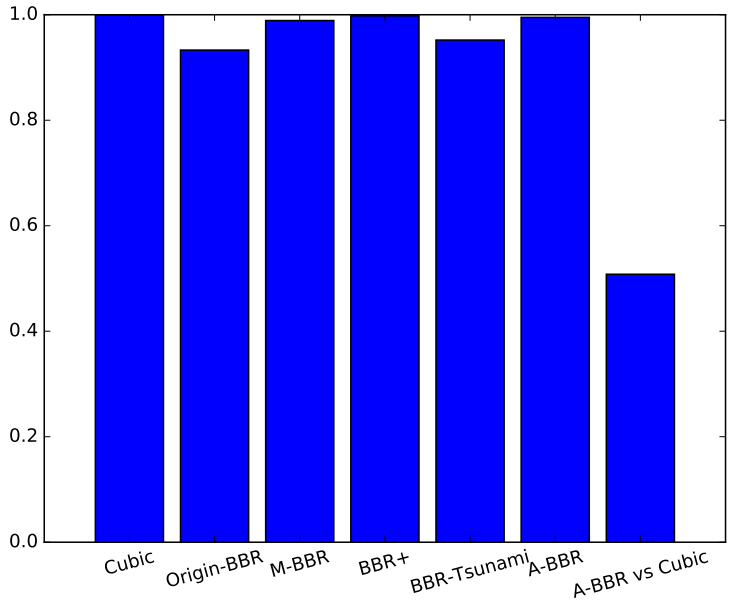


Figure 6.8: Fairness of A-BBR and benchmarks in dynamic network conditions.

Fig. 6.8 indicates the Jain's fairness of the considered congestion control algorithms in network condition with a bandwidth of 10 Mbps, a loss of 5% and a RTT of 100 ms. In this experiment, the file with size of 40 MB is sent from three senders to three corresponding receivers. All considered congestion control algorithms can achieve a good fairness index

because their values approximate to 1. The last column shows fairness between A-BBR and Cubic with a flow of A-BBR and two Cubic flows. The fairness index between A-BBR and Cubic approximates to 0.5, so there is less fair bandwidth sharing between flows of these algorithms. With the loss of 5 percent, Cubic reduces its cwnd according to loss signal while A-BBR can probe to adjust its sending rate. Therefore, the throughput of A-BBR is higher than Cubic which leads to a low fairness index.

6.2.5 Conclusion

This section proposes an adaptive BBR algorithm (A-BBR) for QUIC to adapt to the unexpected changes of network environments because a static congestion control algorithm (original BBR) is not effective across various network conditions. According to RL, A-BBR changes the sending rate adaptively using the feedback from the network environments. Moreover, A-BBR and the benchmarks are evaluated in the context of HTTP/3. The experimental results show that A-BBR achieves good results related to the average reward and fairness index across considered network conditions compared to the benchmarks.

Despite the good results, A-BBR as well as BBR's variants contain a drawback related to the fairness with existing congestion control algorithms (e.g., QUIC Cubic, etc.). Google is developing BBR v2 which offers an opportunity to solve this drawback. In the future, we will implement the adaptive algorithm for BBR v2. Besides, we will extend this study to select appropriate congestion control algorithms (e.g., QUIC Cubic, QUIC New Reno, etc.) across various network conditions.

Conclusions and perspectives

«Software Defined Networks
and the maturing of the Internet»

Nick Mckeown,
IET Appleton, 2014

1 Summary of Contributions

Nowadays, the rapid development of the Internet results in more and more multimedia services (e.g., video streaming, online game, etc.) and diversified network devices such as IoT devices (Internet of Things). The huge number of network devices not only enlarges the network infrastructures but also leads to many critical issues in the network. A downtime on cloud-based services leads to negative economic impacts (e.g., the loss from over $30,000 to $6,700,000) for many service providers (e.g., Youtube, Facebook, Paypal, etc.) [226]. Therefore, network troubleshooting has been increasingly concerned over the past two decades. The purpose is to detect anomalies, identify its root causes and implement remediation approaches to deal with these root causes definitively.

Although network troubleshooting is studied thoroughly by the research community, the emerging of encrypted traffic entails many questions regarding its deployment in the context of encrypted traffic. For example, Google developed QUIC (Quick UDP Internet Connections), a transport layer network protocol from 2012 to improve latency in data communication. QUIC also provides an encryption algorithm to guarantee secure data communication in the network. In this case, from the point of view of NOs (Network Operators), useful information obtained in data packets is hidden such as sequence number, acknowledgment number, payload signature, and so on. This causes to drawbacks for network performance monitoring approaches (e.g., QoE estimation, application identification, etc.) and intrusion detection systems. These approaches are essential components in network troubleshooting frameworks, so encrypted traffic brings certain obsta-

cles to data collection (e.g., QoE estimation, application identification, etc.) and remediation approaches (e.g., application-aware traffic engineering, payload-based intrusion detection systems, etc.) in network troubleshooting.

Moreover, computer networks nowadays become more complex due to many network devices (e.g., IoT devices, etc.), so the processing time for root cause analysis and remediation approaches becomes larger. Therefore, network systems have negative impacts (e.g., high latency, high loss, etc.) during the root cause analysis of anomalies and remediation. Consequently, it is necessary for network troubleshooting frameworks that guarantee network availability during the root cause analysis and remediation and deal with some of the limitations encountered in encrypted traffic. Therefore, in this thesis, we propose a novel troubleshooting framework for NOs in the context of encrypted traffic. The proposed troubleshooting framework contains five main modules: data collection, anomaly detection, temporary remediation, root cause analysis, and definitive remediation. These modules are described as follows:

- Data collection: In this module, we implement a parameter measurement module to collect troubleshooting data and build datasets in static and dynamic networks. A sample in the datasets contains nine time-series network parameters (e.g., latency, loss, etc.) and corresponding labels for three root causes: link failure, switch failure and buffer overload. From the point of view of NOs, network performance metrics (e.g., quality of experience, application classes, etc.) are hidden due to encrypted traffic. Consequently, we propose a novel traffic classification approach for QUIC traffic. The information about application classes plays an important role in application-aware remediation approaches (e.g., application-aware traffic engineering, etc.). There are two main classification stages in the traffic classification approach. The first one analyzes flow-based features using a ML (Machine Learning) algorithm to classify the network traffic into chat, VoIP or elephant flows. Random Forest is considered as a ML algorithm in this stage after evaluating the performance of several ML algorithms (e.g., Support-Vector Machine, Random Forest, etc.). The second classification stage analyzes packet-based features using CNN (Convolutional Neural Network) algorithm to classify the elephant flows into video streaming, file transfer or Google play music. In this stage, performance is evaluated in various scenarios such as different subsets of input vector and various loss functions. The experiment results show that classification results can achieve ap-

proximately 99 percent for video streaming, file transfer and Google play music and approximately 96 percent for other applications such as chat and VoIP.

- Anomaly detection: After collecting troubleshooting data, this information is analyzed in anomaly detection module to detect anomalies in the network. In this thesis, we implement an anomaly detection approach using machine learning algorithm to classify network states into normal or abnormal states. The experimental results show that the precision, recall and F1-score of the proposed anomaly detection approach achieve up to 99 percent in the considered datasets.

- Temporary remediation: If anomalies happen in the network, we need to implement a temporary remediation module that is responsible for reducing negative impacts of problems (e.g., high latency, high loss, etc.) and guaranteeing the network availability during the root cause analysis and definitive remediation. In the temporary remediation module, we propose an application-aware segment routing mechanism in the SDN (Software-defined Networking) environment. This module is to implement a variety of routing strategies corresponding to different applications. First, the temporary remediation module identifies application classes according to traffic classification. Then, this module selects an appropriate routing strategy for a specific application thanks to RL (Reinforcement Learning) algorithm and feedback from the network environment. In the RL algorithm, there are various selection algorithms (e.g., *E-greedy, softmax*, etc.), so we evaluate the performance of these algorithms to select appropriate ones for four scenarios: perfect scenario, delay scenario, loss scenario and scenario with both delay and loss. Besides, the proposed segment routing mechanism is evaluated with benchmarks (*Standard_SR* and *Max_QoE* mechanisms) related to MOS (Mean Opinion Score) against time, average optimal MOS, median MOS, 95% confidence interval of MOS, CPU usage and control overhead. The experimental results illustrate that MOS of the proposed mechanism is nearly equal or higher than the figure for benchmarks while it requires less CPU usage and control overhead compared to benchmarks in the considered scenarios. Concretely, the proposed mechanism reduces up to 64.39 percent of overhead in comparison with *Max_QoE* mechanism.

- Root cause analysis: In parallel with the temporary remediation, we implement a root cause analysis module to identify the root cause of anomalies in the network.

In this module, we implement a root cause analysis approach using time-series network parameters and a ML algorithm. In the thesis, congestion is considered to generate anomalies in the network. The objective of root cause analysis is to identify the root cause of congestion (link failure, switch failure or buffer overload). Besides, the performance and processing time of ML algorithms (e.g., Support Vector Machine, Bagging, Random Forest, Adaboost, etc.) are evaluated to select an appropriate algorithm for datasets in static and dynamic networks. The experimental results show that Random Forest can achieve high accuracy while requiring less processing time than other algorithms. Consequently, it is considered as a ML algorithm in the root cause analysis approach. F1-score of the proposed root cause analysis mechanism achieves up to 97 percent while it only requires 0.00143 ms for the testing time for the dataset in static network.

- After identifying the root cause of anomalies, we implement a definitive remediation module to solve the root cause definitively. If congestion results from link failure or switch failure, we send notifications to administrators so that they can address these root causes. Otherwise, we propose an adaptive BBR (Bottleneck Bandwidth and Round-Trip Time) algorithm using the RL algorithm to deal with the switch's buffer overload in the network. This approach aims to adjust sending rate at sender sides adaptively for various network conditions. The performance of proposed congestion control algorithm is evaluated with benchmarks in the context of HTTP/3 in contrast to HTTP/2 in existing studies. The experimental results show that the proposed congestion control algorithm achieves a high overall average reward and fairness index in comparison with the benchmarks. Concretely, the overall average reward of the proposal is the highest with 0.51 while the figure for the Origin-BBR, M-BBR, BBR+, BBR-Tsunami and Cubic are 0.43, 0.39, 0.45, 0.4 and 0.04, respectively. The overall average reward of our proposal shows an improvement of 18.6, 30.76, 13.3 and 27.5 percent in comparison with the figure for Origin-BBR and three other BBR's variants.

Despite the promising results, the proposed troubleshooting framework has several limitations as follows:

- In this framework, we consider congestion as a use-case to generate anomalies and build troubleshooting datasets related to three root causes: link failure, switch fail-

ure, and buffer overload. However, there are various kinds of anomalies (e.g., server disruption, etc.) as well as its root causes are not taken into account yet in the framework.

- Encrypted traffic not only leads to a lack of information about application class but also results in many obstacles for network performance monitoring approaches (e.g., QoE estimation, etc.) and intrusion detection systems using a deep packet inspection (e.g., DDoS detection, botnet detection, etc.). However, we only take into account the application class as a use case in the proposed troubleshooting framework.

- In the proposed traffic classification approach, we use supervised learning algorithms to classify the network traffic into five applications, but it requires a large labeled dataset. Besides, building large labeled datasets is time-consuming and costly. Consequently, building larger datasets with more kinds of applications will face many obstacles in the future.

- According to the related work, there are many remediation approaches in network troubleshooting such as routing, load balancing, etc. However, we take into account the routing aspect as an example in the temporary remediation module. The routing aspect can effectively reduce negative impacts of congestion in the network devices (e.g., router, switch, etc.), but it is ineffective with other anomalies (e.g., server disruption, switch configuration, etc.) which is not considered yet in the framework.

- Congestion can be caused by link failure, switch failure or buffer overload. In this thesis, buffer overload is taken into account, and we present a congestion control algorithm to solve it definitively. However, remediation approaches for other root causes need to be studied in the troubleshooting framework.

2 Perspectives and Future Work

According to promising results, this work can be extended with various research perspectives as follows:

- First, we contribute troubleshooting datasets in terms of link failure, switch failure and buffer overload. In the future, we will implement simulation scenarios to col-

lect more data related to other root causes to facilitate network troubleshooting. Besides, a root cause analysis need to be studied to identify these root causes in this case.

- The emerging of encrypted traffic causes to limitations for network management and network troubleshooting regarding network performance monitoring approaches (e.g., QoE estimation, application identification, etc.) and intrusion detection system using a deep packet inspection (e.g., DDoS detection, botnet detection, etc.). In this thesis, we place a special focus on traffic classification solutions for identifying application classes in the context of encrypted traffic. Other aspects need to be thoroughly taken into account in the future.

- The proposed traffic classification approach uses supervised ML algorithms to classify encrypted network traffic. It requires large labeled datasets, but capturing large labeled datasets is a costly and time-consuming. Consequently, many studies focus on traffic classification approaches using semi-supervised ML algorithms. The objective is to pre-train a model on a large unlabeled dataset and transfer learned weights to a new model that is retrained with a small labeled dataset. Therefore, these approaches will be taken into account as a part of our future work.

- The application-aware segment routing mechanism proposed in this work is implemented for a certain applications such as video streaming, file transfer and VoIP. This mechanism selects an appropriate routing strategy corresponding to a specific application to meet strict SLA (Service-level Agreement) requirements. Next step of this thesis is to extend to develop a more effective proof-of-concept in a product environment.

- Finally, we propose an adaptive BBR algorithm for QUIC to adapt to the unexpected changes of network environments. The purpose is to change sending rate at sender sides adaptively in dynamic network. We will extend this work to select appropriate congestion control algorithms (e.g., QUIC Cubic, QUIC New Reno, QUIC BBR, etc.) across various network conditions. In a specific algorithm, we will implement RL algorithm to select appropriate sending rate to improve network performance in dynamic network.

Version abrégée en Français

1 Contexte général

Les avancées dans le domaine des réseaux de télécommunication, l'avènement de l'Internet des objets et la multiplication des services font qu'aujourd'hui les mécanismes embarqués dans le cœur des réseaux de transport prennent une place importante dans l'offre de services. Par conséquent, le réseau est devenu plus complexe et nombreux sont les problèmes qui peuvent impacter le fonctionnement continu des services fournis aux usagers. Pour se rendre compte de l'intérêt de cet aspect, Tab. 8.9, publiée en 2020, donne un aperçu du coût engendré par les interruptions des services chez certains opérateurs [7]. Par exemple, nous constatons que YouTube et PayPal ont subi une perte financière entre $ 34,000 et $ 6,700,000 pour quelques heures d'interruption de leurs serveurs dues à des défaillances techniques.

De même, de nombreux services Cloud sont aujourd'hui perturbés par des cyberattaques, comme c'est le cas, des attaques de type DDoS (Distributed Denial of Service) conçues pour surcharger et perturber les services réseau en les saturant de demandes d'accès. En février 2018, Github [8] en a été la cible avec 1,3 Tb/s de trafic issu de la réception de 126,9 millions de paquets par seconde ayant saturé leurs serveurs. Bien que cela soit la plus grande attaque DDoS enregistrée à l'époque, les systèmes de GitHub n'ont souffert que de 20 minutes d'interruption. GitHub a en effet mis en place un mécanisme

	Arrêt (Heures)	Coût ($)
Youtube	0.17	34,000
CloudFlare	1	168,000
Zoho	33.5	600,000
Cisco	5.33	1,066,000
eBay	6.25	1,406,250
Facebook	8.5	1,700,000
Paypal	30.2	6,795,000

Table 8.9: Le coût lié aux interruptions de services.

de migration pour détecter et empêcher ce type d'attaque. Selon le dernier rapport de Radware, au cours des quatre premiers mois de 2021, le volume des attaques DDoS a augmenté de 30 % [8]. Les attaques informatiques ne sont bien évidemment pas les seules causes d'interruption de services, la société Facebook a connu, par example, le 04 octobre 2021, l'une des pannes les plus importantes de son histoire à cause d'un problème de configuration BGP (Border Gateway Protocol) qui eut pour conséquence de ne plus annoncer les routes avec leurs préfixes DNS (Domain Name System) [227].

En conclusion, il est aujourd'hui nécessaire de réfléchir à de nouveaux mécanismes de dépannage efficaces et adaptés aux évolutions actuelles et futures des systèmes et des réseaux afin d'assurer leur résilience.

