Technological University Dublin
ARROW@TU Dublin

Doctoral

Engineering

2023

Investigating the Effects of Network Dynamics on Quality of Delivery Prediction and Monitoring for Video Delivery Networks

Obinna C. Izima
Technological University Dublin, izimaobinna@gmail.com

Follow this and additional works at: https://arrow.tudublin.ie/engdoc

 Part of the Electrical and Computer Engineering Commons

Recommended Citation

Izima, O. C. (2023). Investigating the Effects of Network Dynamics on Quality of Delivery Prediction and Monitoring for Video Delivery Networks. Technological University Dublin. DOI: 10.21427/NMH6-S029

This Theses, Ph.D is brought to you for free and open access by the Engineering at ARROW@TU Dublin. It has been accepted for inclusion in Doctoral by an authorized administrator of ARROW@TU Dublin. For more information, please contact arrow.admin@tudublin.ie, aisling.coyne@tudublin.ie, vera.kilshaw@tudublin.ie.

Funder: Science Foundation Ireland

Investigating the Effects of Network Dynamics on Quality of Delivery Prediction and Monitoring for Video Delivery Networks

Obinna C. Izima

Supervisor: Dr. Ruairí de Fréin

Prof. Mark Davis

School of Electrical and Electronic Engineering

Technological University Dublin

A thesis submitted in partial fulfilment of the

requirements for the award of the degree of

Doctor of Philosophy

January 2023

In loving memory of my late parents,

Hon. Justice Sir & Lady C. O. C. Izima (KSC).

Declaration

I certify that this thesis which I now submit for examination for the award of Doctor of Philosophy is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. This thesis was prepared according to the regulations for graduate study by research of the Technological University Dublin and has not been submitted in whole or in part for another award in any other third level institution. The work reported on in this thesis conforms to the principles and requirements of the TU Dublin's guidelines for ethics in research. TU Dublin has permission to keep, lend or copy this thesis in whole or in part, on condition that any such use of the material of the thesis be duly acknowledged.

Obinna C. Izima

January 2023

Acknowledgements

First and foremost, I would like to express my profound gratitude to God, the almighty, for His grace, mercy, faithfulness, countless blessings, which have guided me from the very inception to the completion of this PhD research work.

I would like to convey my heartfelt gratitude to Dr. Ruairí de Fréin, my first supervisor, for his unwavering support of my Ph.D study and related research, as well as his patience, continuous support, motivation, and vast expertise. He put in a lot of odd hours of work both during week days and weekends to ensure that this project was a success. His guidance, drive and encouragement helped me in the course of carrying out this research as well as the writing of the thesis. Thank you so much, Dr. Ruairí de Fréin.

I would also like to thank Prof. Mark Davis, my second supervisor for his support and encouragement from the beginning stages of this research to the completion of the study. I was greatly assisted by his insightful comments as he helped broaden my understanding of my research from various angles. My sincere thanks also goes to Dr. Ali Malik, whom I worked with on a couple of projects during this research. He assisted me in many ways from prototyping ideas, allowed me adapt some of his Software Defined Network topologies for some experiments and his unwavering support.

I would also like to express my profound gratitude to Science Foundation Ireland (SFI) for funding this PhD Project under Grant No. 15/SIRG/3459.

Finally, my sincere appreciation goes out to my family: my darling wife, Oluchi, and lovely kids, Gabriella and Emmanuella, for their patience, encouragement and support

throughout this PhD journey. This success is yours. I would also like to thank my siblings, Dr. Chidinma Izima-Ukenta, Dr. Alozie Izima, Barr. Chiemela Izima; my in-laws, Elder Dr. Emmanuel and Mrs. Joy Oti; my church, the Redeemed Christian Church of God (RCCG), Balleyjamesduff, County Cavan for their prayers and unwavering encouragement. My appreciation also goes out to my good friend, David Okonkwo. To everyone who supported me in one way or the other, I am grateful.

Publications

■ Journals

- Obinna Izima, Ruairí de Fréin, and Ali Malik. (2021). "A Survey of Machine Learning Techniques for Video Quality Prediction from Quality of Delivery Metrics" *Electronics 10*, no. 22: 2851. https://doi.org/10.3390/electronics10222851.
- Obinna Izima, Ruairí de Fréin, and Ali Malik. (2022). "Using Device-Level Statistics to Label Video Quality of Delivery Metrics" (Under review in *IEEE Access Journal*).

■ Conferences

- Obinna Izima and Ruairí de Fréin, "Load-Adjusted Prediction for Proactive Resource Management and Video Server Demand Profiling," *IEEE 33rd Irish Signals and Systems Conference (ISSC)*, 2022.
- Ruairí de Fréin, Obinna Izima, and Ali Malik, "Detecting Network State in the Presence of Varying Levels of Congestion," In *IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, 2021, pp. 1-6, doi: 10.1109/MLSP52302.2021.9596271.
- Obinna Izima, Ruairí de Fréin, and Ali Malik, "Codec-Aware Video Delivery Over SDNs," *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2021, pp. 732-733.
- Obinna Izima, Ruairí de Fréin, and Mark Davis, "Predicting Quality of Delivery Metrics for Adaptive Video Codec Sessions," *IEEE 9th International Conference on Cloud Networking (CloudNet)*, 2020, pp. 1-7, doi: 10.1109/CloudNet51028.2020.9335813.

- Obinna Izima, Ruairí de Fréin, and Mark Davis, "Evaluating Load Adjusted Learning Strategies for Client Service Levels Prediction from Cloud-hosted Video Servers," *26th AIAI Irish Conference on Artificial Intelligence and Cognitive Science*. 2018. vol. 2259, pp 198-209.
- Obinna Izima, Ruairí de Fréin, and Mark Davis, "Video Quality Prediction Under Time-Varying Loads," *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, 2018, pp. 129-132, doi: 10.1109/CloudCom2018.2018.00035.
- Ali Malik, Ruairí de Fréin, Chih-Heng Ke, Hasanen Alyasiri and Obinna Izima, "Bayesian Adaptive Path Allocation Techniques for Intra-Datacenter Workloads," *IEEE International Conference on Computer Communications and Networks (ICCCN)*, 2021, pp. 1-8, doi: 10.1109/ICCCN52240.2021.9522253.

Abstract

Video streaming over the Internet requires an optimized delivery system given the advances in network architecture, for example, Software Defined Networks. Machine Learning (ML) models have been deployed in an attempt to predict the quality of the video streams. Some of these efforts have considered the prediction of Quality of Delivery (QoD) metrics of the video stream in an effort to measure the quality of the video stream from the network perspective. In most cases, these models have either treated the ML algorithms as black-boxes or failed to capture the network dynamics of the associated video streams.

This PhD investigates the effects of network dynamics in QoD prediction using ML techniques. The hypothesis that this thesis investigates is that ML techniques that model the underlying network dynamics achieve accurate QoD and video quality predictions and measurements. The thesis results demonstrate that the proposed techniques offer performance gains over approaches that fail to consider network dynamics. This thesis results highlight that adopting the correct model by modelling the dynamics of the network infrastructure is crucial to the accuracy of the ML predictions. These results are significant as they demonstrate that improved performance is achieved at no additional computational or storage cost. These techniques can help the network manager, data center operatives and video service providers take proactive and corrective actions for improved network efficiency and effectiveness.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

MACHINE Learning (ML) applications are being incorporated by an increasing number of video streaming networks. Network and video content providers are under tremendous pressure from the expansion of video streaming services to proactively maintain high levels of video quality. In response, numerous studies have applied ML techniques for the prediction of the quality of the video streams and monitoring of the network infrastructure. In most cases, these studies have either applied the ML techniques blindly or have treated the models as black-boxes. A consequence of this is that these applications fail to model the network dynamics in the network infrastructure. This leads to inaccuracies in the ML predictions and failure of the models to generalize well in settings different from the experimental setup.

This thesis considers the problem of video quality predictions and monitoring of video application using ML. A differentiating factor in the contributions proposed in this thesis is that the thesis contributions model the network dynamics present in the systems considered. The thesis contributions focus on Quality-of-Delivery (QoD) metrics, which are independent of the type of video streaming service. These QoD measurements capture the end-to-end performances of network services. Examples of QoD metrics include jitter, packet

count, delay, throughput, rebuffering, frame rate. This chapter provides some overview and motivation for this research followed by a hypothesis statement and a list of contributions to the field. An outline of the contents of the rest of the thesis is also given.

1.1 Research Motivation

Advances in network technologies and modes of delivery, proliferation of smart phones and hand-held devices have all contributed to the growth of video traffic as can be inferred from the Cisco Visual Networking Index (VNI) [1]. Paradigm shifts in remote work culture are driving the enormous use of video conferencing applications such as Zoom, Google Meet, Skype [2]. This trend presents new challenges in guaranteeing the quality of the video content delivered over today's Internet. This is because video delivery deployed over the best-effort IP networks is highly dynamic in nature. Bandwidth fluctuations and time-varying delays make it a challenge to consistently guarantee the QoD of video content over such networks. When it comes to the new remote work structures, businesses have had to deal with disruptions as the performance of these video conferencing applications are closely tied to the state of the network. It is expected that as video traffic continues to dominate, that network performance requirements will increase in tandem. The reason being that the dominance of video traffic will also bring about a strain on the capacity to deliver quality video to satisfy users, devices and services requirements who regularly seek better QoD.

The growth of video streaming over the last decade has generated a lot of interest from both academia and industry. Various studies have considered problems in different aspects of the video delivery pipeline. The popularity of this area has also seen many investigations due to the deployment of popular video streaming services such as Netflix, YouTube and Twitch.

1.2 Problem Statement

A typical delivery pipeline for Over-The-Top (OTT) delivery media companies is shown in Fig. 1.1. Video content Service Providers (SP), for instance, Netflix provide paid content to their customers via third party IP networks which they have little or no control over. The video source files may start from storage devices like AWS S3 buckets. To stream the video over the Internet, the video data gets compressed and may be pushed to CDNs in order to deliver the content quickly and efficiently to the client devices. For these content providers, their level of control over the video delivery ecosystem may be limited to their own infrastructure. As a result, they are not in a position to accurately guarantee the QoD for video content relayed to client devices. More so, video delivery over third-party IP networks may introduce various impairments in the form of packet loss, delay and/or jitter which may deteriorate the audio-visual quality and expected QoD being received by the end-users.

To satisfy Service Level Agreements (SLAs), network operators must ensure that they are in a position to provide service guarantees to the video content SPs. As the volume of video traffic continues to grow, there is a call for some proactive prediction and monitoring of video QoD metrics to preserve the video quality delivered to clients. This is a major concern for the network operator and will be a differentiating factor for consumers when choosing a video delivery service. Also, today's highly competitive environment with similar pricing schemes, the SP and the network operators cannot afford to rely on profit generation based solely on service provision, but must also take into account different factors which may lead to customer churn [3]. They must strive to meet up with the dynamicity challenge in order to remain competitive. Improving learning algorithms without increasing their computational cost is important for cloud services and the video application community because responsivity to poor QoD is a differentiating factor for consumers.

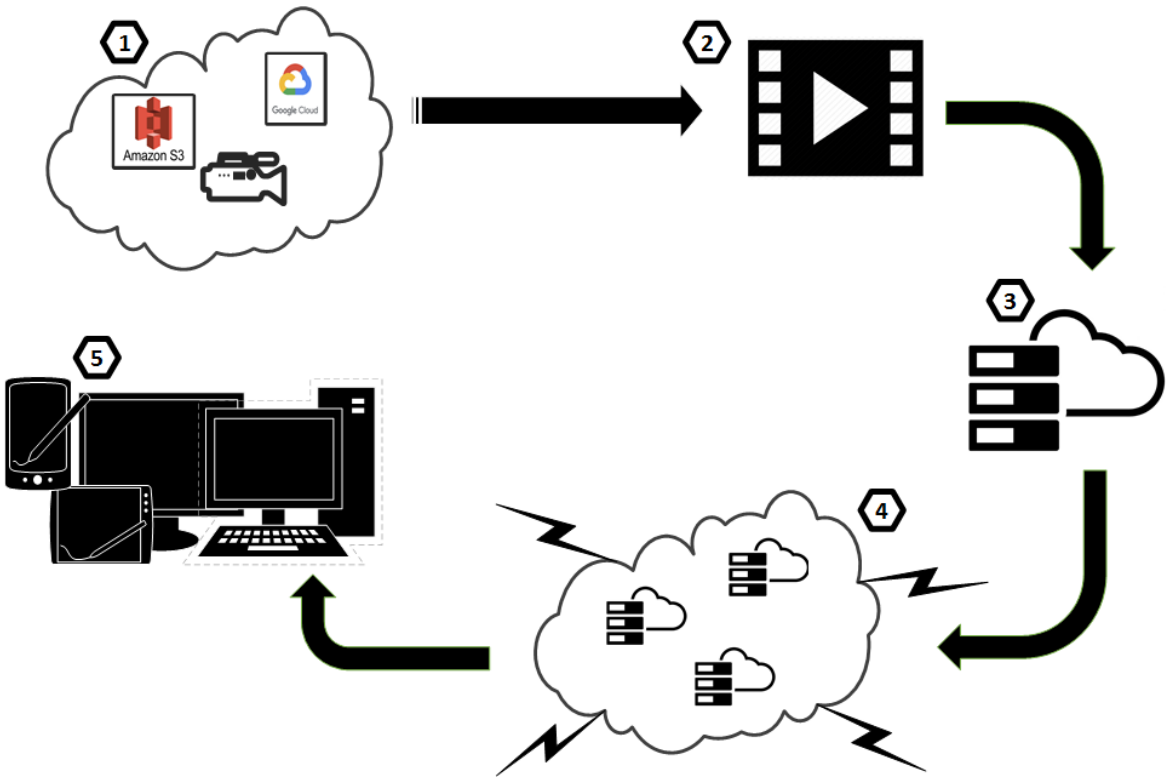

Fig. 1.1 **Typical Pipeline for OTT Media Delivery**: (1) Video input files can start from a variety of sources such as Google Cloud Storage, AWS S3, etc.; (2) These source files are then delivered to an encoder where an adaptive streaming package is generated; (3) The encoded files are then again stored in Google Cloud Storage, AWS S3, Akamai, Level 3, etc.; (4) For efficient global delivery, video SPs may choose to deliver the video content via Content Delivery Networks (CDNs) to provide the best possible quality; (5) The video content is now received at the client devices. The user may choose to initiate video playback in a browser, hand-held devices, workstations, etc.

The field of ML has continued to attract a lot of interest for many practitioners who deploy ML in applications that provide solutions and enable automation in various domains. ML involves building models that can learn to arrive at decisions directly from data without following preset rules. ML empowers a system with the ability to scrutinize data and infer knowledge. This learning process extends beyond knowledge extraction, to utilization and improvement of the inferred knowledge over time and with experience. The networking community is not left out as there are many researchers interested in applying ML in diverse networking problems [4] as evidenced in this survey [5]. In recent years, there has been a

rapid adoption of ML to solve practical problems in computer networks. One reason for the widespread adoption, stems from the recent advances in ML leading to more flexibility and applicability to diverse areas. Furthermore, the success of ML in networks can also be attributed to the colossal amount of data in todays' networks. One of those key areas, is the application of ML in video streaming services and applications.

The need for some cognitive control in network operation and management presents a unique set of challenges for ML [6]. Networks are continually evolving and the dynamics prevent the application of ML solutions blindly in network operation and management [5]. In addition, the heterogeneity of networks also makes it impracticable for ML algorithms to be treated entirely as black boxes in QoD prediction. Studies have confirmed that in order to improve network service performance for QoD optimization, ML approaches can be used to support significantly more control rules and complicated models [7]. However, these network dynamics must be incorporated into the ML solutions for accurate prediction of QoD metrics. This is the premise for this study.

The objective of the thesis is to design ML predictors and network state monitoring techniques that model the underlying network dynamics. Studies in the area in which the underlying effects of the network are ignored and not incorporated in the ML techniques fall short of accurate predictions of video quality and QoD metrics. The solutions treat these ML algorithms as black-boxes. The aim of this research is to demonstrate that adopting the correct models for the right problem is crucial in designing ML techniques.

1.3 Research Hypothesis

The hypothesis that this thesis investigates is that by learning in the presence of network dynamics, it is possible to develop ML techniques that achieve accurate prediction of QoD metrics and the monitoring of the network infrastructure. Formally, the thesis hypothesis is as follows:

Hypothesis. *Machine learning models that explicitly model network dynamics whilst developing predictors and monitoring agents for QoD metrics in video delivery networks offer more accurate predictions than those that do not.*

In simple terms, the thesis hypothesizes that to achieve correct and accurate QoD predictions for video streaming services, ML techniques must model and incorporate the underlying network dynamics. ML techniques that do not consider the network dynamics will fail to achieve accurate QoD predictions. This is because ML techniques that do not consider the dynamicity of networks adopt incorrect models which may yield inaccurate results.

1.4 Contributions

This section outlines the contributions of this research to the field.

1.4.1 Load-Adjusted Learning

The authors of [8] hypothesized that by collecting kernel variables from a cloud-hosted video server, they could learn and predict the behaviour of the client system. They made their data publicly available. This thesis utilizes the dataset in this work. The work in [9] proposed the Load-Adjusted (LA) technique as a means to learn the mapping between the kernel metrics of a machine in a server cluster and service quality metrics on the client machines. This LA technique, unlike the approach taken by the authors in [8], incorporates the effect of the load (i.e. the number of concurrent users accessing the video service) by training the ML models conditional on the load in the system.

This thesis extends the scope of the LA technique, in a novel approach, to methods which enable automation of choice of the regression parameter through the use of the Elastic Net (EN) model. This is the first work that addresses this problem in this way. The thesis conducts simulations that examine the performance of different learning functions using a range of

ML algorithms for the prediction of the video QoD for a cloud-hosted server. The thesis investigates the efficacy of the LA learning for different video QoD metrics, and examine the performance of these learning functions for different QoD metrics under varying network conditions.

This thesis also addresses the problem of resource provisioning for cloud hosted services. It extends the scope of the LA technique to proactive resource management and video server demand profiling. This study formulates the problem of estimating cloud computational requirements as an integrated framework comprising of a learning and an action stage. In the learning stage, the thesis proposes using ML to predict the video QoD metric for cloud-hosted servers and then use the knowledge gained from the process to make resource management decisions during the action stage. The thesis also contributes an LA approach for predicting the load on the cloud video server. The results demonstrate that it is possible to accurately predict different number of users on the system using the LA method. The ability to predict future server demand profiles could help a network manager to proactively manage the dynamicity of cloud provisioned resources.

1.4.2 Learning in the Presence of Congestion and Adaptive Codecs

The fundamental structure of the architecture of video streaming systems comprises of a codec to encode the video stream, and a transport protocol to convey the video content from the source to the destination. Typically, the transport protocol is responsible for estimating the network capacity by looking at congestion, Round Trip Time (RTT) [10]. The codec then uses the network capacity to infer what levels of compression to apply to the video, so that it can adaptively relay the video stream to the destination [11]. Despite the advancements made in compression technologies, the fundamentals barely changed at all. What has changed is that new codecs are equipped with added tools and complexity. Modern video streaming services are codec-agnostic shifting the focus from codecs to how to enable the codec do a

better job. One way this is achieved is by modelling the feedback loop between the adaptive codecs and the network.

A second contribution of this thesis is a new model for estimating jitter, the Codec-aware Network Adaptation Agent (CNAA). The CNAA model is a lightweight data learning engine that achieves accurate predictions of a QoD metric, namely jitter, for video services. This model for estimating jitter is suited to the case where the delivery system uses an adaptive video codec. The CNAA model achieves this prediction accuracy by leveraging the available network information in the presence of congestion and adaptive codecs. By modelling the behaviour of the video codec, the CNAA technique achieves responsive, in-network learning. The performance of CNAA technique is compared with some ML algorithms such as the Exponentially Weighted Moving Average and Autoregressive Moving Average model. The thesis highlights the short-comings of these ML techniques that fail to capture the network dynamics and demonstrate their poor prediction performance in comparison with CNAA model.

The thesis extends the scope of the CNAA model to address the problem of estimating the state of the network for a video delivery system within a shared network infrastructure. The results demonstrate that by using the parameters obtained from the CNAA predictor model, it is possible to achieve network state congestion classifications. This can serve as a network monitoring scheme. These results show that by incorporating the new jitter estimation model for off-the-shelf tree-based supervised ML algorithms (namely Decision Trees and Random Forests), one can achieve accurate network monitoring classification based on congestion levels. The results from these experiments record network state classification accuracies of approximately 80% to 97%. These results outperform traditional ML classifiers which are based on the running average of the jitter statistics. The CNAA classifiers achieve service change-point detection with an accuracy of 80% to 95% and a network responsivity of 89% and above when the congestion levels are changing from 7% to 1.7% per second. The service

change-point is estimated on how effective the ML classifier is able to detect a change in the background congestion. The system responsivity refers to the time lag between a service change-point occurrence and the time the classifiers are able to detect the change in network state.

1.4.3 Automatic Labelling of Machine Learning Data

The third thesis contribution considers the problem of labelling ML data for predicting video quality in the presence of interfering services and applications. In a traditional Human-In-The-Loop (HITL) ML setting [12], the network engineer is involved in a repetitive cycle during which he is tasked with training, tuning, and testing the ML models. This process works like this: (1) first, the network engineer labels the training data. The labelled data is then presented to the ML model for the learning process; (2) the model may require tuning. This can be achieved in a variety of ways, but the most typical is for the network engineer to assess the data to correct for overfitting; (3) a final step in this process is to test and validate the model performance.

This thesis proposes an automatic network-enabled process that eliminates the requirement for the first step, i.e. the process of labelling the ML data. This data labelling process during which the raw data is annotated can be an expensive and a time-intensive process. This contribution poses the problem of automatic data labelling as the problem of estimating the parameters of the network queueing delays in the presence of interfering background traffic. This way, the thesis models the network as a queue and use estimates of the model to label jitter, a video QoD metric. The thesis demonstrates the feasibility of the automatic labelling process using device-level measurements captured from a video client machine in a real-world Software-Defined Network (SDN) topology. Using the automatic labelled data and by training some off-the-shelf ML classifiers, this work achieves network state congestion levels detection. Numerical results from the experiments demonstrate considerable gains of

the automatic labelling process and network congestion detection and classification for all scenarios evaluated.

1.5 Organisation of the Thesis

This thesis is organised as follows. In Chapter 2, a background review of the literature on recent trends in the application of ML algorithms in the prediction of video streaming quality using QoD-derived metrics is provided. This review chapter focuses on contributions that deployed ML techniques using QoD statistics for video quality prediction and monitoring. Chapter 3 extends the scope of the LA technique. This chapter presents techniques that model the dynamic effects of the time-varying workloads present in a video delivery system, along with methods for adjusting the regularization penalties for ML regression models in an adaptive manner. Chapter 4 introduces a new learning framework for adaptive codec sessions, the Codec-Aware Network Adaptation Agent (CNAA). This CNAA model serves as a QoD predictor and a network state classifier. The CNAA model achieves accurate predictions of jitter, a QoD metric, for adaptive video services. This it does by modelling the feedback loop between adaptive video codecs and the network congestion. An empirical evaluation is presented demonstrating the feasibility of the model and the performance of the new approach. In Chapter 5, a novel approach for automatic labelling of ML data is proposed and implemented demonstrating the feasibility of the technique. In Chapter 6, the study conclusions are presented. This chapter also highlights some limitations of the thesis. It addresses some possible future directions for further research in order to advance the research in this field.

Chapter 2

Background and Related Work

THIS chapter provides an overview of ML applications in networking and a background overview of video streaming. A review of literature that leverage ML algorithms for video quality predictions from QoD measurements is also presented.

Introduction

Machine Learning is a subset of Artificial Intelligence (AI) that enables computers to learn on their own to deliver predictions or solutions based on previous experiences. Deep Learning (DL) is a subset of ML, which basically entails a neural network with multiple layers that attempts to mimic the human brain, enabling it to learn from massive amounts of data [13, 14]. DL automates most of the steps involved in the feature extraction process. ML delivers insights to a wide range of disciplines, including computer vision [15], speech recognition [16, 17], SDNs [18], communication networks [19], and Internet of Things (IoT) [20]. In recent years, there has been a rapid adoption of ML techniques to solve practical problems in computer networks [21]. This chapter provides a review of recent applications

of ML techniques that use QoD measurements for the prediction of video quality. QoD measurements capture the end-to-end performances of network services [22].

Video traffic makes up the largest portion of all IP traffic [23], which puts pressure on network operators to manage their resources efficiently while meeting customer expectations. Traditionally, Video Quality Assessments (VQA) and predictions have been conducted through subjective [24] and objective [25] means. The subjective approach relies on quantifying the experience of the end users. This Quality of Experience (QoE)-driven network traffic management relies on monitoring and predicting application-level or QoD performance in terms of video Key Performance Indicators (KPIs), as they affect the end users' experience. The International Telecommunication Union (ITU) defines QoE as the overall acceptability of an application or service, as perceived subjectively by the end user [26]. A major challenge with video streaming is that there is no unified way to measure the QoE [27]. This has given rise to research efforts that investigate QoE models based on network statistics. In fact, QoE modeling can assist in identifying different KPIs for different categories of user. For instance, a network provider may be interested in rebuffering, quality switches, and how these artifacts impact the video stream. This will help quantify the effect on the user's QoE, and how these in turn may be related to network parameters such as delay or jitter. According to the authors of [28], subjective user studies provide reliable evaluations, but are costly, time consuming, and are not suitable for real-world applications. Objective VQA approaches place an emphasis on mathematical modeling aimed at providing a quality score that closely resembles the perceived image/video quality. Although objective VQA metrics such as Peak Signal to Noise Ratio (PSNR) [29] and Structural Similarity Index (SSIM) [30] are fast and relatively easy to implement, these measurements do not always reflect the end user's experience [31, 30]. Unlike those methods, the focus in this chapter is on network performance, as opposed to user experience.

Due to the inherent drawbacks in subjective and objective VQA approaches, researchers have turned their focus to ML-based video quality prediction. This shift has led to numerous studies that leveraged ML models for mapping network measurements to the streamed video quality. Some studies considered Quality of Service (QoS) metrics in video quality prediction. An application or service's QoS is a set of technology employed on a network to make sure it can operate reliably, even when the network's resources are limited [32]. According to the ITU-T Rec. E.800 [33], QoS can be defined as the "totality of characteristics of a telecommunications service that bear on its ability to satisfy stated and implied needs of the user of the service." From the network perspective, the European Telecommunications Standards Institute (ETSI) in its technical report, defined QoS as, "The ability to segment traffic or differentiate between traffic types in order for the network to treat certain traffic differently from others" [34]. Essentially, QoS refers to a group of technology that operates together on a network to ensure that it can reliably run high-priority applications and traffic even when network capacity is constrained. This technology achieves this by providing differentiated handling and capacity allocation to certain network traffic flows. These traffic flows correspond to service types, as QoS measurements are biased by the service under consideration. This offers the network administrator the capability to manage the order in which packets are processed and an ability to throttle the bandwidth available to the service. These QoS mechanisms in IP routers and monitoring protocols take a service view. They are affected by the services considered. QoS measurements for one service may not be relevant for another one. It is important to note that network monitoring and measurements are not always aimed at determining the quality of a particular service or application. In some cases, the aim is to determine the network's overall health. These QoS measurements are normally taken far away from the end users. As a result, they do not provide information that can be easily translated to another service.

QoD is concerned with the quality of the data delivery process [35]. It relates to the ability of the network and transport stacks to ensure quality data delivery. The authors of [36] make the case that QoD is correlated with the network stack's inability to deliver data in a reliable manner. Unlike QoS, QoD is service agnostic. It captures the end-to-end performance of the data delivery process of a network. For video applications, QoD measurements capture the capability of ensuring reliable delivery of the video frames. Some common measures of QoD include packet delivery delays from the source to the destination, i.e., transport delays and queuing delays, packet loss, jitter, and throughput. For video applications, limited QoD may result in frames being delayed, which could lead to non-smooth playback for the client. Some examples of video artifacts introduced as a result of limited QoD include jitter, stalls or freezing, and jerkiness. Fig. 2.1 illustrates the scope and differences between QoD, QoS, and QoE.

With the widespread adoption of end-to-end encryption for privacy and digital rights management reasons, operators lack insights into video quality metrics, such as startup delays, resolution, and stalling events, which are needed to accurately predict the video quality and drive optimum resource management. HTTP Adaptive Streaming (HAS) is becoming the de facto method for video streaming. This introduces some extra dynamics that pose a challenge for network operators attempting to predict the performances of their applications. The situation is further exacerbated by the different design approaches used in Adaptive Bitrate (ABR) algorithms employed in HAS platforms. A range of solutions have been proposed and validated that can predict a video application's stream quality in terms of KPIs or overall QoE, from the QoD measurements. Based on the data collected, these solutions propose models that map QoD measurements to QoE and/or video quality KPIs using ML techniques. This chapter provides a review of research efforts that utilized QoD measurements in ML-enabled video quality prediction.

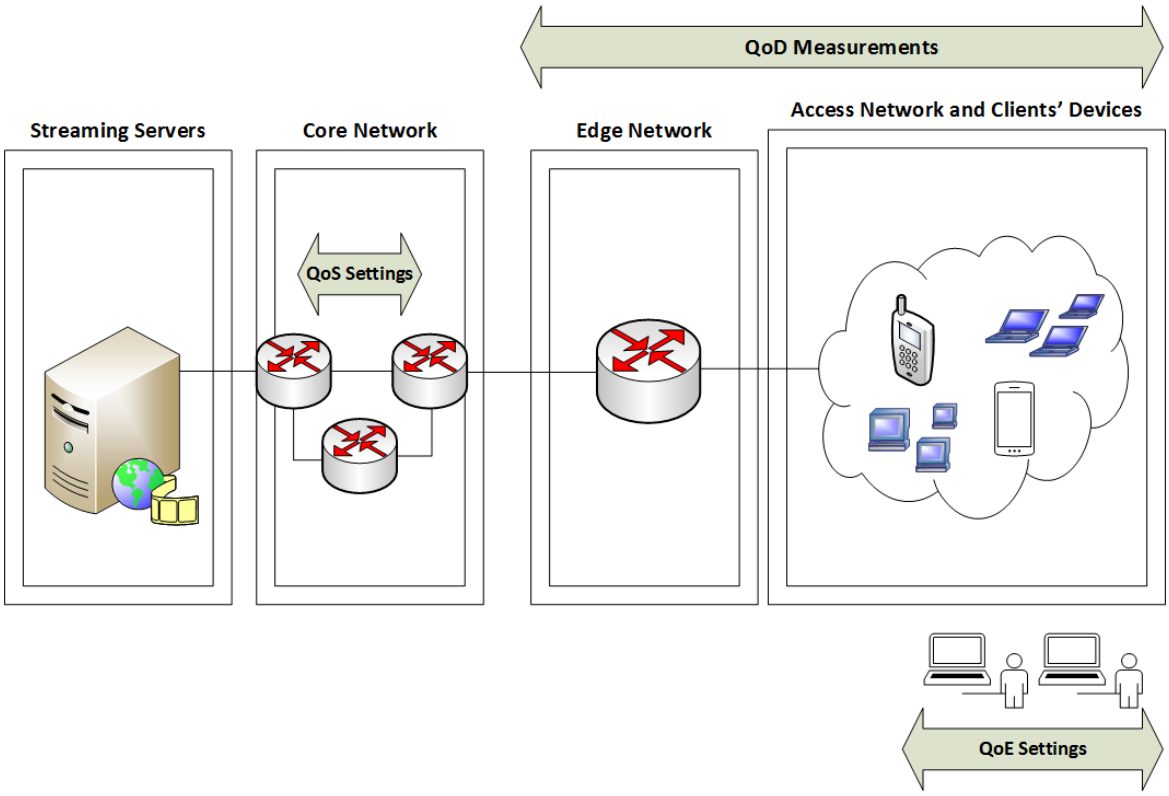

Fig. 2.1 QoD measurements capture the end-to-end performance of the data delivery process from the core network to access networks and client devices. QoS settings are applied at the core IP routers to provide differentiated handling and capacity allocation to certain network traffic flows. QoE evaluations are conducted on a subjective basis. The Mean Opinion Score (MOS) or Perceptual Evaluation of Speech Quality (PESQ), for example is then calculated for these evaluations.

Survey Methodology

Relevant literature in the field published between 2016 and 2021 is reviewed in this chapter. The review focused mainly on papers from databases and publishers such as IEEE Xplore, Elsevier, MDPI, Nature, ACM, Springer, and ArXiv. This review considered more than 200 papers on various ML applications in QoD prediction for video streaming services. This highlights that this is a very active area of research given the growth of video-related IP traffic. The selected papers for analysis and review were based on (1) recent contributions in the area, (2) ML applications for QoD predictions, (3) the actively researched family of

ML used for video services, (4) challenges in applying ML and DL techniques in video streaming and suggested resolutions, and (5) the computational requirements and feasibility of the approaches.

2.1 Scope

The rapid advances in network-level operations, such as SDNs, adaptive codec development, resource allocation, and slicing of networks, highlight the need for a new assessment on how to measure QoD. A review of the most promising future directions in this area is therefore needed. In response, this survey chapter provides a systematic review on recent breakthroughs in video quality prediction that leverage QoD measurements—from task formulation, QoD measurements utilized, ML algorithms to datasets, to future research directions. The review includes aspects such as video quality KPIs, model assumptions, research key findings, and the ML learning paradigms employed.

Fig. 2.2 shows the scope of this review chapter. The chapter provides a review of the recent trends in the application of ML algorithms in the prediction of video streaming quality using QoD-derived metrics from 2016 to 2021. In recent years, there has been an emphasis on end-to-end encryption, HAS streaming applications, video streaming with SDNs, and technological advancements in broadband and wireless networks. This survey focuses on contributions in these areas that deployed ML models using QoD statistics for video quality prediction. This survey also reviews contributions that predicted video quality under QoD impairments or in limited QoD settings and in WebRTC applications. A large number of publications on video quality prediction using ML exist. The survey focused on the research objectives, ML learning algorithms used, and datasets. The review also provides crucial insights into the methodologies used, their benefits, limitations, and applicability to real-world networking scenarios. The contributions of this chapter are outlined as follows:

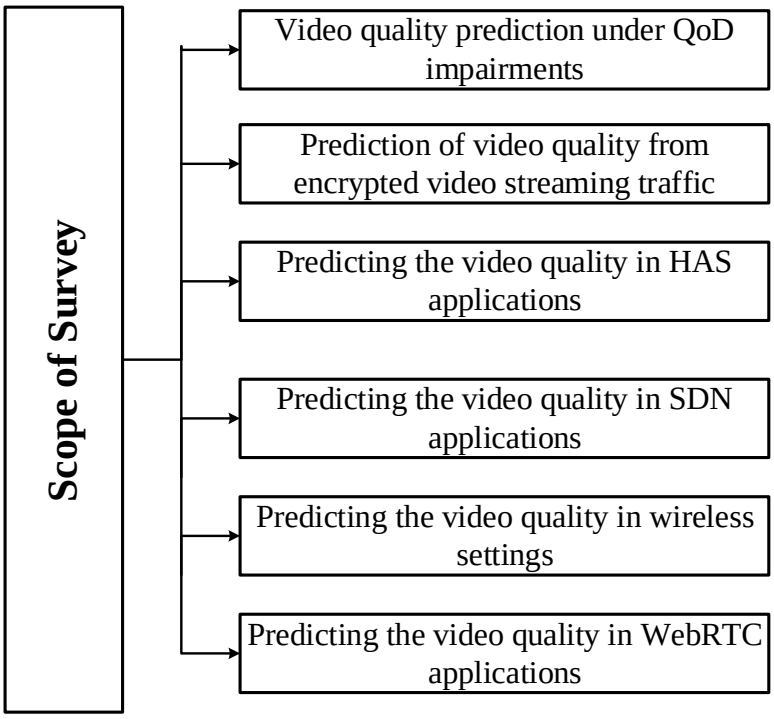

Fig. 2.2 This survey reviews ML video quality prediction using QoD metrics under QoD impairments, encrypted video traffic, HAS video applications, SDN, wireless settings, and WebRTC applications.

- An overview of ML applications in networking. This includes a review of related surveys, the learning paradigms used, and the applications considered.
- A background overview on video streaming. This includes the evolution of video streaming, modeling video quality, and common issues encountered in video streaming protocols.
- A review of the applications of ML techniques for predicting QoD metrics with the aim of improving video quality. The survey also provides a review of works that leveraged ML algorithms for video quality predictions from QoD measurements. This review provides a discussion on the ML techniques used in these studies, and analyze their

benefits and limitations. Fig. 2.2 highlights the scope and areas considered in this survey.

- A discussion of the future challenges and opportunities in the use of ML techniques in video streaming applications.

The rest of this chapter is organized as follows. Section 2.2 provides an overview of ML and DL algorithms. Section 2.2 also reviews some related surveys on the applications of ML in networking. Section 2.3 provides an overview on the background of video streaming. This section also discusses the structure of videos, an overview on the evolution of video streaming, and common protocols deployed over the years in the field of video streaming. The review discusses some common challenges faced in video streaming and highlight some common video quality metrics. Section 2.4 surveys ML-based techniques for predicting video quality from QoD measurements in the research literature. Finally, Section 2.5 concludes the review.

2.2 Overview of ML Applications in Networks

In this section, an overview on how ML is used in networks is provided. The review considered previous surveys on the applications of ML and DL based on the areas of interest and learning paradigms discussed.

ML is a subset of AI that enables computers to learn and improve from past experiences. DL is a subset of ML that relies on Deep Neural Networks (DNN) to train a model. In comparison to traditional ML, DL's main advantage is its automatic feature extraction, eliminating the need for time-consuming feature engineering [37]. This comes at the cost of requiring more computing power, and in most cases, an increase in computational complexity. ML models are easily adaptable to most problems and easy to understand.

The nature of the training data is one of the most important considerations in ML problems. The ML framework is trained to achieve a certain objective during the training process, such as making a decision, predicting a value, or performing a classification task. Without any human intervention, the ML framework learns the relationship between input and output data through training [38]. The online ML algorithm is another type of ML in which the model is updated for each new input feature after each prediction [39].

Fig. 2.3 depicts the basic workflow of machine learning algorithms. To train the algorithms, the dataset is loaded into the ML platform. The ML platform then generates the model, by learning the correct parameters and features for the prediction task. This model is tested for accuracy (against ground truth). If the accuracy is not satisfactory, then more optimization may be needed. This process may include modifying or discarding variables, and adjusting hyperparameters [40] (model-specific settings) to achieve acceptable levels of accuracy [41]. The trained ML system is then evaluated on additional data to guarantee that the model generalizes well to new data. There are generally three main categories of ML models: supervised learning, unsupervised learning, and Reinforcement Learning (RL).

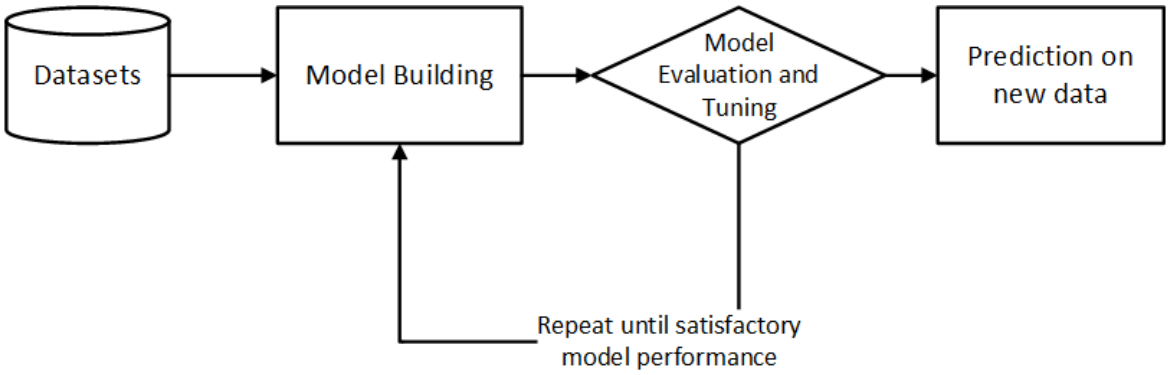

Fig. 2.3 The basic workflow of ML systems is shown. The datasets are loaded into ML algorithm, which learns the appropriate parameters for the task. The model evaluation and tuning process may involve modifying the learned model parameters and hyperparameter tweaking until an acceptable accuracy level is achieved. The model is then tested on new data to verify its accuracy.

Supervised learning relies on labeled datasets to create ML models. The goal in this technique is to predict the value of one or more output variable from a vector of input variables. The training dataset consists of samples of the input variables and the corresponding output values. In the learning method, e.g., regression, there is a function that provides a prediction of the value of the output variable in relation to a change of input variable. Linear Regression (LR) [42], Decision Trees (DT) [43], Random Forests (RF) [44], and Support Vector Machines (SVM) [45] are some examples of supervised learning algorithms. An important application of LR in the type of studies I survey is given in [9].

Unsupervised learning aims to explore the input data space to infer structure directly from unlabeled data. This sort of learning is essential in cases where the applications lack labeled data. While unsupervised learning can be used to solve a variety of problems, clustering algorithms [46] are the most popular. Examples of other unsupervised learning algorithms are k-Means [47], Self-Organizing Maps (SOM) [48], Expectation-Maximization (EM) [49], and Generative Adversarial Networks (GAN) [50]. In this context, the method in [51] is a form of blind deconvolution approach, which can be classed as an unsupervised learning algorithm.

RL is used to handle applications in which the goal is to learn a policy, i.e., a mapping between environmental conditions and actions to be done, while interacting directly with the environment. RL is an agent-based iterative technique where the agent learns by interacting with the environment and exploiting the knowledge. Unlike supervised machine learning, RL does not learn from a specific dataset. Instead, the RL agent learns from the relevance of its actions and selects an action based on previous information and a new choice. RL is fundamentally a trial-and-error learning strategy [52]. The reward from a specific action is learned via RL, which provides a feedback loop to the algorithm. The agent then alters its behavior in response to the preceding reward. Until the reward saturates or reaches a pre-defined threshold, the agent continues to interact with the environment by learning both

the action and the reward [53]. Based on the rewards obtained from the environment, the agent learns how good or bad its action was. Examples of RL include the Q-learning [54] and Deep Reinforcement Learning (DRL) [55].

The applications of ML to networks have been reviewed in numerous studies. Imran et al. in [56] provided a review with a focus on recent research studies and future trends in IoT, SDN, and ML hybrid applications. A range of perspectives were covered in this study, including wide-area networks, edge networks, and access networks. The authors reviewed how ML applications in SDN-enabled IoT environments bring about intelligent network decisions in the areas of traffic classification, routing optimization, QoE/QoS prediction, and resource management. Related surveys to that conducted by Imran et al., which focused on IoT, include [57, 58, 59]. Al-Garadi et al. in [57] surveyed ML and DL applications in developing security mechanisms for IoT frameworks. Mahdavinejad et al. in [58] provided a survey on how various ML methods may provide solutions for the challenges presented by IoT data, focusing on smart cities as the use case. The survey in [59] listed traffic profiling, IoT device identification, security, edge computing infrastructure, and network management as the major applications of ML in IoT. Security is a common area covered in these studies. As IoT systems are complex and integrate multiple components, maintaining security within the IoT wide-scale attack surface is challenging. The works described in [56, 59] considered the role of ML with SDNs in addressing the issue of security. The authors presented cases of ML integration with SDN for anomaly detection, traffic profiling, and classification.

Miller et al., in [60], presented a review of DNNs for defending networks against attacks. The review in [61] by Tang et al. surveyed the literature on applications of ML techniques for preventing phishing attacks in real-time. Meshram et al. in [62] provided a review of ML for anomaly detection in industrial networks. Hodo et al. in [63] conducted a survey which reviewed ML applications in Network Intrusion Detection Systems (NIDS) and their performances in detecting anomalies. The focus in this work was on the applications of DL

in NIDS. The survey in [64] by Sultana et al. reviewed recent works on ML techniques that leverage SDN to implement NIDS. In their paper, the authors argued that the SDN framework can be used to detect vulnerabilities in networks and to monitor the overall network health. Buczak et al. in [65] presented a review of ML applications for cyber security intrusion detection with a focus on wired systems. Otoum et al. in [66] provided a study of DL-based intrusion detection for monitoring critical infrastructures through Wireless Sensor Networks (WSN). The authors of [65, 66] both examined DL-based Intrusion Detection Systems (IDS) as an alternative to ML-based IDS. The DL techniques offered slightly better performances than the traditional ML approaches considered. The authors of [66] proposed ML-IDS for WSN-based infrastructure monitoring due to the shorter training time. In addition to the shorter training times of ML-IDS, the authors of [65] listed model interpretability, accuracy, and computational complexity as other factors to consider when choosing between ML-IDS and DL-IDS. These factors were common to both studies. Buczak et al. argued that the dataset needed to be more representative of the problem task. According to the authors, in order to detect anomalies and misuse, it is desirable for an IDS to reach network and kernel-level data. This granularity in data was not considered in the works described in the survey.

Sharma et al. in [67] reviewed the applications of ML in WSNs in the context of smart cities. The survey in [68] by Zhang et al. focused on mobile and wireless research based on DL. The review by Klaine et al. in [69] examined Self-Organizing Networks (SON) from the point of view of learning and provided an overview of the most common ML techniques used in cellular networks. Musumeci et al. in [70] provided an overview of ML applications in optical networks. These studies focused on specific ML algorithms or applications.

A survey on unsupervised learning applications in networking was provided by the Usama et al. in [71], who pointed out the dominance of supervised learning in networking. The authors surveyed the literature on unsupervised learning applications in the areas of

internet traffic classification, anomaly/intrusion detection, network operations, optimizations, and analytics. Other areas covered in the survey include QoS/QoE optimization and TCP optimization. Fadlullah et al. in [72] provided a survey of ML algorithms relevant to network traffic controls systems. They reviewed and compared the performances of ML and DL techniques proposed in the literature. The authors of Boutaba et al. in [5] conducted a review of the various network applications of ML, which included traffic predictions, traffic classifications, routing, congestion control, resource allocation, and QoS and QoE management. Although Boutaba et al. provided one of the most comprehensive surveys on the use of ML, their review focused on papers before 2018. Their survey missed out on some novel ML applications in networking. For instance, the LA learning first proposed in [9], and used in [21, 73, 51], was not covered in the survey. This thesis background in algorithms as opposed to networks provides a different perspective on the survey. A recent survey was published by Ridwan et al. in [74]. The authors presented a survey of recent trends in the application of ML in networks. The authors considered the applications of ML in congestion control, predictive models, IDS, routing, QoS improvements, and resource management. There are some reoccurring themes in these surveys in the following areas: ML for resource management, congestion control, routing, traffic classification, and network operations. For instance, given the deteriorating effect network congestion has on network performance, numerous studies have considered ML methods to ensure network stability and efficient resource utilization. Fig. 2.4 shows the summary of existing surveys and the branches of AI considered in the review. Table 2.1 provides a cluster of related surveys illustrating the different areas of focus and applications covered in these surveys.

Given the dominance of video traffic, a good number of research efforts from both academia and industry have investigated the applications of ML in the field of video streaming. Oprea et al. in [75] surveyed applications of DL techniques to intelligent video frame prediction from a sequence of context frames. Aroussi et al. in [76] reviewed the literature on

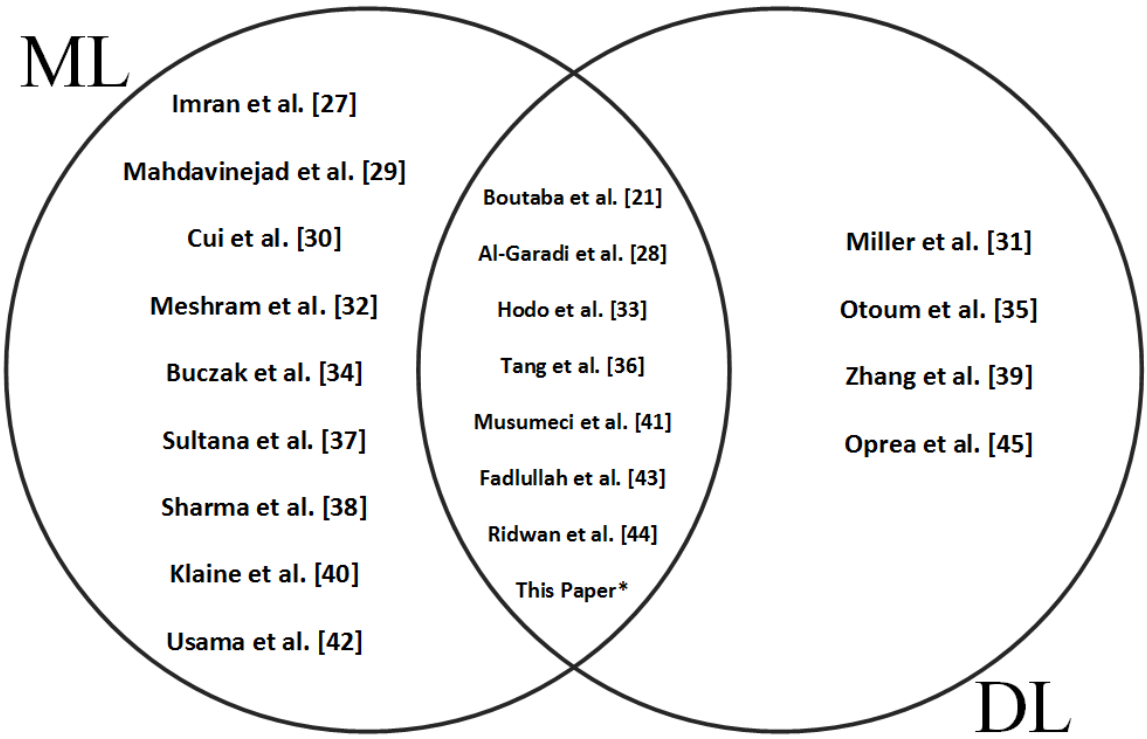

Fig. 2.4 A summary of existing surveys and branch of AI considered. Traditional ML algorithms dominate the literature possibly owing to the lower computational requirements. A good number of studies combine DL and ML techniques.

ML-based mappings of video quality with QoS-derived metrics. That study was similar to the survey provided in this chapter. However, their review considered papers published before 2014. This chapter reviews the literature on studies that leverage the QoD measurements in the prediction of the video quality.

Previous studies on the prediction of the video quality show that QoD measurements can be used to predict the overall quality of the video stream [77]. Recent evidence suggests that for QoD optimization, ML techniques can be leveraged to support more advanced and flexible models than traditional algorithms to achieve higher performing network services [7]. An example of how ML is used for QoD prediction in video networks is provided. Fig. 2.5 depicts a time-varying client population accessing video streams being streamed by a cloud-hosted video server. Using this setup, the authors of [21, 73] demonstrated how Audio

Buffer Rate (ABR), Video Frame Rate (VFR), and Packet count, QoD measurements, could be predicted using System Activity Reports (SAR). The SAR function is used to extract device statistics from the server. These device statistics, feature set x, refer to the operating system's metrics (the server). Some features captured with SAR include: the number of TCP active connections and number of running processes. Video stream requests are served by a VLC player at the clients, which extract QoD measurements: packet counts, VFR, and ABR.

The authors posit that by modeling the effect of the time-varying load [9] from the client population, they can predict the QoD measurements, y. They called this the LA technique. Let θ_n represent a video resource currently being accessed by a client. Equation (2.1) defines the server response to one video request at a time i as the sum of the resources held by a user and some deviation signal specific to a feature, $\epsilon_i[n]$:

$$x_i[n] = \theta_n + \epsilon_i[n], \quad \text{where} \quad i \in \mathbb{Z}, x_i[n], \theta_n \in \mathbb{R}. \tag{2.1}$$

The load signal $K[i]$ is multiplied by a typical usage weighting, θ_i, and produces the level of usage of resources θ_n. The objective is to predict the QoD metrics, packet count, VFR, and ABR at the client device with the device metrics. The study utilizes ML models to learn parameters from the input features in a way that the proposed model estimates closely approximates the actual.

The authors of [9] first considered the LR model. The LR models the relationship between the target QoD metric y and the independent variable x as a linear function of the form:

$$\hat{y_i} = \sum_{n=1}^{N} x_i[n]\beta[n] \tag{2.2}$$

where the intercept is represented by $x_i[n]$, and the remainder of the features are the predictor features. The model coefficients are represented by $\beta[n]$ where $n = 1, \ldots N$.

Using the LA method, the LR models are load-adjusted by training weights for each value of the load signal.

Table 2.1 Areas of focus and applications considered in related works. Majority of the studies reviewed focus on ML applications in the network security domain. The growth of video traffic necessitates a review of existing works; this thesis fills this gap.

Area of focus and application	Ref
Network Security / IDS	[60, 59, 57, 61, 62, 63, 64, 65, 66, 71]
IoT	[59, 56, 58, 57]
Smart Cities	[58, 67]
WSN / Mobile / Cellular Networks	[66, 67, 68, 69]
Networking, Network Operations & Optical Networks	[5, 71, 72, 74]
SDN	[56, 59, 64], This Thesis⋆
Video Frame Prediction	[75]
Video Prediction from QoD measurements	This Thesis⋆

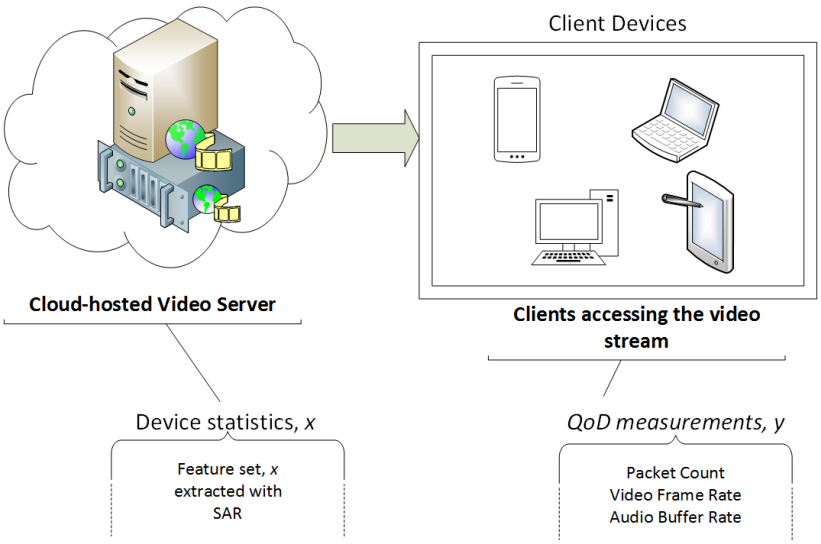

Fig. 2.5 Client devices access video streams served by cloud-hosted servers. Device statistics, feature set x are collected from the servers using SAR; ML models are used for the prediction of QoD measurements, y.

$$\hat{y}_i \Big|_{K(i)=k} = \sum_{n=1}^{N} x_i[n]\beta[n] \Big|_{K(i)=k} \tag{2.3}$$

In a more general sense, $y = f(x)$, where $f()$ is a ML algorithm such as RF or DT.

Fig. 2.6, R1 illustrates the packet count and the system load, recorded during 8000 s. The load $K(i)$ on the server side plays a key role in the QoD of the streamed video at the client device. As the load on the system increases, the TCP active connections' signal increases and may decrease the number of packets delivered to the client because of the limited system resources. The authors employed a range of ML models for the prediction of the QoD measurements. The models used in the study were as follows: LR [9], Ridge Regression (RR) [78], Least Absolute and Selection Operator (LASSO) [79], EN [80], and RF. The EN and RF models achieved the best performances for all QoD targets. Fig. 2.6, R2 compares the prediction accuracy of the EN model with the actual packet count statistics [21]. By utilizing a suitable adaptive regularization function, the load-adjusted learning algorithms outperformed the techniques that did not model the load effect by 10%, and reduced computation.

An example of where a feedback loop is required to improve a service and the overall health of the network is provided. The authors of [81] proposed CNAA, a lightweight and responsive system for predicting jitter, a QoD metric, suitable for a video delivery system that uses adaptive video codecs. The authors, by modeling the adaptive behavior of the codec in response to time-varying levels of network congestion, achieve accurate predictions of jitter. The demonstration in [82] shows how predictions realized from the CNAA learning agent can be integrated with an SDN controller to enable QoD improvements for an SDN-based video delivery system. Fig. 2.7 illustrates the ML-SDN framework demonstrated, which consists of three components: (i) the Topology Discovery and Statistics Gathering Component: creates a topological view of the underlying infrastructure and collects information periodically about the network traffic flow; (ii) Learning Phase Component: the CNAA learning agent achieves accurate predictions of jitter by estimating the adaptive behavior of the code; (iii) Action Phase Component: the reconfiguration component which exploits the information learned from the learning phase to achieve improvements in the QoD of the video.

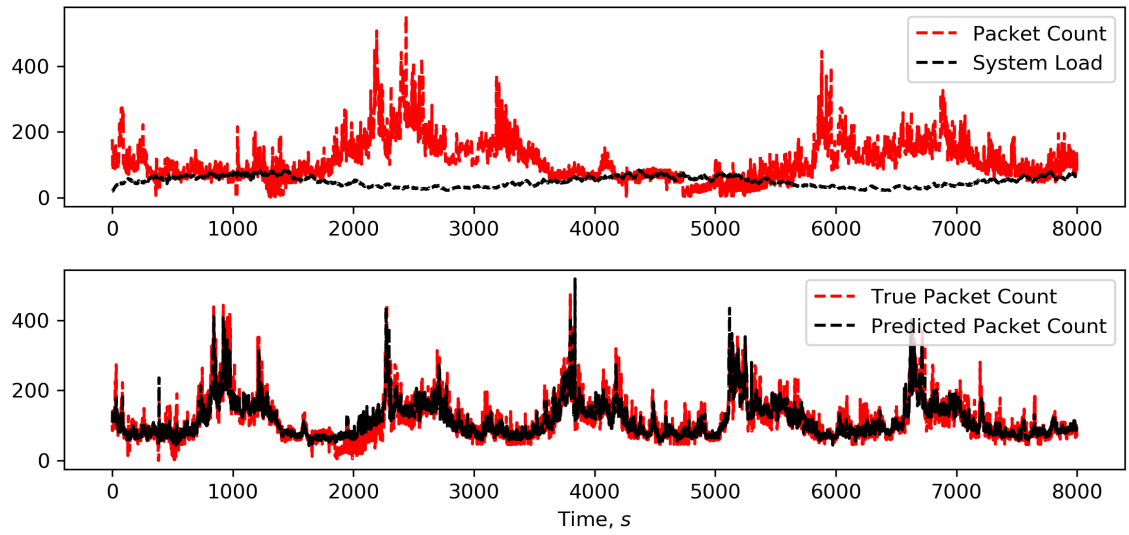

Fig. 2.6 Row 1 (R1): The packet count, y is illustrated for 8000s along with the system load. The load signal is a feature of x. There is a periodic change in request patterns. The load signal increases as the packet count decreases, which illustrates the dependencies between the two statistics. Row 2 (R2): The actual packet count statistics is compared to the predicted packet count.

2.3 Background on Video Streaming

Video streaming over IP networks enables viewers of a video to view the video stream without having to fully download the whole video first. Playback at client or end devices occurs as the media files are being downloaded. This in many ways differs from traditional data transfer over the Internet. The packet sizes in video streaming are much larger and generally require more bandwidth for transmission in comparison to data traffic. Another fundamental difference between video transmission compared with traditional data traffic is the fact that there are real-time constraints for video delivery.

Video streaming can be broadly divided into two flavors, Video-on-Demand (VoD) and live (real-time) video streaming. For VoD streaming, the content is stored on a server, and upon user request, the content is transmitted to the client. In live streaming, the server

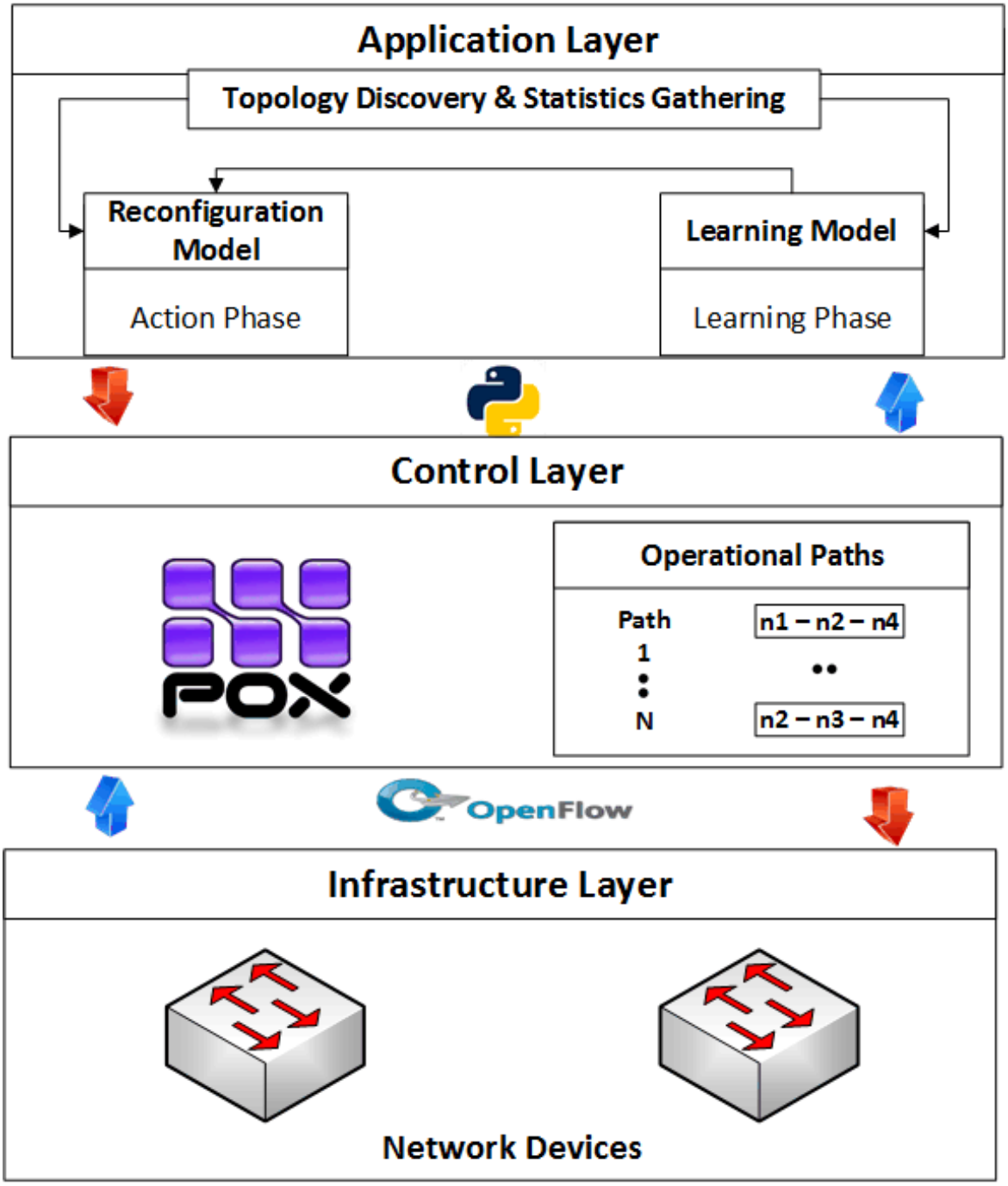

Fig. 2.7 An SDN controller has the ability to adaptively reroute the operational paths for a video delivery system by integrating QoD measurements from an ML model. Integration of CNAA predictions with an SDN controller in a feedback loop is able to preserve the video quality by avoiding congested paths.

generates video content in real-time as it is being injected by the source. Relay of live events via traditional broadcast TV is an example of this type of application [83].

2.3.1 The Anatomy of a Video Stream

A video stream comprises of a sequence of multiple segments. Individual segments are composed of several Group of Pictures (GOP), with each GOP having a header at the beginning. The number of GOPs in that segment, and the types of GOPs, are listed in the segment header. A GOP is made up of a series of frames. An I (Intra) frame comes initially, followed by multiple P (Predicted) and B (Bi-directional predicted) frames. A GOP frame is broken into numerous slices, each of which contains several macroblocks. Video encoding and decoding activities are performed at the macroblock level. There are two types of GOP: closed GOP and open GOP. Regarding closed GOPs, they are independent and can be processed independently. On the contrary, each open GOP is dependent on another GOP; it is impossible to process the GOPs independently [84].

Video streams can be split at various levels to be processed, including segment, GOP, frame, slice, and macroblock levels. Several GOPs can be processed independently at the sequence level. The time needed for the transmission and processing of a sequence, on the other hand, is a bottleneck due to its large size. Processing at the frame, slice, and macroblock levels necessitates dealing with spatio-temporal relationships, which makes the process difficult and time-consuming [85].

2.3.2 Video Streaming over IP Networks

Typically, for a video streaming session, the video streams or frames have to be played at between 24 and 30 frames per second (fps) to realize the illusion of motion. Prior to the delivery of video streams over IP networks, video compression algorithms are deployed to achieve intra and inter-frame compression. These have temporal dependencies of producing

I, B, and P frames. Of the three, I frames are largest due to the fact that they only utilize intra-frame compression, whereas B and P frames are smaller because they use previous I frames for further size reduction [86]. This gives rise to a situation whereby the encoded bits per frame is a variable quantity, which results in Variable Bit Rate (VBR) video.

The VBR encoded video stream is then transmitted on the Internet. As the Internet does not provide a consistent guaranteed bandwidth for the video data, the network can only support the video streams on a best-effort basis. As a result, if the network bandwidth is not enough to accommodate the video bit rate, then the decoder at the client end starts consuming the video content at a greater rate than the rate at which the new data stream is being received from the network. Subsequently, at some point the decoder finally runs out of video data to decode. This situation may result in frozen screens (rebuffering events and video stalls). This is clearly a problem. This is one of the motivations for the ML approaches outlined in this chapter. To address these issues without requiring some costly and complex guaranteed bandwidth mechanisms, the following solutions attempt to balance the video bit rate with the available network bandwidth [86]:

- Use of large playout buffer: The use of a large receive buffer can help overcome temporary variations in the network throughput. The video player can decode the pre-fetched data stored in the playout buffer.

- Transcoding-based solutions: These solutions modify one or more parameter of the raw video data algorithm to vary the resultant bit rate. Examples include varying the compression ratio, video resolution, or frame rate. However, transcoding-based solutions require complex hardware support and are computationally intensive processes.

- Scalable encoding solutions: These solutions are achieved by processing the encoded video data rather than the actual raw data. Hence, the raw video content can be adapted by utilizing the scalability features of the encoder. Some examples of these solutions

involve adaptation of the picture resolution or frame rate by exploiting the spatial and temporal scalability in the data. However, these solutions require specialized servers to implement this enhanced processing.

- Stream switching solutions: This technique is the simplest to implement and is also used for Content Delivery Networks (CDN). This approach involves preprocessing the raw video data to produce multiple encoded streams, at varying bitrates, resulting in multiple versions of the same content. Thereafter, a client-side adaptive algorithm is used to select the most appropriate rate based for the network conditions during transmission. Stream switching algorithms do not need specialized servers and use the least processing power. However, these solutions require more storage and finer granularity of encoded bitrates.

Industry favors using a large playout buffer and stream switching as the preferred solution for video transmission owing to the ease of deployment. To avoid a situation that may bring about a buffer under-run, the video server has to decide on the appropriate sending rate [87].

2.3.3 Evolution of Video Streaming

Video streaming has attracted much attention and research for quite some time. Its popularity has grown tremendously, particularly with the deployment of popular video streaming services such as Netflix and YouTube. In this section, I attempt to go back in time and present an overview of the evolution of video streaming over the Internet. I adopt a chronological order as I overview three stages of video streaming via the Internet.

2.3.3.1 Client–Server Video Streaming

The introduction of video streaming in the 1990s started with the design and implementation of the Client–Server (C/S) streaming architecture. In C/S video streaming, the client device

receives a video data stream pushed by a media server. With respect to the C/S architecture, the existing transport protocols, the TCP and User Datagram Protocol (UDP) were not entirely suitable for video relay over the best-effort Internet infrastructure. Owing to the real-time nature of most video streaming applications, the TCP's features of congestion control and reduction of data transfer windows sizes following packet losses were found to be detrimental to the application setup. The UDP, on the other hand, is a simple connectionless protocol. UDP offers no handshaking dialog with no retransmission, and no guarantee of delivery or ordering of packets [88]. To enable UDP real-time video streaming, the Real-time Transport Protocol (RTP) [89], Real Time Streaming Protocol (RTSP) [90], Session Description Protocol (SDP) [91], and Real-Time Control Protocol (RTCP) [92] were proposed to detect packet loss, compensate for jitter, and control the video streaming clients.

The RTP/RTCP/RTSP protocol suite was standardized by the Internet Engineering Task Force (IETF) [90] specifically for Internet video streaming. However, the characteristics of these protocols result in complex and expensive servers. When Network Address Translation (NAT) devices are present, they may not provide access to control traffic or media traffic [93]. Interoperability among media servers from different vendors was hard to achieve, despite implementing the same baseline protocols. This was largely due to some optional features or differences in design. Failovers often cause presentation snags due to a server fault and are rarely seamless unless certain redundancy schemes are in place. These scalability, vendor lock-in, and vendor dependency issues, along with the high maintenance costs, all present deployment challenges for protocols such as RTSP [27].

2.3.3.2 Peer-to-Peer (P2P) Streaming

P2P networks [94] allow users to share content without the need for centralized servers, making them an appealing option for video delivery over the Internet. In P2P video streaming [95], peers consume resources and supply resources to other peers. The peers

contribute their uplink bandwidths and do not depend on the underlying network infrastructure for any support [96]. There are two types of P2P networks: tree-based [97, 98] and mesh-based [99, 100]. In tree-based video streaming framework, peers are organized in a tree structure such that the video content is pushed from the root to subsequent levels of the tree until it gets to the leaves. This enables lower latency in data dissemination. Such setups are easy and simple to control but can be negatively impacted by peer churn [101]. In mesh-based video streaming systems, each participating peer can connect to a random set of neighbors. Each peer can then download and upload data to different peers at the same time. This is possible as there are no dependencies in this setup, whereas there are in the tree-based overlay. Data distribution here is done in an unstructured manner, and this setup is more suited for applications that can tolerate start-up delays. Since each peer maintains a set of neighbors at any given point in time, this overlay is much less susceptible to peer churn than the tree overlay. P2P offers some level of scalability, but there exist some drawbacks for users. There may be requirements to download and install specific software. In most cases, users have to keep some ports open to allow access beyond firewalls and NAT which could pose a security risk [102].

2.3.3.3 Hyper Text Transfer Protocol (HTTP) Video Streaming

The HTTP protocol is used to deliver video content in today's video streaming services. Using HTTP for video streaming is simple in terms of setup because most firewalls support HTTP/HTTPS traffic, which eliminates the need for additional network configurations to handle video traffic [103]. HTTP Adaptive Streaming (HAS) [104] enables content providers to meet the needs of a wide range of devices and scenarios. The use of HTTP over TCP adds to the benefits of using this technology. HAS can be developed on top of existing content delivery systems for everyday Web use [105]. HAS divides a video file into a number of chunks. Each chunk is encoded at different video rates and stored with a file called

Media Presentation Description (MPD) as a description [106]. Unlike the traditional C/S architecture, which was based on a media server push-based video delivery method, with HAS, the clients pull media streams from the server. HAS treats media files just like regular Web content and delivers them in chunks over the HTTP. A streaming client continuously assesses and evaluates its own capabilities. The chunk with the greatest video rate that is sustainable given the estimated capacity is then requested. Adaptive Bitrate selection (ABR) [103] is the mechanism by which a client determines the profile and schedule of a chunk to download.

The studies in [27, 107] provided extensive classifications of ABR algorithms. Most of these previous studies agreed on dividing ABR algorithms into three categories based on their needed inputs: buffer-based (BBA [108], BOLA [109]), throughput-based (PANDA and CONVENTIONAL [110], FESTIVE [111]), and hybrid-buffer-throughput-based. ML and control techniques have shown much promise in ABR, leading to the design of ML-based [112] and control-based class [113] ABR algorithms.

Most HTTP-based video streaming solutions are based on HAS. Commercial solutions exist, such as Microsoft's Smooth Streaming [114], Adobe's HTTP Dynamic Streaming (HDS) [115], and Apple's HTTP Live Streaming (HLS) [116]. In a move toward ensuring an open standard for video streaming, the Moving Picture Experts Group (MPEG) in conjunction with 3rd Generation Partnership Project (3GPP), released MPEG Dynamic Adaptive Streaming over HTTP (MPEG-DASH) [106].

2.3.4 Common Challenges in Video Streaming

Even though switching from a server-push to a client-pull model has its advantages, HAS faces a few challenges, nonetheless. Some issues encountered in HAS systems include:

- HAS Multiplayer Competition and Stability: It is important for HAS clients not to switch bitrate frequently, as it leads to video stalls, which can negatively affect the

video quality. In a multiplayer HAS environment, it is crucial to achieve fairness. Clients competing for available bandwidth should equally share network resources based on their viewers, content, and device characteristics.

- Consistent Streaming Quality: The correlation between video bitrate and its perceptual quality has been shown to be non-linear by studies conducted on video quality analysis [117]. In general, it is preferable to stream videos at a consistent quality rather than at a consistent bitrate, which results in fewer oscillations in perceptual quality [118].

- Frequent Switches: Depending on the network condition and/or buffer status, the rate adaption algorithm switches video quality. While quality switching is a useful feature of HAS that helps to reduce the frequency of stalling occurrences, frequent quality switching may cause user frustration.

- Throughput: TCP throughput of nearly twice the video bitrate is necessary for effective streaming performance in general, which highlights a fundamental shortcoming of HAS applications [119].

HAS relies on ABR algorithms to dynamically select appropriate video bitrates in response to changing network conditions. To achieve this, it employs two control loops: (1) a TCP congestion control loop that reacts to network congestion by attempting to match the sending rate to the network's capacity; and (2) the ABR selection loop, which adjusts the video bitrate to match the average TCP throughput [120]. In general, the goal of the ABR algorithm is to avoid playback interruptions arising as a result of buffer depletion. The importance of a de-jitter buffer was investigated in [121] to overcome this challenge. This study provided some evidence showing the effect of video codec type and bitrate on video quality. The algorithm also attempts to maximize the quality of the video stream whilst minimizing the number of video quality fluctuations. It is imperative that the algorithm minimizes the startup delay time of the video stream. The main challenge for the ABR

designer and the research community is to strike a balance between these objectives to provide excellent video quality to end users. Although the HAS media display module has been standardized, the ABR adaptation logic is left to the developer's discretion.

The study in [122] lists the initial playback delay, the played out quality, the playback quality changes, and stalling as the key quality degradation factors that affect the video streaming session and impact the video quality. I introduce here the descriptions of these QoD-derived KPIs and some other quality influencing factors.

- Media Session: Media session refers to the start of video playback until the end. It includes the effects of initial loading times, rebuffering events, and switching quality, if applicable. This means that any of these events will cause the media session to be longer than the video/audiovisual playback time.

- Initial Delay: Initial delay is the duration between the video request by the client and the actual time when video playback commences. It is also referred to as initial buffering.

- Playback Quality Changes: It refers to the change in quality throughout the course of the video playback. It is also known as rate or quality adaptation.

- Quality Switching Frequency: The rate at which the quality changes during media playback is referred to as quality switching frequency.

- Stalling: This occurs when video playback is interrupted. In cases where the network throughput is insufficient for the content to be downloaded faster than it is consumed, the buffer depletes, and playback is forced to pause until more data are downloaded and the buffer is refilled.

- Rebuffering: This refers to cases when the data in the buffer are depleted, thereby leading to a video playback stalling. These events in a streaming session are normally represented by a spinning wheel, loading sign, or sometimes a frozen frame.

- Rebuffering Frequency or Ratio: The amount of rebuffering incidents per unit of time is referred to as the rebuffering frequency.
- Rebuffering duration: This is the total duration of all rebuffering incidents in a single media session.

2.4 Review of Video QoD Prediction via ML

Quantifying the effect of network performance on the video quality is critical, as this determines the success, degradation, or failure of a service. The following subsections discuss the applications of ML techniques for the prediction of the video quality from QoD measurements.

2.4.1 Video Quality Prediction Under QoD Impairments

Vega et al. in [123] investigated the prediction of video quality under QoD impairments. In their work on real-time quality assessment of video streaming applications, the authors proposed an unsupervised DL model based on Restricted Boltzmann machines (RBMs) [124]. Using only a subset of features extracted by a client involved in a streaming session, the proposed models were used to infer the no-reference features [125] of the received video stream. The features utilized in the study were mainly from the video-related features (such as the bit stream, frame, inter-frame, and content). In the study, the authors considered the effects of network QoD metrics on the streaming session. Two synthetic datasets (one for generic network conditions and another for extremely lossy networks) were used in the study. Although the authors presented results demonstrating the feasibility of their approach, there were a few issues in this approach. First, the QoD impairments studied (network delay, jitter, throughput) were treated as independent variables. This assumption is not a realistic one, given the way that events in networks cause these variables to have dependencies.

If a client device involved in a streaming session has to assess the video stream quality, it will require information on what impairments may exist already in order to select the appropriate predictor. Secondly, given that this technique is based on a multi-RBM solution, the computational costs of this method would need to be evaluated. This would help ascertain the practicality of using this method in real-time as proposed.