2 Motivations

Les opérations liées au diagnostic et à leur résolution dans les réseaux constituent des tâches souvent chronophages. Il est en effet souvent nécessaire de consacrer un temps plus au moins long pour analyser des causes, poser le diagnostic et ensuite proposer une méthode de résolution dans l'objectif du rétablissement des services. Cette période peut varier de quelques secondes à quelques heures selon l'état des anomalies constatées dans le réseau [11]. Durant cette période, et sans aucune action correctrice, les systèmes continuent à subir une dégradation de leurs performances [12]. De plus, l'augmentation de la part du trafic chiffré dans l'Internet, qui est passé de 40 % en 2016 à 80 % en 2019 [13], conduit à de nouveaux challenges au regard des mécanismes traditionnels [4, 5, 6], en particulier celui de la perception de l'usager au regard des services qui lui sont fournis. En effet, plusieurs mécanismes mis en place aujourd'hui par les opérateurs réseau se basent sur l'identification du type de flux transporté afin de concevoir des stratégies permettant de prendre en compte la satisfaction de l'usager terminal.

3 Contributions

Dans cette thèse, nous proposons une nouvelle architecture de détection et de dépannage pour les réseaux opérateurs qui prend en compte la nature chiffrée du trafic transporté. Cette architecture (Fig. 8.9), modulaire par son aspect fonctionnel, comprend la collecte de données, la détection d'anomalies, la résolution temporaire, l'analyse des causes profondes et la résolution définitive. Ces modules sont décrits comme suit :

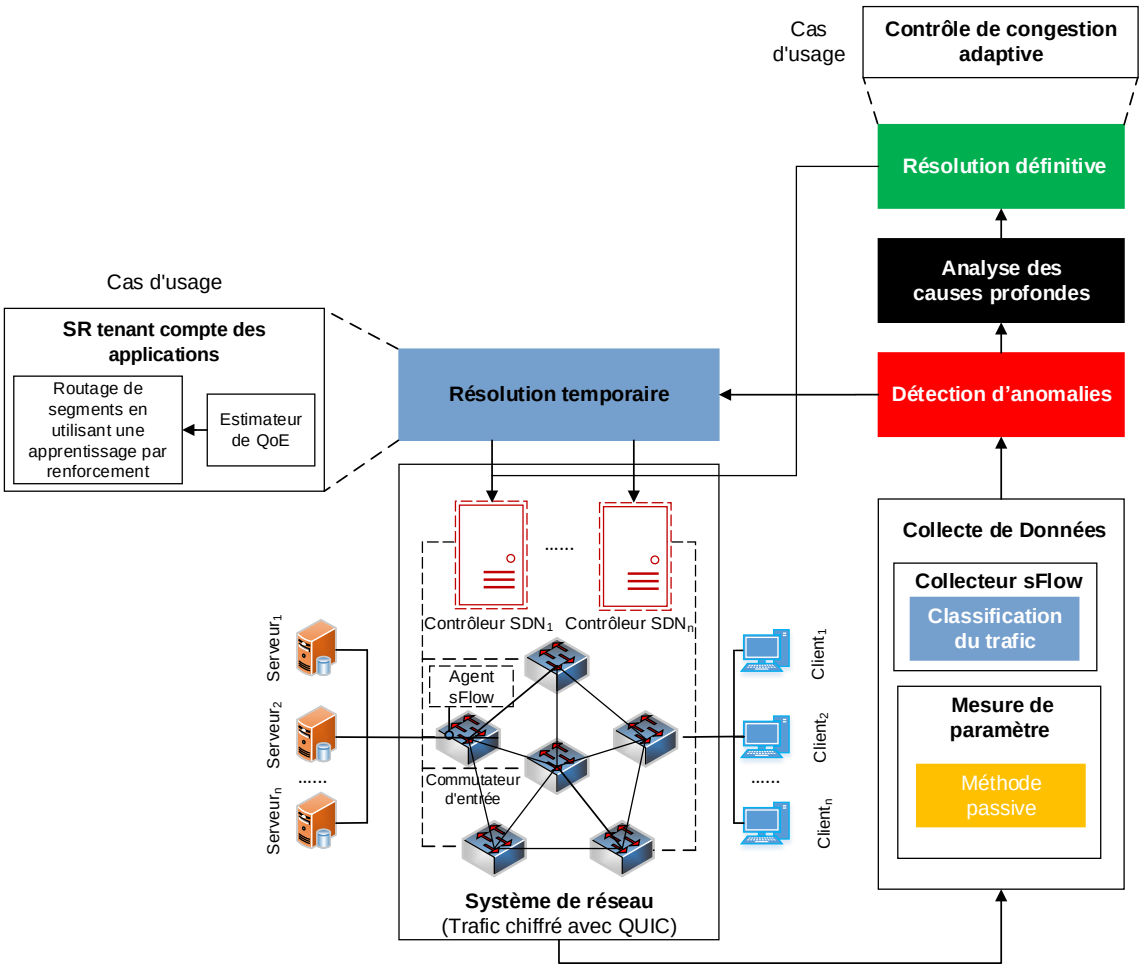


Figure 8.9: L'architecture proposée pour le dépannage des réseaux opérateurs.

- Module de collecte de données : celui-ci est conçu afin de collecter des informations sur l'état du réseau et du trafic. Ainsi, nous proposons une nouvelle approche de classification pour le trafic QUIC qui comporte deux étapes principales. La première phase analyse les caractéristiques basées sur le contenu du flux, à l'aide d'un algorithme d'apprentissage automatique (ML : Machine Learning) de type « Random Forest ». L'objectif est ici de classer le trafic transporté selon le type de l'application utilisée: messagerie instantanée, Voix sur IP et « autres ». La deuxième étape étudie les caractéristiques intrinsèques du type « autres » en se basant sur la nature des paquets. Nous utilisons ici un algorithme de type réseau neuronal convolutif (CNN : Convolutional Neural Network) pour classer ces flux en 3 catégories : « vidéo », « transfert de fichiers » ou « musique ». Les performances de ce module sont évaluées dans divers scénarios et les résultats montrent que la classification obtenue pouvait atteindre les 99 % pour l'identification des flux vidéo, transfert de fichiers et musique et environ 96 % pour les autres types.

- Module de détection des anomalies : Il se base sur les données collectées par le précédent module afin de détecter les anomalies dans le réseau. La méthode mise en œuvre, basée sur des techniques d'apprentissage automatique, analyse l'évolution des caractéristiques fonctionnelles du réseau de transport afin d'identifier les eventuelles anomalies. Les problèmes de réseau peuvent en effet entraîner une dégradation des performances et une fluctuation dans le fonctionnement du réseau qu'il est possible de reconnaître automatiquement afin de classer le système en 2 états : « normal » et « anormal ». Les résultats expérimentaux montrent que la précision du mécanisme de détection d'anomalies proposé atteigne 99 % de taux de succès pour les jeux de données considérés.

- Module de résolution temporaire : Ce module prend la forme d'un mécanisme de routage de segments tenant compte des applications usagers, l'objectif est de résourdre temporairement certaines défaillances constatées dans le réseau, le temps d'en rechercher les causes réelles. L'approche proposée se base sur le paradigme SDN (Software-defined Networking) et a pour objectif de choisir automatiquement des stratégies de routage en fonction de l'application et du contexte en cours. Tout d'abord, le module de résolution temporaire identifie les classes d'applications en fonction de la classification du trafic. Ensuite, il sélectionne d'une manière automatisée, en utilisant un algorithme d'apprentissage par renforcement (RL, Reinforcement Learning), une stratégie de routage appropriée. Les résultats expérimentaux montrent que l'approche mise en place, bien que moins gourmande en ressources CPU et réseau, offre des résultats au minimum comparables aux autres approches existantes. Concrètement, le mécanisme proposé réduit jusqu'à 64.39 % la surcharge par rapport à un mécanisme qui maximise systématiquement la qualité perçue par l'usager.

- Analyse des causes profondes : Parallèlement à la résolution temporaire, un module d'analyse des causes profondes des anomalies a été mis en œuvre. Nous avons considéré la congestion comme cas d'étude et nous nous sommes donnés comme objectifs d'identifier sa cause (défaillance d'un lien, défaillance d'un commutateur ou surcharge de la mémoire tampon). Plusieurs algorithmes et approches ont été évalués et le choix s'est porté sur l'algorithme Random Forest qui offre un bon compromis entre le temps d'exécution et la précision recherchés.

- Enfin, nous nous sommes intéressés aux mécanismes de résolution du problème dans le cas de la congestion. Évidemment, la défaillance d'un lien ou d'un commutateur nécessite une intervention humaine, mais en ce qui concerne la surcharge de la mémoire tampon, nous avons proposé un algorithme adaptatif pour le contrôle de congestion de type BBR (Bottleneck Bandwidth and Round-Trip Time) afin de remédier à la panne constatée. Cette approche, qui se base sur un algorithme d'apprentissage par renforcement, vise à ajuster le taux d'envoi du côté de l'expéditeur, de manière adaptative, en fonction des conditions du réseau. Les performances de l'algorithme proposé sont évaluées dans le contexte du protocole HTTP/3 et comparées aux approches existantes: Origin-BBR, M-BBR, BBR+, BBR-Tsunami et Cubic. La récompense moyenne globale de notre proposition montre une amélioration de 18.6, 30.76, 13.3 et 27.5 % par rapport à la version Origin-BBR ainsi que de ses trois autres variantes.

4 Conclusion et Perspectives

Le développement rapide de l'Internet s'est traduit par une augmentation du nombre de services et de dispositifs connectés. Ceci a eu comme conséquence la complexification du réseau et l'apparition de nombreux problèmes et dysfonctionnements qui peuvent aller jusqu'à provoquer l'arrêt des services entraînant ainsi des répercussions économiques graves pour les fournisseurs de services [226]. Par conséquent, la mise en œuvre d'approches mise en forme de dépannage des réseaux dans le but de détecter les anomalies, d'identifier leurs causes profondes et de mettre en œuvre des approches de résolution automatique, est devenue primordiale. Néanmoins, l'émergence du trafic chiffré a rendu difficile, voire impossible, l'utilisation de certaines approches. Dans cette thèse, nous proposons une nouvelle architecture de dépannage pour les réseaux opérateurs adaptés à ce contexte. L'architecture de dépannage proposé contient cinq modules principaux : la collecte de données, la détection d'anomalies, la résolution temporaire, l'analyse des causes profondes et la résolution définitive.

À l'avenir, ce travail peut être étendu à diverses perspectives:

- La classification et l'identification des applications dans le contexte du trafic chiffré peuvent être améliorées en considérant d'autres aspects non pris en compte dans le cadre de cette thèse, en y intégrant d'autres paramètres applicatifs, plus de données

émanant d'autres sources, voire d'autres modèles.

- La collecte des données étiquetées d'apprentissage nécessaires à la classification de trafic est coûteuse et chronophage. Il serait intéressant d'étudier la possibilité de pré-entraîner un modèle sur un ensemble de données non étiquetées et de transférer les poids appris à un nouveau modèle qui est entraîné de nouveau avec un petit ensemble de données étiquetées.

- Un mécanisme de routage par segment tenant compte de l'application a été proposé dans la cadre de cette thèse. Ce mécanisme sélectionne une stratégie de routage appropriée correspondant à une application spécifique afin de répondre aux exigences strictes du contrat client-opérateur SLA (Service-level Agreement), mais l'analyse que nous avons effectuée a pris la forme d'une preuve de concept qu'on pourrait élargir à d'autres applications afin de confirmer nos conclusions ou de les faire évoluer.

- Enfin, nous nous sommes intéressés à la congestion comme cause probable de dysfonctionnement et il faudra certainement considérer d'autres fonctionnalités afin de faire évoluer l'architecture proposée.

5 Liste des publications

Revues internationales avec comité de lecture

- **Van TONG**, Sami SOUIHI, Hai Anh TRAN and Abdelhamid MELLOUK. "SDN-based Application-aware Segment Routing for Large-scale Network." *IEEE Systems Journal* (Accepté en octobre 2021), 10 pages, doi: 10.1109/JSYST.2021.3123809.

Conférences internationales avec comité de lecture et actes

- **Van TONG**, Hai Anh TRAN, Sami SOUIHI and Abdelhamid MELLOUK. "Machine Learning based Root Cause Analysis for SDN Network." *2021 IEEE Global Communications Conference (GLOBECOM)*, 6 pages, Madrid, Spain, 7–11 December, 2021 (en cours).

- **Van TONG**, Sami SOUIHI, Hai Anh TRAN and Abdelhamid MELLOUK. "Service-centric Segment Routing Mechanism using Reinforcement Learning for Encrypted

Traffic." *2020 16th International Conference on Network and Service Management (CNSM)*, 2020, pp. 1-5, doi: 10.23919/CNSM50824.2020.9269070.

- Lamine Amour, **Van TONG**, Sami SOUIHI, Hai Anh TRAN and Abdelhamid MELLOUK. "Quality estimation framework for encrypted traffic (q2et)." *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1-6, doi: 10.1109/GLOBECOM38437.2019.9014234.

- **Van TONG**, Hai Anh TRAN, Sami SOUIHI and Abdelhamid MELLOUK. "Network troubleshooting: survey, taxonomy and challenges." *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, 2018, pp. 165-170, doi: 10.1109/SaCoNeT.2018.8585610.

- **Van TONG**, Hai Anh TRAN, Sami SOUIHI and Abdelhamid MELLOUK. "A novel QUIC traffic classifier based on convolutional neural networks." *2018 IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1-6,
doi: 10.1109/GLOCOM.2018.8647128.

- **Van TONG**, Hai Anh TRAN, Sami SOUIHI and Abdelhamid MELLOUK. "Empirical study for dynamic adaptive video streaming service based on Google transport QUIC protocol." *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*, 2018, pp. 343-350, doi: 10.1109/LCN.2018.8638062.

Bibliography

[1] Thales e Security. The percentage of encrypted flows. x, 17

[2] Yong Cui, Tianxiang Li, Cong Liu, Xingwei Wang, and Mirja Kühlewind. Innovating transport with quic: Design approaches and research challenges. *IEEE Internet Computing*, 21(2):72–76, 2017. x, 18, 21

[3] José Suárez-Varela and Pere Barlet-Ros. Towards a netflow implementation for openflow software-defined networks. In *2017 29th International Teletraffic Congress (ITC 29)*, volume 1, pages 187–195. IEEE, 2017. x, 16, 38, 39

[4] Paulo César Fonseca and Edjard Souza Mota. A survey on fault management in software-defined networks. *IEEE Communications Surveys & Tutorials*, 19(4):2284–2321, 2017. 1, 9, 16, 28, 38, 82, 137

[5] Yinbo Yu, Xing Li, Xue Leng, Libin Song, Kai Bu, Yan Chen, Jianfeng Yang, Liang Zhang, Kang Cheng, and Xin Xiao. Fault management in software-defined networking: A survey. *IEEE Communications Surveys & Tutorials*, 21(1):349–392, 2018. 1, 9, 16, 28, 38, 82, 137

[6] Sihem Cherrared, Sofiane Imadali, Eric Fabre, Gregor Gössler, and Imen Grida Ben Yahia. A survey of fault management in network virtualization environments: Challenges and solutions. *IEEE Transactions on Network and Service Management*, 16(4):1537–1551, 2019. 1, 2, 9, 16, 28, 37, 38, 137

[7] Maurice Gagnaire, Felipe Diaz, Camille Coti, Christophe Cerin, Kazuhiko Shiozaki, Yingjie Xu, Pierre Delort, Jean-Paul Smets, Jonathan Le Lous, Stephen Lubiarz, et al. Downtime statistics of current cloud solutions. *International Working Group on Cloud Computing Resiliency, Tech. Rep*, 2012. 1, 30, 136

[8] Blair Felter. 7 of the most famous recent ddos attacks. Sep 2021. 2, 136, 137

[9] Eva Abergel. Ddos attacks against financial institutes resurge in june 2021. Sep 2021. 2

[10] Ma łgorzata Steinder and Adarshpal S Sethi. A survey of fault localization techniques in computer networks. *Science of computer programming*, 53(2):165–194, 2004. 2

[11] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. A survey on network troubleshooting. *Technical Report Stanford/TR12-HPNG-061012, Stanford University, Tech. Rep.*, 2012. 3, 10, 30, 34, 43, 110, 137

[12] Maha Mdini. *Anomaly detection and root cause diagnosis in cellular networks*. Theses, Ecole nationale supérieure Mines-Télécom Atlantique, September 2019. 3, 137

[13] CISCO. Cisco white paper report: Encrypted traffic analytics. June 2021. 3, 17, 83, 137

[14] Mirja Kühlewind, Brian Trammell, Tobias Bühler, Gorry Fairhurst, and Vijay Gurbani. Challenges in network management of encrypted traffic. *arXiv preprint arXiv:1810.09272*, 2018. 3, 16, 26, 30

[15] K Moriarty and Al Morton. Effects of pervasive encryption on operators. *draft-mm-wg-effect-encrypt-25 (work in progress)*, 2018. 3, 26, 30

[16] Nancy M Morris and William B Rouse. Review and evaluation of empirical research in troubleshooting. *Human factors*, 27(5):503–530, 1985. 9

[17] David H Jonassen and Woei Hung. Learning to troubleshoot: A new theory-based design architecture. *Educational Psychology Review*, 18(1):77–114, 2006. 9

[18] Ray S Perez. A view from troubleshooting. *Toward a unified theory of problem solving*, pages 127–166, 2012. 9

[19] Alma Schaafstal, Jan Maarten Schraagen, and Marcel Van Berl. Cognitive task analysis and innovation of training: The case of structured troubleshooting. *Human factors*, 42(1):75–86, 2000. 9

[20] Scott D Johnson, Jeff W Flesher, Jihn-Chang J Jehng, and Ahmed Ferej. Enhancing electrical troubleshooting skills in a computer-coached practice environment. *Interactive learning environments*, 3(3):199–214, 1993. 9

[21] Brandon Heller, Colin Scott, Nick McKeown, Scott Shenker, Andreas Wundsam, Hongyi Zeng, Sam Whitlock, Vimalkumar Jeyakumar, Nikhil Handigol, James McCauley, et al. Leveraging sdn layering to systematically troubleshoot networks. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 37–42, 2013. 9

[22] Hongyi Zeng, Peyman Kazemian, George Varghese, and Nick McKeown. Automatic test packet generation. In *Proceedings of the 8th international conference on Emerging networking experiments and technologies*, pages 241–252. ACM, 2012. 9, 10

[23] TONG Van, Hai Anh Tran, Sami Souihi, and Abdelhamid MELLOUK. Network troubleshooting: survey, taxonomy and challenges. In *2018 International Conference on Smart Communications in Network Technologies (SaCoNeT)*, pages 165–170. IEEE, 2018. 9, 10

[24] Kai Bu, Xitao Wen, Bo Yang, Yan Chen, Li Erran Li, and Xiaolin Chen. Is every flow on the right track?: Inspect sdn forwarding with rulescope. In *Computer Communications, IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on*, pages 1–9. IEEE, 2016. 10

[25] Peter Perešíni, Maciej Kuźniar, and Dejan Kostić. Monocle: Dynamic, fine-grained data plane monitoring. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, page 32. ACM, 2015. 10

[26] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *NSDI*, volume 12, pages 113–126, 2012. 10

[27] Petr Slavık. A tight analysis of the greedy algorithm for set cover. *Journal of Algorithms*, 25(2):237–254, 1997. 10

[28] Armin Biere. Picosat essentials. *Journal on Satisfiability, Boolean Modeling and Computation*, 4:75–97, 2008. 10

[29] Y. Zhao, P. Zhang, Y. Wang, and Y. Jin. Sdn-enabled rule verification on data plane. *IEEE Communications Letters*, pages 1–1, 2018. 11

[30] Zhao, P. Zhang, Y. Wang, and Y. Jin. Troubleshooting data plane with rule verification in software-defined networks. *IEEE Transactions on Network and Service Management*, 15(1):232–244, March 2018. 11

[31] Carmelo Cascone, Davide Sanvito, Luca Pollini, Antonio Capone, and Brunilde Sanso. Fast failure detection and recovery in sdn with stateful data plane. *International Journal of Network Management*, 27(2):e1957, 2017. 11

[32] Leonardo Ochoa Aday, Cristina Cervelló Pastor, and Adriana Fernández Fernández. Current trends of topology discovery in openflow-based software defined networks. 2015. 12, 13

[33] George Tarnaras, Evangelos Haleplidis, and Spyros Denazis. Sdn and forces based optimal network topology discovery. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, pages 1–6. IEEE, 2015. 12

[34] Floodlight. Floodlight project. 12

[35] OpenDayLight. Opendaylight project. 12

[36] Farzaneh Pakzad, Marius Portmann, Wee Lum Tan, and Jadwiga Indulska. Efficient topology discovery in software defined networks. In *Signal Processing and Communication Systems (ICSPCS), 2014 8th International Conference on*, pages 1–8. IEEE, 2014. 13

[37] H. Xu, L. Yan, H. Xing, Y. Cui, and S. Li. Link failure detection in software defined networks: an active feedback mechanism. *Electronics Letters*, 53(11):722–724, 2017. 13

[38] Abdelhadi Azzouni, Raouf Boutaba, Nguyen Thi Mai Trang, and Guy Pujolle. softdp: Secure and efficient openflow topology discovery protocol. In *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*, pages 1–7. IEEE, 2018. 13

[39] Dave Katz and Dave Ward. Bidirectional forwarding detection (bfd). Technical report, 2010. 13

[40] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G Andersen, Gregory R Ganger, Garth A Gibson, and Srinivasan Seshan. Measurement and analysis of tcp throughput collapse in cluster-based storage systems. In *FAST*, volume 8, pages 1–14, 2008. 14

[41] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 202–208. ACM, 2009. 14

[42] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 267–280. ACM, 2010. 14

[43] Renuga Kanagevlu and Khin Mi Mi Aung. Sdn controlled local re-routing to reduce congestion in cloud data center. In *Cloud Computing Research and Innovation (ICCCRI), 2015 International Conference on*, pages 80–88. IEEE, 2015. 14

[44] Ramona Trestian, Gabriel-Miro Muntean, and Kostas Katrinis. Micetrap: Scalable traffic engineering of datacenter mice flows using openflow. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 904–907. IEEE, 2013. 14

[45] Seungbeom Song, Jaiyong Lee, Kyuho Son, Hangyong Jung, and Jihoon Lee. A congestion avoidance algorithm in sdn environment. In *Information Networking (ICOIN), 2016 International Conference on*, pages 420–423. IEEE, 2016. 14

[46] Shadi Attarha, Koosha Haji Hosseiny, Ghasem Mirjalily, and Kiarash Mizanian. A load balanced congestion aware routing mechanism for software defined networks. In *Electrical Engineering (ICEE), 2017 Iranian Conference on*, pages 2206–2210. IEEE, 2017. 14

[47] Masoumeh Gholami and Behzad Akbari. Congestion control in software defined data center networks through flow rerouting. In *Electrical Engineering (ICEE), 2015 23rd Iranian Conference on*, pages 654–657. IEEE, 2015. 14

[48] N Sminesh C, Grace Mary Kanaga E, et al. A proactive flow admission and re-routing scheme for load balancing and mitigation of congestion propagation in sdn data plane. *arXiv preprint arXiv:1812.02474*, 2018. 14

[49] Gaolei Li, Mianxiong Dong, Kaoru Ota, Jun Wu, Jianhua Li, and Tianpeng Ye. Deep packet inspection based application-aware traffic control for software defined net-

works. In *Global Communications Conference (GLOBECOM), 2016 IEEE*, pages 1–6. IEEE, 2016. 15

[50] Davide Adami, Gianni Antichi, Rosario G Garroppo, Stefano Giordano, and Andrew W Moore. Towards an sdn network control application for differentiated traffic routing. In *2015 IEEE International Conference on Communications (ICC)*, pages 5827–5832. IEEE, 2015. 15, 25, 49, 53, 84

[51] Li-Chia Cheng, Kuochen Wang, and Yi-Huai Hsu. Application-aware routing scheme for sdn-based cloud datacenters. In *Ubiquitous and Future Networks (ICUFN), 2015 Seventh International Conference on*, pages 820–825. IEEE, 2015. 15, 84

[52] Siva Sairam Prasad, Kotaro Kataoka, et al. Ampf: Application-aware multipath packet forwarding using machine learning and sdn. *arXiv preprint arXiv:1606.05743*, 2016. 15

[53] Ahmed M Abdelmoniem and Brahim Bensaou. Incast-aware switch-assisted tcp congestion control for data centers. In *Global Communications Conference (GLOBECOM), 2015 IEEE*, pages 1–6. IEEE, 2015. 15

[54] Jaehyun Hwang, Joon Yoo, Sang-Hun Lee, and Hyun-Wook Jin. Scalable congestion control protocol based on sdn in data center networks. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2015. 15

[55] Lei Zhang, Yong Cui, Mowei Wang, Zhenjie Yang, and Yong Jiang. Machine learning for internet congestion control: Techniques and challenges. *IEEE Internet Computing*, 23(5):59–64, 2019. 16

[56] Ticao Zhang and Shiwen Mao. Machine learning for end-to-end congestion control. *IEEE Communications Magazine*, 58(6):52–57, 2020. 16

[57] Zhiyuan Xu, Jian Tang, Chengxiang Yin, Yanzhi Wang, and Guoliang Xue. Experience-driven congestion control: When multi-path tcp meets deep reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 37(6):1325–1336, 2019. 16

[58] Nathan Jay, Noga Rotman, Brighten Godfrey, Michael Schapira, and Aviv Tamar. A deep reinforcement learning perspective on internet congestion control. In *International Conference on Machine Learning*, pages 3050–3059. PMLR, 2019. 16

[59] Raja Majid Ali Ujjan, Zeeshan Pervez, Keshav Dahal, Ali Kashif Bashir, Rao Mumtaz, and J González. Towards sflow and adaptive polling sampling for deep learning based ddos detection in sdn. *Future Generation Computer Systems*, 111:763–779, 2020. 16, 32, 34, 39, 57, 87, 88

[60] Petr Velan, Milan Čermák, Pavel Čeleda, and Martin Drašar. A survey of methods for encrypted traffic classification and analysis. *International Journal of Network Management*, 25(5):355–374, 2015. 17

[61] Adam Langley, Alistair Riddoch, Alyssa Wilk, Antonio Vicente, Charles Krasic, Dan Zhang, Fan Yang, Fedor Kouranov, Ian Swett, Janardhan Iyengar, et al. The quic transport protocol: Design and internet-scale deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 183–196. ACM, 2017. 17, 50, 83, 119, 120

[62] Mike Kosek, Tanya Shreedhar, and Vaibhav Bajpai. Beyond quic v1: A first look at recent transport layer ietf standardization efforts. *IEEE Communications Magazine*, 59(4):24–29, 2021. 18

[63] Gabriel Lopez-Millan, Rafael Marin-Lopez, and Fernando Pereniguez-Garcia. Towards a standard sdn-based ipsec management framework. *Computer Standards & Interfaces*, 66:103357, 2019. 22

[64] Eric Rescorla and Tim Dierks. The transport layer security (tls) protocol version 1.3. 2018. 22

[65] Lamine Amour, Sami Souihi, Abdelhamid Mellouk, and SM Mushtaq. Q2abr: Qoe-aware adaptive video bit rate solution. *International Journal of Communication Systems*, 33(10):e4204, 2020. 24

[66] Raimund Schatz, Tobias Hoßfeld, and Pedro Casas. Passive youtube qoe monitoring for isps. In *2012 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pages 358–364. IEEE, 2012. 24, 30

[67] Pedro Casas, Alessandro D'Alconzo, Pierdomenico Fiadino, Arian Bär, Alessandro Finamore, and Tanja Zseby. When youtube does not work—analysis of qoe-relevant degradation in google cdn traffic. *IEEE Transactions on Network and Service Management*, 11(4):441–457, 2014. 24

[68] Irena Orsolic, Dario Pevec, Mirko Suznjevic, and Lea Skorin-Kapov. A machine learning approach to classifying youtube qoe based on encrypted network traffic. *Multimedia tools and applications*, 76(21):22267–22301, 2017. 24, 31

[69] Irena Orsolic and Lea Skorin-Kapov. A framework for in-network qoe monitoring of encrypted video streaming. *IEEE Access*, 8:74691–74706, 2020. 24, 31

[70] HE Lin, Peng KUANG, WANG Shicheng, LIU Ying, LI Xing, and PENG Shuping. Application-aware ipv6 networking. *Telecommunications Science*, 36(8):36, 2019. 25

[71] Manuel Lopez-Martin, Belen Carro, Antonio Sanchez-Esguevillas, and Jaime Lloret. Network traffic classifier with convolutional and recurrent neural networks for internet of things. *IEEE Access*, 5:18042–18050, 2017. 25, 30, 49, 51, 53, 54, 58, 89

[72] Michael Finsterbusch, Chris Richter, Eduardo Rocha, Jean-Alexander Muller, and Klaus Hanssgen. A survey of payload-based traffic classification approaches. *IEEE Communications Surveys & Tutorials*, 16(2):1135–1156, 2013. 25

[73] Luca Deri, Maurizio Martinelli, Tomasz Bujlow, and Alfredo Cardigliano. ndpi: Open-source high-speed deep packet inspection. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 617–622. IEEE, 2014. 25, 30

[74] C Huitema and E Rescorla. Issues and requirements for sni encryption in tls. *Internet Engineering Task Force, Internet-Draft draft-ietftls-sni-encryption-03*, 2018. 25, 26

[75] Pedro Amaral, Joao Dinis, Paulo Pinto, Luis Bernardo, Joao Tavares, and Henrique S Mamede. Machine learning in software defined networks: Data collection and traffic classification. In *2016 IEEE 24th International conference on network protocols (ICNP)*, pages 1–5. IEEE, 2016. 25, 54

[76] Manos Antonakakis, Roberto Perdisci, Yacin Nadji, Nikolaos Vasiloglou, Saeed Abu-Nimeh, Wenke Lee, and David Dagon. From throw-away traffic to bots: Detecting the rise of dga-based malware. In *21st {USENIX} Security Symposium ({USENIX} Security 12)*, pages 491–506, 2012. 26

[77] Jonathan Woodbridge, Hyrum S Anderson, Anjum Ahuja, and Daniel Grant. Predicting domain generation algorithms with long short-term memory networks. *arXiv preprint arXiv:1611.00791*, 2016. 26

[78] Duc Tran, Hieu Mac, Van Tong, Hai Anh Tran, and Linh Giang Nguyen. A lstm based framework for handling multiclass imbalance in dga botnet detection. *Neurocomputing*, 275:2401–2413, 2018. 27

[79] Mohammad M Shurman, Rami M Khrais, and Abdulrahman A Yateem. Iot denial-of-service attack detection and prevention using hybrid ids. In *2019 International Arab Conference on Information Technology (ACIT)*, pages 252–254. IEEE, 2019. 27, 30

[80] Yazan Otoum and Amiya Nayak. As-ids: Anomaly and signature based ids for the internet of things. *Journal of Network and Systems Management*, 29(3):1–26, 2021. 27, 30

[81] Tamer AbuHmed, Abedelaziz Mohaisen, and DaeHun Nyang. A survey on deep packet inspection for intrusion detection systems. *arXiv preprint arXiv:0803.0037*, 2008. 27

[82] Norberto Garcia, Tomas Alcaniz, Aurora González-Vidal, Jorge Bernal Bernabe, Diego Rivera, and Antonio Skarmeta. Distributed real-time slowdos attacks detection over encrypted traffic using artificial intelligence. *Journal of Network and Computer Applications*, 173:102871, 2021. 27, 31

[83] Mikhail Zolotukhin and Timo Hämäläinen. Data stream clustering for application-layer ddos detection in encrypted traffic. In *Cyber Security: Power and Technology*, pages 111–131. Springer, 2018. 27, 31

[84] Pan Wang, Feng Ye, Xuejiao Chen, and Yi Qian. Datanet: Deep learning based encrypted network traffic classification in sdn home gateway. *IEEE Access*, 6:55380–55391, 2018. 31, 49, 50, 55, 83

[85] Wei Wang, Ming Zhu, Jinlin Wang, Xuewen Zeng, and Zhongzhen Yang. End-to-end encrypted traffic classification with one-dimensional convolution neural networks. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 43–48. IEEE, 2017. 31, 49, 50, 55, 83

[86] Fetia Bannour, Sami Souihi, and Abdelhamid Mellouk. Distributed sdn control: Survey, taxonomy, and challenges. *IEEE Communications Surveys & Tutorials*, 20(1):333–354, 2017. 32

[87] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. Onos: towards an open, distributed sdn os. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6, 2014. 32

[88] Fetia Bannour, Sami Souihi, and Abdelhamid Mellouk. Adaptive distributed sdn controllers: Application to content-centric delivery networks. *Future Generation Computer Systems*, 113:78–93, 2020. 32

[89] Gwangsun Kim, Changhyun Kim, Jiyun Jeong, Mike Parker, and John Kim. Contention-based congestion management in large-scale networks. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1–13. IEEE, 2016. 33, 82

[90] Piotr Jurkiewicz, Robert Wójcik, Jerzy Domżał, and Andrzej Kamisiński. Testing implementation of famtar: Adaptive multipath routing. *Computer Communications*, 149:300–311, 2020. 33, 82

[91] Cisco to optimize ntt docomo's 5g mobile backhaul for simpler, more flexible and scalable network operation. June 2021. 33, 82

[92] Cisco and vodafone showcase mobile transport networking advancements via segment routing at mobile world congress. June 2021. 33, 82

[93] Lingxia Liao and Victor CM Leung. Lldp based link latency monitoring in software defined networks. In *2016 12th International Conference on Network and Service Management (CNSM)*, pages 330–335. IEEE, 2016. 34, 41, 100