Accurate throughput prediction and bandwidth prediction have the potential to boost the performances of a wide range of applications. Adaptive multimedia streaming services, for example, can enhance a video stream session by making timely quality modification decisions based on these information. In [126], Raca et al. described how ML and DL techniques can be leveraged for accurate throughput prediction in video streaming applications running over cellular networks. The authors trained a throughput predictor using RF and SVM models. In addition, they compared the performance, training, and data requirements of these ML models with a DL model, the Long Short-Term Memory (LSTM) model. The predictor was trained to estimate the average throughput over a future time range by processing the historical data over a set period of time. In their setup, the collector module collected device and network specific information which was used during the predictor training stage for future throughput estimation. QoD data used included the average throughput of devices connected to a given cell and the load, i.e., the number of devices in the same cell. To evaluate the performance of the throughput predictor in a video streaming application, the authors considered a HAS video client connected to a server using a controlled link whose bandwidth was driven by a 4G trace. They computed the following metrics: bitrate, number of switches, mean count of bitrate switches, number of stalls, and the mean bitrate of selected video chunks. Their technique was compared with a base case scenario; the streaming performance parameters were computed without using the prediction module. In comparison to the traditional player's reactive actions (i.e., the base case), the throughput predictor-assisted player made excellent decisions that were future-aware. As a result, the throughput

predictor-assisted player eliminated stall situations by switching to a lower bit rate as soon as possible, but the traditional player failed to forecast abrupt dips in available bandwidth and chose higher bit rates. The authors also reported that the throughput predictor module reduced quality switching. Results showed improved performance when network-level data were used.

In a study related to that of Raca et al., the authors of [127] proposed ABR for Chunked Transfer Encoding (ACTE). They considered the problem of ensuring low-latency in live streaming applications. The proposed method relied on predicting the bandwidth of the applications and using the information in the selection of the appropriate bitrate setting for the client. In their investigation, they assumed that the client's choices would differ under different bandwidth conditions or impairments. The proposed design employed HTTP chunked transfer encoding instead of short video segments. This approach ensured that long segments can be created and sent in small bits, which are referred to as chunks [128]. The authors reported that ACTE realized accurate bandwidth measurement and prediction in low-latency streaming applications. To make ABR judgments, ACTE relies on three primary components: (1) a chunk-based sliding window moving average bandwidth measurement—this is similar to the baseline prediction approach used in [81]; (2) an adaptive recursive least squares (RLS)-based [129] bandwidth prediction module; and (3) a throughput-based bitrate selection logic. To evaluate the performance of ACTE, the authors utilized bandwidth measurements and packet-level data collected from the Twitch API. The authors measured the performance of their design by computing the average buffer occupancy, average number of switches, the average number of stalls and their average duration, and the average startup delay. When compared to ABR schemes that used segment-based bandwidth measurements, the reported results showed that ACTE achieved higher performance by reducing the number of stalls by 65% and their durations by 83%, maintaining a low latency of 2.3 to 3 s (36% reduction), providing a 28% higher bitrate. These results in [126, 127] are quite promising,

but given that the techniques' successes relied heavily on accurate and timely throughput prediction and bandwidth prediction, the ability to identify abrupt changes in throughput or bandwidth measurements is concerning. According to Cisco [130], the recommended network latency for video should be ≤150–300 ms. The latencies recorded here are much higher. The performances of these techniques across different channels and use cases warrants further investigation.

In [131], Mao et al. proposed Pensieve, a DRL model that makes future bitrate selections based on prior observations collected by the client video players. Pensieve is decribed as a Neural Network (NN) engine that learns a control policy for bitrate adaption entirely through experience without relying on any preset fixed rules. The model learns how to improve on past ABR decisions through reinforcements, in the form of reward signals that are measures of the video quality for previous decisions. To train the NN, the learning agent is fed the state inputs of the following: client playback buffer occupancy, previous bitrate decisions and network throughput measurements, and download time of the video chunks or the throughput interval times. These measurements are fed into the NN which then outputs an action, i.e., the bitrate selection for the next chunk. The authors reported that the ABR RL agent was able to identify policies that outperformed algorithms baseline methods that relied on fixed heuristics or employed erroneous system models. According to the authors, their method recorded this performance gain by optimizing its control policy based on the actual performance of previous selections.

Some related studies that investigated the use of RL as a data-driven technique to automatically optimize ABR algorithms include [132, 133]. The authors of [133] adopted a similar approach as that described in the seminal work reported in [131]. According to the authors, their approach was able to account for latency issues, which were some limitations they observed in the Pensieve RL agent. They also claimed to have improved the performance of the streaming session by employing an ensemble model of two NN models to improve

ABR decisions and enforce latency control. QoD metrics used in the evaluation included the average throughput for about 50 minutes, at a 0.5 second granularity. Similarly, the authors of [132] proposed ABRL, an RL-based ABR module in Facebook's production Web-based video platform. A limitation of their approach is that the authors reported experimental results based on Web-based videos, well-connected traffic patterns. This evaluation may need to consider more generic conditions, network variability issues, and cellular networks to evaluate the RL agent's performance in those real network conditions.

In a related effort, Hiba et al. in [112] demonstrated how ML approaches, namely, supervised classification, can predict the ABR algorithm's class by estimating the most important features (feature importance based on mean decrease in impurity). The authors posit that their technique can forecast any ABR algorithm's bitrate decisions, providing a set of input features that may be seen at the application level. The authors formulated the ABR bitrate prediction problem as a multiclass classification problem. In keeping with their goal of designing an ABR agnostic technique for bitrate prediction, the authors used a set of generic features which did not require knowledge of the ABR design logic beforehand. Some QoD input features used include the buffer level, bandwidth, previous bandwidth level during the last download, download time, and previous bitrate. They tested their approach on some well-known ABR algorithms and reported experimental results on commercial closed-source players, using realistic VoD and live datasets. A range of ML algorithms were evaluated—namely: logistic regression [134], SVM, RF, DT, Adaptive Boosting (AdaBoost) [135], Gradient Boosting (GB) [136], Naive Bayes (NB) [137], and k-Nearest Neighbors (kNN) [138]. The results reported showed that the RF and GB models attained the highest prediction accuracies.

In [139], Yusuf et al. proposed SMASH: a supervised machine learning method for adaptive streaming over HTTP. This work, like Hiba et al's work, was an attempt to develop a more generalized method to ABR selection. The authors presented a scenario where the

streaming client would adapt to a range ABR algorithms in response to changes in network conditions irrespective of the transmission context. The authors generated their dataset by streaming a mix of popular ABR algorithms across a range of different network settings (3G, 4G, and WiFi). During the streaming session, they logged in 22 features which included the buffer level, codec type, arrival and download time of video segments, segment duration, VFR, and average bitrates of the previous two or five chunks. Features used for model training were selected for the evaluation by considering the correlation matrix and some statistical information of the features set. Several ML models were used to evaluate the performance of SMASH. The models used include LR, Quadratic Discriminant Analysis (QDA) [140], kNN, DTs, NB, AdaBoost classifier, RFs, and Multi-Layered Perceptron (MLP) [141]. The model with the best performance in terms of classification accuracy, RF, was then selected for comparison with some state-of-the-art techniques. The authors reported some promising results, with SMASH outperforming all other approaches in all network conditions evaluated.

Most of the studies surveyed in this section adopted the RF algorithm for their experimental evaluations. The authors based this decision on the model's ability to reduce overfitting (via variance reduction) by averaging over several trees compared to other ML models used. In some cases, the GB model improved the accuracies achieved with the RF model. The GB algorithm improves the RF algorithm by modeling the performance of the intermediate trees. GB's performance is boosted by sequentially and repeatedly fitting trees to residuals of the model, correcting errors resulting from previously trained trees with each successive new tree. The review showed that RL models are the current best ML candidates for ABR algorithm optimization. The RL seems well suited to the settings for ABR corrective actions owing to the ability for the RL agent to adaptively refine their behavior in response to the environment. For most of these studies, the RL learner interacted with the learning framework in the presence of degrading QoD, such as throughput fluctuations. This seems to be an obvious

choice owing to the need to take corrective or remedial actions at the client device when the network conditions change.

2.4.2 Prediction of Video Quality from Encrypted Video Streaming Traffic

The quality of the video stream playback at client devices can be correlated with the application QoD features, such as initial loading time (also known as startup delay or join time in the literature), frequency of rebuffering/stalling events, and playout quality (spatial resolution) [142]. However, since most video data are encrypted, network operators rarely have such information on the video traffic generated in their networks. As a result, one way to assess video streaming performance is to use network-level QoD features extracted from encrypted video traffic traces or independent network measurement tools run outside the video application data plane [143]. This is due to the fact that Deep Packet Inspection (DPI)-based technology for analyzing the video data is no longer a viable option [144]. Using this approach contradicts the end-to-end security policies of SSL, and therefore causes an unfortunate set of issues, as outlined in [145, 146]. Internet Service Providers (ISPs) must infer the video quality from network level measurements. This motivates the application of ML techniques to link the network-level QoD measurements to the video quality. In this section, a review of the use of ML techniques in predicting video quality with QoD measurements obtained from encrypted streaming video traffic is provided. The review begins by considering studies that focused on real-time prediction of the video quality. Thereafter, a survey of studies that predicted the video quality by utilizing the video session data is presented.

2.4.2.1 Real-Time Video Quality Prediction:

The authors of [147] proposed ML-based frameworks for real-time prediction of startup delay and resolution for encrypted streaming video services. The authors focused on the YouTube service. In their work on video quality classification, the authors defined three classes: *high, medium, low.* To generate data, the authors ran 39 experiments with varying levels of bandwidth. Video metadata, such as video duration, view like, and dislike count, were included in the dataset. Network-level QoD features captured in the dataset include throughput, packet count, inter-arrivals, and byte counts. They started with a dataset of 54 features, which they divided into six categories: packet length statistics, size of transferred data in 5-second intervals, packet count statistics, inter-arrival time statistics, throughput statistics, and TCP flags count. The authors reduced the feature set number to 33, mostly due to redundancy. They defined two functions which analyzed an instance of a video streaming session, quantifying measured degradation and classifying the session as high or low. If it fell into neither of these categories, it was assigned to the medium category. The authors evaluated different ML models in the classification task; models used included One Rule (OneR) [148], Sequential Minimal Optimization (SMO) [149], J48 decision tree [150], RF, and NB. The RF and NB models were reported as having recorded the highest accuracies on the full dataset and the reduced dataset.

Dimopoulos et al. in [151] captured cellular network measurements and applied ML models to predict typical system performance metrics for streaming services (e.g., playing resolutions and stalling occurrences) using Round Trip Times (RTT), packet loss, and chunk sizes. The authors used a Web proxy in the network to collate data and infer the video quality of the data stream of YouTube sessions. Similarly to the study in [147], the authors solved a classification problem in which they leveraged the RF model in predicting the video quality. They reported high classification accuracies of approximately 94%. One of the paper's primary results was that changes in video segment size and inter-arrival periods were

among the most relevant markers of quality degradation. This is a reasonable assumption because as bitrate varies, so does the size and resolution.

YouTube video quality-relevant KPIs such as stalls, quality changes, and delays can be challenging to measure passively, especially on smartphones and cellular networks [152]. This is because of the difficulty in accessing application-level metrics directly on the YouTube application if the monitoring is done at the device level without root privileges. Considering this problem and the prevalence of end-to-end encryption in video streaming services, the study in [153] used ML and network-layer information to evaluate video-quality parameters (initial delay, stalling ratio or rebufferring rate, number of stalls, total stalling time) and user engagement for YouTube videos watched on smartphones. The authors extracted these network-layer features through the Android API. Simple metrics easily accessible through the Android APIs, such as the numbers of incoming and outgoing bytes, the signal strength, and the number of network switches, were some features extracted from end user Android handsets. The authors collected a dataset of 275 features in total. In general, these features included information about the received signal strength, the number of handovers, the number of network switches, and several statistics about incoming and outgoing traffic, aggregated over time windows of 1, 5, 10, 30, and 60 s intervals. They employed 10-fold cross-validation to evaluate a 10-tree RF model for each statistic [154]. To balance classes for learning purposes, the authors used simple bootstrapping approaches [155]. ML models were used to design predictors, which considered each problem as a classification task with discretized goals. They reported perfect performance in detecting the stalling or rebuffering ratio. Results reported showed that detecting the frequency of stalling events was the most challenging. However, the performance of the RF model was not compared to that of any other ML model. For this type of task, it would be beneficial to evaluate a range of ML models.

Wassermann et al. in [156] presented a ML-driven architecture used to predict YouTube video resolution in real-time using network packet-level data only. The features used in the

study included the numbers of uplink and downlink packets, and transmitted bytes. Others included time-based data such as the time from the start of the slot to the first packet, and the time between the first and last packets of the time slot. To achieve this prediction, the framework analyzed ongoing streaming sessions using 1-second time frames. The system employed a stream-based approach to calculate several lightweight, snapshot-like statistical characteristics from video traffic. It considered three windows (current, trend, and session). The system computed the trend features using a first sliding window which aggregated the latest three time slots. The session-progression features were computed using a sliding window averaging all past slots since the start of the session. At the end of each 1-second time slot, the properties of that slot, as well as those of the adjacent windows, were fed into ML models that predicted the video resolution (144p, 240p, 480p, 720p, or 1080p). The authors employed a diverse dataset of over 15,000 different YouTube videos recorded under varying network settings. The videos used in the work were streamed over an LTE mobile network, or a home or corporate WiFi network. The authors employed a 5-fold cross-validation to test nine ML algorithms: (1) DT, (2) RF with 10 trees (RF10), (3) AdaBoost using 50 trees, (4) an ensemble with 10 Extremely Randomized Trees (ERT10) [157], (5) bagging with 10 trees (BAGGING) [158], (6) NB, (7) kNN with $k = 5$, (8) a Feedforward Neural network (FFNs) with three hidden layers, and (9) an SVM. The results showed that all models correctly detected the 480p-class, with SVM being the least accurate with an accuracy of less than 70%. For the video resolution task, DT, RF10, ERT10, and BAGGING earned near-perfect scores. The authors claimed that this was due to the fact that about 50% of the time slots belonged to this video resolution. In terms of computational complexity and achieving real-time prediction of the video resolution, the authors reported the RF10 as the best ML model. This result is promising, because a model for real-time prediction tasks should be highly efficient in terms of computational time. However, due to the obvious bias towards the majority class (i.e., the 480p-class), ML algorithms are likely to produce good accuracy

scores on these datasets. As such, accuracy is not a very clear measure of performance for algorithms that work on imbalanced data, since the classifier will record poor results over the minority class [159].

In a related effort, Wassermann et al. in [160], considered the prediction of the average bitrate and the video resolution in real-time. Using the same dataset in [156], the authors evaluated the performances of two ML models, namely, RF10 and kNN with $k = 1$ and $k = 3$. The video resolution prediction task was posed as a classification task, and the bitrate prediction was considered as a regression problem. The results reported showed that the RF10 model outperformed the kNN model with an accuracy of 71%, against 66% accuracy recorded by the kNN with $k = 3$. For the bitrate prediction, the RF10 model outperformed the kNN. It was also reported that the RF10 model actually overestimated the actual bitrate for a large proportion of the time slots, ≈59%. The authors hypothesized that this may be advantageous for ISPs as it provides a method through which they can apply the appropriate QoS policies that can help ensure continuous playback of the video stream.

Gutterman et al. [161, 162] proposed Requet—a framework for real-time quality of experience metric detection for encrypted traffic. The authors focused on three metrics: buffer warning (low buffer, high buffer), video state (buffer increase, buffer decay, steady, stall), and video quality. The video state metric can be determined when the video level of the user is in a steady state. The metric also captures occurrences of buffer depletion and stall conditions. The system comprises a chunk detection algorithm, chunk feature extraction, and ML video quality prediction models. The chunk detection algorithm identifies video and audio chunks in encrypted traffic through IP headers. The audio or video chunk metrics recorded were: the protocol used to send the *get* request, start time, time to first byte (TTFB), download time, slack time, chunk duration, and chunk size. The features extracted from the chunks were then fed into ML algorithms for prediction. The authors gathered two datasets: (i) using a laptop running YouTube from a browser over WiFi, Browser-WiFi; (ii) using the

YouTube application on an Android smartphone over an LTE cellular network, App-LTE. In this work, these video quality metrics were predicted with a RF model. The results reported listed Requet's accuracies for predicting buffer warning, video state, and video resolution in the Browser-WiFi setting as 92.0%, 84.2%, and 66.9%, respectively. In the App-LTE setting, the accuracies for predicting buffer warning, video state, and video resolution were listed as 97.8%, 88.2%, and 80.6%, respectively. While these results are promising, given they considered real-time constraints of video services, there are a few limitations: (i) the evaluations considered only one service, the YouTube service; (ii) the technique was not tested on other services. The differences in ABR algorithms may introduce some dynamics that were not captured in the current design.

Seufert et al. in [163, 164] proposed ViCrypt, a system able to realize video quality predictions in real-time. The system was described as a ML-based approach for monitoring YouTube's video quality-relevant metrics in real-time. It was reported to be capable of predicting rebuffering events from such basic features. The system implemented with YouTube achieved these predictions by relying on constant memory stream-like inputs extracted from the encrypted stream of packets for an ongoing YouTube session. Firstly, ViCrypt utilized a sliding window which consisted of the last T time slots to compute trend features. A second sliding window consisted of all past time slots since the start of the streaming session to compute session-progression features. The features from within each time slot, along with the features from the corresponding windows, were fed into a RF model, which determined whether the current time slot of 1 second contained stalling or not. The approach is similar to that described in [156]. Input features utilized in the study included: the numbers of total uplink and downlink packets, count of bytes transferred, volume of TCP and UDP packets, TCP ratio, and UDP ratio, computed from the counts of the packets and byte counts. Other QoD measurements utilized in the study included, the average throughput of the slot (i.e., traffic volume divided by slot length) and the burst

throughput (i.e., traffic volume divided by burst duration). Initial results reported in [163] showed that the technique realized good predictions using all extracted features. In [164], the authors evaluated the relevance of different feature sets. By examining independent time slot stalling, they investigated which features were specifically relevant for accurate prediction of stalling. They investigated the effects of adding prior predictions (i.e., recurrent feature set) from past time slots on the prediction accuracy. Results showed that the full dataset was not required for real-time prediction of rebuffering events, as they realized similar performance using a reduced dataset. Their findings also indicated that by using the recurrent features, the prediction error was reduced. They did not investigate this further. However, it may desirable to do so and also consider a range of other streaming services.

The work in [165] presented BUFFEST. The authors considered the problem of classifying video streaming flows depending on the present buffer conditions of the clients. The authors asserted that the ability to estimate buffer conditions accurately is vital to understand how video streaming flows are experienced by users. Specifically, BUFFEST's focus was on detecting rebuffering events in YouTube. The authors emulated a player that resides on a client's Network Interface Card (NIC) (or wherever the proxies was located) and registered HTTP-level, TCP/IP-level, and stream metadata. Encoding rates, chunk boundaries, and other information generally found in metadata files were included in this data. The authors focused on features based on a simple one-pass metrics generated using only packet-level information to achieve fast processing. The performances of ML classifiers were compared with threshold-based classifiers. Two ML models were evaluated for this study: SVM and DT. Given the reported throughput measurements (using real and synthetic data) over various time periods, the classification problem was to detect whether or not a playback stall would occur. The DT model recorded a better performance than the SVM model. Based on their findings, the authors hypothesized that even if the video data stream was encrypted, network operators could distinguish a significant fraction of low buffer instances.

Mazhar et al. in [166] leveraged network and transport-layer measurements as input features to train ML classifiers for predicting startup delay and rebuffering events in encrypted video data. They considered encrypted video traffic streamed over HTTPS and QUIC, which runs over UDP. Information in IP headers were used to determine network-layer features. TCP/UDP header information were used to extract transport-layer features. Some network-level measurements used in the study included: packet counts, byte counts, throughput, packet inter-arrival times, packet sizes in bytes. For the classification tasks, the authors employed DT classification algorithms. The tree-based ML models achieved up to 90% classification accuracy for HTTPS and up to 85% classification accuracy for QUIC in the experimental evaluations. Given the diversity of ML algorithms, the study requires further investigation. The works described in [165, 166] proposed models that operate in controlled environments for certain services; the ideas could be advanced by exploring these techniques in real-world network deployment scenarios and to a broader range of services.

2.4.2.2 Session-level Video Quality Prediction:

Bronzino et al. in [167] (which builds upon the works in [165, 166]), extended the framework to account for a wider variety of services. The authors proposed models that work in deployment scenarios where video sessions and segments could be detected from a mix of traffic. The gathered traffic statistics have a coarser time precision. The authors devised a single composite model that could be used for a range of platforms (e.g., Netflix, YouTube, Amazon, and Twitch), rather than just one. They created a complete labeled set with over 13,000 video sessions for the four video services. Using network, transport, and application-layer measurements as inputs, the study investigated the feasibility of designing ML models that could predict startup delay and resolution. Some QoD features used in the study included: throughput, average packet count, average downstream throughput difference between consecutive time slots, and packet inter-arrival times. For the task of

predicting the startup delay, the authors evaluated different regression methods, including: LR, RR, SVR, DT, and RF regressors. Using the average absolute error as a performance metric, the study reported that the RF model accounted for the lowest prediction errors. Similarly to [156], the authors trained ML classifiers with five classes, 240p, 360p, 480p, 720p, and 1080p, for the video resolution. The RF was reported as having achieved the best performance in terms of precision and recall. For the composite models, the authors reported findings that suggested that models which included the network and application-layer features outperformed models that depended solely on network and transport-layer features for the startup delay and resolution prediction tasks. For most video sessions data used in the study, startup delay models achieved less than one second error; the average precision of resolution models recorded was above 0.93.

In a related effort, the authors of [168] considered a composite model capable of predicting video quality for the popular VoD services (Netflix, Amazon Prime and YouTube). They extracted network-related features from streaming sessions running over the VoD services based on the streaming patterns and characteristics. Using the extracted features, they trained a single-layer perceptron NN. QoD features used in the study included packet size, port information, bandwidth, resolution, rebuffering time, and bitrate data. Other features used included block size, buffering phase, standard deviation of duration, and progressive download ratio. Results reported indicated that the NN achieved prediction accuracies of 0.929, 0.857, and 0.9333 for YouTube, Amazon and Netflix, respectively. It is not clear if the authors evaluated the NN model using trained models from another service—for example, by evaluating the NN on Netflix data using a YouTube trained NN model or a dataset containing all the services.

Schwarzmann et al. in [169] investigated session-level MOS prediction using ML regression models based on 5G monitoring data. Using QoD statistics from traces generated within an OMNeT++ simulation, the authors tested the feasibility of their technique. QoD measure-

ments used in the study included throughput measurements (access node, user equipment (UE)), downlink channel quality indicator (CQI), uplink CQI measured at UE, RTT measured at the UE, and smoothed RTT computed with a moving average. Two ML regression models, LR and SVR, were evaluated in the study. Results reported indicated that subjective QoE scores could be reliably predicted, solely from network-QoD monitoring data. It was also shown that a limited set of features could lead to a reasonable accuracy in predictions. However, the practicality of this approach could be advanced by testing with real 5G network traces.

As the traffic characteristics of realistic video sessions monitored in the network are inherently affected by end user behavior [170] (e.g., seeking, pausing, and abandoning), Bartolec et al. in [171] investigated these events in their work on video quality KPI classification. The research examined the impact of user interactions on video KPI classification accuracy (resolution, initial delay, video bitrate). Using YouTube as the case study, the authors trained models on datasets that included and excluded user interactions. Two datasets containing IP-level traffic features were used in the study. A dataset containing 299 videos (without user interaction) and another comprising 307 videos (with user interactions) were utilized in training the RF model adopted in the work. The authors compared the performance of their models with ground-truth data from YouTube Stats for Nerds. The Stats for Nerds option displays some video meta-data regarding the stream playback. Results showed that the performance of models trained on datasets not including user interactions were worse when applied to the interactive dataset. The system performance will need to be evaluated with more realistic dataset and under additional scenarios such as a user-initiated quality switch.

In [172, 173], Orsolic et al. investigated session-level MOS and video KPI classification using IP-level traffic features. The work in [172] used ML models namely, OneR, J48, and RF, in predicting the video resolution, stalling, startup delay, and bitrates. Two datasets

were collected in laboratory emulated environments. A total of 394 videos were collected on Android and 383 videos on iOS. Two additional datasets were collected from mobile networks running in iOS. To compare their prediction with ground-truth data, the authors collected data from Stats from Nerds. QoD-level measurements used in the study included: average throughput and average packet size. The initial results reported in the work demonstrated promising applicability; however, further tests using datasets with different quality degradation scenarios maybe required. The study in [173] considered the same targets and used similar QoD inputs in the prediction tasks. In this study, four datasets were used. With the data collected on a laboratory WiFi network, the authors trained ML models to classify the video quality in terms of a range of video streaming KPI metrics (bitrate, resolution, stall events, initial delay) on the iOS platform. The authors used the same ML models: OneR, RF, and J48 (DT). The results reported in the study showed that the prediction accuracy, for all classification targets, ranged from 70 to 90%.

This review found that the input data which were used to train the models varied significantly across different applications, and among different usage scenarios within a single service. For most of the described scenarios, extracting ground-truth data from the application was not feasible. For instance, while YouTube provides video performance metrics in its Stats for Nerds window on desktop and mobile applications, the case is different for services such as Twitch and Netflix. These services only offer video performance reports in the browser. There are differences in datasets, methodology, validation setups, and ML models for the same target objectives in these published papers. However, the features and labels to be extracted from the collected data should be based on the desired objectives. If the objective is to achieve real-time network management, the ground-truth data used for model training should be representative of the conditions that need to be identified. For example, in order to predict instantaneous stall conditions in network traffic, the model must be trained on short time-windows of network traffic. If the desired objective is to provision the network

to meet the time-varying needs of the streaming population, estimating the video quality on a session-level might be more suitable. What is more challenging is identifying features in the network that correlate with the target video quality KPIs. The authors of [77] highlighted some network-level measurements appropriate to detecting certain video quality KPIs, such the startup delay, stalling events, resolution, and quality switches.

From the reviews conducted in this section, the survey found that the current best ML candidate for these settings is the RF algorithm. The RF works well for both regression and classification tasks. The majority of works surveyed in this section reported the RF as the sole ML model or evaluated the RF along with a range of other ML algorithms. The RF seems to be an attractive choice in these settings due to the fact that its default parameters produce good prediction results without much tuning. The hyperparameters are very easy to understand, and there are not a lot of them. The RF is a great choice when dealing with high dimensional data. This set of algorithms is a good candidate when handling unbalanced data. This is because the model attempts to minimize the overall error rate by assigning a low error rate to the larger class, and the smaller class is assigned a larger error rate. However, when dealing with large datasets, the RF may take up a large amount of memory, making it quite resource intensive. In such cases, the DT may offer a lightweight approach for in-network computations at the expense of performance.

2.4.3 QoD Prediction for HAS and DASH

Following the widespread deployment of HAS, a lot of interest has been generated in the area. Research in this area has included some data-driven approaches, deployed with ML models for the prediction of network QoD changes. These studies investigated predictions of network metrics—namely, the throughput and trigger adaptation mechanism aimed at reducing rebuffering at the video client players [174] and selection of the appropriate adaptation action in HAS [175].

Sun et al. proposed cross section stateful predictor (CS2P) to enhance bitrate selection and adaptation in HAS clients in [174]. They made three contributions in their paper. The authors analyzed the throughput characteristics using a dataset sourced from a Chinese video provider, which comprised 20 million sessions covering three million unique client IPs, 8 server IPs, and 87 ISPs. They concluded from the analysis that sessions sharing similar key features (ISP, geographical region, etc.) have similarities in network-layer throughput values and dynamic patterns. They also alluded that there was also an inherent natural stateful presence in throughput variability within a session. However, the relationship between session features and throughput is rather a complex one. A video client's perceived throughput is influenced by multiple factors, such as the load on the server [176], network congestion, and last-mile link technology, which implies that clients sharing one of these features do not have similarities in throughput. A second contribution in the paper was the implementation of CS2P using a Hidden Markov Model (HMM) [177]. The authors used the HMM to model the state transition evolution of the throughput, one model per session cluster, where sessions were clustered by similar characteristics, as mentioned above (e.g., ISP and region). Finally, the authors integrated CS2P in a dash.js client. They demonstrated that it achieved accurate throughput prediction in a real world environment. Using the above-mentioned dataset, the HMM model was trained offline using the EM algorithm, and 4-fold cross validation was applied to tune the number of states. The authors reported that CS2P achieved ≈40% and 50% improvements over existing predictive approaches in terms of initial and midstream throughput prediction error. They compared their results with the Model Predictive Control (MPC) [113], and reported that CS2P achieved an improvement of 3.2% in overall video quality, and 10.9% higher average bitrate compared to state-of-the-art MPC models which used the harmonic mean for throughput prediction.

The authors of [178] described a ML framework that used a range of regression models to predict the video quality of a streaming session. In their approach, each streaming session

was defined by five statistics: the average of video segment quality values, the time over which segment quality decreases occurred, the time since the last impairment event, the total stalling duration, and the number of stalling events. The authors adopted three regression models: LR; SVR; and an ensemble approach using RF, GB, and extra trees regression. SVR was determined to have the best average prediction performance when using a dataset of 112 sessions with a length of about 72 s. However, these streaming session measurements do not adequately reflect the magnitude of quality degradation. This is because the temporal relationships between the impairment events are normally lost. This sort of predictor could be integrated with practical video-quality-aware decision algorithms. The use of such systems could aid in intelligent network resource allocation, thereby increasing the video quality and leading to a reduction in the costs incurred by streaming providers and content delivery networks.

A related study which considered the prediction of video quality for HAS streaming sessions was proposed in [179]. In this study, the authors presented a ML approach for predicting the overall quality of HAS sessions. The authors adopted LSTM networks [180] in the study. The authors made the case that LSTMs can utilize memory to explore temporal relationships between QoD impairment events. In the proposed technique, inputs were taken on a segment-by-segment basis rather than on a session-by-session basis. The authors evaluated their proposed technique over a range of datasets using two different types of LSTM, namely, basic and advanced. Segment quality, content qualities, stalling duration, and padding were some features considered in the study. The authors reported high performance for the LSTM networks in the overall quality prediction of HAS. However, this came at the expense of the method being computationally expensive. Although, the study reported promising results for all use cases considered, the scalability of this technique due to its high computational cost poses a challenge for practical large-scale deployments.

A hybrid-DL architecture for proactive prediction of the video quality from multivariate time-series data was proposed in [181]. The authors hypothesized that using QoD and some video KPI information, their DL framework could predict the client video quality at the next time step before the event occurred. The authors evaluated their technique using data from an industry video streaming testbed for three different scenarios: congestion-free, congestion in the client network, and congested ISP network. QoD features used in the experimental evaluations included buffering length and frequency, and bitrate data. To evaluate the proposed approach, the authors employed a hybrid model comprising a Bidirectional LSTM (BiLSTM) and a Convolutional Neural Network (CNN), which they called BiLSTM-CNN. The proposed hybrid model was compared with some other ML algorithms, such as SVR, and some DL models, such as MLP, LSTM, and BiLSTM. The results showed that the proposed hybrid model, BiLSTM-CNN, outperformed the other models. The SVR had the worst performance, followed by the MLP. Similar results were achieved by the BiLSTM and the LSTM. These results are promising, as they could enable network managers initiate remedial or proactive actions in advance of the occurrence of failures.

2.4.4 Software-Defined Networking (SDN)

The heterogeneous nature of today's networks increases the complexity of networks and presents a number of challenges in effectively organizing, managing, and optimizing network resources. SDN is a networking architecture that decouples the control plane from the data plane [182]. The separation of concerns in SDN gives more flexibility to the network. Network resources in SDN are centrally managed by a logical controller. In addition to this, SDN has more advantages over traditional networks, including network automation, reduced network complexity, and more agility within the network.

Owing to the centralized role of the SDN controller, it can monitor and gather real-time network state, flow statistics (e.g., throughput, packet loss), and configuration data. These

features make ML algorithms appealing for SDN. The work in [183] is a recent survey of ML deployments in SDN.

Network operators rely on QoD metrics such as packet loss ratio, delay, jitter, and network throughput to assess network performance. These QoD metrics correspond to the network KPIs—for instance, transmission rates, queue length, etc.—upon which service-level agreements are made. Quantifying the relationships between these QoD metrics and the KPIs can help improve QoD management in SDNs by offering QoD prediction ability in relation to the KPIs. SDN's centralized architecture makes it possible to gather network statistics from the switches at per port and per flow granularity levels, based on which ML can be utilized to realize QoD predictions [183]. Josep et al. in [184] evaluated the performances of two models to characterize the network delay in an SDN network given the network load and the overlay routing policy. The authors considered a traditional $M/M/1$-inspired ML regressor network model and a NN model in their delay estimation study. Their experimental simulation results showed that the NN estimator performed better than the $M/M/1$-based delay estimator. However, both models treated the network as a *black box* without considering the network dynamics within the system. In order to quantify the effectiveness of these delay estimators in a streaming infrastructure, it would be beneficial to model dynamicity in the network. This could include the effects of concurrent users or interfering loads on the system, the variance in the delay, routing changes, congestion, etc. In addition, a more-detailed description of the trained NN model would aid better understanding and comparison of the approach in relation to NN models.

The authors of [185] proposed a two-step systematic technique towards QoD improvements in an SDN. They gathered a large number of metrics from virtualized and real environments. They then applied ML algorithms to automatically discover a formulae that could quantify the relationships between the network KPIs and the QoD metrics. They first used DTs to determine the correlations between the network KPIs and the QoD parameters.

Thereafter, using a LR ML model, they performed root cause analysis to uncover each KPI's quantitative impact. The proposed approach could also be used to predict traffic congestion and proffer recommendations on QoD improvements to the SDN controller [186]. However, because of a lack of uniformity in networks, rules in one may not work in another. It is therefore pertinent that the network operator adopts this approach and uncovers the correlations or rules as they relate to the network.

The work described in [187] by Rafael et al. focused on application-aware QoD predictions. The work described the use of two ML techniques, namely, RF and regression trees, to predict two QoD parameters: VFR and response time. The authors collected device (or kernel) metrics from a VoD server. Additionally, they collected statistics from the switches on a per port and flow granularity levels in the SDN. To reduce the computational cost, the authors employed subset selection to reduce the feature set size while preserving a low level of QoD prediction error. The authors reported an application-aware QoD prediction accuracy of over 90%. However, given that the setup for this was similar to [21], as was the feature set, incorporating the system load [51] would have better modeled the problem, as demonstrated by the work in [21]. The study in [188] utilized regression ML models to predict mean opinion score (MOS). The authors estimated the MOS value from the QoD parameters such as RTT, jitter, link bandwidth, and delay. The authors reported that the SDN controller was able to dynamically adjust the video parameters (e.g., screen resolution, frames per second, and bitrate) to improve the video quality received at the client devices. This evaluation was carried out using synthetic data, and not a lot of details were provided on the ML models.

To prevent the occurence of video freezes for HAS clients, Petrangeli et al. in [189] proposed an SDN-based approach, where intermediate network elements support video delivery. The authors presented a ML-based framework designed for helping clients avoid video freezes due to network congestion. The ML engine is based on the Random Undersampling Boosting (RUSBoost) algorithm [190] and fuzzy logic. In the proposed framework, an

SDN controller used an ML engine to predict when a HAS client would experience a video freeze so that it could determine if the currently downloaded segment should be prioritized. The decision was made using measurements collected only from network nodes, with no additional input from clients required. The ML freeze predictor used QoD input parameters such as the following: (i) the bandwidth for HAS traffic upon segment request; (ii) the difference in HAS bandwidth between two consecutive samples; (iii) the inter-arrival time between consecutive *get* requests. Other input features used included the quality level for the segment requested (expressed from 0 to q_{max}, the maximum available), and the difference in segment quality between consecutive requests. The authors collected 4500 clients' logs, which amounted to about 1.5 million individual segment requests, with 2.5% being affected by video freeze. They trained the predictor using 85% of the training data and set apart the remaining 15% as the validation dataset.

The performance of the RUSBoost algorithm was compared wit RF, AdaBoost, GB, and 1-nearest neighborhood (1-NN). The authors reported findings which showed that the RUSBoost algorithm outperformed the other classifiers with classification accuracies of 99% and 85% for the training data and validation set, respectively. Results showed that the RUSBoost algorithm could detect if a client was close to freezing, and fuzzy logic confirmed if the queue was good enough to successfully prioritize video segments.

2.4.5 Predicting the Video Quality in Wireless Settings

Video streaming quality can degrade in wireless settings due to performance issues. In such environments, there are various performance indicators, such as link speed, signal strength, interference, medium availability, and latency. This section provides a review of the literature that investigated the prediction of the video quality based on network performance parameters, specifically in Wi-Fi networks.

Hora et al. [191] measured the frequency and duration of rebuffering events, join time, and resolution switches of the YouTube application in an attempt to predict the MOS. The authors collected QoD parameters such as the transmit physical rate, total number of frames sent and retransmitted to station, Received Signal Strength Indicator (RSSI), and medium busy time. They used existing models which predicted the YouTube application's quality as a function of rebuffering events and video bitrate as proposed in [192]. Similarly, for video bitrate changes, they used SSIM to estimate the effect of video resolution on the streamed video quality as proposed in [193]. During the training phase, the authors employed stepwise feature engineering to incrementally increase the number of features used to a maximum of 36 features. They trained an SVR algorithm to predict the video quality and reported a Root Mean Squared Error (RMSE) of 0.60 – 0.79.

Ligata et al. [194] defined four binary classes of the video quality in terms of video streaming buffer ratio, initial buffer delay, rebuffering frequency, and rebuffering duration. In the study, QoD parameters considered were signal strength (SS), retransmission ratio, error ratio, activity factor, channel utilization, and exposed node load. To determine which QoD parameters correlate with the four Wi-Fi problems, such as coverage, overload, contention, and interference, the researchers used Pearson correlation coefficients and selected the features that had the highest correlations. The authors then trained a RF algorithm to predict the impact of these QoD metrics on the binary classes. They reported prediction accuracies of 85–95%.

For priority queue management, clients can measure or report the video quality directly to the Access Point (AP) and controller. FlowBazaar system was presented by Bhattacharyya et al. in [195]. The system leveraged middleware on the client side which sampled the video state, and mapped this state to a Delivery Quality Score (DQS). Application performance metrics considered by DQS included stall duration and number of stalls during streaming. Clients then requested bids in the range of [0–5], with the top two bidders being selected for

the queue assignment, and the third highest bid for other clients being charged. Using the OpenFlow protocol, the controller made policy decisions (such as allocating flows to queues) that were communicated to the AP. At the AP, the statistics collection component collected QoD statistics such as throughput, drop rates, and RTT. These measurements were then sent to the controller in a predefined message format using the OpenFlow protocol. The authors used RL to select which queue should receive clients based on network latency, packet loss, and throughput for the next period of time. The auction-based approach achieved DQS 5 for almost 85–90% of clients.

2.4.6 Predicting Video Quality in WebRTC

The authors of [196] proposed a method to address the problem service providers face in monitoring streaming quality of WebRTC-based audiovisual communication services. They examined the use of ML models in identifying root causes of video quality problems by extracting features from various application-layer performance statistics. The target features considered in the prediction tasks were video blockiness, audio distortion, and identification of the root causes of impairments. They authors employed six ML models in these classification tasks, namely: DT, RF, NB, SMO, kNN, and bagging. QoD input features used in the study included the frame rate, packet loss count, jitter, jitter buffer size, and bitrate. The DT model achieved high accuracy when detecting all three targets. According to the authors, due to the high interpretability of DTs and space constraints, they reported only results with the DT models. Nevertheless, they reported that the other models had comparable results.