[94] Yang Li, Zhi-Ping Cai, and Hong Xu. Llmp: exploiting lldp for latency measurement in software-defined data center networks. *Journal of Computer Science and Technology*, 33(2):277–285, 2018. 34, 41, 44, 90, 110

[95] ONOS. Portstatistics api. June 2021. 34, 43, 44, 90, 110

[96] Javier Alvarez Cid-Fuentes, Claudia Szabo, and Katrina Falkner. Adaptive performance anomaly detection in distributed systems using online svms. *IEEE Transactions on Dependable and Secure Computing*, 17(5):928–941, 2018. 34, 43, 69, 106, 110

[97] Jibum Hong, Suhyun Park, Jae-Hyoung Yoo, and James Won-Ki Hong. Machine learning based sla-aware vnf anomaly detection for virtual network management. In *2020 16th International Conference on Network and Service Management (CNSM)*, pages 1–7. IEEE, 2020. 34, 43, 73, 90, 106, 109, 110, 112

[98] Segment routing application designy. July 2021. 35

[99] Lsquic library. April 2021. 36, 125

[100] Adanma Cecilia Eberendu et al. Unstructured data: an overview of the data of big data. *International Journal of Computer Trends and Technology*, 38(1):46–50, 2016. 36

[101] Lamine Amour, Souihi Sami, Said Hoceini, and Abdelhamid Mellouk. An open source platform for perceived video quality evaluation. In *Proceedings of the 11th ACM Symposium on QoS and Security for Wireless and Mobile Networks*, pages 139–140, 2015. 36

[102] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1285–1298, 2017. 37

[103] Aleksandra Revina, Krisztian Buza, and Vera G Meister. It ticket classification: The simpler, the better. *IEEE Access*, 8:193380–193395, 2020. 37

[104] Xiao Yu, Pallavi Joshi, Jianwu Xu, Guoliang Jin, Hui Zhang, and Guofei Jiang. Cloudseer: Workflow monitoring of cloud infrastructures via interleaved logs. *ACM SIGARCH Computer Architecture News*, 44(2):489–502, 2016. 37

[105] Debanshu Sinha, K Haribabu, and Sundar Balasubramaniam. Real-time monitoring of network latency in software defined networks. In *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, pages 1–3. IEEE, 2015. 38

[106] Alon Atary and Anat Bremler-Barr. Efficient round-trip time monitoring in openflow networks. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016. 38

[107] Ghazi Al-Naymat, Mouhammd Al-Kasassbeh, and Eshraq Al-Harwari. Using machine learning methods for detecting network anomalies within snmp-mib dataset. *International Journal of Wireless and Mobile Computing*, 15(1):67–76, 2018. 38

[108] K Amirthalingam and Robert J Moorhead. Snmp-an overview of its merits and demerits. In *Proceedings of the Twenty-Seventh Southeastern Symposium on System Theory*, pages 180–183. IEEE, 1995. 38

[109] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. Network innovation using openflow: A survey. *IEEE communications surveys & tutorials*, 16(1):493–512, 2013. 39

[110] Lamine Amour. Poqemon qoe dataset. June 2021. 42, 91

[111] Lamine Amour, Van Tong, Sami Souihi, Hai Anh Tran, and Abdelhamid Mellouk. Quality estimation framework for encrypted traffic (q2et). In *2019 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2019. 42, 44, 91, 110

[112] Sihyung Lee, Kyriaki Levanti, and Hyong S Kim. Network monitoring: Present and future. *Computer Networks*, 65:84–98, 2014. 43

[113] Woojoong Kim, Jian Li, James Won-Ki Hong, and Young-Joo Suh. Ofmon: Openflow monitoring system in onos controllers. In *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, pages 397–402. IEEE, 2016. 43

[114] Xuan Thien Phan and Kensuke Fukuda. Sdn-mon: Fine-grained traffic monitoring framework in software-defined networks. *Journal of Information Processing*, 25:182–190, 2017. 43

[115] Servicenow dataset. June 2021. 44

[116] Github dataset. June 2021. 44

[117] Nfv dataset. June 2021. 44

[118] Ana Bildea, Olivier Alphand, Franck Rousseau, and Andrzej Duda. Link quality estimation with the gilbert-elliot model for wireless sensor networks. In *2015 IEEE 26th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 2049–2054. IEEE, 2015. 44, 74, 75, 110, 112

[119] Dalwinder Singh and Birmohan Singh. Investigating the impact of data normalization on classification performance. *Applied Soft Computing*, 97:105524, 2020. 45, 74, 111, 125

[120] Van Tong. Service-centric segment routing using reinforcement learning. June 2021. 45, 95

[121] Selenium. Webdriver. *https://www.seleniumhq.org/projects/webdriver/*, June 2021. 45, 61

[122] lucas clemente. A quic implementation in pure go. *https://github.com/lucas-clemente/quic-go*, June 2021. 45, 61

[123] Van Tong. Network flow of quic. *https://drive.google.com/drive/folders/1cwHhzvaQbiap8yfrj2vHyPmUTQhaYOj?usp=sharing*, June 2021. 46, 51, 61

[124] Shahbaz Rezaei, Bryce Kroencke, and Xin Liu. Large-scale mobile app identification using deep learning. *IEEE Access*, 8:348–362, 2019. 49

[125] Zhong Fan and Ran Liu. Investigation of machine learning based network traffic classification. In *2017 International Symposium on Wireless Communication Systems (ISWCS)*, pages 1–6. IEEE, 2017. 49

[126] Sina Fathi-Kazerooni, Yagiz Kaymak, and Roberto Rojas-Cessa. Tracking user application activity by using machine learning techniques on network traffic. In *2019*

International Conference on Artificial Intelligence in Information and Communication (ICAIIC), pages 405–410. IEEE, 2019. 49, 54

[127] Ola Salman, Imad H Elhajj, Ali Chehab, and Ayman Kayssi. A multi-level internet traffic classifier using deep learning. In *2018 9th International Conference on the Network of the Future (NOF)*, pages 68–75. IEEE, 2018. 49

[128] Mohammad Lotfollahi, Mahdi Jafari Siavoshani, Ramin Shirali Hossein Zade, and Mohammdsadegh Saberian. Deep packet: A novel approach for encrypted traffic classification using deep learning. *Soft Computing*, 24(3):1999–2012, 2020. 49, 50, 55

[129] VNI Cisco. Cisco visual networking index: Forecast and trends, 2017–2022. *White Paper*, 2021. 50, 88, 95

[130] Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*, 2015. 51

[131] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 51, 89

[132] IANA. Service name and transport protocol port number registry. June 2021. 52

[133] Thomas Karagiannis, Andre Broido, Michalis Faloutsos, and KC Claffy. Transport layer identification of p2p traffic. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134, 2004. 53

[134] Nigel Williams, Sebastian Zander, and Grenville Armitage. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *ACM SIGCOMM Computer Communication Review*, 36(5):5–16, 2006. 54

[135] Hyun-Kyo Lim, Ju-Bong Kim, Kwihoon Kim, Yong-Geun Hong, and Youn-Hee Han. Payload-based traffic classification using multi-layer lstm in software defined networks. *Applied Sciences*, 9(12):2550, 2019. 55

[136] Kun Zhou, Wenyong Wang, Chenhuang Wu, and Teng Hu. Practical evaluation of encrypted traffic classification based on a combined method of entropy estimation and neural networks. *ETRI Journal*, 42(3):311–323, 2020. 58, 89

[137] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. 59

[138] Shuhua Gao, Cheng Xiang, Changkai Sun, Kairong Qin, and Tong Heng Lee. Efficient boolean modeling of gene regulatory networks via random forest based feature selection and best-fit extension. In *2018 IEEE 14th International Conference on Control and Automation (ICCA)*, pages 1076–1081. IEEE, 2018. 60, 90

[139] Kaiming He and Jian Sun. Convolutional neural networks at constrained time cost. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5353–5360, 2015. 60, 89

[140] F. CholletKeras. Keras. *https://github.com/fchollet/keras*, 2016. 61

[141] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011. 61, 75, 112

[142] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4):427–437, 2009. 61, 62, 75, 113

[143] Mlp. *https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html*, May 2021. 63

[144] Svm. *https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html*, May 2021. 63

[145] Rf. *shorturl.at/bquyW*, May 2021. 63

[146] F. CholletKeras. loss function. *https://keras.io/losses/*, 2016. 64

[147] Curtis R Taylor, Douglas C MacFarland, Doran R Smestad, and Craig A Shue. Contextual, flow-based access control with scalable host-based sdn techniques. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016. 66

[148] Aliyu Lawal Aliyu, Peter Bull, and Ali Abdallah. Performance implication and analysis of the openflow sdn protocol. In *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pages 391–396. IEEE, 2017. 66

[149] Mohiuddin Ahmed, Abdun Naser Mahmood, and Jiankun Hu. A survey of network anomaly detection techniques. *Journal of Network and Computer Applications*, 60:19–31, 2016. 68, 70

[150] Gilberto Fernandes, Joel JPC Rodrigues, Luiz Fernando Carvalho, Jalal F Al-Muhtadi, and Mario Lemes Proença. A comprehensive survey on network anomaly detection. *Telecommunication Systems*, 70(3):447–489, 2019. 68

[151] Maha Mdini. *Anomaly detection and root cause diagnosis in cellular networks*. PhD thesis, Ecole nationale supérieure Mines-Télécom Atlantique Bretagne Pays de la Loire, 2019. 69, 106

[152] Pedro Casas, Pierdomenico Fiadino, Sarah Wassermann, Stefano Traverso, Alessandro D'Alconzo, Edion Tego, Francesco Matera, and Marco Mellia. Unveiling network and service performance degradation in the wild with mplane. *IEEE Communications Magazine*, 54(3):71–79, 2016. 69

[153] Jing Chen, Ming Chen, Xianglin Wei, and Bing Chen. Matrix differential decomposition-based anomaly detection and localization in nfv networks. *IEEE Access*, 7:29320–29331, 2019. 69, 106

[154] Kristof Böhmer and Stefanie Rinderle-Ma. Mining association rules for anomaly detection in dynamic process runtime behavior and explaining the root cause to users. *Information Systems*, 90:101438, 2020. 70

[155] Bhupendra Ingre, Anamika Yadav, and Atul Kumar Soni. Decision tree based intrusion detection system for nsl-kdd dataset. In *International conference on information and communication technology for intelligent systems*, pages 207–218. Springer, 2017. 71

[156] Ansam Khraisat, Iqbal Gondal, and Peter Vamplew. An anomaly intrusion detection system using c5 decision tree classifier. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 149–155. Springer, 2018. 71

[157] Rifkie Primartha and Bayu Adhi Tama. Anomaly detection using random forest: A performance revisited. In *2017 International conference on data and software engineering (ICoDSE)*, pages 1–6. IEEE, 2017. 71

[158] MA Jabbar, Rajanikanth Aluvalu, and S Sai Satyanarayana Reddy. Intrusion detection system using bayesian network and feature subset selection. In *2017 IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, pages 1–5. IEEE, 2017. 71

[159] MJ Vargas-Muñoz, Rafael Martínez-Peláez, Pablo Velarde-Alvarado, E Moreno-García, DL Torres-Roman, and JJ Ceballos-Mejía. Classification of network anomalies in flow level network traffic using bayesian networks. In *2018 International Conference on Electronics, Communications and Computers (CONIELECOMP)*, pages 238–243. IEEE, 2018. 71

[160] Huiwen Wang, Jie Gu, and Shanshan Wang. An effective intrusion detection framework based on svm with feature augmentation. *Knowledge-Based Systems*, 136:130–139, 2017. 71

[161] Mukaram Safaldin, Mohammed Otair, and Laith Abualigah. Improved binary gray wolf optimizer and svm for intrusion detection system in wireless sensor networks. *Journal of ambient intelligence and humanized computing*, 12(2):1559–1576, 2021. 71

[162] Pynbianglut Hadem, Dilip Kumar Saikia, and Soumen Moulik. An sdn-based intrusion detection system using svm with selective logging for ip traceback. *Computer Networks*, 191:108015, 2021. 71

[163] H Taud and JF Mas. Multilayer perceptron (mlp). In *Geomatic Approaches for Modeling Land Change Scenarios*, pages 451–455. Springer, 2018. 73

[164] Iftikhar Ahmad, Mohammad Basheri, Muhammad Javed Iqbal, and Aneel Rahim. Performance comparison of support vector machine, random forest, and extreme learning machine for intrusion detection. *IEEE access*, 6:33789–33795, 2018. 73, 112

[165] Abdul Rehman Javed, Zunera Jalil, Syed Atif Moqurrab, Sidra Abbas, and Xuan Liu. Ensemble adaboost classifier for accurate and fast detection of botnet attacks in

connected vehicles. *Transactions on Emerging Telecommunications Technologies*, page e4088, 2020. 73, 112

[166] Donghwoon Kwon, Kathiravan Natarajan, Sang C Suh, Hyunjoo Kim, and Jinoh Kim. An empirical study on network anomaly detection using convolutional neural networks. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1595–1598. IEEE, 2018. 73, 112

[167] Dan Frost and S Bryant. Packet loss and delay measurement for mpls networks. Technical report, RFC 6374, september, 2011. 74

[168] Dhruvkumar Dholakiya, Tanmay Kshirsagar, and Amit Nayak. Survey of mininet challenges, opportunities, and application in software-defined network (sdn). In *International Conference on Information and Communication Technology for Intelligent Systems*, pages 213–221. Springer, 2020. 75, 112, 125

[169] Mark JF Gales, Anton Ragni, H AlDamarki, and C Gautier. Support vector machines for noise robust asr. In *2009 IEEE Workshop on Automatic Speech Recognition & Understanding*, pages 205–210. IEEE, 2009. 76

[170] Pier Luigi Ventre, Stefano Salsano, Marco Polverini, Antonio Cianfrani, Ahmed Abdelsalam, Clarence Filsfils, Pablo Camarillo, and Francois Clad. Segment routing: A comprehensive survey of research activities, standardization efforts and implementation results. *IEEE Communications Surveys & Tutorials*, 2020. 82

[171] Zahraa N Abdullah, Imtiaz Ahmad, and Iftekhar Hussain. Segment routing in software defined networks: A survey. *IEEE Communications Surveys & Tutorials*, 21(1):464–486, 2018. 82, 90

[172] Aniruddha Kushwaha, Sidharth Sharma, Naveen Bazard, Ashwin Gumaste, and Biswanath Mukherjee. Design, analysis, and a terabit implementation of a source-routing-based sdn data plane. *IEEE Systems Journal*, 2020. 82

[173] Navin Kukreja, Rodolfo Alvizu, Ana Kos, Guido Maier, Roberto Morro, Alessandro Capello, and Carlo Cavazzoni. Demonstration of sdn-based orchestration for multi-domain segment routing networks. In *2016 18th International Conference on Transparent Optical Networks (ICTON)*, pages 1–4. IEEE, 2016. 82, 86

[174] Shuping Peng, Jianwei Mao, Ruizhao Hu, and Zhenbin Li. Demo abstract: Apn6: Application-aware ipv6 networking. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1330–1331. IEEE, 2020. 82, 86

[175] Ana Custura, Raffaello Secchi, and Gorry Fairhurst. Exploring dscp modification pathologies in the internet. *Computer Communications*, 127:86–94, 2018. 83, 86

[176] Cong Dong, Chen Zhang, Zhigang Lu, Baoxu Liu, and Bo Jiang. Cetanalytics: Comprehensive effective traffic information analytics for encrypted traffic classification. *Computer Networks*, 176:107258, 2020. 83, 86

[177] Quic statistic. January 2021. 83

[178] Ming-Chieh Lee and Jang-Ping Sheu. An efficient routing algorithm based on segment routing in software-defined networking. *Computer Networks*, 103:44–55, 2016. 83

[179] Zhenbin Li, Shuping Peng, Daniel Voyer, Chongfeng Xie, Peng Liu, Zhuangzhuang Qin, Kentaro Ebisawa, Stefano Previdi, and Jim Guichard. Problem Statement and Use Cases of Application-aware Networking (APN). Internet-Draft draft-li-apn-problem-statement-usecases-01, Internet Engineering Task Force, September 2020. Work in Progress. 84

[180] U Pongsakorn, Yasuhiro Watashiba, Kohei Ichikawa, Susumu Date, Hajimu Iida, et al. Application-aware network: Network route management using sdn based on application characteristics. *CSI Transactions on ICT*, 5(4):375–385, 2017. 84, 85

[181] Seyeon Jeong, Doyoung Lee, Jonghwan Hyun, Jian Li, and James Won-Ki Hong. Application-aware traffic engineering in software-defined network. In *2017 19th Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 315–318. IEEE, 2017. 85