In a related effort, the authors of [197] conducted evaluations using WebRTC measurements from a Wi-Fi network. Using QoD metrics such as Wi-Fi RTT, link quality, and RSSI, the authors employed ML models to predict whether the video quality would be acceptable during the next time window. They focused on video freezing events. ML models used in the

study were DTs, RFs, SVMs, and extra trees classifier. Results reported showed that the RF outperformed the other models. These techniques aim to produce self-healing frameworks where the system can adjust its servicing strategy. The results can be improved by adopting multi-dimensional models and by incorporating a broader range of realistic and perceptible audio-visual impairments into the evaluations.

2.5 Conclusions

This chapter presented a survey of the applications of ML techniques for video quality prediction with QoD metrics. The survey covered the video QoD prediction via ML in QoD degrading conditions, encrypted video stream traffic, HAS video services, SDN video streaming, video streaming over wireless networks, and WebRTC video streaming applications. It is of paramount importance for service providers and network operators to develop reliable models that are capable of monitoring, predicting, and even controlling the video quality in order to satisfy the ever-increasing demand for video services. The review found that there are a number of research efforts that are looking at video quality prediction from QoD measurements for encrypted video traffic. The lack of visibility caused by encryption and digital rights management drives the need for ML-based QoD prediction. Furthermore, the survey showed that ML models are increasingly being used to predict video quality under poor QoD conditions. Several studies evaluated similar QoD video quality KPIs, such as rebuffering, quality switching, video resolution, and startup delay. For the application designers, the media player buffer design is crucial, because rebuffering impacts the quality of the video stream. The buffer size affects startup delay and rebuffering time [198]. Most players are typically designed to buffer a predetermined number of segments, before initiating playback. With 1 and 2 second segments, this has been shown to be fine, and it provides under 10 second latency if there are no more than 3 segments being buffered [199].

Generally, it is envisaged that to design accurate ML models that can predict video quality from QoD statistics, the models should be reliable given the QoD metrics and all externally influencing factors. For instance, there is evidence that models which assume independency among QoD variables may be inaccurate, since, for example, bitrate and buffering are correlated [200]. In order to perform real-time video QoD via ML, the models must have acceptable computational complexities and storage requirements. Finally, to aid with the scalability of such proposals, the models should be able to readily take new variables into account as the network evolves over time.

Chapter 3

Predicting Video QoD with the Load-Adjusted Learning Strategy

THIS chapter presents techniques that model the dynamic effects of the time-varying workloads present in a video delivery system, along with methods for adjusting the regularization penalties for ML regression models in an adaptive manner. This technique for compensating for loads in a video delivery system is referred to as the Load-Adjusted learning model. The LA technique has been proposed in literature as an efficient technique for video quality prediction in the presence of interfering user sessions in a video streaming session. Regression models, namely LR and RR, have been used in previous LA model research to learn the mappings between a cluster of servers delivering video streams to client devices. Results reported in these studies demonstrated the superior performance of the LA learning technique over the Un-Adjusted (UA) techniques and showed that the LA method is computationally cheaper. The UA learning techniques do not model the effects of the interfering users or loads present in the streaming session. This thesis chapter describes empirical studies undertaken to extend the scope of the LA technique. This chapter builds on earlier LA contributions by conducting experimental simulations that

(1) examine the performance of different learning functions using a range of ML models; (2) investigate the efficacy of the LA learning for different video QoD metrics, and (3) investigate the performance of these learning functions for different QoD metrics under varying network conditions. This chapter contributes novel and adaptive methods using the EN ML model that enables automation of the choice of the regression parameter. The use of subset selection as a means of speeding up the computation process has been considered in previous research studies based on the UA technique. This thesis chapter investigates whether subset selection, the UA technique, or LA learning improves the condition number of a variety of learning algorithms. The findings of this reveal that subset selection alone reduced the effectiveness of the prediction. The chapter also presents empirical studies and results showing the performance gains achieved with the LA learning strategies over the UA. The contributions proposed in this chapter can be applied within the video service provider network infrastructure, edge locations and content delivery networks.

3.1 Problem Statement and Motivation

3.1.1 Problem Statement

Online videos can be accessed, downloaded, and viewed anytime, anywhere due to widespread internet connectivity. Users can choose from a wide range of services, including web-based video services like YouTube, Netflix and Internet Protocol Television (IPTV) and VoD. The rapid growth of IP traffic resulting from video and the growing number of internet users worldwide necessitates that network performance requirements increase in tandem. This is due to the fact that the surge in video-driven data traffic will put pressure on network and service providers' ability to deliver high-quality video to global consumers who seek improved QoD. Video delivery over dynamic IP-based network infrastructures may suffer from rebuffering events, delayed playback, frequent quality switches, packet loss, delay,

and jitter, reducing the audio-visual quality and service quality expected by users. For the network and service providers, it is important to be able to automatically and accurately predict the quality of video content being delivered in order to remain competitive. The ability to improve learning algorithms without increasing their computational costs is important for the cloud services and applications community since consumers differentiate products based on their ability to respond to poor network QoD [201].

3.1.2 Motivation

Cloud-based video delivery services are particularly difficult to model due to their dynamic nature. Such services are also hard to predict. It is therefore imperative that learning algorithms deployed in these types of settings are computationally cheap and capable of facilitating timely network management [202]. The authors of [8] investigated video QoD prediction for cloud-based video delivery services. They applied a range of ML techniques for client-side video QoD prediction. The ML aglorithms used in the study were LR, RF, RR and LASSO. The authors made their dataset public. Using their approach as a baseline and with the publicly available dataset, this thesis chapter contributes adaptive learning techniques that have lower computational complexity and yield more accurate predictions.

The main contributions of this chapter can be summarized as follows: (1) this thesis chapter considers the role of regularization via the EN [80] which automates the choice of the regression parameter; (2) this chapter demonstrates the efficacy of the LA technique, originally proposed in [9] using the EN and other shrinkage methods such as RR and LASSO. The LA learning technique, uses the TCP socket count (this feature is an indicator of the number of users or load in the system) feature to improve the prediction algorithm's performance; (3) this thesis chapter advances the scope of the LA technique by contributing LA learning models with non-linear ML algorithms; a comparison of the performance of these non-linear LA models with the LA linear models is also presented; (4) the authors

of [8] relied on subset selection to record performance gains in their study, reduce the feature space and to speed up the computation time. The thesis investigates whether subset selection achieves performance gains over the LA or not; and finally, (5) this chapter results demonstrate the performance gains in prediction accuracy achieved by the LA technique over the UA approach under a range of network test conditions. Finally, the thesis compares these LA performance gains recorded by the linear and non-linear ML algorithms.

The rest of the chapter is organized as follows. Section 3.2 presents a review of some past works on client video QoD prediction. Section 3.3 describes the LA and UA techniques. Section 3.4 provides some background on the ML techniques used in this chapter. In Section 3.5, a description of the experimental evaluation of the proposed models is presented. This section outlines the model fitting process, data preparation and evaluation of the LA and UA learning strategies. Section 3.6 presents results of the evaluations and analysis. Section 3.7 summarizes the LA implementation for cloud resource management. In Section 3.8, the thesis chapter highlights the advantages of the LA learning technique. Finally, Section 3.9 provides a conclusion of the thesis chapter.

3.2 Client Video QoD Prediction

Yanggratoke et al. in [8] proposed that by gathering thousands of video server kernel features, they could model and predict the video QoD of the client devices. In the study, the authors streamed video using VLC to a group of client devices from a networked cluster of video servers. During the streaming session, they collected kernel or device statistics from the streaming servers using the SAR function. Some examples of kernel metrics collated using SAR include the number of TCP sockets currently in use, number of active processes, etc. The authors by instrumenting the VLC player, collected some client-side QoD metrics such as the frame rate and packet counts of the video stream.

Fig. 3.1 depicts the system under study in this chapter which comprises of a cloud-based server infrastructure that serves dynamically varying client requests over a network. Resources are shared between multiple clients (RHS), with the video service (LHS) delivering video to clients. With a rapidly fluctuating number of users accessing the video stream, it can be challenging to predict the quality of the video stream received by the client devices. It is imperative that the streaming server infrastructure be capable of handling dynamic and time-varying requests since users may choose to start and stop the video streams at any time. The authors of [8] computed the feature set, x using the SAR function. This function extracts the selected aggregate activity counter values in an operating system for a specified period of time. Here, features are metrics about the server operating system, such as the number of running processes, TCP active connections or TCP Socket count (TCPSCK). The TCPSCK feature of x is a measure of the number of active clients, the load signal at time i. A load generator dynamically distributes client video requests to the servers using either a periodic-load pattern or a flashcrowd-load pattern. The Periodic-load patterns introduce clients following a Poisson process at an average rate of 30 per minute. This arrival rate is modulated by a sinusoidal function with a period of one hour and an amplitude of 20 clients. The flashcrowd-load pattern starts with a Poisson process where clients arrive at an average rate of 5 clients per minute, peaking at random events at a rate of 10 events per hour. Flash events see an increase in arrival rates to 50 clients per minute for about a minute and then gradually reduce to 5 clients per minute within 4 minutes.

VoD requests are handled by VLC media player which extracts information about the client video QoD metrics, y, such as the packet count, Video Frame Rate (VFR) and Audio Buffer Rate (ABR). The server device statistics, x in Fig. 3.1 are used to predict the client video QoD metrics, y, at clients A, B, C, and D. The objective of this thesis chapter is to predict the (i) VFR, the number of displayed video frames for any time i; (ii) ABR, audio

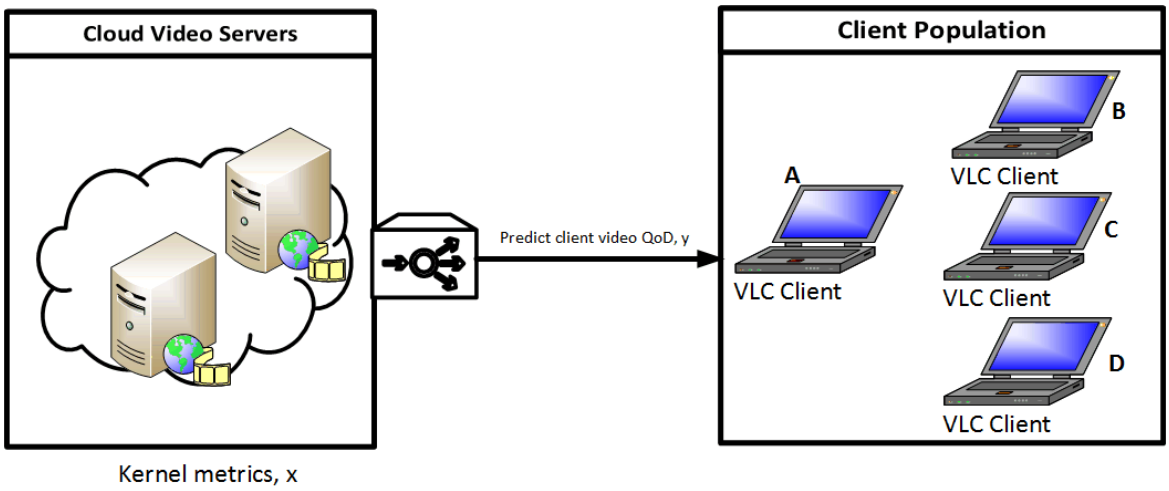

Fig. 3.1 This work investigates the prediction of the video QoD for client A, y, given a time-varying number of other clients B, C, D, etc. by using kernel metrics, x, collected from the video server in the cloud. Previous works have not considered the time-varying number of clients.

buffers played for time i; and (iii) packet count, the number of received packets in time i at clients A, B, C, and D.

Fig. 3.2 (R1) shows the packet count recorded during 15000 seconds along with the system load, i.e., the TCPSCK for the flashcrowd-load trace. The TCPSCK feature value plays a key role in determining the performance of the client video QoD metrics. The value of the TCPSCK feature increases with the load and may result in fewer packets being sent to the client since the system does not have unlimited resources. Other metrics show a similar trend. Fig. 3.2 (R2) illustrates a plot of the TCPSCK feature measured for a period of 15000 seconds along with the packet count for the periodic-load trace. The dependent relationship between the statistics is shown for both traces.

The LA technique was first proposed in [9] where the authors proposed a generative model for QoD and kernel-level metrics for video streaming services. In LA learning, ML models are trained conditional on the load signal. Its simplest form would imply learning ML models for each individual value of the load signal. This significantly speeds up the training time and is computationally cheap. In the UA learning, ML models are trained regardless of

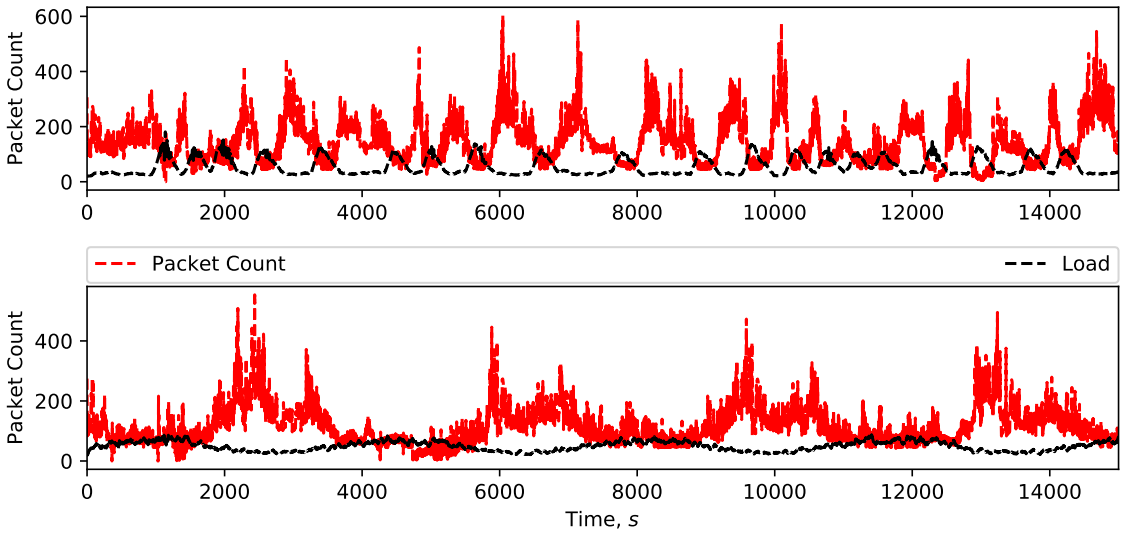

Fig. 3.2 Row 1 (R1): For 15000s, the flashcrowd-load trace packet count is shown alongside the TCPSCK, which is the system load. There is a periodic change in request patterns. An increase in the TCPSCK leads to a decrease in the packet count received at the client, as demonstrated in [9] for the first time. This shows the dependence of the packet count on the system load. Row 2 (R2): The packet count is illustrated for 15000s along with the TCPSCK for the periodic-load pattern. The dependency holds true for this trace too.

the load value. This learning mechanism fails to capture the effect a user streaming session may have on another in a shared network resource. Given that the load signal can be directly measured from the TCPSCK, the authors demonstrated the efficacy of predicting a QoD metric, video packet count using a LR model. They demonstrated that by relying on the dependency between the QoD metric and the load signal as illustrated in Fig. 3.2, the LA technique could achieve accurate predictions of the video packet count by load-adjusting a LR model.

The study in [51] advances the results of the seminal work on LA learning proposed in [9]. Considering that time-varying loads in the system affect the estimation of QoD predictor parameters, the authors formulated the video quality prediction task as a supervised deconvolution problem. They proposed a LA version of UA video QoD predictions that takes source separation into account. The authors reported an improvement in the Signal-to-Noise-

Ratio (SNR) for LA learning. They provided results from their evaluations using traces from the baseline UA approach [8] demonstrating that the LA learning was (1) faster, and (2) more accurate compared to the UA learning technique.

In a related effort, the authors of [9] expressed the LA technique as a graphical model in [176] to handle cases where there were no packet count statistics in the system for a pre-determined number of users in [203]. The proposed graphical model accepts QoD and kernel-level measurements into the system. Using a K-level switch, a feature, such as the TCPSCK count, is used to deduce under which load settings the data was produced. For every value the load takes, a model is learned. The mappings between the kernel-level features and the QoD feature, the packet counts were dependent on a hidden parameter, which was the system load. The authors with the help of interpolation between different states, demonstrated the feasiblity of predicting the packet count for system loads that had never been observed before. Experimental evaluations of the proposed model demonstrated the suitability of the LA graphical models for the packet count, i.e. the QoD metric prediction in cases of missing training data.

3.3 Proposed Approaches

This section presents the proposed learning approaches to predict the client video QoD at the user devices by modelling the system load, namely, the LA learning technique. The LA technique examines the effect of concurrent requests on the server in its QoD predictions. The UA learning technique assumes the load effect is negligible and does not model its effect in QoD predictions. The LA technique models the dependent relationship between the system load and the video QoD metrics. By doing so, the ML models trained via the LA technique outperforms the baseline approaches which do not capture the load effect.

3.3.1 Load-Adjusted Learning Model (LA)

The theoretical justification for LA learning was proposed in [9]. The authors presented a linear model for the relationship between kernel statistics, client video QoD metrics, and the system load. According to the authors, the response of the server, with respect to kernel metric n, the n-th feature, to one request for video at time i is expressed as the sum of a load-based component $\hat{u_i}[n]$ (sum of resources held by a user), and some deviation signal specific to a feature, $\epsilon_i[n]$,

$$x_i[n] = \hat{u_i}[n] + \varepsilon_i[n], \quad \text{where} \quad i \in \mathbb{Z},\ x_i[n],\ \hat{u_i}[n] \in \mathbb{R}. \tag{3.1}$$

A feature is a metric at the operating system level, such as the number of active TCP connections. The feature set $x_i[n]$ was constructed using SAR function information, which provides system metrics for a given time period. In Eqn. 3.1, $x_i[n]$ refers to the n-th feature observed at time index i. The observed client QoD metric, y_i, is the packet count. The deviations from the expected performance are captured by the noise signal $\epsilon_i[n]$. The signal $\hat{u_i}[n]$ represents an increase in the server load for each user. For example, a request for additional resources θ_n made by the current client or a new client would initiate a feature response of the form:

$$x_i[n] = 2\hat{u_i}[n] + \varepsilon_i[n,1] + \varepsilon_i[n,2]. \tag{3.2}$$

The deviation from the ideal performance arising from the second user is denoted by $\epsilon_i[n,2]$. Let us assume that at time i, the number of users requesting the service is $k[i]$. For instance, in Fig. 3.1 when clients A, B, C and D are receiving video at time i, $k[i] = 4$. The response of the n-th feature to the time-varying load is

$$x_i[n] = \theta_n K[i] + \sum_{k=1}^{K[i]} \varepsilon_i[n,k]. \tag{3.3}$$

The load signal $\theta_n K[i]$ denotes the number of active users at time i times the resources one user uses, θ_n. The TCPSCK or load signal is $K[i]$. The simplest form of a load-adjusted learning algorithm is one that adjusts prediction weights according to changes in the load value.

3.3.2 Un-Adjusted Learning Strategies (UA)

The baseline UA approach for predicting client video QoD metrics from device statistics does not model the effect of the time-varying load. They assume that $K[i]$ is a constant, C.

$$x_i[n] = \theta_n C + \sum_{k=1}^{C} \varepsilon_i[n,k]. \tag{3.4}$$

The implication of this is that during training and testing phases, all samples of the data are used irrespective of the load value. These techniques fail to account for the strain on the limited resources of the streaming server. There is a relationship between a QoD metric, the packet count and the number of users accessing the video stream, the load as illustrated in Fig. 3.2. The fact that a dependency exists between the target metrics makes a case for training the ML models conditional on the load.

3.4 Overview of Machine Learning Techniques

This section provides a brief discussion on the ML models adopted for the chapter experiments.

3.4.1 Linear Regression (LR)

The LR model is the first algorithm considered for the analysis. Using the LR, the relationship between the target metrics, y, i.e. the dependent variables and the independent variables of predictors, x, is modelled as a linear function. The training dataset consists of n number of observations, i.e., $(x_1, y_1), \ldots, (x_n, y_n)$ with p number of predictors. Let y_i be the predicted client video QoD for the ith case. Using a LR model, the aim is to predict the response, $\hat{y_i}$ using a linear function of the form:

$$\hat{y_i} = \beta_0 + x_1\beta_1 + \ldots + x_p\beta_p. \tag{3.5}$$

Given a dataset, the LR model fitting procedure computes a set of coefficients, $\beta = (\beta_0, \ldots, \beta_p)$ that minimize the Residual Sum of Squares (RSS) using the Least Squares fitting procedure:

$$RSS = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 \tag{3.6}$$

where, y_i is the true observed client video QoD metric for the ith case and x_{ij} is the i-th observation for the j-th predictor.

3.4.1.1 LA Learning Using the LR Model

LR models are load-adjusted by training weights for each value of the load signal.

$$\hat{y_i} \Big|_{K(i)=k} = \sum_{j=1}^{p} \beta_j x_{ij} \Big|_{K(i)=k} \tag{3.7}$$

When all samples, regardless of load value are used for training, the model is not load-adjusted. In the UA learning approach, the learning and prediction models use all samples irrespective of the load value.

3.4.2 Ridge Regression (RR)

The Ridge Regression (RR), a variant of the LR is similar to the Least Squares, except that the RR coefficients are estimated by minimizing a different quantity. The RR includes an ℓ_2-norm loss function on the coefficients, and maintains a small amount of energy in each coefficient [204]. RR seeks model coefficients that best suit the data by minimizing the RSS in the LR equation with the addition of a regularization parameter, λ.

$$RR(\beta) = RSS + \lambda \sum_{j=1}^{p} \beta_j^2 \tag{3.8}$$

where $\lambda \geq 0$ is the regularization or tuning parameter that determines the amount of shrinkage; the larger its value, the greater the shrinkage. The regularization parameter, λ is used to adjust the relative influence of the RSS value and the shrinkage penalty, $\lambda \sum_{j=1}^{p} \beta_j^2$ on regression coefficient estimates. When $\lambda = 0$, the penalty has no effect and defaults to the least squares estimates produced in Eqn. 3.6.

3.4.3 Least Absolute Shrinkage and Selection Operator (LASSO)

A common problem with the RR model is that it includes all p predictors in the final model. Despite shrinking all coefficients toward zero, the shrinkage penalty will not set any to exactly zero. This may pose a challenge in model interpretability in cases where the number of predictors, p is very large [205]. The LASSO, another LR variant imposes an ℓ_1-norm on the regression coefficients. Similar to the RR, the LASSO solves the LR model with the addition of a regularization parameter $\lambda \sum_{j=1}^{p} |\beta_j|$, where $\lambda \geq 0$. In contrast to RR, the ℓ_1-norm in LASSO performs a form of automatic variable selection and continuous shrinkage by forcing some model coefficients to zero and effectively *turning-off* some features.

$$LASSO(\beta) = RSS + \lambda \sum_{j=1}^{p} |\beta_j| \tag{3.9}$$

The feature space the thesis examines is a high dimensional one and LASSO is known to obtain sparse linear models in such cases [206].

3.4.4 Elastic Net (EN)

The LASSO algorithm has poor prediction accuracy due to high correlations between features, especially when the number of observations is greater than the feature space, as is the case in the dataset [80]. The EN model was chosen for analysis owing to its ability to perform shrinkage and variable selection just like the LASSO method. The EN combines both the LASSO and RR, that is, it applies a mixture of ℓ_1-norm and ℓ_2-norm penalties on the coefficients. The EN algorithm is a penalized LR model that incorporates the ℓ_1 as well as ℓ_2 penalties during training. A hyperparameter, α, is used in the EN algorithm for determining what weight each of the ℓ_1 and ℓ_2 penalties should receive. The α value ranges between 0 and 1. The contribution of the ℓ_1 penalty would be weighted using the α value provided during the EN algorithm's training. To compute the contribution of the ℓ_2 penalty, the α value is subtracted from 1. The EN penalty is a trade-off of the form

$$\sum_{j=1}^{p} \left(\alpha \left| \beta_j \right| + (1-\alpha)\beta_j^2 \right). \tag{3.10}$$

For instance, using an α value of 0.75 implies that the α in $\alpha \left| \beta_j \right|$ contributes 75% and the other $((1-\alpha)\beta_j^2)$ contributes 25% to the loss function. If the α value is 0 then everything is weighed toward the ℓ_2 penalty, while an α value of 1 assigns all the weights to the ℓ_1 penalty [207].

Regularization in the EN algorithm is controlled by the λ parameter. The larger λ is, the more the coefficients will be shrunk toward zero (and each other). If the λ value is set to zero, regularization is deactivated, and standard generalized liner models are fitted. The choice of the α parameter is directly related to the λ value. Table 3.1 below illustrates the types of penalized models that can be obtained using the listed λ and α settings.

Table 3.1 The alpha and λ settings determine the amount of regularization applied.

λ	α	Resulting Model
$\lambda = 0$	Any number	No regularization
$\lambda > 0$	$\alpha = 0$	RR
$\lambda > 0$	$\alpha = 1$	LASSO
$\lambda > 0$	$0 < \alpha < 1$	EN

The EN automates the choice of the regularization parameter and produces sparse solutions just like the LASSO. However, the EN tends to select more features than the LASSO does as the EN overcomes the grouping effect situation in LASSO. The grouping effect is a situation where the LASSO tends to select only one feature from a group of features with high pairwise correlations [208].

3.4.5 Random Forest (RF)

A non-linear method, the RF algorithm, which is an ensemble method was chosen for analysis to compare with the performance of the linear models. The RF algorithm constructs multiple DTs and combine their output in order to make stable and more accurate predictions. For the ith client video QoD metric, y_i, the RF estimates the $\hat{y}_i$ by using the average predictions from a large number of trees [44, 209].

3.5 Experiments, Model Fitting and Evaluation Procedure

This section presents a description of the data preparation process, model fitting and evaluation strategies using both the LA and UA learning techniques.

Problem Statement: The objective is to predict the client video QoD metrics, y_i: packet count, VFR and ABR using the features $x_i[n]$ given the time varying load $k[i]$.

3.5.1 Data Preparation

The data preparation steps taken by the authors of [9] was adopted. The periodic-load pattern has 51043 observations for 297 features, whereas the flashcrowd-load pattern has 275 features with 15150 observations. First, all non-numeric and constant value features are removed from the datasets. The dataset is prepared for evaluation by adpoting a 60-40 split between the training and test data. First, the ML models are trained with the training dataset. Thereafter, the test dataset is used to evaluate the performance of the trained ML models. The purpose of this test data is to evaluate the model's generalization ability to make accurate predictions on new, unseen data. The regularization techniques, RR, LASSO and EN required a method for selecting the regularization parameter, λ, for the penalty function. RR, LASSO and EN use an ℓ_2-norm, ℓ_1-norm and combination of both norms as a weighted (by λ) penalty term. The entire path for the results for these models was calculated using path-wise cyclical coordinate descent algorithms. To determine the value of the regularization parameter, λ, a 10-fold Cross-Validation (CV) is applied for both the LA and UA learning approaches. The value obtained was used in subsequent learning and prediction experiments. The 10-fold CV was applied during training and while testing the models. Different values for λ were determined for both the UA and LA algorithms. A sequence of values between 0.0001 and 1 was selected and CV applied to automate selection of the λ parameter.

3.5.1.1 UA Learning

The baseline or UA approach outlined in [8] trains the ML models without modelling the time-varying load effect. In the UA experiments, the approach described by the authors was adopted. The training and test data is generated using any sample from the data regardless of the load value. Using the validation set approach, the dataset is split in a 60-40 ratio for training and test purposes respectively. The 60/40 split was done for both traces with 60% set aside for training the models and 40% for test data and prediction.

3.5.1.2 LA Learning

The LA learning strategy was implemented based on [9], and the results compared with the UA approach. A subset of the entire dataset with the load value, TCPSCK, fixed and more than 500 samples is obtained. In order to maximize the number of train samples that can be used for the LA learning, the top subsets with the most samples within them is used. The dataset is then split using the validation set method with 60% of the dataset in the training set, and 40% in the test set. For each of the LA algorithms, the CV procedure described above is employed to determine the best λ value. The LA strategy learns a set of weights for each value of the load signal. By learning different models for each load value, the models removed the conditional dependence on the load in each of these sets of data and reduced the computational complexity of learning them.

3.5.2 Packet Count Prediction Using the Periodic-Load Trace

The comparison between the LA and UA models begin by considering packet count prediction using the set of features from the periodic-load dataset. As a comparison to the UA algorithms, a demonstration of the effectiveness of LA learning using LR, RR, the LASSO and EN, by explicitly modelling the effect of the load on the learning algorithms is shown.

Considering the high-dimensional space of the dataset, the authors of [8] had proposed subset selection as means to reduce the variance in the predictor performance. Variable or feature selection [210], also known as subset selection, involves selecting a subset of explanatory variables from which to construct a regression model. This approach facilitates understanding causal relationship between explanatory variables and response variables. The use of subset selection can also reduce the time and costs of gathering data and estimating the values of model parameters. Furthermore, the predictive performance of regression models can be improved since overfitting is mitigated with subset selection [211]. The thesis

examines the effects of subset selection on learning and prediction performance for both the UA and LA learning approaches.

The LA learning approach assumes that system statistics, along with any learning models that can be learned from them, will change over time as the number of users accessing video content changes. Learning algorithms that are able to capture this effect will do better. However, it does not automate the selection of the learning algorithm. In the experiments on packet count prediction, the thesis investigates the problem of parameterization of learning algorithms. Precisely, it addresses the problem of automating the choice of the learning algorithm via the EN model. The UA learning discountenances the load effect and proposed subset selection as an alternative.

For the LA case, the LASSO was less sensitive to the selection of λ than for the UA case. This was due to the LASSO's tendency to turn off components, making its tuning more sensitive to changes in load. There seems to be more influence of the system load on the choice of best λ value in the UA case than the LA case. However, it is encouraging that when LA data is used, LASSO is easier to tune. In both the UA and LA cases, the best value of λ for RR was approximately the same. This can be explained by the fact that RR turns on many coefficients, so as the load changes, so does the sensitivity to λ.

3.5.3 Packet Count, VFR and ABR Prediction Using Both Traces

In the extended experiments on LA versus UA learning, both datasets are used in all experiments described in this section. The effectiveness of the LA model is explored using both traces. The performance of the UA models are compared with the LA models for LR and the shrinkage methods: RR, LASSO and EN. The performance of the best performing linear method, EN is then compared with a non-linear ML model, the RF. Finally, the performance of the LA technique is analysed for both linear and non-linear ML models.

The EN algorithm is trained using the LA method using the same CV procedures as the UA models. The EN models are trained for each subset of data based on the TCPSCK values. Then, the EN model is trained via the UA technique to predict the VFR, ABR, and packet count. These models are referred to as Un-Adjusted Elastic Nets (UA-EN). With the same number of samples as the UA-EN models, LA models are trained to predict the same target video QoD metrics. These models are referred to as Load-adjusted Elastic Net (LA-EN) models .

3.5.4 Evaluation Metrics

All models are evaluated in terms of two accuracy measures. The first is the Root Mean Squared Error (RMSE), which is computed as

$$\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}. \tag{3.11}$$

The best model is the model with the lowest RMSE. The second measure is the R-squared which is essentially a statistical measure that explains the goodness of fit of our regression models. R-squared achieves this by comparing our regression models with a baseline model, one which is simply the average of observed responses of the dependent feature. The R-squared is computed as

$$R^2\,(\%) = 1 - \frac{SSE}{SST} \times 100, \tag{3.12}$$

where the Sum of Squared Errors (SSE) in our model is computed as

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \tag{3.13}$$

and, the Sum of Squared Total (SST) of our baseline model is computed as

$$\sum_{i=1}^{n} (y_i - \bar{y}_i)^2, \tag{3.14}$$

and $\bar{y}_i$ is the mean of the observed data. The model with the highest R-squared value is the best, and a model with an R-squared value of 1 is a perfect model. In this thesis, R-squared values are reported in percentages.

3.6 Results

This section discusses the results of the LA and UA experiments. The results demonstrate that it is possible to parameterize learning algorithms and automate the choice of the tuning parameter, λ with the EN algorithm to boost the performance models learned using the LA and UA techniques. With the EN model based on the LA technique, the thesis contributes an adaptive learning algorithm, which improves the prediction algorithm's performance by modelling the effect of the system load. These results also provide evidence that shows that subset selection alone does not improve prediction performance. Furthermore, this section also presents the results of the LA versus UA learning approaches using both linear and non-linear ML models. The evaluation results of the LA technique versus UA learning is reported for different test scenarios, namely the flash-crowd and periodic-trace patterns. The LA models record superior performance compared with the UA models. In general, the thesis adopts the naming convention of *"Learning Technique-ML Model Name"* in referring to the LA and UA ML models. For instance, LA-RR refers to the ridge regression model learned via the load-adjusted technique.

Table 3.2 The results of the LA versus UA learning using the ML models is shown. The LR algorithm offered the worst performance and is not listed here. The poor performance of the LR estimates motivated the study of the shrinkage methods.

Model	LA-RMSE	UA-RMSE
Ridge	**28.49**	30.74
LASSO	**28.46**	29.85
Elastic Net	**28.36**	28.53

3.6.1 Packet Count Prediction Using the Periodic-Load Trace Results

In Table 3.2, the results of the RR, the LASSO and the EN are presented for both the LA and UA algorithms on 20000s of test data. The RMSE is better for all algorithm predictions if the algorithm is LA (bold font in table).

The majority of y values lie between 20 and 100 packets/s. The LA-RR estimates are approximately 2 packets/s better than the UA-RR based estimates. The prediction performance gain achieved by the LA-RR over the UA-RR is $\approx 10\%$. The best performance is obtained with the EN algorithm in both the UA and LA learning. RR performs the worst. Based on these findings, the LA learning technique improves client video quality metrics prediction.

In order to provide some context on what a RMSE of 28.36 means for this QoD metric, Fig. 3.3 provides an illustration of the accuracy of the LA-EN predictions of the packet count compared with the true packet counts. LA-EN makes better predictions when the packet count is low, when the TCPSCK is high; and it makes 2 packets/s better predictions than UA learning.

The improvements in prediction are significant because service level agreements that guarantee the desired levels of packets/s should be delivered to clients may be affected by the wrong predictions.

To determine if load-adjusting the data or performing subset selection explains the improvement in performance, the effect of subset section is evaluated on the LA learning.

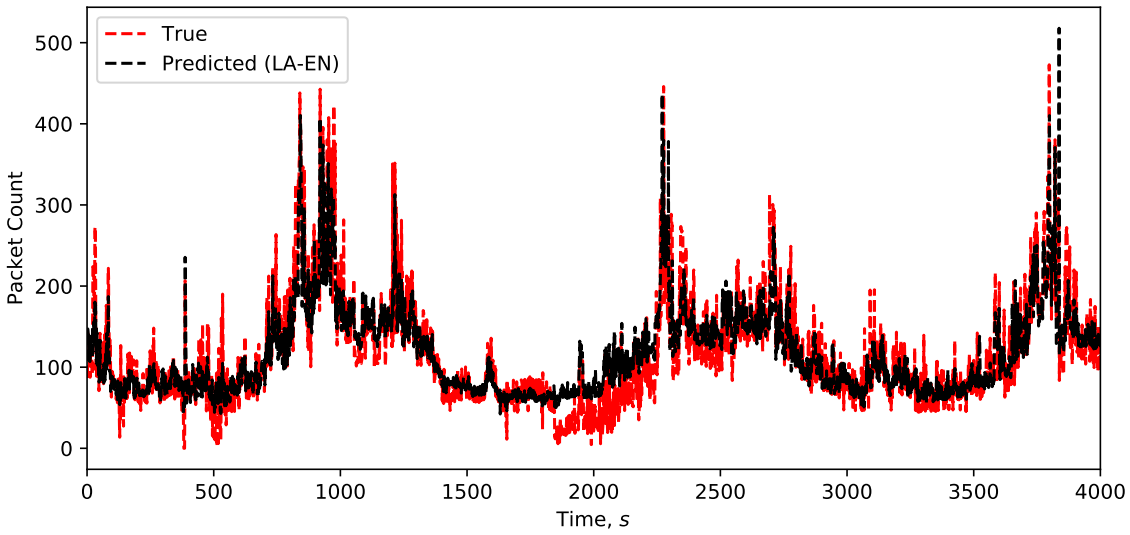

Fig. 3.3 The accuracy of the LA-EN prediction is compared with the true packet count, y.

Table 3.3 provides a comparison of the performance of LA learning alone versus LA learning with subset selection applied to the features. The EN, the best performing learning algorithm, is used in these experiments. The table lists the results for predictions when the TCPSCK on the system is 30, 40, 50, 60, 70, to illustrate how the predictors behave under various loads of the system. The results show that LA learning without subset selection performs better than LA learning with subset selection. LA learning produces packet count predictions which are 2, 7, .1, 5, and 5 packets/s more accurate than when LA learning is combined with subset selection for successive load values. These results are also evaluated using the adjusted R-squared score, which shows that these gains in performance are not due to the measurement method. A higher adjusted R-squared score indicates better performance than a lower one. The performance gain achieved by LA learning compared to LA learning combined with subset selection is confirmed by the adjusted R-squared score.

Table 3.3 RMSE, R Squared (in percentages) for LA versus LA with Subset Selection. LA learning without Subset Selection offers better predictions.

	Load-Adjusted		LA & Subset Selection	
Load Values	RMSE	R^2	RMSE	R^2
30	38.43	71.2	40.11	69.20
40	25.48	64.40	32.73	42.10
50	24.20	42.20	24.34	37.70
60	19.15	47.90	24.61	2.30
70	18.91	43.40	23.46	4.20

3.6.2 Performance Evaluation of Packet Count, VFR and ABR Predictions Using Both Traces

This section compares the performance of the four linear models, namely LR, RR, LASSO and EN using the baseline or UA approach as described in [8] and the LA models. The best-performing linear model, the EN model is then compared with the RF algorithm. Finally, an evaluation of both models using both the periodic-load and flashcrowd-load datasets is provided for both the UA and LA learning techniques.

3.6.2.1 Comparison of the Linear ML Models Using UA Learning

Table 3.4 lists the performance of the linear models LR, RR, LASSO as well as the EN on the test data for the periodic-load and flashcrowd-load traces using the UA technique.

For both traces, the EN model gives the best result out of all four linear models. The RMSE for all three metrics predicted is the lowest for the EN, and R-squared is also the highest for the EN models (bold in table). For all three target QoD metrics, the EN performs similarly to the LASSO and the LR for both traces. However, the EN is the most accurate due to its ability to adapt the loss function based on a data-driven approach, which overcomes the limitations of LASSO. The EN model performs well in both traces, but offers lower RMSE

Table 3.4 RMSE, R Squared for Video Frame Rate, Audio Buffer Rate and Packet Count for UA Models. For both traces, Elastic Net produces the best RMSE and R Squared results for the linear models (bold font); Random Forest produces the best results (bold font) out of all models evaluated.

		ABR		VFR		Packet Count	
Un-Adjusted Method	Model	RMSE	R^2	RMSE	R^2	RMSE	R^2
Periodic-load Trace	Linear Regression	14.75	42.01	3.27	63.34	28.09	82.63
	Ridge Regression	16.70	28.99	3.31	58.58	28.76	80.36
	LASSO	14.19	53.30	3.18	63.80	28.09	82.98
	Elastic Net	**14.03**	**55.28**	**2.99**	**64.20**	**25.82**	**83.63**
Non-linear	**Random Forest**	**4.42**	**95.44**	**1.95**	**84.58**	**22.74**	**87.59**
Flashcrowd-load Trace	Linear Regression	11.26	63.20	2.67	73.82	29.73	85.03
	Ridge Regression	11.96	58.48	2.73	72.67	31.49	84.13
	LASSO	11.36	63.16	2.66	74.27	29.74	85.56
	Elastic Net	**11.17**	**64.37**	**2.62**	**74.76**	**29.64**	**85.93**
Non-linear	**Random Forest**	**4.76**	**94.37**	**2.09**	**84.70**	**25.53**	**89.63**

values for the VFR and ABR QoD metrics using the flashcrowd-load trace. This may be due to the *spiky* nature of clients joining and leaving the streaming session in this data. The worst prediction perfromance for all UA linear models is offered by the RR model. Table 3.4 shows the performance of the RF algorithm using the UA approach for both load traces. The RF algorithm offers a significant improvement in RMSE and R-squared compared to the EN. The RF performance gain indicates that non-linear methods perform significantly better than linear methods, as evident from the results.

3.6.2.2 Comparison of the LA ML Models with UA Models

Table 3.5 lists the results for the EN and RF models learned using the LA and UA approaches. For both periodic-load and flashcrowd-load traces, the LA models for EN and RF outperform the UA models across all three target video QoD metrics. In both load traces, LA-EN estimates indicate improvements of over 5 audio buffers/second over UA-EN. For both load traces, LA-EN estimates for VFR and RTP show similar improvement.

Table 3.5 Load-Adjusted Technique Versus Un-Adjusted Technique; LA techniques offer better estimates than the UA approach for both traces. The RF produces more accurate predictions than the EN using the LA technique.

		ABR		VFR		Packet Count	
Trace	Method	RMSE	R^2	RMSE	R^2	RMSE	R^2
Periodic-load	**Load-Adjusted-EN**	**12.03**	**65.84**	**3.08**	**66.54**	**30.51**	**83.64**
	Un-Adjusted EN	12.10	57.53	3.28	63.36	36.47	73.16
	Load-Adjusted RF	**6.46**	**92.52**	**1.79**	**86.66**	**28.29**	**84.29**
	Un-Adjusted RF	9.49	72.89	2.11	85.68	33.62	80.62
Flashcrowd-load	**Load-Adjusted EN**	**8.38**	**84.94**	**2.20**	**85.82**	**23.94**	**80.32**
	Un-Adjusted EN	9.79	65.04	2.71	78.84	25.66	74.33
	Load-Adjusted RF	**4.12**	**97.11**	**1.88**	**89.09**	**20.55**	**87.57**
	Un-Adjusted RF	4.40	94.07	1.89	78.89	29.55	82.11

For the LA technique, the RF algorithm is more accurate than the LA-EN. The LA-RF model shows a big improvement in predictions. A comparison of the packet count prediction to illustrate what the RMSE values imply is provided. The LA-RF estimates for the packet count metric using the flashcrowd trace indicate values between 90 and 449 packets/s. The UA-RF estimates for the same trace range between 8 and 347 packets/s. The true RTP values lie in the range of 83 to 545 packets/second. When expressed in percentages, the average improvement in prediction performance is $\approx$ 50% to 60% better using the LA-RF learning.

In general, it is observed that the LA models for the flashcrowd-load trace offer better predictions compared with the periodic-load trace. This is interesting to note as training the prediction weights conditional on the load seems to be able to handle the *spiky* nature of clients joining and leaving the stream.

3.7 Efficient Resource Management with the LA Technique

Resource management involves managing the key network resources, such as switches, routers, processors, memory, disks, etc. These resources are used independently or in

combination to provide services over a network. To implement a specific function or service, service providers can choose to provision a minimal predetermined set of resources. However, predicting is difficult, as both over- and under-estimation can result in low utilization and revenue loss. As a result, predicting demand and dynamically provisioning and replenishing resources so that the network is resilient to variations in demand for services is a fundamental challenge in resource management. Learning ML models for a video streaming session conditional on the load on the system via the LA technique can enable a network manager understand the health of the network for a preset number of users. This information can be useful for either knowing when to provision more server resources or when to reduce the server resources depending on the number of users involved in the streaming session. In this section, the thesis investigates the role of how the LA learning technique can enable efficient resource management.

Integrating elasticity in video delivery systems would allow the delivery infrastructure to adapt to shifting workloads in today's cloud environments. This allows the system to autonomously decide when to allocate and de-allocate resources at any given time in order to match the requests for the system with available resources. This section explores how the LA technique can be used in a feedback control system to detect or predict when there is a change in service demand. This service change point could be the point or time interval when the system detects anomalies or increased requests for system resources. One question this section seeks to answer is this. If we know that the system load has doubled, can we use this information to improve the delivery system? To demonstrate this, the LA-EN model is adopted which allows the automation of the regularization parameter suitable for predicting the video quality conditional on the load.

A demonstration of what happens when the server load increases for the video delivery system shown in Fig. 3.4. The packet count recorded at the client device for 100 seconds for the server load value, K, for 30 and 60 users is shown in Fig. 3.4, R1. In R2, the same plot

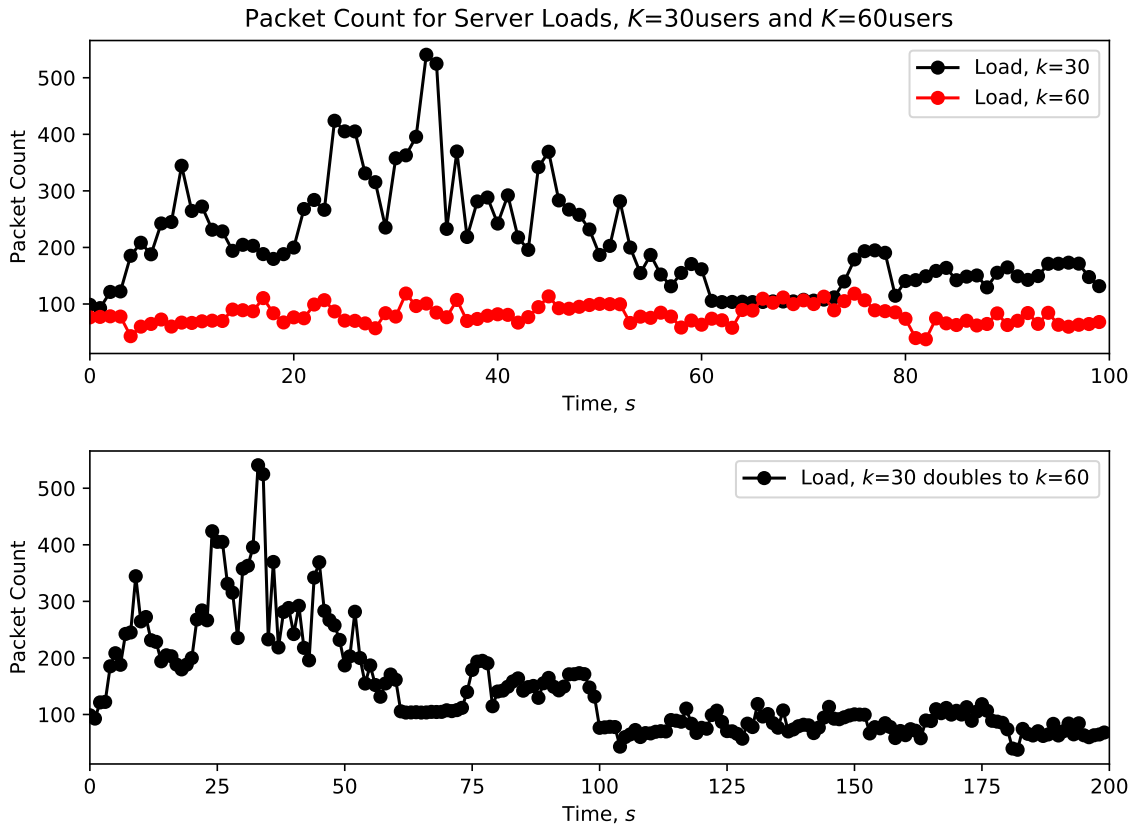

Fig. 3.4 The packet count delivered to the client for server load value, K=30 and K=60 users recorded for 100 seconds is shown in Row 1. There is drop in the packet count when the K value doubles. I demonstrate this drop in video QoD in Row 2.

for the K values of 30 and 60 users recorded for 100 seconds each is shown. It is evident from Fig. 3.4, R2, that there is a drop in the received packet counts delivered to the client devices. This is due to the strain for the server resources in this shared environment. Fig. 3.4, R2 illustrates a plot of the recorded packet count showing the transition from when the server load is K=30 to K=60 to demonstrate the drop in the video QoD. The mean packet count for K=30 users and K=60 users is $\approx$ 208 packets/second and 80 packets/second respectively.

Using the EN ML model, the thesis examines the performance of the LA technique for a specified number of users; for example, when the server load, K is 30. The scope of the LA models in detecting when the load on the server doubles is investigated. For instance, if the number of concurrent users viewing a video streaming doubles from K=30 to K=60,

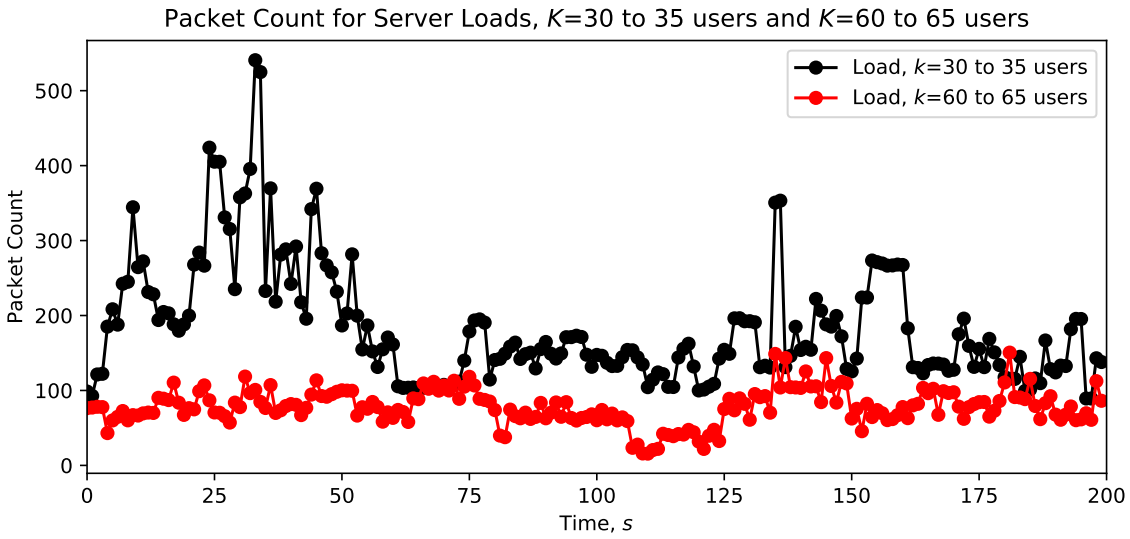

Fig. 3.5 The packet count received at the client for server load value range, K = 30 to 35 users and K = 60 to 65 users captured for 200 seconds are illustrated. The packet count delivered to the client is better when there are less number of users. As the streaming sessions increase, there is a drop in the video QoD.

over a time interval, can this enable accurate prediction of the video QoD including detecting the transition stage? Given that this is possible, this information could be used in enabling auto-scaling of the cloud resources in order to handle the surge in requests. An infrastructure of this nature enabled by the LA models would prevent the degradation of the received video stream at the client devices that may arise as a result of interference from other users.

To investigate this, a feature set with the packet count for the load values, K of 30 - 35 users and 60 - 65 users is extracted. There are 8385 observations and 6323 observations for the 30 - 35 users range and the 60 - 65 users respectively. In Fig. 3.5, a plot of the video packet count received at the client devices for the load value ranges of 30 to 35 users and 60 to 65 user sessions is shown. There is a significant reduction in the video packet count received at the client devices as the number of streaming sessions increase.

The evaluation begin by first fitting the LA-EN model to the two datasets. Both models are evaluated using a validation set method with a 60-40 training-test data split. Using a grid of alpha values ranging from [0.0001, 0.001, 0.01, 0.1, 0.3, 0.5, 0.7, 1], CV is applied

Table 3.6 The results of the LA-EN model for the 30 - 35, 60 - 65 user ranges, and the time-varying combined trace are shown. The LA-EN model RMSE values illustrate the accuracy in predictions.

Dataset	LA-EN RMSE
30 - 35 users	38.06
60 - 65 users	16.95
Time-varying combined trace	26.93

to obtain the λ values for the experiments. These values were used for training the LA-EN models and for predictions. The original packet counts recorded for the 30 - 35 users and the 60 - 65 users ranges are plotted against the LA-EN predictions in Fig. 3.6. The LA-EN model accurately approximates the actual packet count observations recorded at the client devices.

To investigate the efficacy utilizing the LA technique to make resource allocation and management decisions, an examination of the performance of the LA-EN model for a time-varying dataset where the packet counts recorded for the 30 - 35 users transition to the 60 - 65 users range is carried out. The questions of interest here are: (1) Is it possible to rely on the LA-EN automatic parameterization to gracefully handle the transition from a region where there is less demand for the system resources to a moment where this suddenly doubles? (2) Does the learning function adaptively respond to these changes especially around the time intervals leading to and following the increased demand for resources?

To investigate this, a combined dataset comprising of the 60% training splits from the 30 - 35 users range and the 60 - 65 users range is generated. The remaining 40% test data is used for predictions. The same grid of alpha values described above for learning the λ values for the combined dataset is utilized here. Table 3.6 lists the performance of the LA-EN learning algorithms for the 30 - 35 users trace, 60 - 65 users trace and the combined time-varying dataset of both traces.

Fig. 3.6 The LA-EN predicted packet counts for the 30 - 35 users (R1) and 60 - 65 users models (R2) are plotted against the actual observed packet counts. The LA-EN model predictions accurately approximates the original recorded packet counts.

The actual packet counts for the time-varying trace lies in the range 51 to 370 packets/s. The LA-EN predictions realized for the dataset lies in the range 38 to 296 packets/s. The R-squared score for the data, which is a measure of the fit of the LA-EN regression model to the observed time-varying trace is 87.10%. Both metrics demonstrate the accuracy of the LA-EN model in adaptively learning the time-varying change in resource demands. In Fig. 3.7, a plot of the accuracy of the LA-EN model predictions for the time-varying combined trace is shown to understand what an RMSE of 26.93 and an R-Squared score of 87.10% means. The original packet count recorded for the time-varying trace is compared with the LA-EN model predicted values. The mean true packet counts when there is less demand for the system resources, i.e., the 30 - 35 users region is 191 packets/s. The mean true packet

count when the demand increases to double the amount, i.e., the 60 - 65 users region is 133 packets/s. The mean LA-EN packet count predictions are 187 packets/s and 131 packets/s for the 30 - 35 users region and 60 - 65 users region respectively. The main point observed here is that the LA-EN predictions are much better during increased levels of demand. This accuracy in predictions is crucial as a misprediction would impact service level agreements that guarantee the expected video QoD. In Fig. 3.7, it is observed that there is a spike for the original packet count at timestamp 18s; the LA-EN model accurately tries to approximate this peak. The original packet count value at this point is 370 packets and the LA-EN predicts 296 packets. The prediction is off by 74 packets.

Let us examine the time intervals leading up to the transition and following the transition point from 30 - 35 users to 60 - 65 users requesting the system resources. A focus here is on the five time indices leading up to the transition point; and the five time indices following this transition point. The true packet count corresponding to these time intervals are : [136.97, 254.46, 186.18, 212.08, 116.12, 68.52, 65.91, 60.10, 79.72 and 100.44]. The LA-EN predictions for this time interval is [155.74, 244.93, 217.23, 169.82, 146.60, 71.21, 70.51, 65.48, 72.60 and 86.44]. The mean true packet count for this time period is 130 packets/s. The mean LA-EN predicted packet count realized is 128 packets/s. The predicted values are just off by 2 packets.

The thesis results have given evidence that the LA-EN automatic parametrization is able to realize accurate predictions of the received packet count for cases when the server demands increases. It has shown that by load-adjusting the EN model, it is achievable to accurately and adaptively respond to the changes in the system without a need to provision extra resources. The thesis posits that owing to the limited server resources, there may be cases that may require integrating service elasticity in cloud environments. The ability of the LA-EN model to capture points of transition from low to high service utilization makes a case for integrating this model with an auto-scaling facility or a feedback loop as proposed in [82]. The gains of

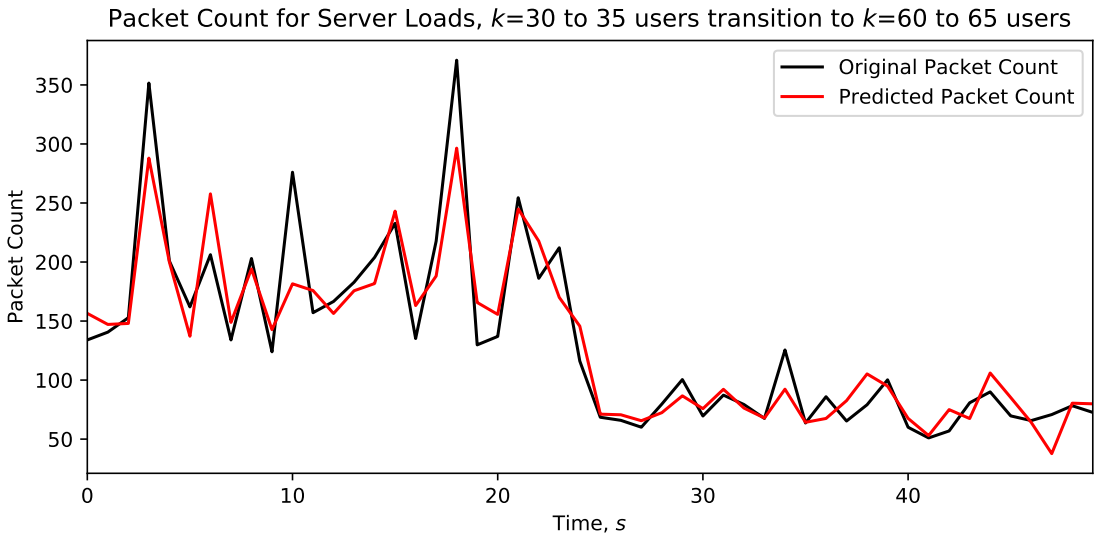

Fig. 3.7 The LA-EN model accurately predicts the time-varying load for $30 < K < 35$ which doubles to $60 < K < 65$. The packet count predictions are plotted against the true packet counts. The LA-EN model predictions accurately approximates the original recorded packet counts.

integrating service elasticity with the LA technique can aid in preserving and maintaining the agreed service agreements. More so, the ability to adaptively handle increased loads may be advantageous over a given time interval up to a known resource threshold. Beyond this threshold, the output from an adaptive LA-EN framework can serve as input to initiate or provision resources at times of increased system utilization. This technique would help avoid over-provisioning the system resources which could lead to a waste of resources.

3.8 Key Advantages of the LA Learning Technique

Predicting client video QoD metrics using the statistics of the streaming server assumes no detailed knowledge of the system is required for modelling the learning process. This has the advantage of enabling a network manager to provision the predictor without significant supervision or set-up. The results shown in this chapter are relevant to the networking community. The authors of [212] evaluated monitoring of the quality of a compressed video

transmitted in a lossy packet network using only bitstream measurements from the standpoint of a network service provider. In their study, they utilized Mean Squared Error (MSE) as an estimation of video quality. They investigated three different techniques for MSE estimation: NoParse (NP), QuickParse (QP), and FullParse (FP). The FP method calculates the effects of packet loss on video, whereas a QP method is only concerned with calculating the video bitstream quality; as a result, the QP method requires less computation time than a FP method. The NP method estimates MSE based only on network-level measurements. The authors concluded that the FP was the most accurate method. It is possible that no measurement may be available for in-network video quality estimation in practical network system spanning multiple ISPs over a large geographical area, except for packet loss rate and bitrate. In such cases, the NP may be the viable option. The LA approach, however, can learn the client video QoD without having to know too much about the system or the video itself, by using device statistics of the server(s) delivering the video resources.

In a novel approach, this chapter investigated how the LA-EN models could be applied to training data to boost performance of (1) baseline UA learning approaches and (2) LA algorithms with subset selection. Using adaptive methods based on the EN model, the analysis carried out in this chapter provided evidence demonstrating the efficacy of the LA-EN technique in different network conditions. The LA approach is computationally cheaper than the UA learning. The proposed ML models investigated in this chapter, namely the LA-EN, LA-RF, LA-RR and LA-LASSO, all have lower computational requirements compared with the UA equivalent models. The LA models learned separate models for the different load values, eliminating the conditional dependence on the load within each set of data and reducing computational complexity. This way, the models improved the prediction accuracies of the UA methods. It is of note that the original LA models based on the LR model are computationally cheaper than the models contributed in this chapter. However, the chapter contributions investigated the LA technique using a range of ML models different

from those proposed in [9, 51, 176, 203] and also considered a number of different QoD metrics. The thesis also contributed extended results showing the efficacy of the LA technique using a range of linear and non-linear ML models. The results presented in this chapter demonstrated that the LA models are able to record similar performance gains for other QoD metrics other than the packet count QoD metric. These results highlight the superiority of the LA over the UA models in all these scenarios.

Furthermore, a baseline method taken in the literature to reduce variance in predictive performance is to examine the feature set via subset selection. The variance could be attributed to the poor condition number or low rank of the feature set matrix used to learn the prediction coefficients. To improve prediction, the authors of [8] initially used the full set of infrastructure statistics which included device statistics from the cluster and network, and then successively reduced it. They failed to specifically account for the changing load of their system in their subset selection methodology. The shrinkage method is another way to boost the predictive performance. Some examples of shrinkage methods are RR, LASSO and EN. The results from the evaluations here suggest that subset selection may not be the best method to improve predictions from regression algorithms. By load-adjusting the data based on the system load, it has been shown that the prediction performance of the regression algorithms can be improved at no additional data or computational cost. Most notably, this chapter contributes a novel approach using the EN to automate the choice of the λ to speed up the learning process.

The chapter has demonstrated how the LA-EN learning model can be used to handle periods of increased demand for server resources. This framework can be used to initiate auto-scaling in the cloud, service elasticity and integration of a feedback loop. The LA-EN framework can be used as input to enable auto-scaling of resources in a reactive or proactive manner. The LA-EN framework can be used as an input to a queue management model to initiate resource provision for the server. The accuracy of the LA-EN model in cases of

increased demands can be reverse-engineered to learn the load on a cluster of distributed servers. This can allow the network manager to accurately estimate the optimal number of resources that must be allocated to satisfy the predicted demand in order to optimize the service response time and reduce over-provisioning. The network manager can use this data to determine the appropriate number of resources required to meet the projected demand, allowing for faster service response times and less or no over-provisioning of resources [213].

3.9 Conclusion

Using a load-adjusted learning strategy, this thesis chapter presented a method for enhancing QoD metric predictions. This method based on the LA-EN model automates the process of choosing the regularization technique for linear shrinkage methods used for QoD predictions. It was demonstrated that the EN algorithm delivers the best prediction performance among the LR variants using the baseline UA approach. In this chapter, the results from the evaluations provided evidence that the LA learning algorithm enhances UA prediction performance for all three metrics studied, namely: VFR, ABR and packet count evaluated under two different network or test conditions. The chapter provided extended results demonstrating the efficacy of the LA learning technique using non-linear, ensemble ML models such as the RF algorithm. The chapter results showed that the LA-RF models improved the results of the LA-EN under the two test conditions. The LA-RF and LA-EN predictions both show superior performance compared with the UA ML models. For instance, the LA-RF and LA-EN models offer $\approx$ 10 packets/s and 7 packets/s improved predictions respectively over the UA equivalent models. It was also demonstrated that subset selection alone does not offer improved predictions compared with the LA models; they decreased the accuracy of the predictions. These findings are interesting to practitioners because they show that off-the-shelf techniques may be enhanced and automated without the need of additional data

sources, which is important given the growing dynamicity of current service delivery and host infrastructures.

Chapter 4

Codec-aware Network Adaptation Agent Model

THIS chapter introduces a novel online lightweight learning framework, Codec-Aware Network Adaptation Agent (CNAA), a model able to serve as a QoD predictor, and as a network state classifier. The CNAA predictor model achieves accurate predictions of jitter, a QoD metric, for adaptive video services. This it does by using the available network information in the presence of congestion and adaptive codecs. The CNAA model is also able to estimate the network state by utilizing the predictor model estimates. A description of the network test-bed is provided, followed by an experimental analysis which describes the evaluation procedure of the proposed ML learning model. The CNAA model is first evaluated by collecting QoD data for a target video client from a six router networking test-bed where video is transmitted across a shared infrastructure with varied levels of interfering sources that induce congestion. Secondly, the CNAA model is then evaluated using QoD measurements collected from a real-world network topology obtained from the SNDlib [214] using a SDN emulator. Based on the results, some further directions are suggested in the context of a feedback loop in an SDN environment, able to initiate a path re-routing in the face of network

disruptions or link failures. The contributions proposed in this chapter can be applied at any point within the video delivery system either in an online or offline manner. For example, a video service provider may choose to predict the video QoD within their own infrastructure or at edge locations; a network manager may utilize the the combined video QoD predictor and network congestion level classification models to help initiate remedial and corrective actions in poor network conditions.

4.1 Motivation and Problem Statement

Integrated network monitoring and network state classification tools are critical to network managers to help in resolving network problems on a timely and proactive basis. This is because to effectively manage networks, they need to be in a position to predict the network behaviour [215]. This is not a trivial issue as most applications do not support QoD integration which might have some negative impacts on their solution [5]. Since the architecture of the IP network was not designed for real-time services like video streaming, there are many factors that can influence the final QoD. Common video QoD metrics include latency due to packet delays arising from transmission delay (the time it takes to send them across communications links), processing delay (the time it takes each network element to handle the packet), queueing delay (waiting the packet to be handled), packet loss and throughput [216]. The need to understand latency characteristics of networks and devices has become even more critical with the increased use of delay-sensitive voice and video traffic over IP and Ethernet networks. Characterizing the temporal performance of a network requires measuring latency and jitter [217]. According to RFC 4689 [218], jitter is the absolute value of the difference between the forwarding delay of two consecutive packets that belong to the same stream. For interactive or multimedia services, estimating packet delay variation is critical as this variability impacts signal reconstruction at the receiver. It is common for end-user devices and applications to be able to tolerate jitter. This is

normally achieved by buffering the data flow and designing processing algorithms in order to compensate for slight changes in latency between packets. These buffers are essential to the proper functioning of packet networks, but excessive, poorly managed, and poorly coordinated buffers create excessive delays [219].

Network congestion impairs video delivery and results in variations in jitter [220], both of which are problematic for real-time video applications [10]. The basic architecture for video streaming systems consists of two major components: a codec for encoding the video, and a transport protocol for transmitting it. The video codecs estimate the capacity of the network so that the video data can be compressed adaptively according to the available network capacity [11]. The estimated network capacity may change instantaneously, which may lead to capacity under or over utilization, video stalling ultimately degrading the video quality [221].

This thesis chapter contributes the CNAA learning framework, a lightweight online ML-learning engine that achieves accurate QoD metric estimation of jitter statistics. It achieves efficient and accurate network monitoring for adaptive video codec sessions. By modeling the behaviour of the video codec deployed by the server, the model achieves responsive, in-network learning. The main goal of this chapter is to demonstrate that the network conditions are valuable sources of information that can be incorporated into the ML learning algorithms to achieve accurate QoD metrics predictions for video services. The CNAA learning framework comprises of a predictor model and network-state classifier model. The CNAA predictor model introduced in this chapter incorporates the underlying effect of congestion and the adaptability of codecs in its learning engine to demonstrate how learners must incorporate these network dynamics to achieve accurate predictions of jitter. Considering the fact that the video codec already estimates the level of congestion in the network and adjusts its playout to meet network conditions, the technique provides a representation of jitter (i.e. the CNAA network-state classifier model) that models how the

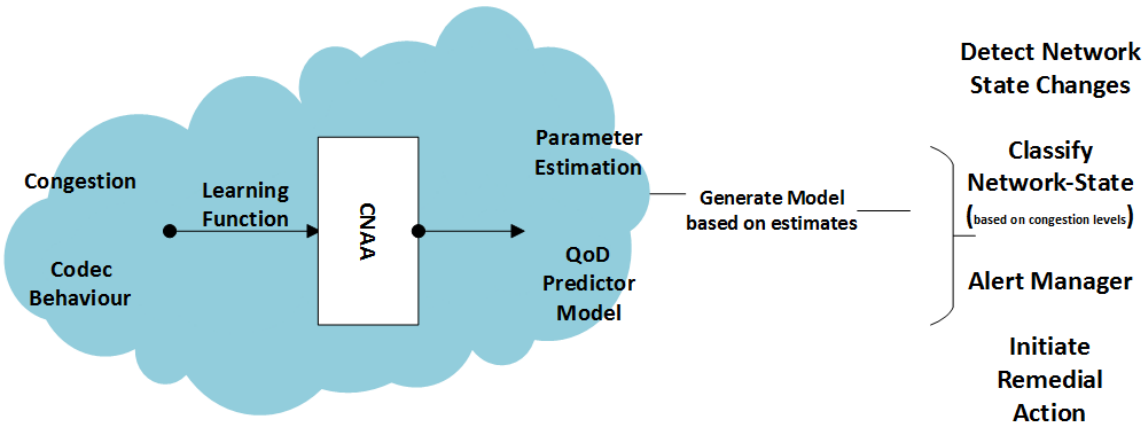

Fig. 4.1 The role of the CNAA agent in the network prediction and monitoring ecosystem. The learning function incorporates network dynamics which are caused by congestion and codec behaviour in QoD prediction and parameter estimation. The estimated parameters are then used as inputs to generate tree-based models for network-state classification of the service. The model includes network state change-point detection which, practically could be used to initiate a remedial action.

codec behaves under different levels of congestion (interfering network users). This method of estimating the adaptation of the codec makes it possible to detect changes in network congestion as the network conditions vary. The effect of the codec adaptation on jitter is expressed in three parameters: its period, its base, and its decay rate. Fig. 4.1 illustrates the purpose of the proposed in-network learning agent, CNAA. A summary of the contributions of this chapter is as follows:

- A new model for estimating jitter is introduced which is suited to the case where the delivery system uses an adaptive video codec. The results from the CNAA predictor model shows that it achieves better prediction performance when compared with some baseline approaches. The performance gain of CNAA was achieved by leveraging the information in the network;

- This chapter contributes a network-state classifier model which models the codec behaviour in the presence of interfering network users. The output of the CNAA predictor model is fed as inputs to the network-state classifier model which then uses

Off-the shelf classification algorithms. The results from these analysis demonstrate the effectiveness of this approach which records classification accuracies of approximately 80% to 90% accuracy.

- This chapter demonstrates that the CNAA classifiers achieve service change-point detection with an accuracy of 80% to 95% and a responsivity of 89% and above when the congestion levels are changing from 7% to 1.7% per second.

The rest of the chapter is organized as follows. Section 4.2 describes the laboratory and SDN network test-bed implementation used to evaluate the models proposed in this chapter. In Section 4.3, a description of the CNAA model is presented. Section 4.4 describes the parameter estimation process for the CNAA jitter model. Section 4.5 describes the CNAA predictor model fitting process. Section 4.6 provides some background on the ML techniques used in this chapter. Section 4.7 describes the model fitting for the baseline ML models. This section benchmarks the performance of these baseline models with the CNAA model. In Section 4.8, an evaluation of the efficacy of using features of the CNAA jitter model for network state classification is presented. Finally, the chapter conclusions are presented in Section 4.9.

4.2 Network Test-Bed

In order to test the proposed CNAA method, the model evaluations are based on both a laboratory physical test-bed and on SDN network topologies using the Mininet network emulator [222]. This section describes the network topologies from which QoD measurements were extracted to evaluate the CNAA predictor and network state classifier models.

4.2.1 Laboratory Network Implementation

Fig. 4.2 shows the laboratory network set-up for a video streaming session between a server and a client machine. The server is connected to the client using six Cisco 2911 Integrated Services Routers (ISRs), $R1, R2, \ldots, R6$. The server and client machines are 64-bit Windows 7 machines. To match observations from the server and client, both clocks are synchronized using the Network Time Protocol. Jitter measurements, $x[n]$, are taken using Wireshark from the client machine. The server machine streams the "Big Buck Bunny" mp4 video, ($\approx$ 10.34 minutes long) with the VLC player to the client machine, using RTP. In separate experiments, the transcoding of the pre-encoded video files is set to either H.264 or H.265.

To evaluate the efficacy of the proposed learning algorithm, the following steps are taken:

1. During the streaming session, routing changes are initiated so that the video stream is forced to switch paths between paths offering the most bandwidth, i.e. the connections over the Gigabit Ethernet (GE) links (1000Mbps) and paths offering lower bandwidth, the Fast Ethernet links (FE) (100Mbps) and the serial links (64kbps);

2. Using the Ostinato Packet Generator [223], some additional background interference congestion is introduced by injecting bursts of interfering UDP traffic over the GE interface of the router, $R2$, to a GE interface of $R6$.

A series of time-varying periodic falling exponential curves are represented in Fig. 4.3 as a result of the codecs' adaptive responses to network congestion. Each of the time series data shown in Fig. 4.3 has a periodic component. Additionally, they decay exponentially during each of these periods due to the video codec's behavior. Video playback is tailored by the codec based on what it understands about the network conditions. The codec uses estimates of the network capacity to decide what levels of compression to apply during transcoding of the video data. By doing this, the video server can adaptively relay the video stream to the client device in response to changes in the network conditions [11]. As a result, each of the

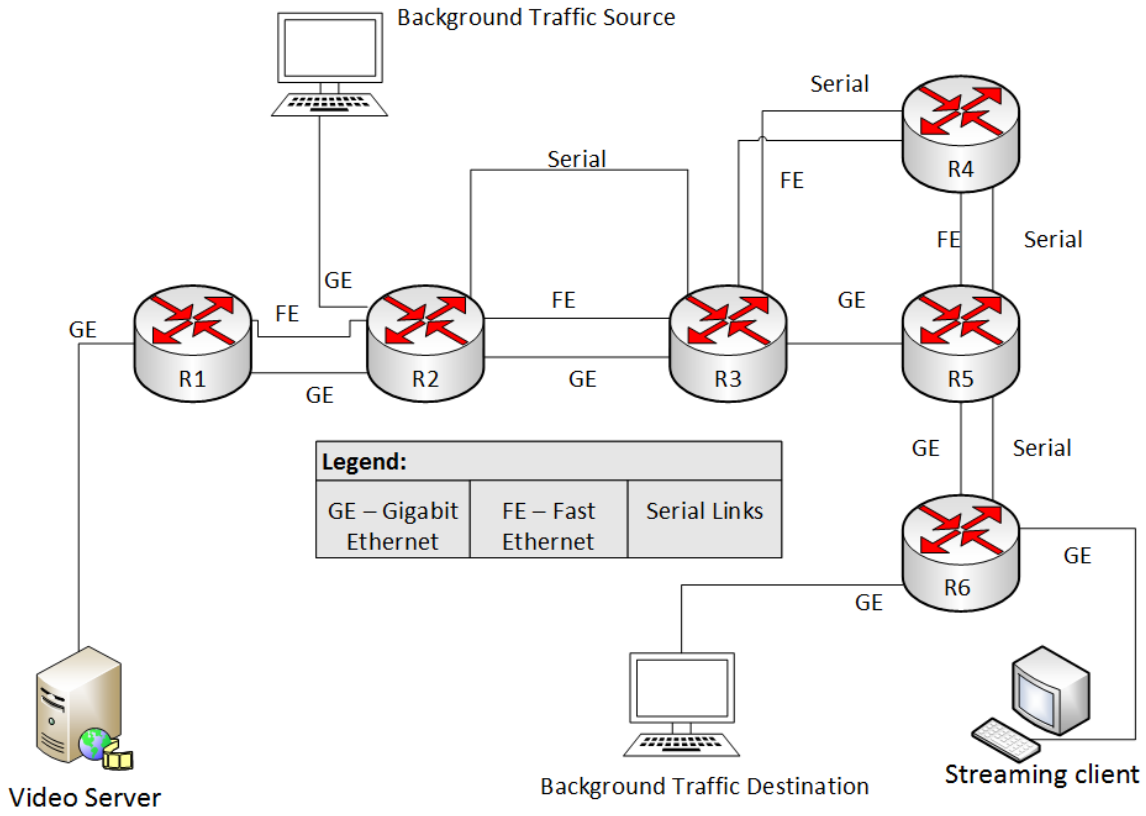

Fig. 4.2 A video server streams H.264 video over RTP to a streaming client over a topology which consists of 2911 ISR routers. Different link technologies (Gigabit Ethernet, Fast Ethernet and Serial) are used to create a range of different physical transport media, which in turn cause variability in instantaneous jitter values. Ostinato is used to create congestion on different paths through the network.

periods of the time series shows an exponential decay which corresponds to the time-varying, periodic network transit times resulting from different congestion levels.

Using the H.264 codec, jitter measurements are collected for a 10.34 minutes run of the streaming session. In this experiment, the streaming session is divided into three parts of 3.44 minutes ($\approx$ 206 seconds) intervals. During the first 3.44 minutes into the streaming session, routing changes are invoked that force the video streams mainly through the GE links. In the next 206 seconds afterwards, i.e. minutes 3.45 - 6.88 minutes, routing changes are initiated that force the streaming session over the FE and serial links. In the last run, the

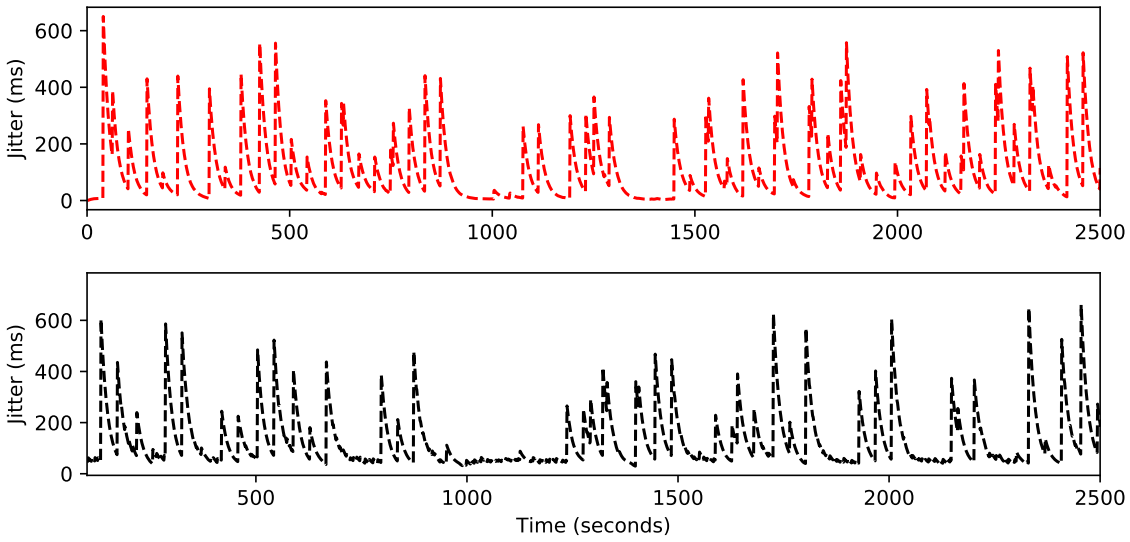

Fig. 4.3 In Rows 1 and 2, I illustrate the adaptive behavior of the H.264 and H.265 codecs in response to network congestion. The maximum, minimum and mean instantaneous jitter values in R1 and R2 are (650ms, 1.17ms, 107ms) and (750ms, 3.91ms, 126ms) respectively.

video stream traverses the serial links only. Ostinato is used to introduce these background traffic for the entire streaming session.