[182] Albert Rego, Sandra Sendra, Jose M Jimenez, and Jaime Lloret. Dynamic metric ospf-based routing protocol for software defined networks. *Cluster Computing*, 22(3):705–720, 2019. 85

[183] Alcardo Alex Barakabitze, Is-Haka Mkwawa, Lingfen Sun, and Emmanuel Ifeachor. Qualitysdn: Improving video quality using mptcp and segment routing in sdn/nfv. In *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 182–186. IEEE, 2018. 85, 95

[184] Ayoub Bahnasse, Fatima Ezzahraa Louhab, Hafsa Ait Oulahyane, Mohamed Talea, and Assia Bakali. Novel sdn architecture for smart mpls traffic engineering-diffserv aware management. *Future Generation Computer Systems*, 87:115–126, 2018. 86, 90

[185] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 432–445, 2017. 86

[186] Van Tong, Hai Anh Tran, Sami Souihi, and Abdelhamid Mellouk. A novel quic traffic classifier based on convolutional neural networks. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018. 89

[187] Seunghyeon Lee, Jinwoo Kim, Seungwon Shin, Phillip Porras, and Vinod Yegneswaran. Athena: A framework for scalable anomaly detection in software-defined networks. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 249–260. IEEE, 2017. 90

[188] Lorenzo Fernández Maimó, Ángel Luis Perales Gómez, Félix J García Clemente, Manuel Gil Pérez, and Gregorio Martínez Pérez. A self-adaptive deep learning-based system for anomaly detection in 5g networks. *IEEE Access*, 6:7700–7712, 2018. 90

[189] Jin Wang, Yangning Tang, Shiming He, Changqing Zhao, Pradip Kumar Sharma, Osama Alfarraj, and Amr Tolba. Logevent2vec: Logevent-to-vector based anomaly detection for large-scale logs in internet of things. *Sensors*, 20(9):2451, 2020. 90

[190] Zoubir Mammeri. Reinforcement learning based routing in networks: Review and classification of approaches. *IEEE Access*, 7:55916–55950, 2019. 92, 125

[191] Sven Koenig and Reid G Simmons. Complexity analysis of real-time reinforcement learning. In *AAAI*, pages 99–107, 1993. 93

[192] Chi Jin, Zhuoran Yang, Zhaoran Wang, and Michael I Jordan. Provably efficient reinforcement learning with linear function approximation. In *Conference on Learning Theory*, pages 2137–2143. PMLR, 2020. 93, 125

[193] VN Temlyakov. Greedy approximation in convex optimization. *Constructive Approximation*, 41(2):269–296, 2015. 93

[194] Yu-Lin He, Xiao-Liang Zhang, Wei Ao, and Joshua Zhexue Huang. Determining the optimal temperature parameter for softmax function in reinforcement learning. *Applied Soft Computing*, 70:80–85, 2018. 93

[195] Hai Anh Tran, Said Hoceini, Abdelhamid Mellouk, Julien Perez, and Sherali Zeadally. Qoe-based server selection for content distribution networks. *IEEE Transactions on Computers*, 63(11):2803–2815, 2013. 93

[196] Mininet. February 2021. 94

[197] Open network operating system (onos). June 2021. 94

[198] Shih-Chun Lin, Ian F Akyildiz, Pu Wang, and Min Luo. Qos-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach. In *2016 IEEE International Conference on Services Computing (SCC)*, pages 25–33. IEEE, 2016. 95

[199] Elisavet Grigoriou, Alcardo Alex Barakabitze, Luigi Atzori, Lingfen Sun, and Virginia Pilloni. An sdn-approach for qoe management of multimedia services using resource allocation. In *2017 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2017. 96

[200] Sergio Fortes, Raquel Barco, Alejandro Aguilar-García, and Pablo Muñoz. Contextualized indicators for online failure diagnosis in cellular networks. *Computer Networks*, 82:96–113, 2015. 106

[201] Karthik Nagaraj, Charles Killian, and Jennifer Neville. Structured comparative analysis of systems logs to diagnose performance problems. In *9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12)*, pages 353–366, 2012. 106

[202] Giorgos Dimopoulos, Ilias Leontiadis, Pere Barlet-Ros, Konstantina Papagiannaki, and Peter Steenkiste. Identifying the root cause of video streaming issues on mobile devices. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, pages 1–13, 2015. 106

[203] Jordan Hochenbaum, Owen S Vallis, and Arun Kejariwal. Automatic anomaly detection in the cloud via statistical learning. *arXiv preprint arXiv:1704.07706*, 2017. 106, 107

[204] Junichi Kawasaki, Genichi Mouri, and Yusuke Suzuki. Comparative analysis of network fault classification using machine learning. In *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–6. IEEE, 2020. 107, 109

[205] Honggang Zhou, Yunchun Li, Hailong Yang, Jie Jia, and Wei Li. Bigroots: An effective approach for root-cause analysis of stragglers in big data system. *IEEE Access*, 6:41966–41977, 2018. 107

[206] Leila Bennacer, Laurent Ciavaglia, Abdelghani Chibani, Yacine Amirat, and Abdelhamid Mellouk. Optimization of fault diagnosis based on the combination of bayesian networks and case-based reasoning. In *2012 IEEE Network Operations and Management Symposium*, pages 619–622. IEEE, 2012. 108

[207] Fernando Benayas, Alvaro Carrera, and Carlos A Iglesias. Towards an autonomic bayesian fault diagnosis service for sdn environments based on a big data infrastructure. In *2018 Fifth International Conference on Software Defined Systems (SDS)*, pages 7–13. IEEE, 2018. 108

[208] Juan Qiu, Qingfeng Du, Yu He, YiQun Lin, Jiaye Zhu, and Kanglin Yin. Performance anomaly detection models of virtual machines for network function virtualization infrastructure with machine learning. In *International Conference on Artificial Neural Networks*, pages 479–488. Springer, 2018. 109

[209] A Niranjan, Anusha Prakash, N Veena, M Geetha, P Deepa Shenoy, and KR Venugopal. Ebjrv: An ensemble of bagging, j48 and random committee by voting for efficient classification of intrusions. In *2017 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*, pages 51–54. IEEE, 2017. 112

[210] Jordan Frery, Amaury Habrard, Marc Sebban, Olivier Caelen, and Liyun He-Guelton. Efficient top rank optimization with gradient boosting for supervised anomaly detection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 20–35. Springer, 2017. 112

[211] Van Tong. May 2021. 113

[212] Qasem Al-Tashi, Helmi Md Rais, Said Jadid Abdulkadir, Seyedali Mirjalili, and Hitham Alhussian. A review of grey wolf optimizer-based feature selection methods for classification. *Evolutionary Machine Learning Techniques*, pages 273–286, 2020. 115

[213] Global data volume of consumer ip traffic 2017-2022. March 2021. 119

[214] Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. Pcc vivace: Online-learning congestion control. In *15th USENIX Symposium on Networked Systems Design and Implementation NSDI 18*, pages 343–356, 2018. 119

[215] Xuewei Wu, Li Wang, Bo Bai, Jianyong Qiao, and Aiguo Fei. Hopf bifurcation in dctcp congestion control. *IEEE Communications Letters*, 24(7):1424–1427, 2020. 119

[216] Mario Hock, Roland Bless, and Martina Zitterbart. Experimental evaluation of bbr congestion control. In *2017 IEEE 25th International Conference on Network Protocols (ICNP)*, pages 1–10. IEEE, 2017. 120, 121, 125

[217] Xiaohui Nie, Youjian Zhao, Zhihan Li, Guo Chen, Kaixin Sui, Jiyang Zhang, Zijie Ye, and Dan Pei. Dynamic tcp initial windows and congestion control schemes through reinforcement learning. *IEEE Journal on Selected Areas in Communications*, 37(6):1231–1247, 2019. 120, 123, 124

[218] Yue Wang, Kanglian Zhao, Wenfeng Li, Juan Fraire, Zhili Sun, and Yuan Fang. Performance evaluation of quic with bbr in satellite internet. In *2018 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, pages 195–199. IEEE, 2018. 120

[219] Robin Marx, Joris Herbots, Wim Lamotte, and Peter Quax. Same standards, different decisions: A study of quic and http/3 implementation diversity. In *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, pages 14–20, 2020. 121

[220] Sanjeev Patel, Yash Shukla, Nikhil Kumar, Tejasv Sharma, and Kulgaurav Singh. A comparative performance analysis of tcp congestion control algorithms: Newreno, westwood, veno, bic, and cubic. In *2020 6th International Conference on Signal Processing and Communication (ICSC)*, pages 23–28. IEEE, 2020. 121

[221] Imad Abdeljaouad, Hicham Rachidi, Sherwin Fernandes, and Ahmed Karmouch. Performance analysis of modern tcp variants: A comparison of cubic, compound and new reno. In *2010 25th Biennial Symposium on Communications*, pages 80–83. IEEE, 2010. 121

[222] Jing Wang, Yufan Zheng, Yunzhe Ni, Chenren Xu, Feng Qian, Wangyang Li, Wantong Jiang, Yihua Cheng, Zhuo Cheng, Yuanjie Li, et al. An active-passive measurement study of tcp performance over lte on high-speed rails. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019. 123

[223] Songyang Zhang. An evaluation of bbr and its variants. *arXiv preprint arXiv:1909.03673*, 2019. 123

[224] Keith Winstein and Hari Balakrishnan. Tcp ex machina: Computer-generated congestion control. *ACM SIGCOMM Computer Communication Review*, 43(4):123–134, 2013. 124

[225] Hai-Anh Tran, Sami Souihi, Duc Tran, and Abdelhamid Mellouk. Mabrese: A new server selection method for smart sdn-based cdn architecture. *IEEE Communications Letters*, 23(6):1012–1015, 2019. 125

[226] Christophe Cérin, Camille Coti, Pierre Delort, Felipe Diaz, Maurice Gagnaire, Quentin Gaumer, Nicolas Guillaume, J Lous, Stephane Lubiarz, J Raffaelli, et al. Downtime statistics of current cloud solutions. *International Working Group on Cloud Computing Resiliency, Tech. Rep*, 1:2, 2013. 130, 140

[227] Facebook outage. October 2021. 137

Annex (Paper published in IEEE System Journal)

To be published in IEEE System Journal

SDN-based Application-aware Segment Routing for Large-scale Network

Van TONG*, Sami SOUIHI*, Hai Anh TRAN† and Abdelhamid MELLOUK*
*Univ Paris Est Creteil, TINCNET/LISSI, F-94400 Vitry-sur-Seine, France
Email: (van-van.tong, sami.souihi and mellouk)@u-pec.fr
† SOICT, HUST, Vietnam
Email: anhth@soict.hust.edu.vn

***Abstract*—With the rapid growth of 5G, network operators have met with difficulties in ensuring user satisfaction due to diverse SLA (Service-level Agreement) requirements. Application-aware routing is able to potentially address this issue in implementing differentiated routing policies corresponding to various applications. To facilitate the application-aware routing, SDN (Software-defined Networking) and SR (Segment Routing) are potential solutions with programmability property to steer network traffic into appropriate routing paths. However, application-aware SR encounters two main problems including application identification for encrypted traffic and path identification with human intervention. Therefore, this paper proposes a new SDN-based SR mechanism that could help network operators overcome these problems. This approach implements Reinforcement Learning (RL) to adapt to dynamic networks in order to optimize the QoE (Quality of Experience) that network operators must guarantee. The proposal also considers the application class to implement corresponding routing policies for various applications to meet the stringent SLA requirements. Identifying the applications is sometimes complicated due to encrypted network traffic. Hence, our proposal implements a traffic classification approach to classify encrypted traffic into different kinds of applications. Obtained results under considered conditions illustrate that our proposal outperforms the standard SR mechanism related to QoE and decreases up to 64.39 percent of overhead compared to several benchmarks.**

***Index Terms*—Segment Routing, Application-aware routing, Traffic classification, Quality of Experience, SDN**

I. Introduction

Network operators have faced a challenge related to improving network services [1]. Concretely, the rapid development of 5G leads to many strict user's network requirements (e.g., low latency, high reliability, etc.). Therefore, differentiated application treatment is expected by end-users. However, network operators lack application awareness in their networks which results in dissatisfaction of the end-users. Application-aware routing which implements different kinds of routing policies corresponding to differentiated applications, can overcome this drawback. To facilitate the application-aware routing, it needs two requirements related to a routing technique supporting Traffic Engineering and a network architecture that can obtain a global view of network.

For the first requirement, Multi-Protocol Label Switching (MPLS) is a potential solution which is deployed by network operators to improve their IP networks. However, MPLS contains three main drawbacks [2]. First, it requests an IP network to maintain an explicit state at network nodes along an MPLS path, bringing a scalability problem in both control and data plane. Second, MPLS can not benefit from the load balancing given by Equal-cost Multi-path routing (ECMP). Finally, MPLS with the support of IGPs (Interior Gateway Protocols) for routing protocol can not be easily implemented on multiple Autonomous Systems (ASs). Therefore, many network operators (e.g., NTT, Vodafone, etc.) implement Segment Routing (SR) in their network infrastructure as a solution for these issues [3], [4]. The core idea of SR architecture is based on the notion of source routing [5] and tunneling to guarantee the scalability property in decreasing the amount of state information to be processed in the core network. Besides, SR's main benefit is to fix the scalability issues and limitations of the MPLS approach. Concretely, SR does not require any state maintenance in core network nodes. Moreover, it takes advantage of the ECMP routing and the implementation on multiple ASs [6].

For the second requirement of the application-aware routing, Software-defined Networking (SDN) is a promising solution. It decouples the control layer from the infrastructure layer that offers an opportunity to obtain a global network view. Therefore, application-aware SDN-based SR is concerned by the research community [6], [7]. However, this SR mechanism contains two main disadvantages:

- First, this routing mechanism considers the application class to meet different SLA (Service-level Agreement) requirements, but application identification is sometimes complicated due to encrypted traffic. In the past, the application class can be obtained by DSCP (Differentiated Services Code Point) in packet's header or deep packet inspection. However, DSCP field can be changed during packet's transmission [8] while the deep packet inspection is not effective with encrypted traffic [9]. Indeed, many service providers encrypt their data during the transmission to protect the user's privacy. According to a recent Cisco report [10], 80 percent of web traffic was encrypted by 2019 compared to 40 percent by 2016. Consequently, there is a necessity for a novel traffic classification approach to identify the application class in this case. Much existing research work focus on the classification approaches for VPN [11] and TLS/SSL [12] traffic. However, a novel solution is studied here to obtain the application class for the traffic of QUIC (Quick UDP Internet Connection) [13] which is a new transport layer network protocol developed by Google in 2012. The amount of QUIC traffic comprises 35 percent of Google's egress traffic, which corresponds to approximately 7 percent of all Internet traffic and continues to increase in the future [14].
- Second, routing paths are identified according to the

human intervention, so it is not adaptive with the unprecedented change of network environments. For the past few decades, QoS (Quality of Service) played an essential role in the network systems, so much research work [15] concentrated on QoS-aware SR mechanisms. The main objective is to optimize the network resource utilization as well as meet network requirements related to QoS. Nevertheless, selecting appropriate QoS parameters is sometimes complicated. Therefore, QoE (Quality of Experience), which network operators need to assure, is considered in SR in this work. QoE is a metric correlated to QoS metrics, but it is a perception of end-user that facilitates network policies to guarantee the user's SLA requirements.

In this paper, a novel SDN-based SR mechanism for encrypted traffic is proposed for network operators. There are traditional algorithms in the QoE-aware SR scheme (e.g., selecting a path with maximal QoE value, etc.). However, it is not effective with dynamic networks. With the development of 5G, the networks have become more and more complex. Using the RL algorithm in the QoE-aware SR mechanism offers opportunities to select appropriate paths to adapt to the dynamic changes of network environments and improve long-term performance. Moreover, this SR mechanism also plays an important role in a network troubleshooting framework. The rapid development of the Internet leads to many complex problems frequently in network systems. Network operators need to react quickly to reduce its negative impacts. However, this requires so much time (e.g., a few hours, etc.) to identify and solve its root causes definitively [16]. The proposed segment routing mechanism can reduce the negative influence of problems (e.g., optimize QoE, etc.) until the root cause of problems is solved.

This research is an extended version of our previous work [17]. The additional contributions are presented as follows:

- A more comprehensive analysis of the SDN-based SR framework for network operators is comprised. The objective is to implement the corresponding routing policies for various SLA requirements because each application has a specific SLA requirement. In a particular routing policy, this mechanism selects the appropriate paths using Reinforcement Learning (RL) according to the network environment's feedback (QoE) to adapt to dynamic network conditions. Unlike our previous work which investigated overloaded network scenario, the impact of perfect network scenario and fault tolerance on the proposed SR mechanism are studied here.
- The RL algorithm is enhanced by our adaptive selection algorithm which implements different selection policies corresponding to various scenarios. This contrasts with our previous work which implemented only the *softmax* selection algorithm.
- The proposed SR mechanism is evaluated thoroughly with the benchmarks according to MOS score (Mean Opinion Score), controller's CPU usage and control overhead while our previous work evaluates cumulative MOS score and CPU usage of the PC implementing a testbed.