The time series shown in Fig. 4.4, R1 illustrates the entire jitter measurement captured during one run of the video streaming session. This figure is reproduced from [224] with the permission of the authors. Three levels of congestion are experienced by the video packets during this video session. R2 illustrates 200 jitter measurements which cover low, medium, and high levels of congestion. The characteristics of the traces are similar to the ones shown in Fig. 4.3. The characteristic nature of time series data which decays exponentially within the periods can be seen from the plot. This is the result of the adaptive codec trying to match playout with the observed network conditions. In addition, differing peaks of different parts of the time series (in R2) can be attributed to buffering delays or out-of-order packet arrivals [225], which in this case is due to different levels of background congestion.

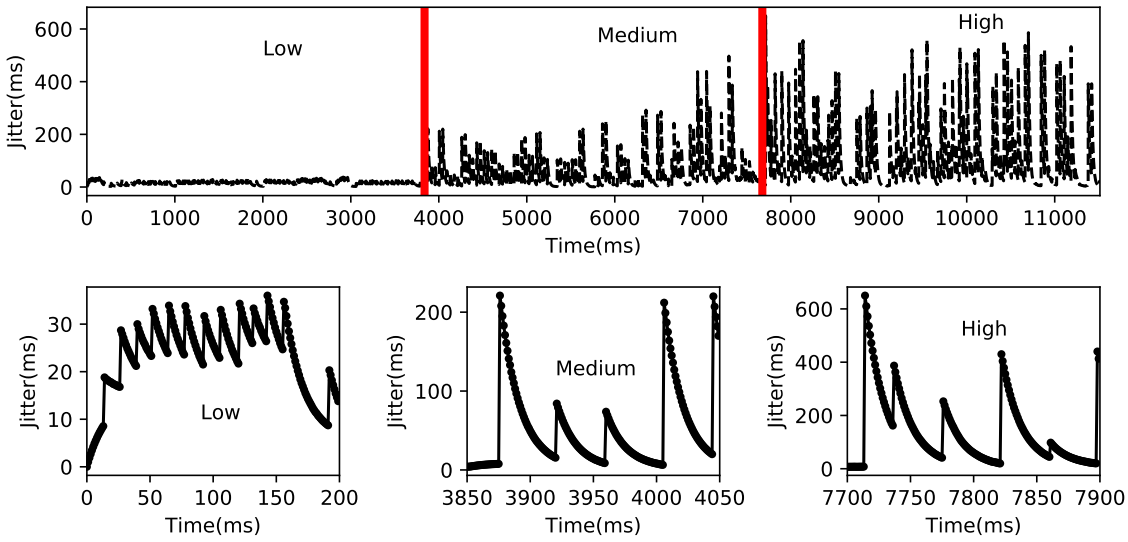

Fig. 4.4 Rows 1 (R1) and R2 illustrate traces depicting the adaptive behaviour of the H.264 codec in response to network congestion. The minimum, maximum and mean instantaneous jitter values for the curve peak in R1 for the low congestion area ($1 < t < 3838ms$) are (12.18ms, 37.53ms, 23.89ms); times $3839 < t < 7677ms$ represent the medium congestion area with minimum, maximum and mean instantaneous jitter values of (15.73ms, 295.05ms, 125.81ms) respectively; times $7678 < t < 11514ms$ represent the high congestion levels with minimum, maximum and mean instantaneous jitter of (27.61ms, 649.38ms, 254.79ms). R2 focuses on 200 measurements of observed jitter during low, medium, and high congestion.

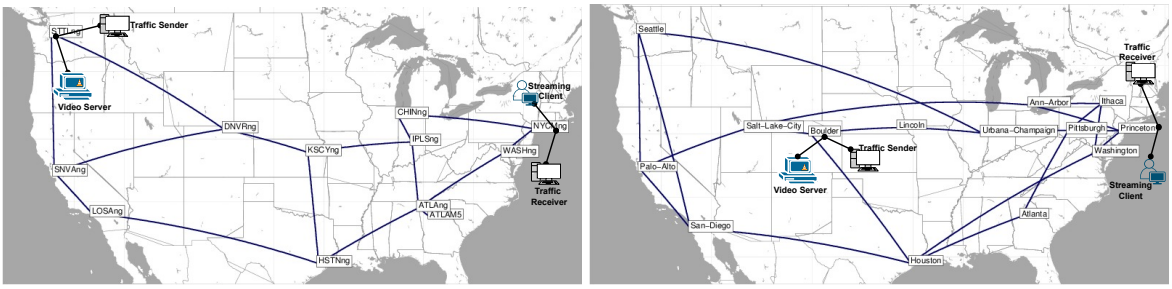

Fig. 4.5 The Abilene topology (LHS), connects eleven regional sites across the United States; the Nobel reference network topology (RHS) originated from the European project NOBEL. Both topologies were obtained from SNDlib.

4.2.2 SDN Network Topologies

In order to investigate the efficacy of the CNAA predictor and classifier models in SDN environments, the Mininet emulator is used. Mininet is a container-based emulator that

may be used to generate a realistic virtual network on a single system by running actual kernel, switch, and application code. Fig. 4.5 depicts real-world network topology examples produced from the SNDlib which are adopted for the evaluation of the CNAA models. These SDN toplogies were adapted from the published works in [226, 227]. The Abilene topology, (LHS), with 12 nodes and 15 edges is a real-world topology which consists of eleven regional sites spread across the United States. The Nobel-US network topology, (RHS) with 14 nodes and 21 edges have been chosen to represent the realistic core network topology. In each topology, two hosts are set up as server and client, which are required to launch the video streaming session. A random path with a length equal to the network diameter for each topology such that the client and server hosts form the two ends of the path was selected. The video streaming sessions between the client and server machines are established using the VLC server as was the case in the laboratory set up. The Big Buck Bunny mp4 video, transcoded using the H.264 codec was used for the streaming session.

Similar to the laboratory test-bed implementation, some interfering traffic is introduced during the session. The Distributed Internet Traffic Generator (D-ITG) [228] was used to create realistic packet-based network traffic by properly emulating real-world traffic and existing Internet applications. Two additional nodes, H3 and H4 are attached in each topology to serve as the traffic source and traffic receiver respectively. The introduction of the background interfering traffic is automated with a script. The D-ITG traffic generator was set up in a multi-flow mode with UDP flows having constant inter-departure time between packets. The traffic flows start with 200 packets/s for about 3 minutes, then builds up to 500 packets/s for the next 3 minutes of the session. The remainder of the streaming session for the 10.34 minutes video file is set at 700 packets/s of interfering UDP traffic. This configuration is chosen in order to generate three different levels of congestion as shown in Fig. 4.4. The network jitter statistics are captured using the Wireshark network protocol analyser. Using

the Linux traffic control[1] (tc) command, the bandwidth usage of a streaming session is limited to 10MB, 15MB and 20MB in order to simulate a range of Internet connections.

The POX controller [229], a Python-based open source SDN controller, is adopted for the video streaming sessions. The POX controller offers a range of APIs that can be used to implement different network applications. POX is chosen because it is ideal for research purposes, comes bundled together with the Mininet system, as well as convenient for quick prototyping. Data plane elements are programmed using the controller's instructions via the standard OpenFlow protocol [230]. The NetworkX tool [231] was adopted to process the network graph in the Mininet emulated topologies. NetworkX is a Python package for examining the structure of networks, creating graphs, manipulating graphs and calculating metrics, such as the shortest path between nodes.

Fig. 4.6, Row 1 shows a plot of an extract of the jitter time series data captured using the Abilene topology with the bandwidth set to 10MB. Three levels of congestion are observed for the time series data similar to the traces shown in Fig. 4.4 taken from the laboratory test-bed. Similar to the traces taken from the laboratory setup, there is a periodic component in each of the time series data across the three level of congestion captured from the SDN test-bed. The jitter measurements of each time series shown in Fig. 4.6, R2, decay exponentially during each of these periods due to the adaptive behavior of the video codec in response to congestion. Finally, the respective jitter peak values, of the different regions of the traces depicted in R2 differ, as shown in the lab traces. As noted earlier from the laboratory measurements, jitter statistics can increase in response to buffered data or distorted packet arrival times. In this case, this is due to the varying levels of interfering traffic or congestion introduced in the network during the streaming session.

[1] https://tldp.org/HOWTO/Traffic-Control-HOWTO/intro.html

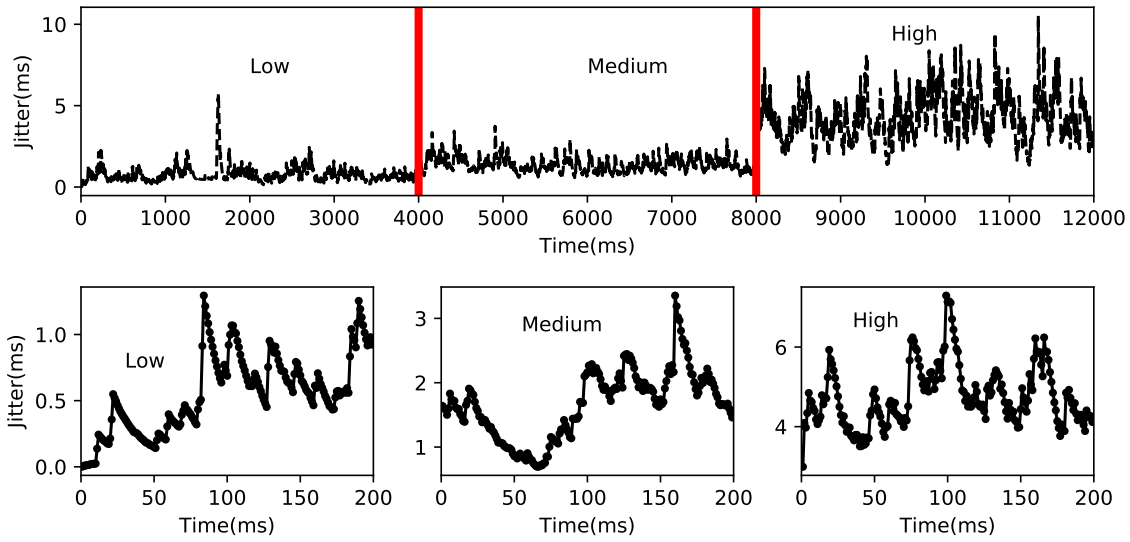

Fig. 4.6 Rows 1 (R1) and R2 illustrate traces depicting the adaptive behaviour of the H.264 codec in response to network congestion for the Abilene 10MB topology. The plots in R2 shows 200 measurements of the jitter measurements during the low, medium and high congestion periods.

4.3 CNAA Jitter Model

This section presents the CNAA model for jitter prediction and network state monitoring. The need for a new jitter representative model is made by highlighting the failure of traditional linear ML models in estimating jitter given the characteristics nature of the traces. The analysis in this section also shows that ML models based on moving averages fail to accurately model jitter in these scenarios.

The mathematical formulation for the jitter time series, $x[n]$, as proposed in [224] is described. The authors divide the time series data into time component regions where the jitter can be described by decaying exponentials. This process is represented with a synthetic jitter time series illustrated in Fig. 4.7. In Fig. 4.8, the complete time series in Fig. 4.7 is divided into its component parts. The amplitude for each function is captured by a rectangle window, $r[n]$, in this case, P time indices,

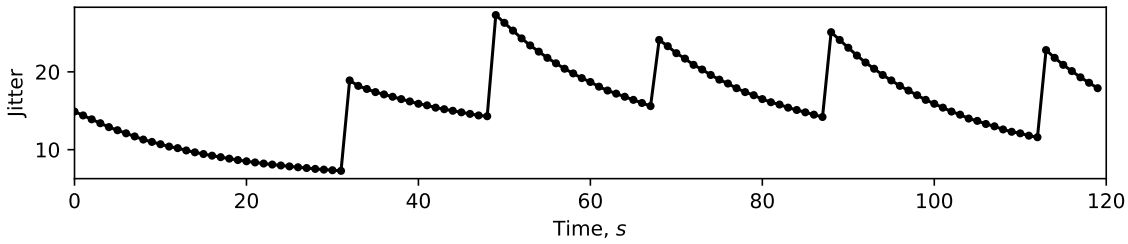

Fig. 4.7 Synthetic jitter time series.

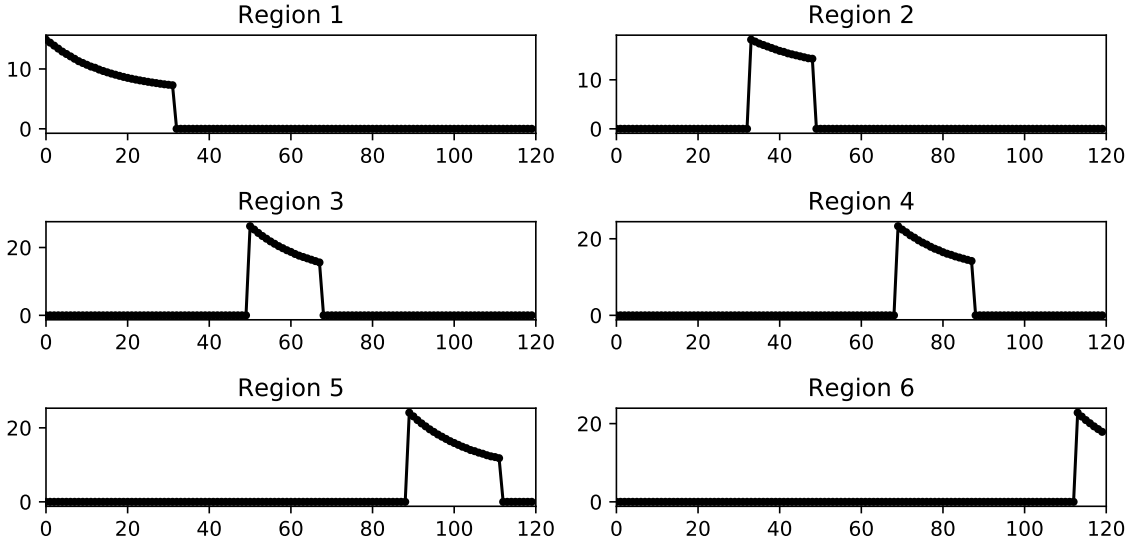

Fig. 4.8 The composite parts that make up the synthetic jitter trace.

$$r[n] = \begin{cases} 1, & 1 \leq n \leq P \\ 0, & \text{otherwise.} \end{cases} \tag{4.1}$$

The authors of [224] demonstrate that each of the component decaying exponential shown in Fig. 4.7 is centered at different time indices, so the rectangular window is shifted to capture its amplitude. For time, i, in the time series, $x[n]$, the i-th decaying exponential is defined by the window,

$$r[n-\omega_i] = \begin{cases} 1, & \omega_i + 1 \leq n \leq \omega_{i+1} \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

The i-th region in the time series $x[n]$ is represented by the set of observation indices,

$$\Omega_i = \{\omega_i \leq n \leq \omega_{i+1}\}. \tag{4.3}$$

These regions are characterized by their respective periods. The period of the i-th region is defined as

$$P_i = \omega_i - \omega_{i+1}. \tag{4.4}$$

The periods in each region of the jitter time series, changes based on the level of congestion in the network (cf. Fig. 4.4, R2). The data is represented using the exponential function,

$$f[n] = be^{-n\lambda} \quad n \geq 0. \tag{4.5}$$

The function, $f[n]$ is its own derivative and any exponential b^x can be written using the Euler's expression as $b^x = e^{x\log_e b}$. For the ith region, the peak of the amplitude of each $f_i[n]$ is b_i. The jitter data's decay rate is represented by λ_i. The function is undefined for $n < 0$. In each of the regions, Ω_i, in the jitter dataset, a sub-dataset (with origin shifted time) is used or a truncated decaying exponential of the form

$$f_i[n] = \begin{cases} r[n-\omega_i] b_i e^{\lambda_i (n-\omega_i)}, & \text{if } n \in \Omega_i \\ 0, & \text{otherwise.} \end{cases} \tag{4.6}$$

According to the authors of [224], it turns out that the jitter time series can be described by the sum of segmented, decaying exponentials:

$$x[n] = \sum_{i=1}^{I} f_i[n], \tag{4.7}$$

where I is the count of regions in the observed N samples of the jitter trace.
Using Equation 4.7, a model of a variety of the adaptive reactions of the video codec occurring during the streaming session in response to time-varying levels of congestion in the network can be realized. Equation 4.7 summarizes the jitter time series data with the feature set

$$\{b_i, \lambda_i, P_i\}, \tag{4.8}$$

where the peak value b_i, the decay rate, λ_i and the period P_i, are the result of the adaptive nature of the codec in response to different network conditions.

A summary of the interpretation of the feature set of the CNAA jitter model is as follows:

- b_i - As the network transit times increases, the b value increases. We expect large b values during times of high network congestion and vice versa. An increase in network congestion is a result of increased network delays.
- λ_i - Large λ_i values lead to more rapidly decaying exponential curves. In situations of low congestion, we expect the exponential to decay faster.
- P_i - For high congestion regions, the period is large; and when the period is small, the congestion is low.

4.4 Parameter Estimation

In this section, a description of the parameter estimation process for the model introduced in Equation 4.7 is presented.

4.4.1 Period Estimation with Auto-Correlation Function (ACF)

The period can be estimated using the Auto-Correlation Function (ACF) [232]. The ACF is a common tool for detecting the period in a signal. It is a measure of the statistical dependence between values of a time series data at different times, and summarises its time-domain structure. ACF, also known as serial correlation calculates the similarity between time series data as a function of the time lag between them. The ACF can be used as a tool for finding repeating patterns, such as the presence of periodic signals in time series analysis. A periodic signal's ACF sequence usually has the same cyclic features as the signal itself. As a result, the ACF is used to identify the existence and duration of cycles [233]. For periodic signals, we expect the autocorrelation to peak when the lag, l matches the period. The time series $x[n]$ shows a trend as illustrated in Fig. 4.4. In order to obtain accurate estimates of the period, the time series $x[n]$ is denoised by applying a low-pass filter and removing the trend, $y[n]$, = lpf($x[n]$),

$$\hat{x}[n] \leftarrow x[n] - y[n]. \tag{4.9}$$

The smoothed signal is then shifted along a time lag and the correlation is computed. After a certain time lag, the correlation spikes as shown in Fig. 4.9, R1. The peaks indicated by the red circles in the figure reveal the periods, P_i and the associated time lags on the x-axis.

4.4.2 Estimating the b and λ

To estimate the model parameters for the ith region of the time series data, the model parameters defined in Equation 4.8 are designated as b_i and λ_i and P_i. The following steps are taken to determine these parameters:

1. Using the most recent period estimate p, the start indices of about the last three functions, $f_i[n]$ is located. The set of the starting indices are denoted as $\{\hat{\omega}_{i-2}[n], \hat{\omega}_{i-1}[n], \hat{\omega}_i[n]\}$, where i represents the index of the function in which the nth time index occurs.

2. Each function is separated and origin-shifted. Then, each individual curve is shifted back to the origin, $n = 0$ using the starting indices. A fresh set of time indices is then assembled $\hat{n}[n]$ so that the time indices of this time series is the same as the shifted function.

3. A two parameter exponential model is used, $x[n] = be^{\lambda\hat{n}[n]}$, to estimate the parameters b and λ. A pair of values for the estimations of $\{x[n], \hat{n}[n]\}_{n=\hat{\omega}_{i-2}}^{\hat{\omega}_i}$ is used.

   To obtain a linear expression of $x[n]$, the natural log of both sides is computed

$$\log x[n] = \log b + \lambda\hat{n}[n] = \hat{y}[n] = \hat{b} + \lambda\hat{n}[n], \tag{4.10}$$

   and introduce the notation $\hat{y}[n] = \log x[n]$ and $\hat{b} = \log b$. The least squares objective is minimized

$$E = \sum_{j=\hat{\omega}_{i-2}[n]}^{\hat{\omega}[n]} \left(\hat{b} + \lambda\hat{n}[j] - \hat{y}[j]\right)^2 \tag{4.11}$$

   by computing the partial derivatives of E with respect to $\hat{b}$ and λ and setting them to zero.

4.5 CNAA Predictor Model

This section describes the CNAA predictor model fitting process. To compare the performance of the predictor model, some baseline ML models are trained using the jitter time series data. The baseline models compared with the CNAA predictor model are LR, Simple

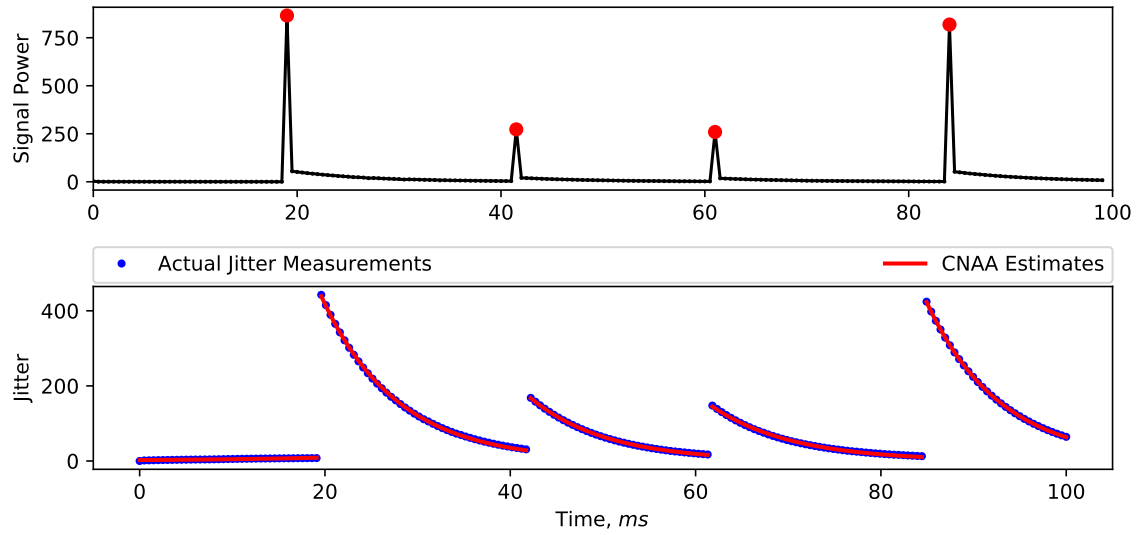

Fig. 4.9 Precision of the CNAA Jitter Predictor Model. The CNAA predictions are plotted over the true jitter values showing accuracy in the model's estimations in R2. The plot in R1 shows the period detection mechanism illustrating the points where the peaks occur (or the inflection points) signalling the period locations in the signal.

Moving Average (SMA), AutoRegressive-Moving Average (ARMA) and the Exponential Weighted Moving Average (EWMA).

4.5.1 Split Data and Fit CNAA Predictor Model to Independent Curves

To examine the suitability of the CNAA predictor model, the first 100 samples from the jitter time series is extracted. The first step is to figure out how long each curve in the dataset lasts, i.e. the period, P_i. The ACF described in Section 4.4.1 is used to estimate the periods. The next step is to split the data using the estimates of the period. Then, the single curve fit is applied by shifting the data the to origin using the process described in Section 4.4.2 (Equation 4.11).

Fig. 4.9, R2 illustrates the accuracy of the CNAA predictions plotted against the true jitter time series data. In R1, the plot shows the period detection process indicating the peak or inflection points which illustrate the period time locations.

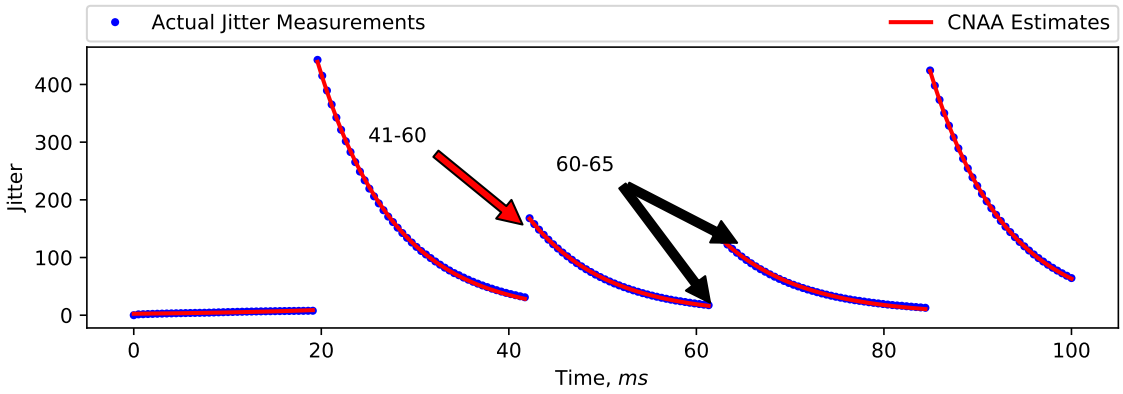

Fig. 4.10 Prediction of the jitter measurements for time indices 60 - 65 will rely on the captured jitter measurements for 41 - 60; the plot explores jitter predictions from historical values which can determine instantaneous estimates of jitter.

4.5.2 Estimating the Parameters from Historical Data

The suitability of the CNAA jitter predictor model at estimating the jitter time series data has been demonstrated. Next, jitter predictions from on historical jitter data is explored in this section. As an example, if the task was to predict timestamps between 60 and 65 in Fig. 4.10, this could be achieved with a model that incorporates the timestamps from 41 to 60. Here, an examination of an approach that is able to learn the model parameters for a range of different jitter time series curves is explored. This section explores jitter estimation and prediction models based on historical data values.

With this approach, some curves are overlaid on the time axis and the best fitted model based on all data points is computed. Specifically, the parameter estimates could be used to determine what the jitter estimates would be between 60 and 65. It is reasonable to assume that the timestamps from 0 - 10, will reflect the current network conditions necessary to predict the jitter statistics between timestamps 11 - 23. Additionally, the timestamps 24 - 30, will provide information about 35 - 40. Fig. 4.11 illustrates a model fit obtained for three curves superimposed on the time axis. The thick blue line corresponds to the fit obtained, which includes the relevant information to estimate all jitter values.

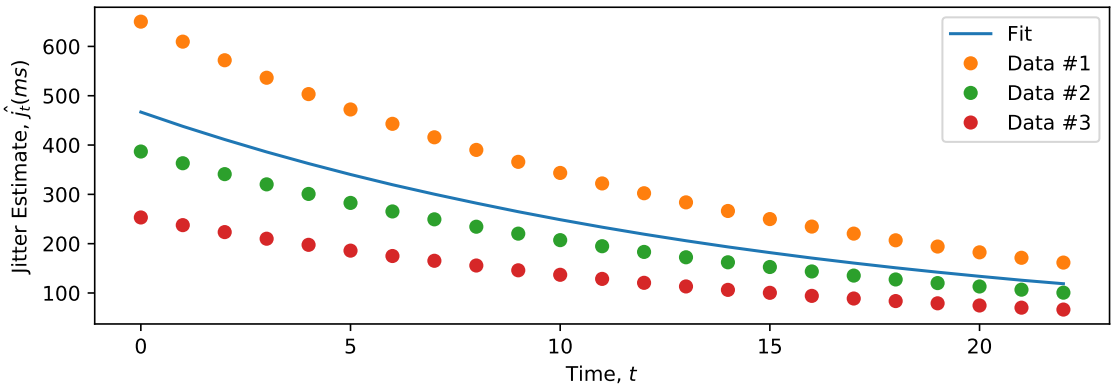

Fig. 4.11 This approach relies on historical data for parameter estimation and prediction of model parameters.

4.6 Review of Baseline ML Models

This section presents a review of the baseline ML models adopted for performance comparison with the proposed CNAA predictor model. The baseline ML models used in this chapter include the LR (introduced in Section 3.4.1), the Simple Moving Average (SMA) model [234], the Exponential Weighted Moving Average (EWMA) model [235], and the AutoRegressive-Moving Average (ARMA) model [236].

The jitter measurements captured with WireShark as a time series is of the form: $j = (j_1, j_2, \ldots, j_n)$, a sequence of data points measured in equally spaced time intervals, where $j_t \in \mathbb{R}$ represents an element at time t (t = 1, 2, ..., n), such that $1 \leqslant t \leqslant n$ (where n is the length of the time series data). The task is to predict jitter at time t, j_t using the ML models introduced in this section.

4.6.1 Simple Moving Average (SMA)

Moving averages are a straightforward and widely used smoothing technique in time series analysis and prediction. Calculating moving averages requires the specification of a window size, or window width. This specifies the number or the size of the subset of actual observa-

tions which will be used to compute the moving average value. The "moving" aspect of the moving average refers to how the window indicated by the window width is shifted along the time series to compute the new series' average values.

It is calculated as the unweighted mean of the past k data points. For example, given the time series data, j, the model is defined as

$$\begin{aligned} SMA_k &= \frac{j_{n-k+1} + j_{n-k+2} \ldots + j_n}{k} \\ &= \frac{1}{k} \sum_{i=n-k+1}^{n} j_i \end{aligned} \tag{4.12}$$

4.6.2 Exponential Weighted Moving Average (EWMA)

There are some limitations in the predictions realized using the SMA. The SMA will always lag by the size of the window, and it cannot reach the full peak or valley of the data due to its averaging. Furthermore, the SMA can be significantly biased by extreme historical values. The EWMA reduces the lag effect observed in the SMA and assigns more weight and significance on recent measurements. In doing so, the EWMA is set up in a way that older observations are weighted less heavily. The formula to compute the EWMA at the time t is

$$EWMA_t = \begin{cases} x_0 & \text{if } t = 0 \\ \alpha j_t + (1-\alpha) EWMA_{t-1} & \text{if } t > 0 \end{cases} \tag{4.13}$$

where j_t is the value of the observation at time, t and α is the smoothing factor. The smoothing factor, α is defined for the range $0 < \alpha \leq 1$. In the computation of the EWMA, the smoothing factor defines how relevant recent observations are. The greater the α value, the closer the EWMA follows the original time series.

4.6.3 Auto-Regressive Moving Average (ARMA) Model

The ARMA is a combined model comprising of the Auto-Regressive (AR) and Moving Average (MA) model for prediction. In time series analysis, Auto-Regression and Moving Average (ARMA) models are used to describe stationary time series [237]. The properties of a stationary time series such as its mean, variance are independent of the time at which they were observed [238]. For such processes, the ARMA model is defined as follows

$$j_t = \sum_{i=1}^{p} \varphi_i j_{t-i} + \varepsilon_t + \sum_{j=1}^{q} \theta_j \varepsilon_{t-j}, \tag{4.14}$$

where $\varphi_1, \ldots, \varphi_p$ are the autoregressive parameters to be estimated, $\theta_1, \ldots, \theta_q$ are the moving average parameters to be estimated, and $\varepsilon_1, \ldots, \varepsilon_t$ are a series of unknown random errors assumed to be normally distributed [239]. This is known as an ARMA(p,q) model.

4.7 Model Evaluation and Analysis

This section describes the model fitting procedure adopted for the baseline ML models. The performance of these baseline models is examined using the same dataset and under the same conditions as described in the CNAA model fitting section. The performance of the baseline ML models are compared with the CNAA predictor model. The model performance are evaluated using the RMSE and the Mean Absolute Error (MAE). The absolute error refers to the difference between the predicted jitter values and the true values captured via Wireshark. MAE provides a measure of the magnitude of errors for a series of predictions and observations by taking the average of the absolute errors for the series. MAE is computed as

$$\frac{\sum_{i=1}^{n} |\hat{y}_i - y_i|}{n}, \tag{4.15}$$

where y_i and $\hat{y}_i$ is the actual data point and prediction for the i-th observation respectively.

An automated grid search using Python's forecast tool, Pmdarima [240], is used to identify the (p,q) order of the ARMA model to apply on the time series data. An ARMA (2,1) model is suggested by the tool. The Partial Autocorrelation Function (PACF) and the ACF are used to confirm these model suggestions. The ACF of a time series data explains how the current value of the series compares to previous values. The PACF, rather than seeking correlations between the current data point and the lags as ACF does, looks for correlations between the residuals and the next lag value. In the event that the residual contains any latent information which can be modeled by the next lag, then a good correlation may be obtained that includes the lag in the model [241]. The PACF can indicate the recommended AR(p) order, whereas an ACF plot can reveal the MA(q) order [242]. To identify the order of the ARMA parameters, the difference in the lags for both functions are examined. If the graphs reveal a clear drop after a certain amount of lag, that indicates the parameters of the ARMA model. For example, if the ACF curve drops significantly after the first lag, then, that is an indication that a model with one MA component should be considered (MA(1)). Similarly, suppose the PACF graph reveals a significant drop-off after the 2nd lag, that would indicate a AR(2) process. The findings in using the PACF and ACF to compute the order of the ARMA model suggests that ARMA model (2,1) might be the best fit as proposed by the Pmdarima tool.

The evaluations begin with the SMA and EWMA baseline models by setting the sliding windows for both models with the period estimates derived from the data. The resulting predictions are shown in Fig. 4.12. The periods are then halved, and both algorithms are re-run. In this case, this is an attempt to try to incorporate some historical data into the model predictions. In the time series data shown in Fig. 4.9, R2, timestamps 0-39, reducing the window integrates some previous data from 0-19 in calculating timestamps 20-39. Fig. 4.13 depicts the resulting predictions.

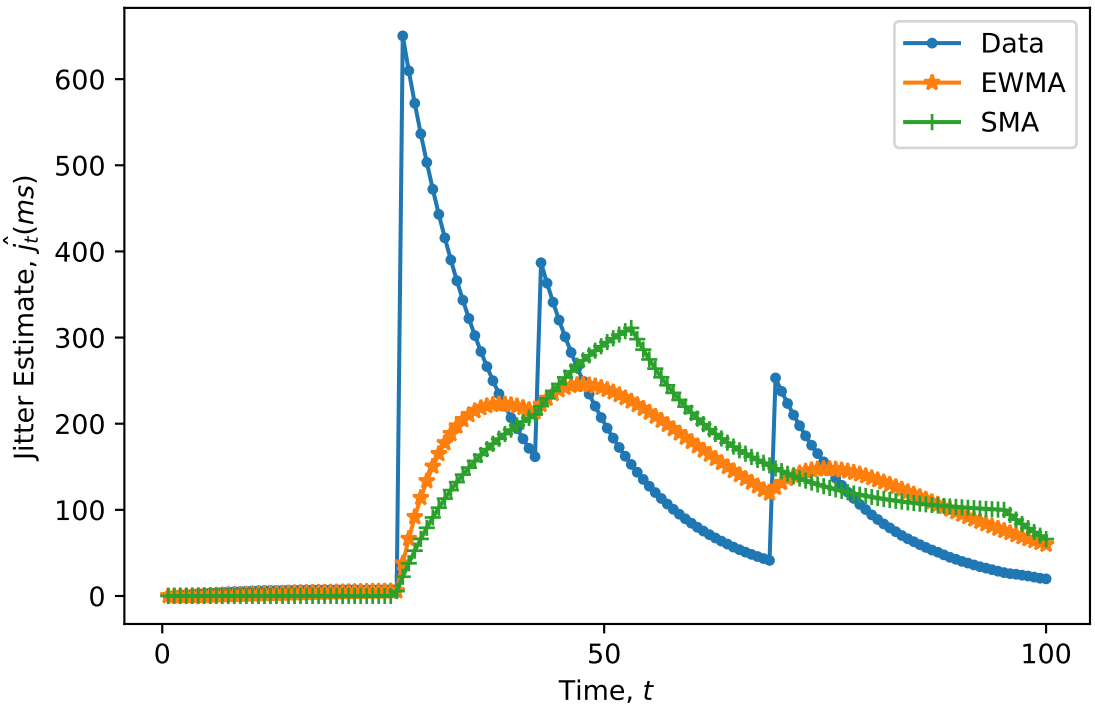

Fig. 4.12 Comparison of the baseline models, EWMA and SMA, to the actual jitter data. EWMA performs better than SMA. However, both algorithms, compared with CNAA, offer very poor predictions.

LR models are unable to capture the network dynamic, the time-varying periodic data, and the data shape. The LR model takes a weighted sum of the previous jitter values to estimate the next jitter value. The weights are selected to give good estimates, based on criteria such as the sum of the squares of the error. For short time durations, the data behaves like a falling exponential (Fig. 4.14, Row 1). As a result, this dataset can only be adequately modeled with a linear model over a short time horizon, as illustrated in Fig. 4.14.

The LR estimator is ineffective for this time series because the data has points of inflection. There are huge jumps in the time series when one falling exponential zone ends and the next begins. The ability to change directions is not available in a linear model. This is depicted in Fig.4.14, Row 2. There is a large jump in the function at time index $n = 20$ where one region ends and another begins. A linear fit based on the slope computed during the range of points

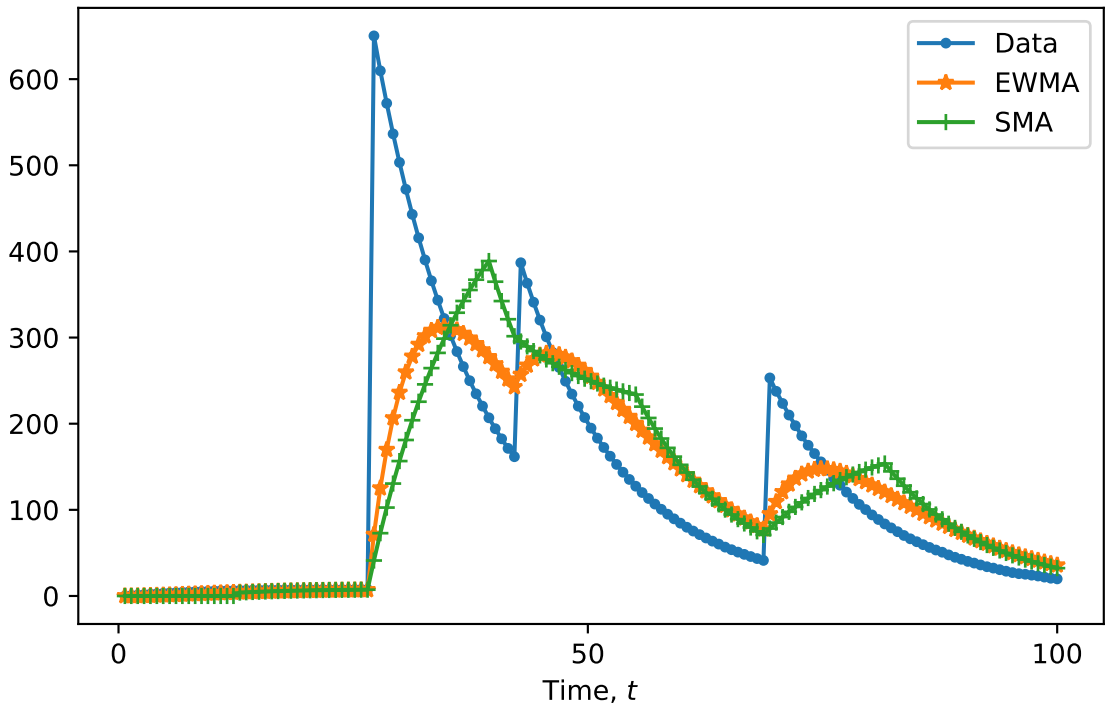

Fig. 4.13 The EWMA and SMA predictions are improved by including network dynamics in the models. There is ≈ 19% and 15% improvements for EWMA and SMA respectively.

$15 \leq n \leq 20$ is shallow. The slope from the first sample of the exponential function, starting at $n = 20$, produces a steep linear fit. Fitting a line through all the points results in a linear fit which is not a good approximation for the entire set of points. Fitting a linear function to all these points is not a good solution. Fitting an exponential curve to all the points is also ineffective.

The running average estimator is explored as a solution to predict the jitter statistics. Consider taking the average of an exponential's first six indices to predict the seventh number. The first six values are $x[n]_{n=0}^{5} = [1, 0.7270, 0.5286, 0.3840, 0.2794, 0.2031]$. The mean of these values is 0.5203, which is bad estimation of $e^{-6} = 0.1477$. Running averages are unlikely to yield good jitter estimates.

The performance of the CNAA predictor along with the baseline ML models measured in terms of the RMSE and MAE are listed in Table 4.1. The CNAA RMSE and MAE is

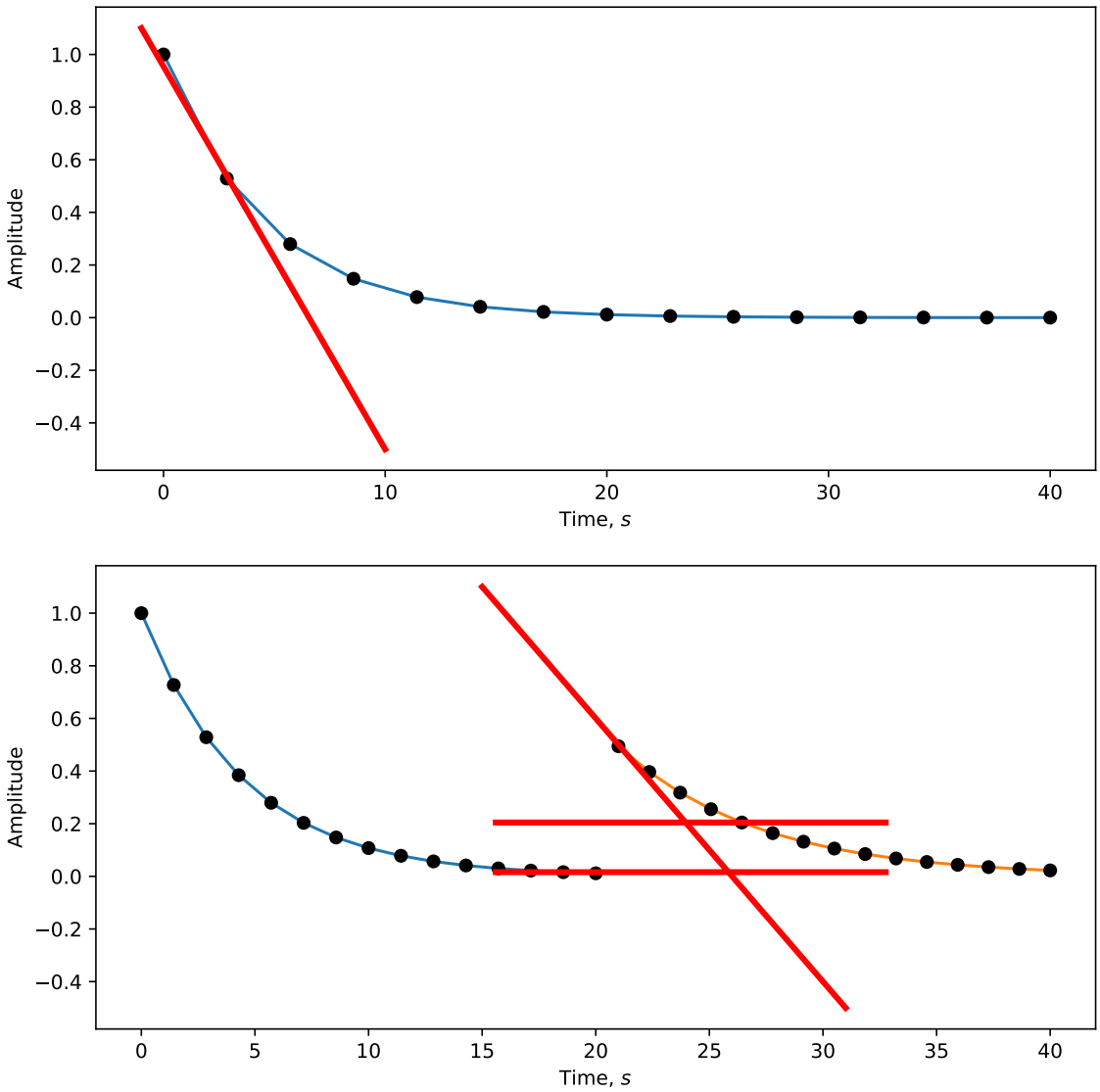

Fig. 4.14 Linear approximations of jitter time series yield inaccurate estimates of jitter values as they are not flexible enough to capture the characteristics of jitter time series.

≈ 95.8% and 93.8% better respectively over the best performing baseline model, EWMA. ARMA with a RMSE of 119.16 and MAE of 89.76 is approximately 114.34 and 84.94 packets less accurate than CNAA. The SMA is the worst performing baseline model with a RMSE difference of about 134 packets. The CNAA model records the best prediction because the model takes into account the network dynamics which are a result of the codec adaptive response to congestion in the network. In an attempt to incorporate some of the

Table 4.1 Codec-aware Network Adaptation Agent (CNAA) Versus Baseline Models; compared with the best performing baseline model, EWMA, CNAA predictor offer ≈ 95.8% and 93.8% performance improvements in terms of RMSE and MAE respectively.

Method	RMSE	MAE
CNAA	**4.82**	**4.07**
EWMA	115.84	65.50
SMA	138.88	81.24
LR	119.66	91.92
ARMA(2,1)	119.16	89.76

network dynamics (the period estimates) in the SMA and EWMA, as seen in Fig. 4.13 the period estimates are halved to incorporate some historical information in the sliding windows of both algorithms. The predictions realized by both algorithms around the inflection points vary significantly from the actual measurements. This is due to the inability of the algorithms to distinguish these time indices as points in different exponential regions. However, incorporating some network dynamics via halving of the period estimates improves the prediction performance of both the SMA and EWMA algorithms as shown in Table 4.2.

Table 4.2 The performance of the SMA and EWMA algorithms are boosted by incorporating some network information in the models. Improvements in predictions of ≈ 19% and 15% are realized.

Method	RMSE	MAE
EWMA	93.71	49.48
SMA	113.16	62.41

The CNAA predictor model has been shown to achieve accurate jitter predictions, in the face of congestion and adaptive codecs. Next, an examination of the performance of the CNAA model by varying the data rates of the streaming session is explored. Different transcoding scenarios are simulated by varying the bitrate of the codecs during the streaming session. Two datasets are generated for two different streaming sessions. The bitrates for the two datasets is set to 56 kbps and 1000 kbps respectively. The video frame rates are set to 24 frames per second.

The 1000 kbps trace begins slowly, with many minor spikes and brief pauses. In general, there are several fluctuations in the system which are the reactions of the codec to the interfering background traffic and higher data rates. The 56 kbps trace follows the same pattern as the previous traces. The data starts slowly, then surges, while the network transit times between the curves preserve some periodicity.

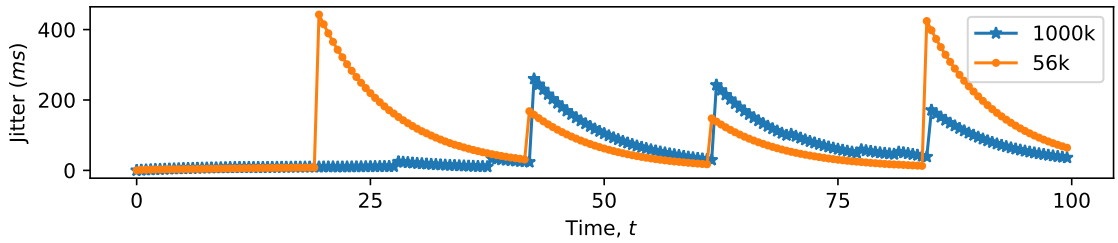

Fig. 4.15 Variations of the decoding parameters were made to produce varying levels of instantaneous jitter by setting the video bitrate to 56kb/s and 1000kb/s, respectively.

For the 56 kbps dataset, the CNAA predictor model achieves good prediction accuracies. The jitter estimates generally match the actual values very well. However, the CNAA predictor model fails to correctly predict the slow and almost negligible periodicity at the beginning of the 1000 kbps dataset. There are around $3-4$ curves virtually overlapping on each other between time indices 65 and 85. Instead of the expected independent curves, the CNAA estimates given here attempt to approximate all data points as one. Table 4.3 lists the performance of the CNAA predictor model for these two datasets.

Table 4.3 CNAA prediction performance in varying congestion levels and bitrates. Higher data rates in the face of congestion lead to more variablity in the system. The CNAA model is able to yield accurate jitter predictions under the system perturbations.

Model	RMSE	MAE
56 kbps trace	4.61	3.90
1000 kbps trace	14.93	12.31

Experimental evaluations have shown that the CNAA is the model of choice for predicting, jitter, for adaptive video codec sessions. The model is able to deliver improved prediction

performance and overcome the constraints of some baseline ML models by incorporating network dynamics into its learning process. The performance of the model was evaluated under varying video data rates and congestion levels. The performance of the CNAA model allows for accurate predictions of the QoD metric, jitter, without requiring additional computational resources. In the next section, the functionalities of the CNAA model is extended to network monitoring and management.

4.8 Using the CNAA Predictor Model Estimates for Network State Classification and Monitoring

Network state or traffic classification is a core requirement for network operators to achieve a variety of network operation and management activities. Some of these activities include detecting anomalous network patterns, capacity planning, QoS planning and provisioning, network service differentiation, network performance monitoring, intrusion detection [216]. The authors of [243] hypothesized that by gathering video flow data from a live network and analyzing packet arrival time, transport protocol, packet size, and other data, they could investigate the internal transmission characteristics and protocol transmission characteristics of video flows and propose a set of features for video traffic classification. The authors of [244] in their work on network state classification considered the use of the RTT as a proxy to determine conditions of the network. In their work, the authors evaluated two baseline approaches on the use of RTT statistics for inferring the network state. The first method, described in [245], takes advantage of RTT statistics over the course of a data transmission and provides a network state classification model. The second method is described in [246], in which the TCP sender is updated (i.e., Adaptive TCP) to assess the degree of congestion by analyzing RTT evolution since data transfer began. In good network conditions, the estimates realized could be used to prevent an unnecessarily low TCP transmitting rate.

This section decribes experiments aimed at examining the suitability of the jitter model features for network state classification. The efficacy of the CNAA predictor model at realizing accurate jitter predictions has been demonstrated. The proposed predictor model delivers accurate estimates of the QoD metric by incorporating network dynamics into the learning algorithm. Referring back to Fig. 4.1, the suitability of using the CNAA model for network state classification based on congestion levels and interfering user traffic is examined. To do this, the estimated parameters from the CNAA model, $\{b_i, \lambda_i, P_i\}$,, are used as inputs for network monitoring via traffic state classification. Off-the shelf classification ML algorithms are used on these features; the results demonstrate that they improve congestion level accuracy compared with traditional running average ML models computed on the smoothed jitter statistics [89]. The network responsivity of the CNAA network monitoring classifiers is computed. This is the ability of the system to accurately detect network service change-points. The findings demonstrate that the CNAA classifiers achieve network responsivity of 89% and above when the congestion levels are changing. This section also highlights the shortcomings of traditional classifiers based on the running average of the jitter statistics in comparison with the CNAA technique. The CNAA classifier offers approximately 20% and 34% performance gains in terms of its classification accuracy over the running average classifier for the worst and best-case scenarios respectively.

4.8.1 Machine Learning Techniques

This section briefly introduces the different ML classification techniques adopted for the experiments described here. Two tree-based ML models, namely the RF and DT algorithms, are adopted.

4.8.1.1 Decision Trees (DT)

DTs are a class of supervised ML techniques that classify data based on information gains by computing the entropy of the dataset. The root node will be calculated using the entropy with the highest information gain in the dataset. The process continues to split the tree branches. The internal nodes represent a test on the features and the branches represent the outcomes. The class labels are represented by the leaf nodes at the base of the tree. A 5-fold Cross Validation (CV) technique to find the optimal depth for the DTs is used. The method selects a tree depth of five. The tree depth chosen via CV helps avoid over-fitting and gives a better chance to reproduce the accuracy and generalize the model on test data [247].

4.8.1.2 Random Forest (RF) for Classification

RFs are an ensemble learning method for classification or regression. For classification, the RF algorithm consists of many decisions trees. It uses bagging and feature randomness when building each individual tree to create a forest of trees whose prediction by committee is more accurate than that of any individual tree. Each DT classifies the same problem and the overall decision will be computed by considering the majority vote of the results. For regression problems, the final prediction is the average prediction of each tree [248]. The Scikit-Learn RandomizedSearchCV [249] method is used to obtain the optimal values for the model hyperparameters to avoid over-fitting.

4.8.2 CNAA Traffic Classification Model

This section begins by describing the dataset realized from the CNAA predictor model estimates using the traces shown in Fig. 4.4, R1. This is followed by a description of the classification experiments to examine the performance of the CNAA classifier model.

4.8.2.1 Dataset

The CNAA jitter model estimates and updates the three parameters, $\{b, \lambda, p\}$ as the dataset evolves. These parameters form the input feature space, X. To fix ideas and notation, the dataset as shown in Fig. 4.4, R1 is broken into three classes for the target variable, y. The low congestion area ($1 < n < 3838$), is categorized as class A; the medium congestion area, $3839 < n < 7677$, is designated as class B; and class C represents the high congestion area from time indices $7678 < n < 11514$ of the trace. This procedure is repeated for the jitter measurements gathered from the SDN topologies. The dataset is divided into three distinct regions of congestion as done for the traces collected from the laboratory test-bed above. The outline of the experiments is described in the next section.

4.8.2.2 Batch Experiment

This section presents the batch experimental setup adopted in this thesis. First, the jitter time series data captured from the laboratory test-bed is evaluated and the results presented. Then, the experiments are repeated using the traces captured from the SDN test-beds.

Using the CNAA jitter estimator model, parameter estimates for the traces in classes A, B and C are generated. Each class consists of 3838 observations making a total of 11,514 observations. A labeled dataset of the jitter model parameters for the entire dataset: $\{b, \lambda, p\}$ is generated. The labeled dataset is divided into two parts. The training dataset is composed of 70% of the data with 30% set aside as test data. The test data would be used to validate the performance of the ML models. All the models are trained with the training dataset separately and model accuracies are then calculated using the test dataset. The goal of these experiments is to design a model that can accurately classify the network state data to the correct class category based on the congestion levels.

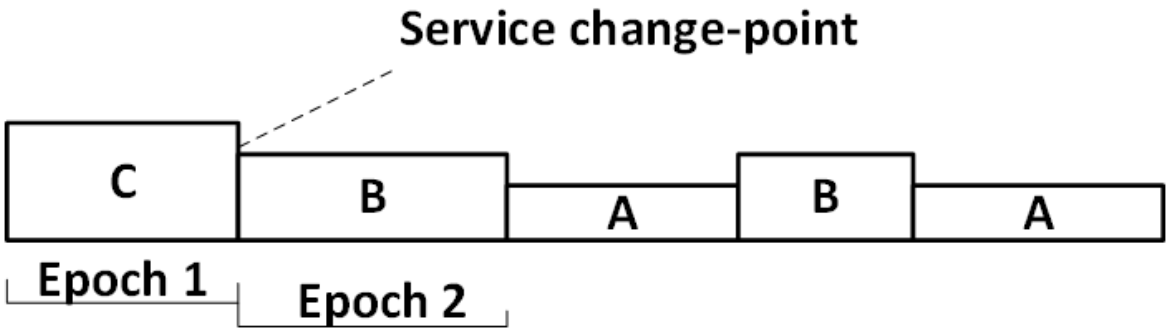

Fig. 4.16 Time-varying system experiments set-up. The responsivity of the system is measured by the time lag between when a service change-point occurs and the time the classifier detects the change in congestion levels.

4.8.2.3 Time-varying System Classifier

Fig. 4.16 illustrates the set-up for the time-varying system experiments. The CNAA model parameter estimates are used as the feature set. The traces are from the varying classes of congestion in the form depicted. First, traces from class C are placed in epoch 1, class B in epoch 2, class A in epoch 3 and so on. The goal is to detect the service change-point and measure the responsivity of the CNAA classifier in response to changing levels of congestion in the system.

The responsivity of the system R, is the time lag between when a service change-point occurs and the time it takes the system to detect and know about the change. Reference Fig. 4.16, assuming that there are 10 timestamps in epochs 1 & 2 represented by classes C and B respectively. The responsivity of the CNAA classifier is computed by how soon the CNAA classifier detects a service change from epoch 1 (class C) to epoch 2 (class B). Does the CNAA classifier detect the service change by the 12th, 15th, or 18th timestamp? The accuracy of the metric is expressed as a percentage of the timestamps in the epochs. The initial simulations contain fixed samples from the various classes in each epoch. The time varying experiments are then repeated with varied number of samples from the different classes to evaluate the effect of the frequency on the sampling.

4.8.2.4 Classifier based on Jitter Running-Average

Using the full dataset shown in Fig. 4.4, R2, the running average of the entire dataset using the most-occurring period value for each corresponding class range is computed. Then, a RF model classifier is trained on the running average estimates of the jitter (i.e. the feature space) and classified against the same three classes [250]. The RF classifier is chosen due to its superior performance over the DT classifier as shown in Table 4.4. Jitter running average estimators predict future values of the jitter statistics as a floating average of previous values. Finding the optimum sliding window is crucial because if the window is too small, then the estimation of playout delay is likely to be poor. On the other hand, if the window size is too large, large memory is wasted for keeping tracks of long and unnecessary history [251].

4.8.3 Numerical Evaluation and Results

The CNAA classifier offers $\approx$ 20% and 34% improved performance gains in terms of its classification accuracy over the running average classifier for the worst and best-case scenarios respectively.

Three results are presented: (1) The performance of the DT and RF classifiers for the batch experiments using labeled data which is generated from the jitter estimator, CNAA model is reported. (2) Results from the time-varying experiments are presented. (3) Finally, the performance of the CNAA classifiers with a traditional classifier based on the running average of the jitter statistics is examined. All models are evaluated with the accuracy metric derived from the models' classification confusion matrix. The accuracy metric is computed thus:

$$M_A(\%) = \frac{T_P + T_N}{T_P + T_N + F_P + F_N} \times 100. \tag{4.16}$$

where T_P and F_P represent True and False Positives; T_N and F_N represent True and False Negatives. In addition to the accuracy metric, the service change-point measurements for the time-varying results are also evaluated with the system responsivity, R.

4.8.3.1 Batch Experiments

Table 4.4 lists the performance of the CNAA classifier with both the DT and RF algorithms for the batch experiments using the jitter measurements taken from the laboratory test-bed. For the DT models, the best classification accuracy which is achieved is 91.67%. This is acheved using the class A data. The CNAA classifier records only 4 misclassification errors with 3 samples wrongly classed as C and 1 sample wrongly classified as class B. The DT classification errors occur with the highest frequency with the class B data. The classification accuracy is 75.68%. Similarly, the best classification accuracy for class A is achieved using the RF algorithm. The accuracy achieved is 97.92%. The worst performance is recorded with the class B traces. The accuracy achieved is 83.33%. This is an improvement of $\approx 8\%$ over the DT model performance. The class C trace also records an improvement in performance with the RF model. The accuracy achieved is 88.89%. This is an improvement of $\approx 10\%$ over the DT model.

Table 4.4 The results of the batch experiments. The RF models outperform the DT models for all classes. The CNAA classifier achieves the best classification accuracy for class A; class B records the worst performance with high classification accuracy of 75.68% and 83.33% for DT and RF models respectively.

Model	Batch Classification	Accuracy (%)
Decision Tree	Class A	91.67
	Class B	75.68
	Class C	78.57
Random Forest	Class A	97.92
	Class B	83.33
	Class C	88.89

These results are quite significant. Fig. 4.17 illustrates the distribution of the peak (b) of the curves, the periods and the decay rate, λ, of the curves. Row 1 shows the distribution of the jitter feature-set $\{b, \lambda, p\}$ for class A. The best classification accuracy for both models is achieved with class A. Evidently, from the histogram for the b parameter, it is easy to notice that in relation to the other classes (Rows 2 and 3, Column 1, (C1)), the data has more symmetry around the mean. There is less variability in the data for this class. The class A data distribution for p and the λ values, (R1, C2 & C3) also exhibit similar characteristics with the b feature. The data has less variability. This explains the high classification accuracies for this data.

More variability within the samples are observed for classes B & C, Fig. 4.17, R2 & R3. For class B, R2 the data values for the b parameter, (C1) indicates a random distribution of the data. There are several peaks among the data with one pronounced peak around the mean value. More importantly, the period between the curves also display noticeable variance and distinct outliers as shown in R2, C2. There are distinct outlier values for the period. The periods in conjunction with the variability in the λ values exhibit much variability for class B. These statistics underline the high misclassification errors in comparison with the other classes. In contrast, it is evident that the periods between the curves for class C exhibit less variability with a few bounded outliers. Class C's decay rate (R1, C3) also shows less variability with almost some negligible outlier values compared with class B's decay rate.

Using the jitter time series measurements gathered from the SDN topologies, the applicability of the CNAA models for SDN environments was examined. Table 4.5 lists the classification accuracies for the Abilene and Nobel-US topologies under different bandwidth configurations using the three feature-set $\{b, \lambda, p\}$. DT [3] and RF [3] refer to classification accuracies using these three features for each topology under the different bandwidth configurations. The traces for the two topologies are collected with the bandwidths set to 10 Mbps, 15 Mbps and 20 Mbps.

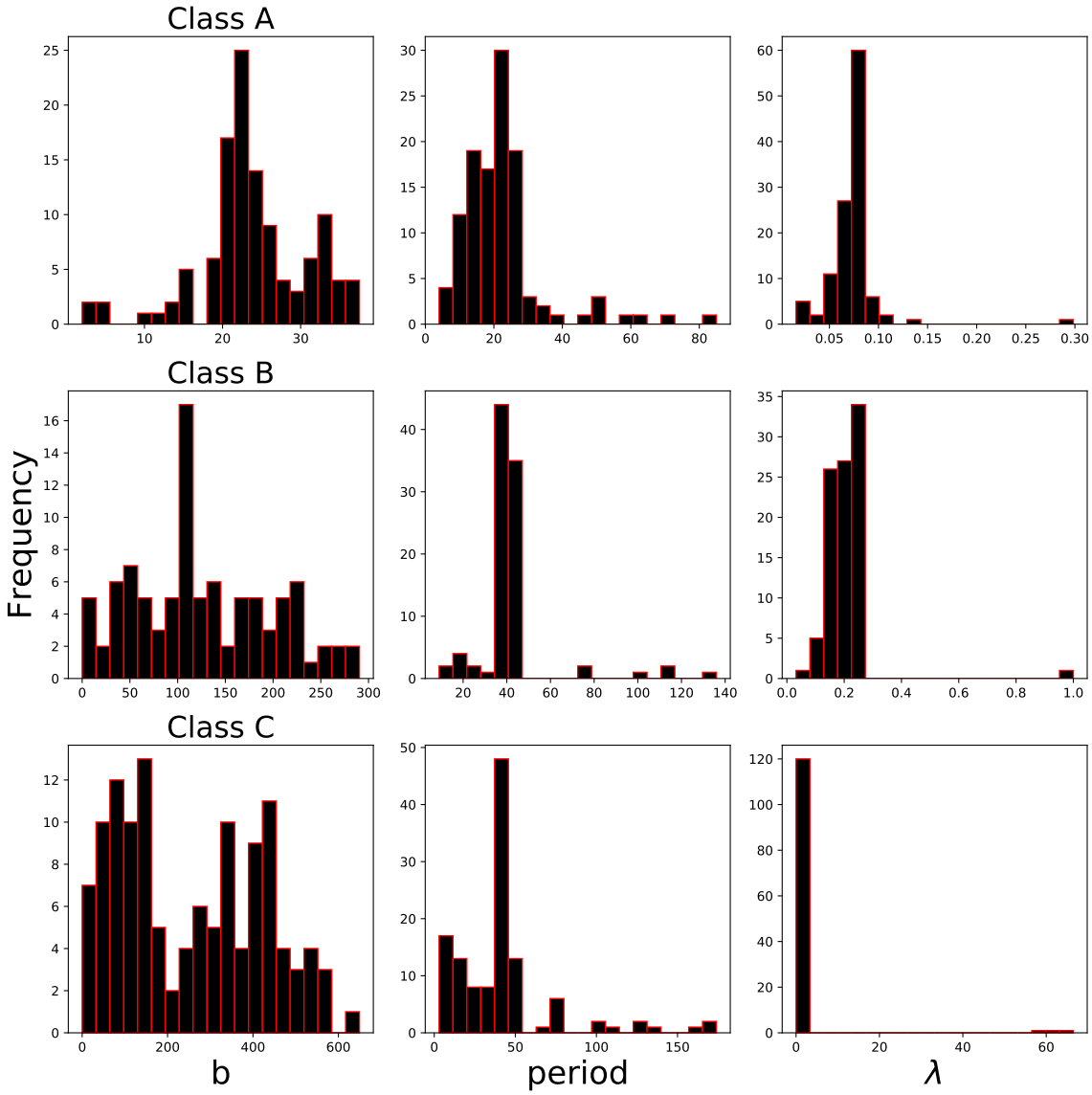

Fig. 4.17 I histogram the distribution of my feature space for the CNAA classifiers. The periods taken together with the b feature exhibits randomness, variability and some outliers which underline the poor performance in relation to the other classes. Better symmetry across all feature distribution are observed for class A which records the best classification accuracy.

The classification accuracies range from 42% to 76% for all cases. The classifiers achieve the best performance for the Nobel-US topology set to 15 Mbps bandwidth. The estimated parameters for each of the network states using the Abilene topology observations with the 10 Mbps bandwidth is examined using the figure shown in Fig. 4.18.

Table 4.5 The results of the batch experiments for the SDN topologies. Congestion classification using the three features $\{b, \lambda, p\}$ in DT and RF classifiers for the Abilene and Nobel-US topologies. Accuracy is in %.

		10 Mbps		15 Mbps		20 Mbps	
SDN Topology	Congestion	DT [3]	RF [3]	DT [3]	RF [3]	DT [3]	RF [3]
Abilene	Low	70.10	64.54	64.39	70.48	69.57	61.16
	Medium	73.90	75.15	56.82	52.52	60.35	47.20
	High	64.96	52.05	53.16	53.67	64.92	50.41
Nobel-US	Low	71.92	69.91	76.26	65.18	52.00	59.36
	Medium	73.59	70.79	75.33	51.00	45.48	42.63
	High	68.41	65.24	74.89	47.51	63.50	60.60

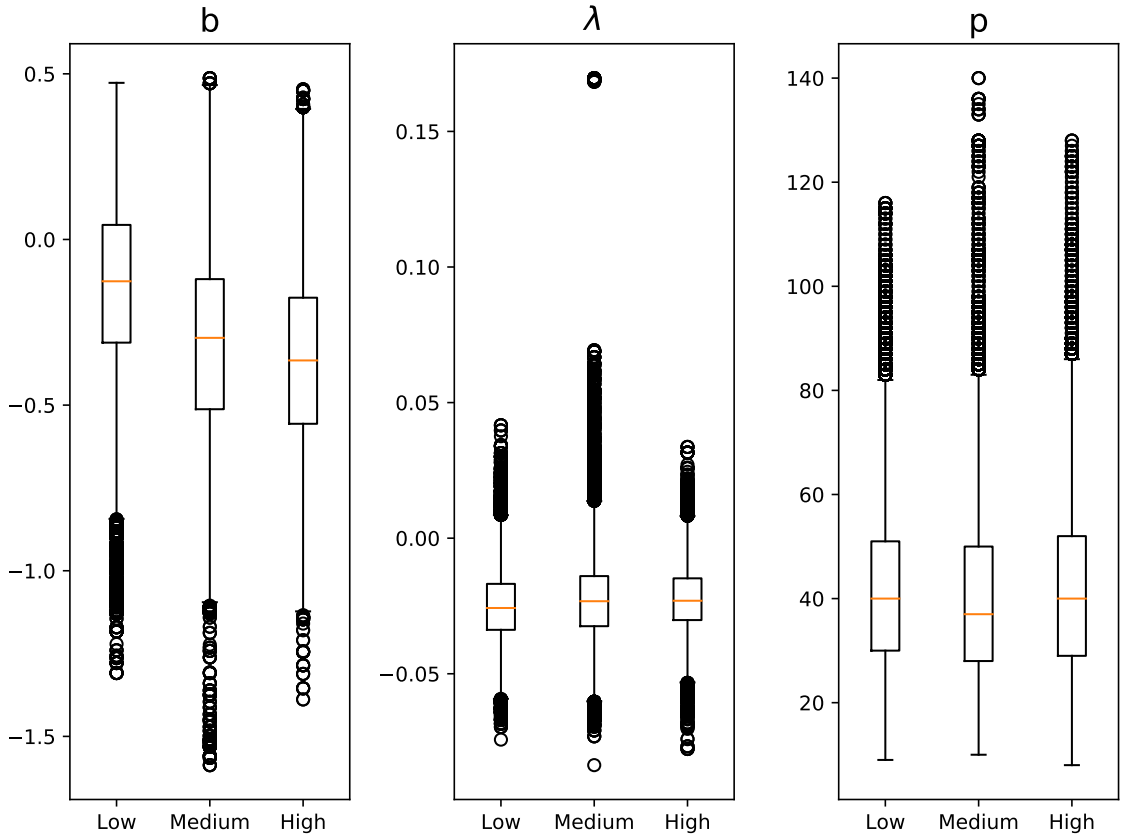

Fig. 4.18 The $\{b, \lambda, p\}$ paraemeters for the SDN Abilene topology with bandwidth set at 10 Mbps; the parameters are marginally different for the three network states.

The 25th, median, and 75th percentiles for each of the jitter feature-set $\{b, \lambda, p\}$, are shown in the first, second, and third columns of Fig. 4.18. The boxplots in each of the

columns provide summaries of the jitter feature-set in low, medium and high congestion states.

The boxplots reveal similarities in the statistics for the jitter feature-set which explains the less impressive classification results tabulated in Table 4.5. Low, medium and high congestion are characterized by large ranges for both the b and p parameters. The λ feature shows a narrow range in the 25th to 75th percentile for all three congestion levels. These overlaps with the spread of the 25th to 75th percentile for the jitter feature-set makes distinguishing between all three states a challenge for the classifier. The boxplots in Fig. 4.18 are complemented with the illustrations in Fig. 4.19 which capture the jitter feature-set. The red lines indicate the transition points from low to medium and medium to high congestion. Similar characteristics are observed for the different levels of congestion which accounts for the miss-classifications.

Using the jitter feature-set, $\{b, \lambda, p\}$, an evaluation is conducted on how to include the differences in the 25th to 75th percentile range for the different levels of congestion in order to improve the classification accuracy for the network traces tabulated in Table 4.5. The moving average of each parameter b, λ, and p, with a 100-tap moving average filter is computed. The difference between the parameter estimate and the moving average is then squared. This is done to estimate the variance in the parameter estimates. These additional three features are used in conjunction with the original jitter feature-set, $\{b, \lambda, p\}$, in the RF [6] and DT [6].

Table 4.6 lists the classification accuracies for the Abilene and Nobel US topologies using the original jitter feature-set, $\{b, \lambda, p\}$, and the corresponding deviation features for the three features designated as DT [6] and RF [6]. The inclusion of the three deviation features boost the classification accuracies of the DT [3] and RF [3] classifiers using the original jitter feature-set. The classification using the six parameter dataset is better for all cases for both topologies. The DT [6] and the RF [6] out-performs the DT [3] and

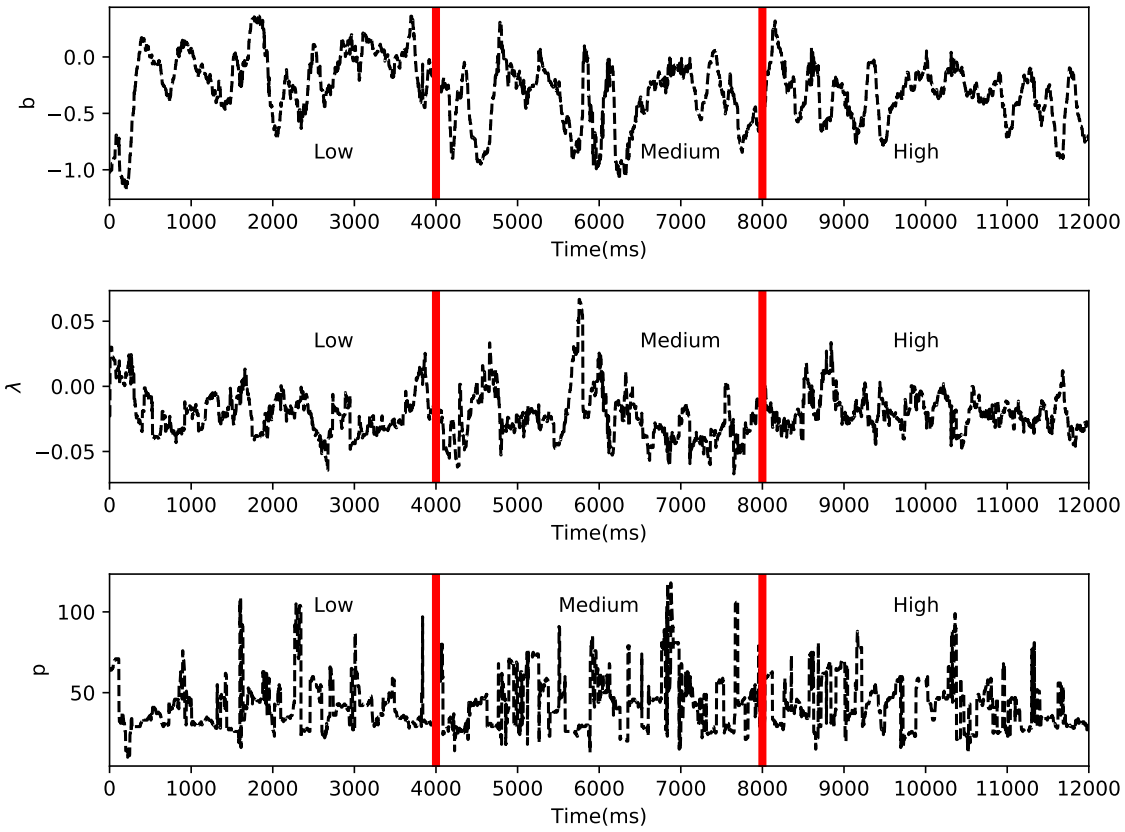

Fig. 4.19 Changes in state are possible but marginally detectable.

Table 4.6 The results of the batch experiments for the SDN topologies using the jitter feature-set, $\{b, \lambda, p\}$ and the deviation features for all three parameters. The inclusion of the deviation features boost the classification accuracy. Accuracy is in percentage (%).

		10 Mbps		15 Mbps		20 Mbps	
SDN Topology	Congestion	DT [6]	RF [6]	DT [6]	RF [6]	DT [6]	RF [6]
Abilene	Low	95.37	98.23	94.15	98.27	83.16	88.32
	Medium	94.68	96.24	67.53	78.85	84.09	92.56
	High	91.09	93.47	71.00	77.62	70.82	77.57
Nobel-US	Low	80.66	80.08	85.77	92.27	76.12	84.29
	Medium	78.91	78.04	74.08	73.36	67.83	77.65
	High	91.78	93.95	85.34	87.89	84.72	89.59

the RF [3] in all congestion cases. The DT [6] and the RF [6] significantly boosts the classification performances for the medium congestion level compared with the initial three

jitter feature-set. These boost in classification performance suggest that considering the deviations in parameter estimates may yield positive results. However, it is observed that the classification results for the medium congestion level is generally below those for the low and high congestion cases. Furthermore, the dynamic nature of networks raises concerns about the practicality of achieving smooth graduated congestion levels, as regular corrective measures may lead to network instability. From these analysis and results, being able to identify and correctly classify network congestion as low or high, is enough.

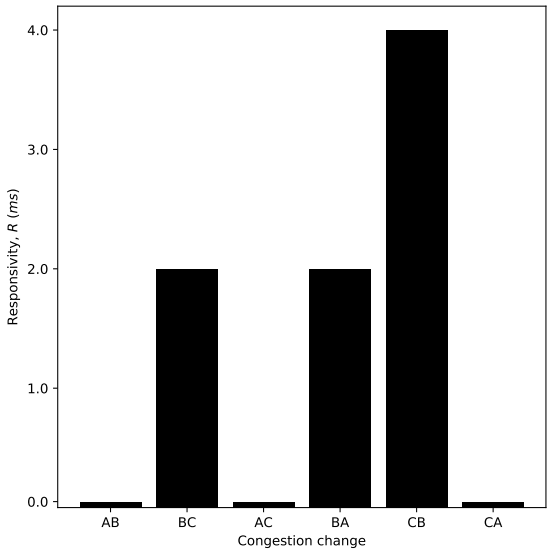

Fig. 4.20 The responsivity of the time-varying system experiments are shown. The responsivity for transitions from A to B, A to C and C to A are 0ms. Transitions from B to C and B to A record a 2ms lag. The responsivity for transition from C to B is 4ms.

4.8.3.2 Time-varying System Experiments:

Fig. 4.20 illustrates the results of the online experiments depicting the service change-point detection times. For same number of samples in epochs (10 time-points), service change-points from classes $A \rightarrow B$, $A \rightarrow C$ and $C \rightarrow A$ all record a 0ms lag. The CNAA classifier classification accuracy is 100% with no mis-classification errors. These results make sense as the data values for all features are quite distinct for classes A and C. The mean and

maximum b values for class A is (24ms, 38ms); for class C, the mean and maximum b values are (255ms, 650ms) respectively. Similarly, the p parameter differs significantly with mean values of 22 and 40 for class A and C respectively. The same holds for classes A and B where there is a clear difference in the b and p parameter values.