Outline: The remainder of the paper is structured as follows. Section II introduces the related work in SR. In section III, the paper presents the proposed SR mechanism. Section IV describes the experimental results of the proposed mechanism. Finally, the paper concludes with section V which highlights our future work.

II. Related work

This section presents related work on segment routing, application-aware routing and encrypted traffic classification methods.

Barakabitze et al. [18] proposed QoEMultiSDN, a QoE-based multipath source routing algorithm to optimize the network resource by mapping subflow paths into the SR paths in SDN and NFV environments. Concretely, this approach selects routing paths using a constrained shortest path model and QoS parameters (e.g., delay, packet loss, etc.). Although QoEMultiSDN adaptively selects bitrate of video streaming using QoE, the routing path is still identified according to QoS parameters.

Li et al. [1] emphasized that network operators have encountered a challenge of providing better services with the rapid growth of 5G and multimedia services which requires diverse network requirements (e.g., low latency, high reliability, etc.). However, network operators are unaware of which applications in their networks, so they are unable to obtain a global view to manage the networks effectively. Therefore, much research work focus on application-aware routing (e.g., application-aware MPLS, etc.).

Rego et al. [19] proposed an improvement of Open Shortest Path First (OSPF) routing protocol in SDN environments. This approach changes the metric calculation (e.g., bandwidth, delay, etc.) adaptively corresponding to different applications to select appropriate routing paths. However, identifying an effective metric calculation is sometimes complicated. Similarly, Bahnasse et al. [20] proposed an application-aware MPLS in SDN to optimize network resource. This approach identifies VoIP, video, HTTP, and ICMP traffic based on DSCP field in the packet's header. Then, a specific routing policy is applied for each application to meet bandwidth constraints. However, identifying the application class using DSCP sometimes is inaccurate because it can be changed during data transmission [8]. Google [21] proposed Espresso, an SDN-based Internet peering edge routing infrastructure which offers an opportunity for the application-aware MPLS mechanism at Internet-peering scale. According to integrating application-aware MPLS mechanism, Espresso delivers 13 percent more network traffic on their infrastructures and improves link utilization and user perception compared with BGP (Border Gateway Protocol)-based routing. Nevertheless, MPLS suffers from several hindrances related to scalability problem and ECMP routing. Consequently, many other proposals on SR are proposed to address these hindrances.

In fact, Kukreja et al. [6] presented a demonstration of SDN-based SR for multi-domain networks. In this demonstration, an orchestrator finds the suitable routing paths and encodes to packet's header corresponding to diverse applications. The

objective is to meet different resource requirements of these applications (e.g., bandwidth, delay constraints, etc.). Nevertheless, it is assumed that class application is known by the orchestrator. Peng et al. [7] proposed an application-aware network framework that takes advantage of SR to meet their SLA requirements. This framework identifies the application characteristics according to deep packet inspection mechanism and then forwards packets into corresponding paths (policy or traffic engineering tunnel). Nevertheless, these SR mechanisms identify the routing paths thanks to human intervention which is not effective with the unprecedented change of network environments. In contrast, the proposed SR mechanism identifies the routing paths using RL to adapt to dynamic networks. Moreover, classifying the traffic using deep packet inspection is not effective due to encrypted traffic nowadays [9]. Therefore, the application-aware routing mechanism needs a traffic classification approach to classify the encrypted network traffic and identify the application class.

Wang et al. [11] presented an encrypted traffic classification method with a one-dimensional convolution neural network (CNN). This method uses the first 784 bytes in the payload of each packet to create a one-dimensional vector and analyzes them using CNN to classify VPN traffic into email, chat, streaming, file transfer and VoIP. Similarly, Pan et al. [12] proposed an encrypted traffic classification approach using deep learning algorithm. This approach monitors HTTPS traffic and collects the first 1480 bytes in the packet's payload to create a one-dimensional vector. Next, this vector is analyzed according to deep learning algorithms (e.g., CNN, etc.) to classify the network traffic into 15 kinds of applications. In this paper, an encrypted traffic classification method is presented to identify the QUIC traffic into different types of applications.

III. Proposed Adaptive Segment Routing Mechanism for Encrypted Traffic

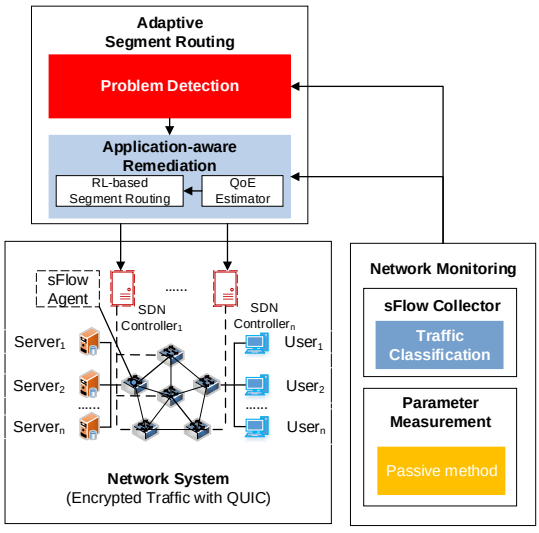


Fig. 1: The SDN-based adaptive SR framework.

A. Overview of SDN-based Adaptive Segment Routing Framework

The proposed multi-modular system is depicted in Fig. 1. During the transmission from servers to clients, a network flow is transmitted from ingress switches to a sFlow collector [22] to obtain the class of application thanks to traffic classification module. In the beginning, the network flow is forwarded using the shortest paths from the standard SR algorithm. After that, network parameters from parameter measurement module and the class of application are analyzed in problem detection module to detect abnormal symptoms (e.g., increase of latency, packet loss, etc.). When the problems occur in the network, the network flow is forwarded using an adaptive SR mechanism that selects the appropriate routing paths that meet strict user requirements related to the QoE corresponding to each network application. When the problems are solved, the network flow will be forwarded using the standard SR algorithm. The major components in this framework are described as follows:

1) *Network monitoring* includes two essential modules containing traffic classification and parameter measurement modules. The former aims to identify the class of application on network flows (video streaming, file transfer and VoIP) in the context of encrypted traffic while the latter aims to monitor and collect network parameters for further modules.
2) *Problem Detection* takes into account time-series network parameters and the class of application to detect an abnormal symptom of network problems.
3) *Application-aware Remediation* is used to consider various routing strategies corresponding to different kinds of applications to optimize the QoE in the network.

The proposed SR mechanism considers the class of application to implement appropriate routing policies corresponding to each kind of application. When network traffic is encrypted, this information is hidden. Therefore, there is a necessity of a traffic classification module for identifying this information. The traffic classification is presented in the following section.

B. Network Monitoring

1) Traffic Classification

A novel traffic classification approach is presented in this section to identify different kinds of applications for encrypted traffic. In [23], it is reported that video streaming, file sharing, and VoIP will comprise over 80 percent of global IP traffic by 2022. Consequently, these applications are considered in the traffic classification module. Regarding the network traffic for traffic classification, it can be collected directly from Openflow switches, but it is not effective with a huge amount of network traffic. Therefore, sFlow [22], a standard for network monitoring supporting packet sampling technique, is deployed to reduce the collected traffic volume and offer opportunities to implement the traffic classification module in real-time. When the network traffic traverses the ingress switches (edge switches), the sFlow agents send the network traffic to the sFlow collector after sampling. At the sFlow collector, the traffic classification module collects the network traffic to identify the class of application. This module is described as follows (Fig. 2).

There are two kinds of flows in the network traffic including mice flows (small continuous flows in total bytes) and elephant flows (huge continuous flows in total bytes). After investigating their characteristics, flow-based features (handcrafted

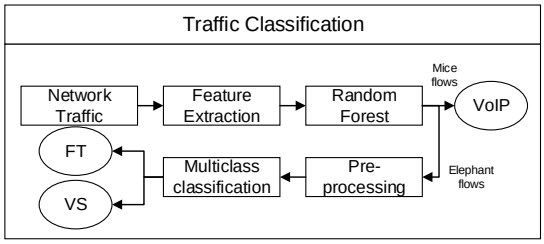


Fig. 2: The novel traffic classification approach for encrypted traffic.

features) of the first few packets in each flow [24] are collected and analyzed using the random forest algorithm [25] to classify the network traffic into the mice flows (VoIP) or the elephant-flows (video streaming and file transfer). This approach considers the random forest algorithm in mice-flow identification over several traditional machine learning algorithms (e.g., SVM, MLP, etc.) based on the research in [25]. After that, the elephant flows are classified into video streaming (VS) or file transfer (FT) using packet-based features (implicit features) and the convolutional neural network (CNN). The latter learning algorithm is taken into account in this approach because it contains characteristics including spare connectivity, parameter sharing, and equivariant representations. This helps to learn more effective representations in comparison with traditional machine learning algorithms. The detail of the traffic classification module is described in our previous work [26].

Regarding time complexity of this module, it is based on CNN, and the total time complexity of all convolutional layers [27] is estimated by Equ. 1:

$$O(\sum_{l=1}^{d} n_{l-1} \cdot s_l^2 \cdot n_l \cdot m_l^2). \quad (1)$$

where d is the number of convolutional layers, n_{l-1} is the number of input channels of the l-th layer, n_l is the number of filters in the l-th layer, s_l is the spatial size of the filter and m_l is the spatial size of the output feature map.

This time complexity is applied in both training and testing phase, and training time is approximately three times of testing time.

2) Parameter Measurement

Many SLAs of service providers depend on several performance metrics such as latency, packet loss, and link utilization [20]. Consequently, a parameter measurement module is implemented to measure these parameters on each link in the network. Latency is measured according to an existing work [28] while packet loss and link utilization are measured thanks to *PortStatistics* API [29] in the controller.

C. Problem Detection

In this section, a problem detection approach is presented to identify abnormal symptoms of network problems. Unlike the existing rule-based approaches that trigger the alarms when network parameters (e.g., packet loss, delay, etc.) exceed a threshold, its fluctuations are monitored to early detect the unusual symptoms in this approach [30]. The class of application and the time-series parameters on network flows such as latency, packet loss, and link utilization are taken into account. These parameters are concatenated into a 1-dimensional vector. Then this vector is analyzed according to random forest algorithm as in some existing work [31], [32] to classify the network states into normal or abnormal states to early detect the network problems.

Time complexity of random forest algorithm in training phase is $O(n_0 * log(n_0) * d * k)$ where n_0 is the number of input sample, d is dimensional of data and k is the number of trees in the forest while the time complexity in testing phase is $O(k * d^{'})$ where $d^{'}$ is the depth of tree [33].

D. Application-aware Remediation

1) QoE Estimator

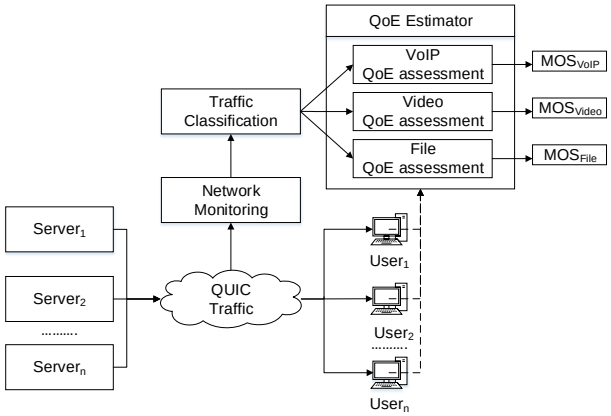


Fig. 3: The QoE estimator for encrypted traffic.

QoE is considered as a factor to evaluate the performance of the proposed SR mechanism. Using a common QoE model for various applications is not effective because each application has a particular QoS requirement. Consequently, a novel QoE estimator (Fig. 3) is implemented to calculate the QoE of different applications. According to the related research work, QoE estimation approaches are divided into three types: subjective, objective, and hybrid solutions. The first one requires participants to evaluate their perceptions about multimedia applications, so it is costly, time-consuming, and ineffective with real-time monitoring due to human intervention. The second one constructs an objective model estimating user perception using parameters (e.g., network, application, etc.), but identifying an effective model is sometimes complicated. The third one is a hybrid approach that uses a subjective dataset to learn QoE models based on ML algorithms. Consequently, it can combine advantages of both subjective and objective approaches. Therefore, we implemented the hybrid approach for QoE estimation, offering opportunities to estimate MOS in real-time. The MOS dataset and description about testbed building it (e.g., number of participants, understanding level of participants, etc.) are published to Github [34]. We have chosen the Random forest algorithm to estimate the MOS score because it has less root mean square error than other ML approaches [35]. MOS is estimated according to network parameters (e.g., latency, packet loss, etc.). There are five levels of MOS including 1 (Bad), 2 (Poor), 3 (Fair), 4 (Good) and 5 (Excellent). According to the traffic classification module, an appropriate pre-trained QoE model is implemented

for each application. The detail of this module is described in [35].

2) RL-based Segment Routing

The SR algorithm is formalized as a RL task that contains *agent, state, action, reward,* and *policy* (Fig. 4). The detail is described as follows:

Agent: An entity in the network system applies a learning algorithm to perform its tasks. In a routing problem, an agent selects appropriate paths to optimize a reward.

State s: A snapshot of the network environment which is observed by the agent.

Action a: An action illustrates how an agent replies to the network environment. In the routing problem, the action is a routing path between a server and a client in the network. All routing paths in the network can be obtained in the SDN controller.

Policy π: A policy is a map from an observed state to action in the network environment.

Reward r: A reward is a feedback of the network environment corresponding to the agent. In the routing problem, the agent monitors a network state s and performs an action a from the routing policy. Then, the agent moves to next state s' and receives a reward *r*. The reward is the MOS score of a chosen path which is calculated via the pre-trained QoE estimator (section III-D1).

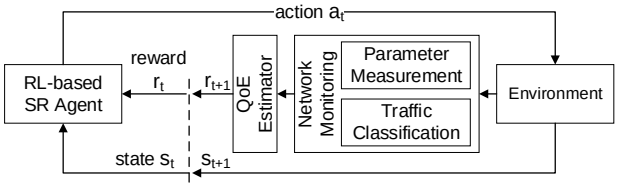


Fig. 4: The RL-based SR mechanism.

The main goal of RL is to optimize an objective function O_f (Eq. 2):

$$O_f = Max\ E[\sum_{t=0}^{\infty} \gamma_t \times r_t]. \tag{2}$$

where $\gamma_t \in [0, 1]$ is a discount factor.

In RL, there are two kinds of approaches including model-based and model-free approaches. In the first approach, the agents learn the environment model and enhance its policies to obtain optimality while the agents optimize its policies without prior information about the network environment in the second approach. The first one can learn faster than the second one. It is still less popular because of a large storage cost and dependence on accuracy of an initial information [36]. Therefore, the model-free approach is used in this work. In this approach, Q-value estimates how good it is to execute a given action in a given state. $Q(s, a)$ is the expected return starting from state *s* and taking action *a* following policy π. At a time step, the agent is in state s, performs action a, receives reward r and moves to next sate s'. The Q-value is updated as in Eq. 3:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a')]. \tag{3}$$

where α is a learning rate and γ is a discount factor.

Time complexity of reinforcement learning algorithm is $O(m * N_0)$ where N_0 is the number of action, and m is the size of state space [37], [38].

3) Exploration-Exploitation Trade-off

An exploration and exploitation phase in RL needs to be balanced to obtain an optimal cumulative MOS score (cumulative reward). The exploitation phase which selects the routing path (action) with maximal Q-value, can not be implemented systematically because each routing path needs to be evaluated frequently to achieve the optimal MOS score. In this paper, the trade-off between the exploration and exploitation phase is formalized as a MAB problem (Multi-Armed Bandit). MAB problem is a formalization of sequential decision-making tasks. At a time step, a decision-maker selects an action and receives a reward from an unknown distribution corresponding to this action. The main objective is to maximize the total reward received through a sequence of actions. In this paper, three selection algorithms are presented to resolve the MAB problem: ϵ-*greedy* [39], *softmax* [40], and *UCB1* (Upper Confidence Bounds) [41].

First, ϵ-*greedy* is the simplest algorithm to resolve the bandit problem. Concretely, the agent selects the routing path with the highest Q-value with a probability of (1-ϵ). Otherwise, the agent selects the routing path randomly. Then, the ϵ value reduce against the time so that the agent can learn more about the network environment and become more confident.

Second, the *softmax* algorithm selects the routing paths according to a probability function of Q-value. Each routing path a_i is assigned to a probability p_i as in Eq. 4:

$$p_i = \frac{e^{\frac{Q_{a_i}}{\tau}}}{\sum_{j=1}^{N_0} e^{\frac{Q_{a_j}}{\tau}}} \tag{4}$$

where τ is a temperature parameter, N_0 is a number of routing paths and Q_{a_j} is a Q-value of routing path a_j.

When the temperature parameter τ is reduced, the routing paths are exploited more frequently. In that way, the temperature parameter τ is reduced each episode (forwarding time). Therefore, *softmax* algorithm not only explores the less-used routing paths but also selects the best routing path in terms of expectation gains.

Finally, the *UCB1* algorithm is related to an index-based algorithm. UCB-index is defined as a sum of a current Q-value and a confidence bound. The UCB-index is described as in Eq. 5:

$$UCB - index_{a_i} = Q_{a_i} + \sqrt{\frac{2ln(N)}{n_{a_i}}}. \tag{5}$$

where Q_{a_i} is a Q-value of routing path a_i, n_{a_i} is a number of chosen time of routing path a_i and N is an episode number (forwarding time).

After calculating the UCB-index for each routing path, *UCB1* algorithm selects the path with maximal UCB-index. As shown in Eq. 5, the UCB-index comprises two parts including Q-value Q_{a_i} and confidence bound $\sqrt{\frac{2ln(N)}{n_{a_i}}}$. A routing path is chosen when the Q-value is large or the confidence bound is high. When the routing path with the large Q-value is chosen, this choice is an exploitation trial. When the confidence bound is high, this choice is an exploration trial. The confidence bound is higher when the number of chosen times of the routing path is smaller in comparison with other paths. In other

words, the less routing path is selected, the more it has the opportunity to be selected.

IV. Experimental results

A. Experiment Setup

TABLE I: Configuration of the PC used in the testbed.

Operation System	Ubuntu 16.04.6 LTS
Processor	Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz
Memory	2133MHz DDR4 8GB

Concerning the experiments, we used the tool mininet [42] to emulate a network with Openflow switches. We implemented a real SDN controller using ONOS [43] and connected it to Openflow switches in the network emulated by mininet. The latter is a popular emulator in the networking research community to emulate network topologies. ONOS merely supports leaf-spine topology, so we use this topology for both simple scene (five spine nodes and two leaf nodes) and a complex one (more than 15 nodes). The number of nodes in the topology is set to 7, 17, 32, and 47. Moreover, ONOS also supports segment routing through *org.onosproject.segmentrouting* application, so we customized it to implement the proposed SR mechanism. To generate QUIC traffic of video streaming in the network, we replay a pcap file of QUIC traffic from servers to clients using tcpreplay application. That pcap file is collected by watching videos on Youtube with Google Chrome. The testbed is deployed in a PC described as in Tab. I. For the exploration-exploitation trade-off, ϵ and τ are set to 1 and 2, respectively. For the RL algorithm, α and γ are set to 0.7 and 1, respectively. These parameters are selected according to an existing work [44]. The source code of the proposed framework is available at [45].

TABLE II: Scenarios.

Scenarios		Descriptions
perfect scenario		No delay and loss
Scenario with faults	*delay*	25, 50, 75, 100 and 125ms
	loss	5, 10, 15, 20 and 25%
	delay and loss	(25ms, 5% loss), (50ms, 10%), (75ms, 15%), (100ms, 20%) and (125ms, 25%)

The experiments are considered in four scenarios (Tab. II). The network parameters are chosen to cover the maximum QoE range. The first scenario is to evaluate the performance of the proposed approach in the network condition without delay and loss, and other scenarios are used to consider the proposed mechanism in the context with faults. Loss and delay parameters in the routing paths are generated according to *mininet*. In the last three scenarios, each routing path is randomly set to a specific delay, loss or both delay and loss. These delay and loss parameters are changed every 50 episodes which are chosen according to the experiments to generate dynamic network states. An uniform link capacity is set to 10 Mbps, and a sending rate is set to 2.5 Mbps as in an existing work [46]. The topology in the testbed is the leaf-spine topology which contains two leaf nodes and five spine nodes.

The proposed application-aware SR mechanism is designed for three applications including video streaming, file transfer and VoIP. Among these applications, video streaming comprises the highest global IP traffic [23], so we consider video streaming as a proof-of-concept to thoroughly validate the performance of the proposed SR mechanism.

B. Benchmark

To validate the performance of the proposed SR mechanism, our proposal is compared with the benchmarks including:

1) *Standard SDN-based SR (Standard_SR)*: This algorithm uses the Dijkstra algorithm to determine the shortest paths between servers and clients.
2) *SDN-based SR with maximal QoE (Max_QoE)* [47]: This approach calculates the MOS score of all routing paths and selects the paths with maximal MOS score. In contrast, the MOS score of a chosen path is calculated in the proposed mechanism which helps to reduce resource consumption.

These approaches are evaluated via the following performance metrics:

1) *MOS*: the perceived quality at the user's side.
2) *CPU Usage*: the percentage of the CPU's capacity which is calculated via *ps* command (process status) in the Unix-like operation systems.
3) *Control Overhead*: To discovery network topology and update status of links (e.g., latency, loss, etc.), a routing algorithm needs to generate the control packets (e.g., LLDP packets, etc.). Control overhead refers to the ratio of the control packet number to the total number of the sent packets.

C. Performance Analysis

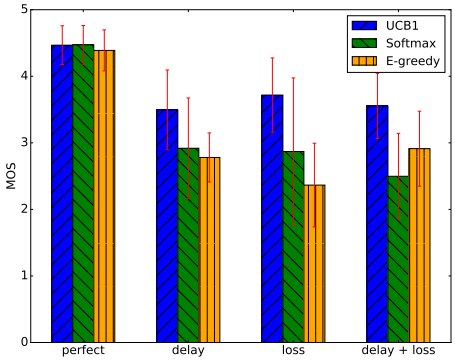
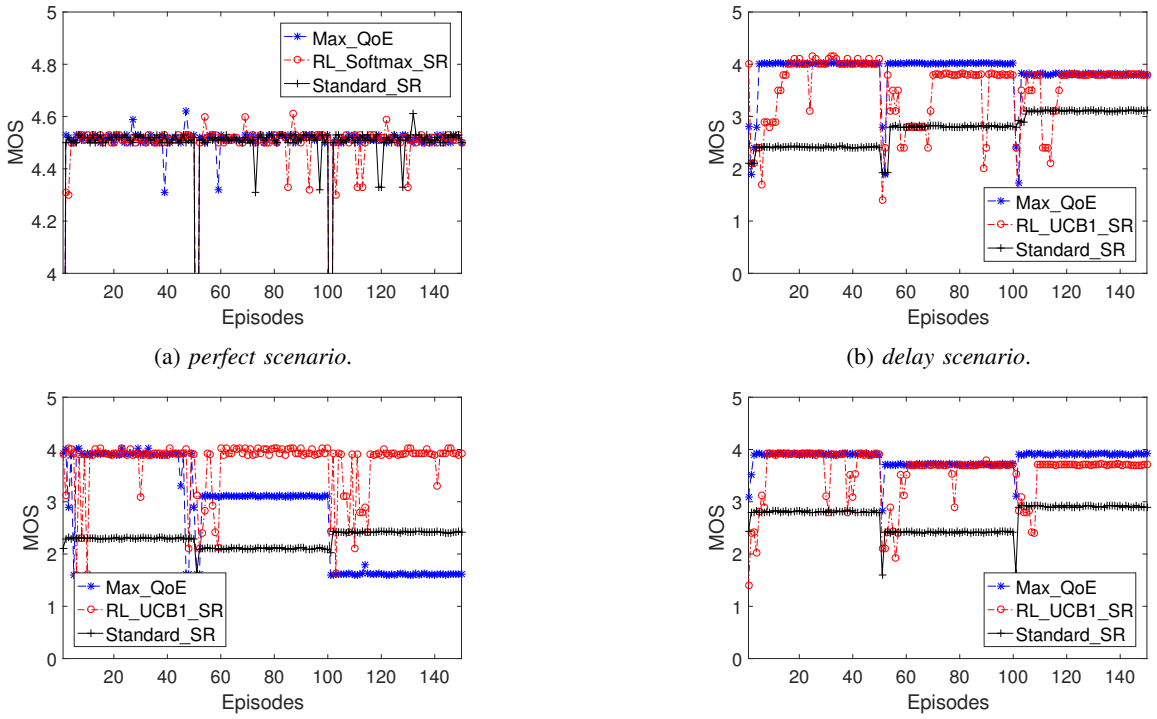


Fig. 5: Average MOS score and standard deviation of three selection algorithms in the proposed SR mechanism.

The selection algorithms for MAB formalization are presented including *UCB1, softmax and ϵ-greedy*. Their performances are first evaluated to select the appropriate selection algorithm. Then, the performance of our proposal is compared to the benchmarks including *Standard_SR* and *Max_QoE* mechanisms related to MOS score, CPU usage, and control overhead. Besides, the running time of each module in the proposed SR mechanism is thoroughly evaluated.

1) Time Analysis

In section III, we explained the mathematical time complexity of the proposed SR mechanism's algorithms. According to this time complexity, the traffic classification requires higher processing time in comparison with other modules. It requires



(a) *perfect scenario*.

(b) *delay scenario*.

(c) *loss scenario*.

(d) *scenario with loss and delay*.

Fig. 6: The MOS score of three SR mechanisms.

0.0316 ms to identify the network flows of VoIP and 3.4 ms to identify the network flows of video streaming and file transfer. When there is a huge amount of network traffic, collecting all packets of network flows for the traffic classification module is not effective due to high computational resources. Therefore, we use sFlow to reduce the amount of traffic collected for this module. In the testbed, we configure sFlow agents to collect 10 percent of network traffic going through sFlow agents. Regarding the running time of the problem detection module, it requires 0.0127 ms to process a network flow. As for the RL-based Segment Routing, it only needs to search in the Q-table, which representing routing policies, to select an appropriate path, so it can make instant decisions.

2) Selection algorithms

Fig. 5 illustrates the average MOS score and standard deviation of *UCB1, softmax* and ϵ-*greedy* in the proposed SR mechanism with four scenarios. In the *perfect scenario*, there is no delay and loss in the network. The sending rate is 2.5 Mbps while the link capacity is 10 Mbps. Consequently, changing the routing paths does not lead to the fluctuation of the MOS score in three selection algorithms. Although there is no significant difference between these algorithms, the MOS score of *softmax* is slightly better than the others. The average MOS score of *softmax, UCB1* and ϵ-*greedy* reach to 4.47, 4.46 and 4.39, respectively. Besides, their standard deviations are 0.29, 0.291 and 0.31, respectively. In this scenario, the MOS score varies from 4.3 to 4.5. The MOS score of ϵ-*greedy* converges to 4.3 while the MOS score of the others converges to 4.5. Therefore, the average MOS score of ϵ-*greedy* is lower than two other selection algorithms.

The average MOS score of *UCB1* is the highest, and the standard deviation is the lowest in three selection algorithms for the other scenarios. The average MOS score of *UCB1* in *delay scenario*, *loss scenario* and the final scenario are 3.55, 3.7 and 3.5, respectively. *UCB1* first explores the routing path during the first few episodes and then converges to optimal MOS score. When the network states are changed, *UCB1* can explore the less-used routing paths. Consequently, it continues to reach an optimal value quickly. *softmax* chooses the routing paths according to probability function, so it takes much time to converge to an optimal value. As a result, the MOS score of *softmax* is lower than the figure for *UCB1*. For ϵ-*greedy*, it first explores the routing paths with a high ϵ value. When ϵ value reduces to approximately 0, it can not explore the routing paths frequently. Therefore, the MOS score of ϵ-*greedy* is lower than two other selection algorithms.

According to Fig. 5, our proposal will implement different selection algorithms corresponding to various scenarios. *softmax* will be implemented for *perfect scenario* while *UCB1* will be implemented for other scenarios in our proposal in the following experiments. Our proposal refers to *RL_Softmax_SR* in *perfect scenario* and *RL_UCB1_SR* for other scenarios.

3) Segment routing mechanisms

Fig. 6 illustrates the MOS score against the episodes (forwarding time) of three SR mechanisms in four scenarios. In *perfect scenario* (Fig. 6a), the role of the routing paths is the same due to no delay and loss in the network. As a result, the MOS score of three SR mechanisms is nearly equal. The average MOS score of our proposal (*RL_Softmax_SR*), *Max_QoE* and *Standard_SR* are 4.47, 4.47 and 4.46, respectively. Besides, their standard deviations are 0.29, 0.29 and 0.34, respectively. Network states at different episodes are different. Moreover, the proposed approach using the reinforcement learning algorithm sometimes explores less-used

routing paths to balance exploration and exploitation phases. Therefore, there are the MOS fluctuations between 4.3 and 4.6 in Fig. 6a.

In *delay scenario* (Fig. 6b), each routing path is set to a specific delay parameter which leads to the differences in the MOS score between the routing paths. Therefore, there is a significant difference between three SR mechanisms. The MOS score of *Standard_SR* is the lowest in three SR mechanisms because it selects the shortest paths to forward the packets. The MOS score of this mechanism depends on the delay parameter which is set to the shortest path. The network states are changed every 50 episodes, so the MOS score of this mechanism changes periodically. The MOS score of *Standard_SR* in the first, second and last 50 episodes are approximately 2.4, 2.8 and 3.1, respectively. *Max_QoE* monitors the network topology, calculates the MOS score of all routing paths and selects the path with the best MOS score. Therefore, it achieves a high MOS score (approximately 4) in the majority of episodes. In this scenario, our proposal (*RL_UCB1_SR*) implements *UCB1* to determine the routing paths to forward the packets. *UCB1* explores the routing paths in the first 16 episodes, so the MOS score varies between 1.7 and 4. After that, the MOS score of our proposal converges to an optimal value (approximately 4). At 50th and 100th episodes, the network states are changed to create dynamic network environments. UCB-index of the less-used routing path becomes larger when a routing path is not chosen frequently. Consequently, *UCB1* can explore the less-used routing paths to select the appropriate path. As a result, the MOS score of our proposal can converge to optimal value quickly after a few episodes. Besides, our proposal can choose another routing path at 24th and 89th episodes after converging to an optimal value. In other words, our proposal does not always select the routing path with the best UCB-index to guarantee the exploration-exploitation trade-off. In Fig. 6b, the MOS score of our proposal is slightly lower or equal to the MOS score of *Max_QoE*.

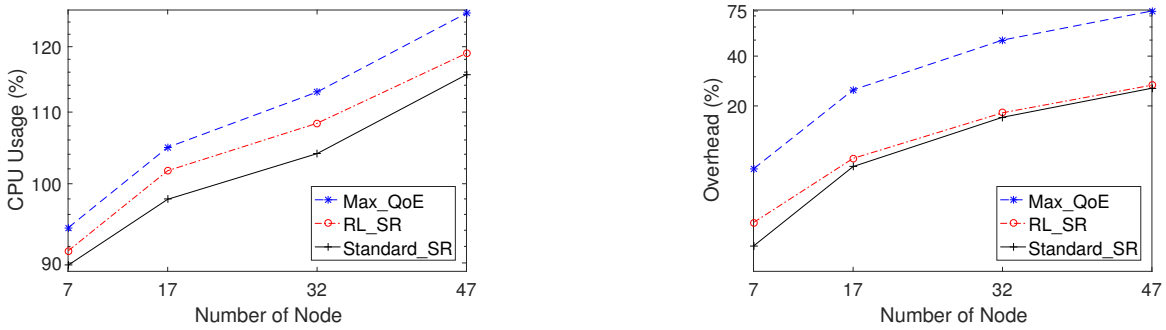
In *loss scenario* (Fig. 6c), the MOS score of *Standard_SR* is lower than the figure for our proposal. The MOS score of this mechanism in the first, second and last 50 episodes are 2.3, 2.1 and 2.4, respectively. Similar to previous scenarios, our proposal using UCB1 selection algorithm (*RL_UCB1_SR*) first explores the routing paths during the first few episodes and then converges quickly to optimal value (nearly 4). A remarkable feature from Fig. 6c is that the MOS score of *Max_QoE* is lower than the MOS score of our proposal from the 50th episode. ONOS controller sends the control packets (e.g., LLDP packets, etc.) to discover the network topology every 3 seconds, so the delay on the routing paths is measured according to control packets as in an existing work [48]. Therefore, *Max_QoE* can monitor the network and select the path with the best MOS score in *delay scenario*. The loss parameter is measured via *PortStatistics* API in ONOS controller after data is transmitted in the network. Consequently, the loss on a routing path is not updated when this path is not chosen. At the beginning of the first 50 episodes in *loss scenario*, the loss of all routing paths is initiated to 0. Therefore, *Max_QoE* can select the path with the best MOS score. At the 50th episode, the network states are changed. The routing paths with low MOS score in previous network state becomes the paths with high MOS score in current network state, but the loss on these paths are not updated. Therefore, the MOS score of *Max_QoE* is lower than the figure for our proposal.

In *scenario with delay and loss* (Fig. 6d), the MOS score of *Standard_SR* is the lowest in three SR mechanisms. Similar to *delay scenario*, the MOS score of our proposal (*RL_UCB1_SR*) is equal or slightly smaller than the MOS score of *Max_QoE*. The MOS score of *Max_QoE* in the first, second and last 50 episodes are 3.9, 3.7 and 3.9, respectively. Besides, the optimal MOS score of our proposal are 3.9, 3.7 and 3.7, respectively. Although the MOS score of *Max_QoE* is higher than the others, it requires much resource consumption related to CPU usage and overhead.

There is less difference between the CPU usage and overhead in four scenarios, so the average CPU usage and overhead of these mechanisms in four scenarios are depicted in Fig. 7. The number of leaf nodes is increased while the number of spine nodes is not changed to obtain a larger network topology in these experiments. Fig. 7a shows the CPU usage of three SR mechanisms. The testbed is implemented in a computer with 4 cores described in Tab. I, so the CPU usage can reach 400 percent in maximum. The CPU usage of *Max_QoE* is the highest in three SR mechanisms because it monitors the entire network topology in order to obtain the routing path with the best MOS score. The CPU usage of *Max_QoE* for 7 nodes is 94.3 percent, and it increases to 125.48 percent for 47 nodes. The CPU of our proposal is lower than the figure for *Max_QoE*. It raises from 91.45 to 118.95 percent when the number of nodes increases from 7 to 47. Besides, there is an increase in the CPU usage of *Standard_SR* from 89.73 to 115.63 percent when the number of nodes raises from 7 to 47.

Fig. 7b indicates the control overhead of three SR mechanisms. The overhead of *Max_QoE* is higher than two other mechanisms because *Max_QoE* sends the control packets to monitor the entire network topology. It increases rapidly from 8.33 to 75 percent when the number of nodes increases from 7 to 47. Our proposal only monitors the chosen paths, so the overhead of our proposal and *Standard_SR* are nearly equal. The overhead of our proposal with 7 and 47 nodes are 3.95 and 26.71 percent, respectively. The overhead of *Max_QoE* is nearly three times the overhead of our proposal for 47 nodes. In other words, our proposal reduces up to 64.39 percent of overhead in comparison with *Max_QoE*. Although *Max_QoE* can achieve a high MOS score in the majority of episodes, it requires much resource consumption in terms of CPU usage and overhead. Consequently, it is not appropriate for a large-scale network. In contrast, the MOS score of our proposal is nearly equal or higher than the figure for *Max_QoE* in considered scenarios, but it requires less resource consumption in comparison with *Max_QoE*.

Some important results are summarized as in Tab. III and



(a) Average CPU usage.

(b) Average overhead.

Fig. 7: The average CPU usage and overhead of three SR mechanisms.

TABLE III: Summarization of average optimal MOS, median and 95 % confidence interval of MOS in the SR mechanisms.

Scenarios	Mechanisms	Average	Improvement of proposal (%)	Median	95% Confidence Interval
Perfect	Proposal	4.47	-	4.53	4.47±0.047
	Standard_SR	4.46	Nearly equal	4.52	4.46±0.056
	Max_QoE	4.47	Equal	4.53	4.47±0.047
Delay	Proposal	3.86	-	3.82	3.86±0.1
	Standard_SR	2.76	139.8	2.80	2.76±0.05
	Max_QoE	3.93	Nearly equal	4.01	3.93±0.062
Loss	Proposal	3.9	-	3.93	3.9±0.089
	Standard_SR	2.26	172.6	2.11	2.26±0.023
	Max_QoE	2.86	136.4	3.11	2.86±0.154
Delay+ Loss	Proposal	3.76	-	3.70	3.76±0.078
	Standard_SR	2.7	139.3	2.40	2.7±0.04
	Max_QoE	3.83	Nearly equal	3.72	3.83±0.024

TABLE IV: Summarization of average overhead in the SR mechanisms.

Mechanisms	Average overhead (%)			
	7	17	32	47
Max_QoE	8.33	24.99	50	75
Proposal	3.95	9.64	18.18	26.71
Improvement of proposal (%)	**52.58**	**61.43**	**63.64**	**64.39**

IV. Tab. III indicates important results related to the median MOS, 95 % confidence interval of MOS and the average optimal MOS score which is the average value of the optimal MOS score in each 50 episode of three SR mechanisms. Tab. IV illustrates the essential results in term of the average overhead against the number of nodes of our proposal and *Max_QoE*. Confidence interval (CI) gives an estimated interval for an unknown population parameter. It is associated with a confidence level, representing a probability which the estimated interval includes a true value of the parameter. 95% confidence interval is computed at the 95% confidence level containing the parameter. It is calculated by Equ. 6.

$$CI = \overline{x} + z^{\star} \cdot \frac{\sigma}{\sqrt{n}} \tag{6}$$

where $\overline{x}$ is mean of MOS, σ is its standard deviation, n is number of samples and $z^{\star}$ is 1.96 for 95% of confidence level.

In Tab. III, 95% confidence interval of the proposed SR mechanism is wider than the figure for *Max_QoE* in four scenarios. The reason is that the proposal needs to explore the routing paths to select the appropriate one, so its MOS fluctuates more frequently than the figure for *Max_QoE*.

V. Conclusion

Segment Routing needs to be performed more adaptively to avoid network problems (e.g., congested links, etc.) and meet different service-level agreement requirements. To cope with these demands, we propose a novel SDN-based adaptive segment routing framework for network operators in the context of encrypted traffic. Our proposal is developed on the SDN controller which can be integrated into networks supporting virtualized architectures related to SDN. The proposed segment routing mechanism implements different routing policies corresponding to various applications and meets strict service-level agreement requirements. Moreover, the appropriate routing path is selected according to reinforcement learning policy and the feedback of the network environment (QoE). The experimental results show that the proposed SR mechanism using reinforcement learning outperforms the standard SR mechanism in terms of QoE and reduces up to 64.39 percent of overhead in comparison with *Max_QoE* mechanism.

Segment list is one of the important factors of the segment routing mechanism, so path encoding algorithm needs to be investigated thoroughly to optimize the performance of the segment routing mechanism. After detecting the network problems and implementing the adaptive segment routing mechanism to reduce its influences, the root causes of the issues needs to be considered to deal with it definitely. Therefore, the root cause analysis mechanism will be investigated in our future work.

Acknowledgment

This research is funded by Vietnam National Foundation for Science and Technology Development (NAFOSTED) under grant number 102.02-2019.314.

References

[1] Z. Li, S. Peng, D. Voyer, C. Xie, P. Liu, Z. Qin, K. Ebisawa, S. Previdi, and J. Guichard, "Problem Statement and Use Cases of Application-aware Networking (APN)," Internet Engineering Task Force, Internet-Draft draft-li-apn-problem-statement-usecases-01, Sep. 2020, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-li-apn-problem-statement-usecases-01

[2] P. L. Ventre, S. Salsano, M. Polverini, A. Cianfrani, A. Abdelsalam, C. Filsfils, P. Camarillo, and F. Clad, "Segment routing: A comprehensive survey of research activities, standardization efforts and implementation results," *IEEE Communications Surveys & Tutorials*, 2020.

[3] "Cisco to optimize ntt docomo's 5g mobile backhaul for simpler, more flexible and scalable network operation," January 2021. [Online]. Available: https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=2057092

[4] "Cisco and vodafone showcase mobile transport networking advancements via segment routing at mobile world congress," January 2021. [Online]. Available: https://newsroom.cisco.com/press-release-content?type=webcontent&articleId=1913303

[5] A. Kushwaha, S. Sharma, N. Bazard, A. Gumaste, and B. Mukherjee, "Design, analysis, and a terabit implementation of a source-routing-based sdn data plane," *IEEE Systems Journal*, 2020.

[6] N. Kukreja, R. Alvizu, A. Kos, G. Maier, R. Morro, A. Capello, and C. Cavazzoni, "Demonstration of sdn-based orchestration for multi-domain segment routing networks," in *2016 18th International Conference on Transparent Optical Networks (ICTON)*. IEEE, 2016, pp. 1–4.

[7] S. Peng, J. Mao, R. Hu, and Z. Li, "Demo abstract: Apn6: Application-aware ipv6 networking," in *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2020, pp. 1330–1331.

[8] A. Custura, R. Secchi, and G. Fairhurst, "Exploring dscp modification pathologies in the internet," *Computer Communications*, vol. 127, pp. 86–94, 2018.

[9] C. Dong, C. Zhang, Z. Lu, B. Liu, and B. Jiang, "Cetanalytics: Comprehensive effective traffic information analytics for encrypted traffic classification," *Computer Networks*, vol. 176, p. 107258, 2020.

[10] CISCO, "Cisco white paper report: Encrypted traffic analytics," February 2021. [Online]. Available: https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf

[11] W. Wang, M. Zhu, J. Wang, X. Zeng, and Z. Yang, "End-to-end encrypted traffic classification with one-dimensional convolution neural networks," in *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*. IEEE, 2017, pp. 43–48.

[12] P. Wang, F. Ye, X. Chen, and Y. Qian, "Datanet: Deep learning based encrypted network traffic classification in sdn home gateway," *IEEE Access*, vol. 6, pp. 55 380–55 391, 2018.

[13] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. ACM, 2017, pp. 183–196.

[14] "Quic statistic," January 2021. [Online]. Available: https://sniansfblog.org/the-potential-impact-of-quic-will-it-replace-tcp-ip/

[15] M.-C. Lee and J.-P. Sheu, "An efficient routing algorithm based on segment routing in software-defined networking," *Computer Networks*, vol. 103, pp. 44–55, 2016.

[16] H. Zeng, P. Kazemian, G. Varghese, and N. McKeown, "A survey on network troubleshooting," *Technical Report Stanford/TR12-HPNG-061012, Stanford University, Tech. Rep.*, 2012.

[17] V. Tong, H. A. Tran, S. Souihi, and A. Mellouk, "Service-centric segment routing mechanism using reinforcement learning for encrypted traffic," in *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020.

[18] A. A. Barakabitze, I.-H. Mkwawa, A. Hines, L. Sun, and E. Ifeachor, "Qoemultisdn: Management of multimedia services using mptcp/sr in softwarized and virtualized networks," *IEEE Access*, 2020.

[19] A. Rego, S. Sendra, J. M. Jimenez, and J. Lloret, "Dynamic metric ospf-based routing protocol for software defined networks," *Cluster Computing*, vol. 22, no. 3, pp. 705–720, 2019.

[20] A. Bahnasse, F. E. Louhab, H. A. Oulahyane, M. Talea, and A. Bakali, "Novel sdn architecture for smart mpls traffic engineering-diffserv aware management," *Future Generation Computer Systems*, vol. 87, pp. 115–126, 2018.

[21] K.-K. Yap, M. Motiwala, J. Rahe, S. Padgett, M. Holliman, G. Baldus, M. Hines, T. Kim, A. Narayanan, A. Jain *et al.*, "Taking the edge off with espresso: Scale, reliability and programmability for global internet peering," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 432–445.

[22] R. M. A. Ujjan, Z. Pervez, K. Dahal, A. K. Bashir, R. Mumtaz, and J. González, "Towards sflow and adaptive polling sampling for deep learning based ddos detection in sdn," *Future Generation Computer Systems*, vol. 111, pp. 763–779, 2020.

[23] V. Cisco, "Cisco visual networking index: Forecast and trends, 2017–2022," *White Paper*, 2021. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.pdf

[24] M. Lopez-Martin, B. Carro, A. Sanchez-Esguevillas, and J. Lloret, "Network traffic classifier with convolutional and recurrent neural networks for internet of things," *IEEE Access*, vol. 5, pp. 18 042–18 050, 2017.

[25] K. Zhou, W. Wang, C. Wu, and T. Hu, "Practical evaluation of encrypted traffic classification based on a combined method of entropy estimation and neural networks," *ETRI Journal*, 2020.

[26] V. Tong, H. A. Tran, S. Souihi, and A. Mellouk, "A novel quic traffic classifier based on convolutional neural networks," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–6.

[27] K. He and J. Sun, "Convolutional neural networks at constrained time cost," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 5353–5360.

[28] Y. Li, Z.-P. Cai, and H. Xu, "Llmp: exploiting lldp for latency measurement in software-defined data center networks," *Journal of Computer Science and Technology*, vol. 33, no. 2, pp. 277–285, 2018.

[29] ONOS, "Portstatistics api," February 2021. [Online]. Available: http://api.onosproject.org/1.5.1/org/onosproject/net/device/PortStatistics.html

[30] L. F. Maimó, Á. L. P. Gómez, F. J. G. Clemente, M. G. Pérez, and G. M. Pérez, "A self-adaptive deep learning-based system for anomaly detection in 5g networks," *IEEE Access*, vol. 6, pp. 7700–7712, 2018.

[31] J. Wang, Y. Tang, S. He, C. Zhao, P. K. Sharma, O. Alfarraj, and A. Tolba, "Logevent2vec: Logevent-to-vector based anomaly detection for large-scale logs in internet of things," *Sensors*, vol. 20, no. 9, p. 2451, 2020.

[32] J. Hong, S. Park, J.-H. Yoo, and J. W.-K. Hong, "Machine learning based sla-aware vnf anomaly detection for virtual network management," in *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020, pp. 1–7.

[33] S. Gao, C. Xiang, C. Sun, K. Qin, and T. H. Lee, "Efficient boolean modeling of gene regulatory networks via random forest based feature selection and best-fit extension," in *2018 IEEE 14th International Conference on Control and Automation (ICCA)*. IEEE, 2018, pp. 1076–1081.

[34] L. Amour, "Poqemon qoe dataset," June 2021. [Online]. Available: https://github.com/Lamyne/Poqemon-QoE-Dataset

[35] L. Amour, V. Tong, S. Souihi, H. A. Tran, and A. Mellouk, "Quality estimation framework for encrypted traffic (q2et)," in *2019 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2019, pp. 1–6.

[36] Z. Mammeri, "Reinforcement learning based routing in networks: Review and classification of approaches," *IEEE Access*, vol. 7, pp. 55 916–55 950, 2019.

[37] S. Koenig and R. G. Simmons, "Complexity analysis of real-time reinforcement learning," in *AAAI*, 1993, pp. 99–107.

[38] C. Jin, Z. Yang, Z. Wang, and M. I. Jordan, "Provably efficient reinforcement learning with linear function approximation," in *Conference on Learning Theory*. PMLR, 2020, pp. 2137–2143.

[39] V. Temlyakov, "Greedy approximation in convex optimization," *Constructive Approximation*, vol. 41, no. 2, pp. 269–296, 2015.

[40] Y.-L. He, X.-L. Zhang, W. Ao, and J. Z. Huang, "Determining the optimal temperature parameter for softmax function in reinforcement learning," *Applied Soft Computing*, vol. 70, pp. 80–85, 2018.

[41] H. A. Tran, S. Hoceini, A. Mellouk, J. Perez, and S. Zeadally, "Qoe-based server selection for content distribution networks," *IEEE Transactions on Computers*, vol. 63, no. 11, pp. 2803–2815, 2013.

[42] "Mininet," February 2021. [Online]. Available: http://mininet.org/

[43] "Open network operating system (onos)," February 2021. [Online]. Available: https://wiki.onosproject.org/display/ONOS/ONOS

[44] S.-C. Lin, I. F. Akyildiz, P. Wang, and M. Luo, "Qos-aware adaptive routing in multi-layer hierarchical software defined networks: A reinforcement learning approach," in *2016 IEEE International Conference on Services Computing (SCC)*. IEEE, 2016, pp. 25–33.

[45] V. Tong, "Service-centric segment routing using reinforcement learning," February 2021. [Online]. Available: https://github.com/vanvantong/rl-sr

[46] A. A. Barakabitze, I.-H. Mkwawa, L. Sun, and E. Ifeachor, "Qualitysdn: Improving video quality using mptcp and segment routing in sdn/nfv," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 182–186.

[47] E. Grigoriou, A. A. Barakabitze, L. Atzori, L. Sun, and V. Pilloni, "An sdn-approach for qoe management of multimedia services using resource allocation," in *2017 IEEE International Conference on Communications (ICC)*. IEEE, 2017, pp. 1–7.

[48] L. Liao and V. C. Leung, "Lldp based link latency monitoring in software defined networks," in *2016 12th International Conference on Network and Service Management (CNSM)*. IEEE, 2016, pp. 330–335.