The responsivity, R for service change-points from $B \rightarrow C$ and $B \rightarrow A$ are both 2ms for same number of samples in each epoch. The system detects the change in service after 2ms. It achieves a classification accuracy of 90%. The worst service change-point detection occurs for transitions from class C to B. The responsivity is 4ms. The classification accuracy is 80%.

4.8.3.3 Effect of Frequency on Responsivity:

An evaluation of the effect of varying the samples in each epoch is investigated here. The responsivity of the CNAA classifier is reported as a percentage of time the first k samples are correctly classified, where the score is computed over intervals from length 1 to k, in increasing interval lengths of 1.

$$R_1^k, \quad \text{where} \quad k \in 1...10 \tag{4.17}$$

Table 4.7 lists the responsivity of the CNAA classifier when the epoch samples are varied from $(10, 15, 20)$ to $(20, 10, 15)$ respectively for classes A, B, C. In all cases, the CNAA classifier achieves a 100% classification for class A. A similarity is observed in the system responsivity for congestion level transitions from $C \rightarrow B$, $C \rightarrow B \rightarrow A$ and $C \rightarrow A \rightarrow B$. In these cases, there is a downward ramp from highly congested traffic areas to lower areas of congestion.

The case is slightly different for trace $A \rightarrow B \rightarrow C$ which involves a gradual build up of congestion from low congestion to medium congestion levels and then highly congested traffic. Traces $A \rightarrow B \rightarrow C$ and $C \rightarrow A \rightarrow B$ also show similarities in service detection. The

Table 4.7 Responsivity of the CNAA classifier for the epoch samples variation. The samples are varied from (10,15,20) to (20,10,15) respectively for classes A,B,C. The responsivity for the service change-points are calculated for R_1^k where k = 1,3,5,7,10. R_T is the responsivity for the entire transition. Class A achieves a 100% responsivity for all cases (The reported results are only for A→B→C). The best overall responsivity is reported for trace C→B→A.

Transitions	Epoch	R_1^1	R_1^3	R_1^5	R_1^7	R_1^{10}	R_T
C→B	C	100%	66.7%	80%	85.7%	80%	92.59%
	B	100%	100%	100%	100%	70%	
A→B→C	A	100%	100%	100%	100%	100%	
	B	0%	0%	40%	57.1%	70%	90.7%
	C	100%	100%	100%	100%	90%	
C→B→A	C	100%	66.7%	80%	85.7%	80%	95.35%
	B	100%	100%	100%	100%	80%	
C→A→B	C	100%	66.7%	80%	85.7%	80%	90.7%
	B	0%	0%	40%	57.1%	70%	

system responsivity for $k = 1$ and $k = 3$ is the same. The system misclassifies the traces early on. Then, for the interval, $4 < k < 10$, the system detects the change in congestion levels and classifies the traffic correctly. These results are significant. The time lag between a change in network congestion state and the time the classifier realizes the change is critical as this could be an indicator of a problem in the network. Overall, the CNAA classifier achieves system responsivity accuracy greater than 90%. The worst case performance has a classification accuracy of 90.7%.

4.8.3.4 CNAA vs RF Running Average Classifier:

Table 4.8 lists the accuracy of the RF running average classifier. A 70% - 30% train-test split is used to divide the data. First, the running average window is set to the mean of the class A trace. In this setting, a classification accuracy of 56.84% is achieved. This is 18.84% off the worst performance recorded by DT-based CNAA with 75.68%. The DT-based CNAA

classifier records better classification accuracy than this RF running average classifier. In a second approach, the window length of the running average estimator is set to the most occurring period value for all classes. For class A, this value is 23. For classes B and C, the value is 39. This is done to incorporate some information about the curve periodicity. An improvement in classification accuracy of $\approx$ 1.26 and 6.69% is achieved for window sizes of 23 and 39 respectively as shown in Table 4.8. The performance of the RF running average classifier is improved by explicitly modelling the network dynamics in the model [252]. Increasing the window beyond 39 does not improve the classification accuracy. However, the worst performance recorded by the CNAA classifier for the class B trace has $\approx$ 12.15 and 19.8% performance gains in classification accuracy for the DT and RF models respectively over the RF-running average classifier. The CNAA classifier models boasts the ability to overcome the running average classifier by incorporating the underlying network dynamics in its learning phase.

Table 4.8 The performance of RF-based running average classifier is listed. The mean period for class A records the worst performance with a classification accuracy of 56.84%. The classification accuracy is increased by incorporating some network dynamics by $\approx$ 1.26 and 6.69% through adjusting the window length to the most occurring period value in classes B and C. The **CNAA** classifier offers $\approx$ **20%** and **34%** performance gains over the running average classifier for the worst and best-case scenarios respectively

Model	Window length	Accuracy (%)
Running Average	22	56.84
	23	58.10
	39	63.53

4.8.3.5 Computational Complexity:

The running average algorithm is based on a simple moving average algorithm which computes the unweighted average of the last N samples. By using a recursive formulation of the algorithm, the number of Floating-Point Operations Per Second (FLOPS) required per

sample is reduced to one addition, one subtraction and one division. The formulation of the algorithm is independent of the window size N, with a runtime complexity of $O(1)$.

If a Neural Network is considered based on a one feed-forward pass algorithm for a Multilayer Perceptron with 4 layers and 3 matrices, the computational complexity would be proportional to $O(N^4)$. This is high.

The CNAA model relies on parameter estimation for network state traffic classification. First, the N-point period estimation which costs $O(N)$ FLOPS. The logarithmic computation to express the individual curves costs $O(log_2(N))$ FLOPS. Then, computation of the b parameter and λ cost $O(N)$ FLOPS. The overall computational compexity of the CNAA model is $O(N)$. The CNAA technique is computationally fast when compared with a Deep Learning method. Even with the additional cost over the running average model, the CNAA model is an ideal solution for real-time applications. Above all, the performance gain in classification accuracy demonstrated above outweighs this additional cost.

4.9 Conclusions

In this chapter, a new model for jitter prediction was proposed which was dubbed the Codec-aware Network Adaptation Agent (CNAA). The CNAA model achieves accurate and correct predictions of QoD metrics, jitter in the presence of congestion and adaptive codecs. The chapter results demonstrated the superior prediction performance gains of the model over some baseline ML approaches. The results from the analysis showed that by incorporating the available network information in the form of the adaptive behaviour of the codec, the model was able to improve prediction performance. This performance gain comes at no additional requirement for computational power or data. A second contribution of this chapter is a demonstration of the CNAA parameter estimates could be used for accurate network traffic classification based on congestion levels. The novel tree-based classifiers achieve approximately 80 - 97% classification accuracy. The CNAA technique achieves

an overall system responsivity of approximately 89% and above in terms congestion level change-point detection. These results are of relevance to network service practitioners. The high accuracy and response times of the CNAA classifiers demonstrate that they could be used to proactively re-route traffic or initiate an action to forestall further problems in the network.

4.9.1 Chapter Contributions

The SDN topologies used in this chapter were adapted from the published works in [226, 227]. The first named author of this publications, Ali Malik setup the topologies and extracted the jitter traces from the SDN test-beds. Ruairí de Fréin conceived the idea for the experimental framework. He proposed the initial idea of modelling the jitter data using quasi-periodic falling exponentials, derived the model parameters, accurate period detection techniques and developed the model for fitting parameters to the training data. He also formulated the correct levels of D-ITG experimental interference traffic required to generate the congestion levels that resulted in reasonable experimental data. The rest of the work described in this chapter including the numerical evaluation, ML evaluation and analysis were carried out by the PhD candidate.

Chapter 5

Automatic Labelling of Video Quality of Delivery Metrics with Device-Level Statistics

THIS chapter considers the problem of labelling jitter for predicting video quality in the presence of interfering services and applications, and for detecting the congestion level in the network. Existing techniques for inferring the state of the network health for such types of applications require that the network manager supply labelled data. Estimates of jitter, a QoD metric, have been used to detect the network state in the presence of interfering traffic workloads from other users. Such methods require clearly labelled data. This data labelling process can be an expensive process. In this thesis chapter, the network is modelled as a queue and parameters of the queueing system were used to label jitter training data. To evaluate the approach, port statistics from a target video client over a shared substrate in a real-world SDN topology were captured. This chapter demonstrates the performance of the automatic labelling process for different levels of background traffic. Numerical results based on data from the experiments demonstrate accuracies of 91% and above are achieved for the

automatic labelling process under constant bitrate traffic sources. Using the automatically labelled data and by training some off-the-shelf ML classifiers, the thesis chapter contributes an unsupervised network state detection technique based on the congestion levels. The performance of the unsupervised classifiers are compared with the traditional supervised approach. Empirical results demonstrate the performance gains of the unsupervised approach over the supervised method. The results from the analysis demonstrate that unsupervised network state classification offers approximately 4% and 25% performance gains in terms of its classification accuracies over the supervised baseline approach for the best and worst-case scenarios respectively. The contributions proposed in this chapter can be applied for offline and online training of video QoD predictors at any point of the video delivery ecosystem particularly within the video service provider infrastructure.

5.1 Motivation and Problem Statement

5.1.1 Motivation

AI has been applied in a variety of ways to address video quality prediction tasks [253]. Some of these AI solutions range from traditional ML solutions [254] to DL algorithms [71, 255] aimed at predicting a video QoD metric, such as jitter [256]. QoD measurements represent the end-to-end performance of a network [22]. Jitter is a variation in the time delay between when a signal is transmitted and when it is received. For video applications, the video server sends out the video data as a stream of packets. However, due to network congestion and the adaptive reaction of the codec in response to network conditions, the delay between the packets can vary. Other applications of ML in video streaming networks such as [224] aim to infer the state of the network by classifying estimates of jitter. However, the majority of these AI-based solutions require the human element for learning purposes. According to the authors of [257], it is estimated that about 90% of ML tasks rely on supervised ML where

there is a target label. A report from an AI research body in [258], estimated that over 80% of the time is spent on wrangling and labelling the data for AI projects. Strategies aimed at incorporating a combination of human knowledge and machine intelligence for AI projects is known as Human-in-the-loop Machine Learning (HITL ML) [12]. By integrating human expertise and knowledge, the goal of HITL ML is to train ML models accurately at minimum cost [257]. The challenge is that most of these supervised forms of ML require enormous amounts of the data that have been properly labelled.

An example of where this approach may be a challenge is with the work described in [224]. The authors proposed a new approach for estimating jitter, a QoD metric, under varying conditions of interfering traffic from other sources. Using the jitter model, the authors computed the estimates of the jitter model in the presence of different levels of background traffic. They then trained some off-the-shelf ML classifiers using the estimated model parameters to classify congestion in the network. However, before training the classifiers, a network engineer had to label the different regions of the trace and the estimated model features with the corresponding level of background traffic as low, medium or high. This is not a trivial task when you consider the amount of time that would be spent on data preparation, and the enormous amount of data that may be involved. Another challenge with this method is that the computed network characteristics may not generalize well in a network setup. For instance, if the classifiers are deployed in a different network architecture with varying levels of interfering traffic, the trained model may not work as it is did in the experimental framework.

5.1.2 Problem Statement

Data labelling platforms are now mainstream, and the industry is witnessing a boom. This has given rise to a thriving industry in developing countries [259]. The Economist in [260] noted that while investment in AI is expected to increase over the years, only a handful of

organizations have deployed ML models due to a lack of labelled data. To emphasize the value of labelling in AI, the authors of [261] referred to unlabelled data as "garbage data." ML models require accurately labelled data in order to be effective and efficient in their tasks. However, data labelling tasks need a lot of time and effort, can be costly, or even inaccurate.

Given that it is possible to model the adaptive response of the video codec in response to varying levels of network congestion [224], the chapter objective is to enable an integrated automatic ML data labelling system which incorporates this feedback loop between the video delivery system and the network. In this chapter, a new method for automatic labelling of jitter is proposed. The chapter describes a framework in which the network substrate for a video streaming session and interfering sources of traffic is modelled as a queue based on the principles of queueing theory [262]. The application of queueing theory to networks can reveal how performance evolves as traffic amounts fluctuate [263]. By modelling the network as a queue, estimates of the queue parameters are used to infer the behaviour of the network [264]. As the network becomes congested due to interfering traffic sources, the codecs attempt to match the video playout rate with the prevalent network conditions. This feedback loop between the video codec and the network ensures that the streaming infrastructure is capable of adaptively streaming the video data based on network state. By modelling the network element using a queueing model, this chapter demonstrates how automatic labelling of jitter can be achieved. This method is an unsupervised approach in which the system is modelled as a network of queues. This unsupervised method proposes a novel technique in which the network automates the labelling of jitter using estimates of a simple $M/M/1$ queue [265]. This eliminates the HITL element and contributes a technique that scales easily. In [224], the authors found that the measured jitter time series consisted of falling exponential curves [266] due to the codec adaptation to changes in the network congestion [11]. They performed network-state classification with off-the-shelf ML classifiers using features of the jitter model. The proposed model is related to the approach taken in [224] because the same

QoD metric, jitter is considered. A vital difference between these two studies is that there is no requirement for a network engineer to label the different levels of congestion in the data as was the case in [224]. Instead, by using a simple queuing model, the labelling process was automated. Finally, the results demonstrate that the approach described in this chapter generalizes well to new data and achieves network state classification.

A summary of the contributions of this chapter are as follows:

- A new model for automatically labelling jitter data is introduced. This technique removes the HITL element as the network handles the data labelling process using estimates of the queue parameters. This reduces the time and resources required for data labelling.

- This chapter demonstrates the efficacy of the automatic labelling process using jitter traces for different levels of background traffic. An evaluation of the system for 2, 3, 4 and 5 levels of background traffic was carried out. Numerical results based on data from the experiments demonstrate accuracies of approximately 95% and 92% for the best and worst cases respectively.

- Using the automatic labelling method, this thesis chapter presents an unsupervised ML technique for accurately classifying and detecting the state of the network using estimates of the queue and jitter. The performance of this unsupervised ML technique is compared with the supervised ML procedure described in [224]. The results of this evaluation demonstrate the superior performance of the unsupervised technique. The unsupervised ML process achieves accuracies of approximately 99% and 68% for the best and worst case scenarios.

This chapter is organized as follows. Section 5.2 introduces the network queue estimation model. Section 5.3 describes the automatic ML data labelling procedure and outlines the algorithm definitions. In Section 5.4, a description of the network test-bed implementation

used to evaluate the learning algorithms is presented. This section outlines the experimental evaluation of the proposed model and benchmarks the unsupervised ML technique proposed in this study with the traditional supervised network state classification. In Section 5.5, the results of the evaluations are presented. Finally, the chapter conclusions are presented in Section 5.6.

5.2 Modeling Network Queues

An important performance indicator of a data network is the average delay it takes to deliver a packet from origin to destination. Fig. 5.1 illustrates the notion of network queues and delays as the packets traverse the network. There are typically four factors that account for a packet delay in the network. These factors are the processing delay, queueing delay, transmission delay and propagation delay. The processing delay is the time the system spends on analysing the packet header and deciding where the packet must be sent. The formation of queues in the switch buffers lead to delayed packets. The length of the queueing delay differs depending on the traffic type, amount and inter-packet arrival times. The transmission delay is the time taken to place the packet's bits onto the link. Finally, the propagation delay is time spent on pushing the first bit from the sending device to the target device.

There are differences in packet delays among packets transmitted continuously over the network. This is due to differences in packet routes and the queueing process at network devices. This variance in delay between packets is known as jitter. To compensate for jitter, a jitter buffer is used in network devices to temporarily hold packets awaiting transmission. This queueing delay is what is modelled in this chapter. Fig. 5.1 highlights the elements of the queueing system. (1) Packet arrivals: On the left-hand side (LHS), there is a server which transmits the video packets and is the source of the interfering traffic; (2) Service Rate: this is the average time expended on servicing each packet in the network; (3) Queue Length: the

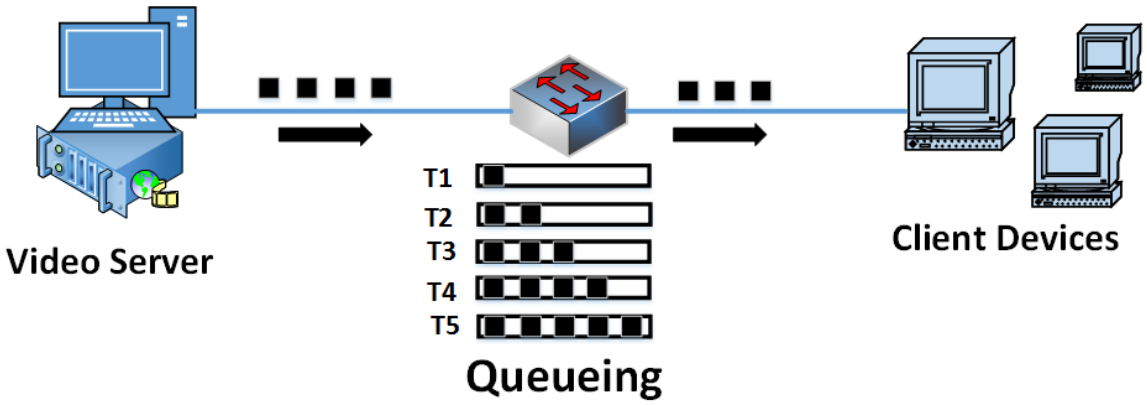

Fig. 5.1 Queues begin to build in the switch buffers represented by the rectangles as video packets and interfering traffic (the black boxes) traverse the network. The build-up accounts for the delay in packet reception. T1 - T5 depicts the switch buffer size at various times.

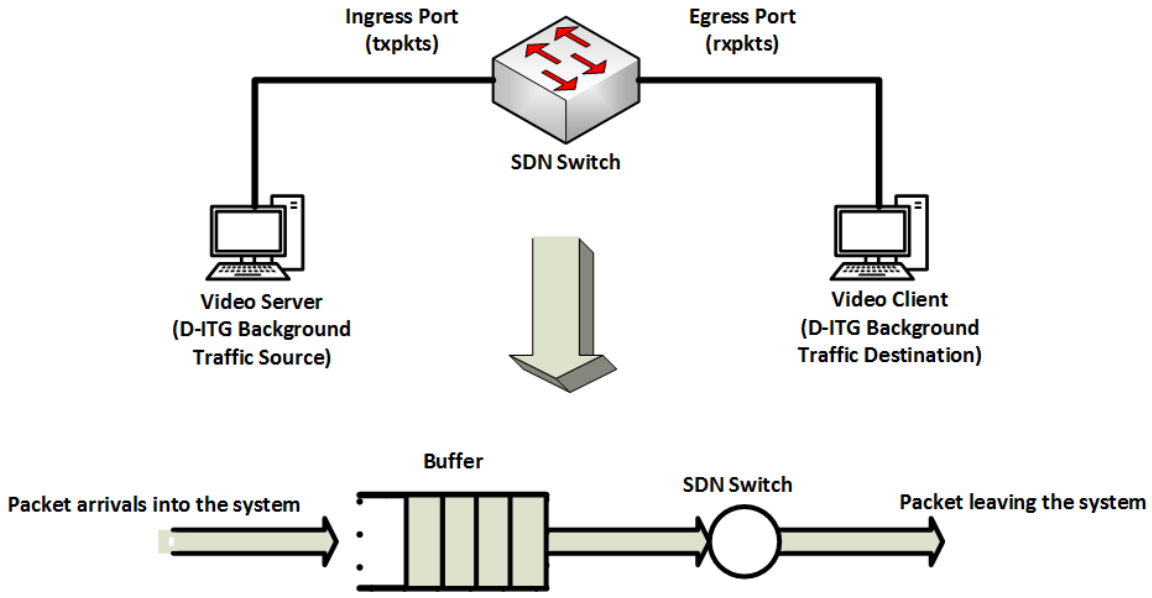

Fig. 5.2 The network of switches is modelled using a simple $M/M/1$ queue. Packets are temporarily enqueued at the switch buffers. Using the SDN Ryu controller inbuilt-thread API, I query the controller for port statistics at the ingress and egress ports of the switches.

capacity of the queues; (4) Queue Discipline: Typically, this is on a First Come, First Served principle (FCFS). It could also be random.

The contribution of this chapter is based on the following. The SDN controller monitors and aggregates network traffic. The Ryu controller [267], used in this study ships with an in-built thread API which is used to collect port statistics at the ingress and egress ports

of the switches connected to the video server and client respectively. The port statistics captured are the number of transmitted packets (txpkts) and received packets (rxpkts). Fig. 5.2 illustrates the notion of queues. Suppose, the video streaming session and the interfering traffic begins at Time 1 (T1). Let us assume that the server is sending the traffic at a rate of four packets for every three the switches can process. The switch has a five-packet buffer to accumulate unprocessed data. The buffer has a packet in it by T1. This waiting time accounts for the delay in transmitting this packet. The delay increases as more packets wait in line to be transmitted leading up to Time 5 (T5) when the buffer is near saturation. The hypothesis this thesis chapter investigates is that by using a simple $M/M/1$ queue, with an infinitely large buffer, it is possible to use estimates of the model parameters to automatically label jitter. A representation of the $M/M/1$ model is shown in Fig. 5.2. The $M/M/1$ model assumes that the packet arrivals are based on a Poisson process [268]. The switches in the SDN topologies used in this study are connected in tandem to each other. In the real world, traffic may enter multiple queues, leave multiple queues, merge and split repeatedly. The Kleinrock Independence Assumption [269] is adopted as a basis for the network models. This assumption posits that the effect of many streams combining and dividing at hops restores the randomness of the inter-arrival attribute, making Poisson arrivals a viable assumption for packets [270].

Consider a queueing system with buffers, (cf. Fig. 5.2) where random packets arrivals are defined at an arrival rate of λ packets per second. These packets are queued while waiting to receive service. The packets are transmitted at a rate of C bits per second which is the link transmission rate. Suppose that the packet length in bits is L, the packet service rate, μ can be estimated as follows

$$\mu = \frac{C}{L}. \tag{5.1}$$

The traffic intensity, ρ is estimated by dividing the average packet arrival rates with the average service rate,

$$\rho = \frac{\lambda}{\mu}. \tag{5.2}$$

To ensure the system is stable, the traffic intensity should satisfy the condition, $\rho < 1$. Using the $M/M/1$ model, the following quantities need to estimated: (1) The average number of packets in the system, N, and (2) The average delay per packet, T. This is a combination of the queueing and service completion time. To determine the quantities N and T, Little's law [271] is applied which makes it possible to determine one quantity given the other. Little's Law is of the form

$$N = \lambda T. \tag{5.3}$$

Little's law expresses a general principle of queueing theory. Under very general conditions, large buffer occupancy levels (large N) correspond to long packet delays (large T) and vice versa. The average number of packets in the system is

$$N = \frac{\rho}{1-\rho}. \tag{5.4}$$

It follows then that the average delay per packet can be estimated using Little's law as follows

$$T = \frac{N}{\lambda}. \tag{5.5}$$

5.3 Automatic Labelling Procedure for Jitter

A network of SDN switches is modelled using a simple $M/M/1$ model. There are some shortcomings of the $M/M/1$ model in terms of the Poisson assumption and the infinite buffer size. However, the analysis presented in this chapter exploits the simplicity of the estimators it uses for different network statistics. Later experiments show that these estimators are

Algorithm 1 Process of Evaluation

1: Packets at the ingress and egress ports	▹ RYU Thread API
2: Capture Jitter traces	▹ Wireshark
3: Compute the instantaneous packet counts	▹ Pairwise differences
4: Compute estimates of the $M/M/1$ model	▹ Equations 5.4 & 5.5
5: Cluster estimates	▹ K-Means algorithm
6: Label the jitter traces with cluster estimates	▹ Assignment Matrix

accurate enough to make automatic labelling achievable, and their low computational burden means that automatic labelling should be achievable in real-world deployments.

The OpenFlow protocol maintains a count of the cumulative packets processed at each port. Algorithm 1 presents the methodology taken to achieve the automatic labelling process. In Line 1 of Algorithm 1, at the start of the streaming session from a server to a client, the cumulative packet counts at the ingress and egress ports of the first and last SDN switches in the topology respectively are captured. In Line 2, jitter measurements for the session are captured using Wireshark [272]. In Line 3, the instantaneous packet counts are obtained by taking the pairwise differences in the cumulative packet counts at the end of each second. Using Equations 5.4 and 5.5, the Queueing Delay (QD) estimates of the $M/M/1$ model are computed in Line 4. In Line 5, the QD estimates are clustered using a k-means clustering algorithm [273]. The goal is to divide the dataset into a distinct number of clusters with each queue estimate belonging to the cluster that has the closest mean. Line 6 describes the automatic labelling of jitter data using the clustered queue estimates. The jitter timestamps are matched with the associated queue estimates computed for that specific timestamp. For instance, all jitter data captured during the first minute of the streaming session will be matched to the queueing estimates for the first minute. The procedure outlined in Algorithm 1, occur in the presence of background traffic from the server to the client using the Distributed Internet Traffic Generator (D-ITG) [228].

The Assignment Matrix described in Algorithm 2 is used to label the jitter measurements using the QD estimates. The jitter data captured with Wireshark includes timestamps for the packets. Unique values for the timestamps (Line 2) are obtained and saved to a times

Algorithm 2 Assignment Matrix

1: Input: Jitter, QD ▷ timestamp, jitter, queue estimates
2: times := unique(timestamp) ▷ times object
3: Require: MAX(timestamp) == LENGTH(QD)
4: For *i* in times, if *i* == QD[i], assign the QD to the jitter value

series object. This is because for video transmission, a single video frame can be divided up and sent in many packets, each with the same timestamp [274]. This corresponds to having multiple packets transmitted in a specific timestamp with the same QD and belonging to an individual adaptive jitter curve [224]. Line 3 describes a requirement that must be fulfilled to ensure that only the jitter measurements associated with the video streaming session is available for labelling. The maximum value of the timestamp in the jitter data must be equal to length of the QD estimates. To ensure that this is the case, the collection of jitter statistics is stopped at the same time when the streaming session ends. Line 4 describes how the assignment function works. The assignment matrix function looks at the timestamp column and inputs the corresponding QD estimate for the timestamp. Finally, the assignment matrix iterates over the times object and matches all the jitter timestamps with the QD estimate computed for that particular timestamp.

5.4 Test-beds and Benchmarks

This section describes the experimental framework starting with a description of the SDN network test-bed. Then, an outline of the process for evaluating the automatic labelling procedure and the implementation of the network congestion state detection using the automatically labelled data is provided. This is the unsupervised approach for classifying the network state congestion levels. The performance of the unsupervised approach is then compared with the supervised method described in [224].

5.4.1 SDN Network Test-bed

The evaluations on SDN emulation platforms are conducted using Mininet [275]. The Ryu component-based SDN controller [267] is used. The Abilene and Nobel-US real network topology instances that are obtained from the SNDlib [214] are adopted. Two hosts are added in each topology to represent the client and server, which are required to launch a video streaming session. For each topology, a random path equal in length to the network diameter was selected so that the client and server hosts became the two ends of the path. A VLC server streams the "Big Buck Bunny" MPEG-4 video, which is approximately 10.34 minutes long (approximately 620 seconds), to the client device using RTP [276]. The H.264 video compression standard for high-definition digital video is used for the experiments. The D-ITG traffic generator is used to generate varying levels of UDP background traffic to emulate additional workloads. In a first set of experiments, the D-ITG tool is set up in multi-flow mode with UDP flows having constant inter-departure time between packets. The D-ITG load background traffic characteristics are tabulated in Table 5.1. The load generating pattern is distributed to generate the required number of congestion levels for each profile to suit the video file playing time of 620 seconds. For instance, to generate the 2-flows profile, the D-ITG load generator at the video server starts by injecting 300 packets per second (pps) for 310 seconds. The packet arrival rate is doubled for the remaining 310 seconds to double the packet arrival rates. Arrival rates of (300→600) indicates that the packet arrival rate increases to 600 pps after 310 seconds.

Table 5.1 The constant bitrate UDP traffic settings. The → in the arrival rates column indicates the rate at which the arrival rates increase with time.

Traffic Mode	Number of Levels	Arrival Rates (pps)	Duration (seconds)
2-flows	2	300→600	310
3-flows	3	200→500→700	206
4-flows	4	200→400→600→800	155
5-flows	5	200→500→900→1500→2300	124

Similarly, for the 3-flows profile, the traffic is introduced at a rate of 200, 500 and 700 packets per second at intervals of 206 seconds respectively. The default packet size of 512 bytes is used for these experiments.

5.4.2 Automatic Labelling of Jitter

This section describes the procedure for using a simple $M/M/1$ model to estimate network queue parameters for labelling jitter. The D-ITG load generator is set up according to the profiles listed in Table 5.1. In Fig. 5.3, Rows 1 (R1) and 2, plots of the instantaneous packet counts captured from the egress and ingress ports respectively in the Abilene network topology for three levels of interfering traffic using the 3-flows traffic model are illustrated. In Fig. 5.3, Row 3, a plot of the computed QD estimates is shown. The network is in one of three states, low, medium or high congestion. In its current state, there are no labels for the data.

Using the data in Fig. 5.3, R3, a k-means clustering model [273] is trained to cluster the QD estimates. The k-means algorithm divides the dataset into k clusters (where k is the number of clusters). The assumption is that each queue estimate can only belong to one cluster and should be representative of the instantaneous network state. A k value of 3 was chosen. This is a design choice which is confirmed by computing the silhouette coefficient,

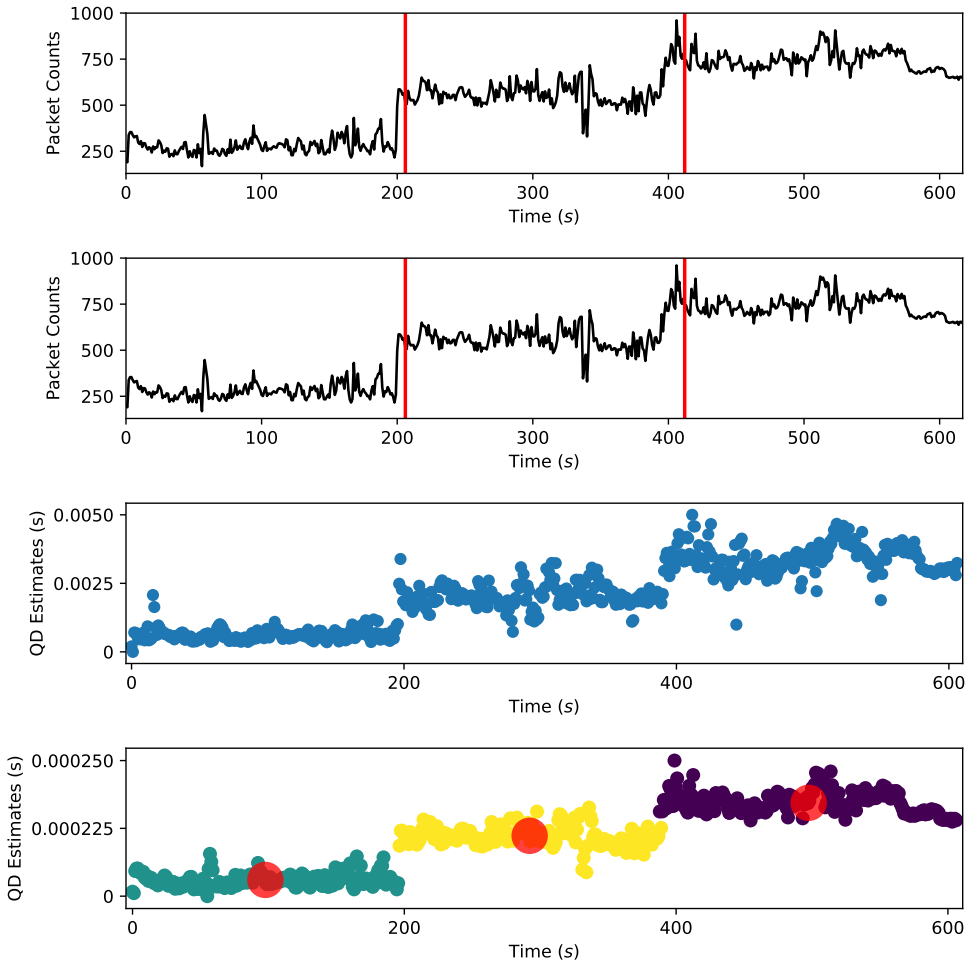

Fig. 5.3 Rows 1 (R1) and R2 illustrate the transmitted and received packet counts captured at the ingress and egress ports for a video streaming session for the 3-flows profile using the Abilene topology. R3 shows the estimates of the queue parameters, QD. The data in R3 serves as training data for the k-means clustering algorithm. R4 reveals the clusters formed with testing data captured using the Nobel-US topology. The red circles indicate the cluster centroids.

(c) [277]. The c measures the quality of clusters created with clustering algorithms such as the k-means with respect to how well data samples are clustered compared to other samples that share similarities. It has a range of -1 <c< 1. Values near +1 imply that the data observation is far away from the neighboring clusters. A c value of 0 indicates that the observation lies on or is in close proximity to the decision boundary between two neighboring clusters, whereas negative values indicate that the samples were assigned to the incorrect

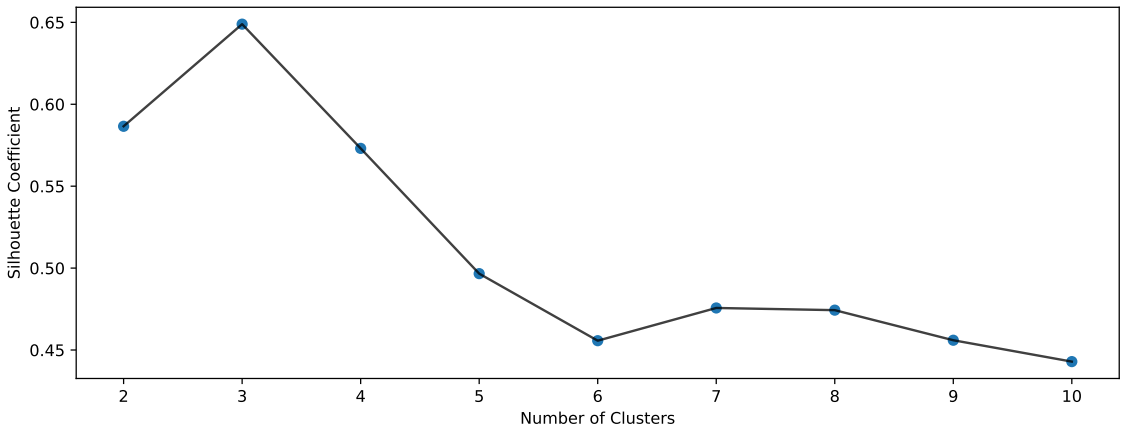

Fig. 5.4 Silhouette Coefficients for the range $2 \leq k \leq 10$.

cluster. The k value is changed over a range of $2 \leq k \leq 10$ to obtain the k number of clusters by evaluating the silhouette coefficients. The ideal number of k is when the c is highest. Fig. 5.4 shows the resulting c values. The maximum c value is observed when k is 3. This is line with the design choice of 3.

The plot in Fig. 5.3, R4 shows the clusters formed. The assignment matrix is to label the jitter traces. To evaluate the automatic labelling models under the traffic modes shown in Table 5.1, network packets are captured using the same configuration with the Nobel-US network topology. This dataset serves as the test dataset. The estimates of the queue parameters are obtained as before. Using the pre-trained k-means model (on the Abilene dataset), the new dataset is used to predict and assign these estimates to clusters as shown in Fig. 5.3, R4. For each network configuration, the training dataset is generated for the clustering algorithm using one SDN topology. Then, the estimates of the queue parameters obtained from the other SDN topology is used for testing and prediction purposes.

5.4.3 Network State Detection Analysis: Unsupervised Versus Supervised

The authors of [224] proposed a model for jitter time series data, $x[n]$. They summarized the ith decaying exponential in $x[n]$ using a jitter model feature set with $\{b_i, \lambda_i, p_i\}$, where the peak value b_i, the decay rate, λ_i and the period p_i, are the result of the adaptive nature of the codec in response to different network conditions. To compute the periods, they used the Auto-Correlation Function [278]. They represented the data using the exponential function,

$$f_i[n] = b_i e^{-n\lambda_i} \quad n \geq 0. \tag{5.6}$$

The authors segmented the data with estimates of the periods and determined that $x[n]$ was the sum of decaying exponentials where each exponential, f_i, was shifted in time, scaled by a different base parameter b_i, and had a different rate of decay parameter λ_i. They formulated the jitter data as

$$x[n] = \sum_{i=1}^{I} f_i[n], \tag{5.7}$$

where I is the count of regions in the observed number of samples of the jitter trace. Using estimates of the jitter model, they trained off-the-shelf classification algorithms to detect the network congestion state. This required knowledge of the different levels of the congestion in the jitter trace to manually label the data beforehand. This approach is referred to as the supervised network state detection.

This thesis chapter contributes an unsupervised network state classification technique where the automatically labelled jitter data and the QD estimates are used as features for training off-the-shelf ML classifiers. In this approach, a network engineer is not required to label the traces as the network via the $M/M/1$ model handles this process. The performance of the baseline supervised ML approach [224] is compared with this novel unsupervised

approach proposed in this chapter. The classifiers used in this chapter are the DT, RF and Extreme Gradient Boosting (XGB) [279].

The experimental evaluations are organized as follows:

1. Full Dataset Multi-class Classification: Using the combined and shuffled dataset of 61805 observations derived from the 3-flows traffic mode, where there are 20601 observations in each state, multi-class classification is evaluated. The objective is to classify the network congestion state as either being in low, medium or high. The dataset is divided using a 70-30 split into training and test data. The test data is used to validate the performance of the ML models. The performance of the unsupervised approach is compared with the supervised approach [224] using the full dataset.

2. Batch Experiments: Batch test dataset from the three possible network states: low, medium or high are generated. Using the pre-trained multi-class classification models, the task is to predict the network congestion levels using the batch test data. The goal of these batch experiments is to investigate the accuracy of the models in classifying the network state data to the correct class category using test data from each of the congestion levels. The task in the full dataset experiments was to classify a shuffled testing dataset from all congestion levels. For the batch tests, the task is for the trained models to classify test data from low, medium or high congestion levels.

3. Time-varying System Classification: The responsivity of the system using the unsupervised approach is evaluated. The time-varying experiments were conducted using the approach described in Section 4.8.3.2. Time-varying datasets of all the three network congestion levels are generated. For instance, to generate a time-varying dataset of 300 observations, 100 observations are taken from the low, medium and high congestion states to form a combined dataset with varying congestion levels. This process is repeated for different use cases by varying the number of samples drawn from each network state. The objective of these experiments is to detect the service change-point

and compute the responsivity of the ML classifier in response to changing levels of congestion in the system.

5.5 Results and Discussion

Five results are reported from the model evaluations.

1. The first reported results are the performance of the automatically labelling technique. The performance of the automatic labelling process is examined for different use cases. The automatic labelling process achieves an accuracy of $\approx 91\%$ and above for all scenarios examined using the flow characteristics described in Table 5.1.

2. The performance of the automatic labelling procedure is examined with a mix of application-level background traffic. The effectiveness and limitations of the technique are evaluated in these settings. The accuracies of the automatic labelling models in these settings range from $\approx 78\%$ to 90%.

3. The results from the network state classification using the baseline supervised approach [224] and the unsupervised approach introduced in this chapter are then presented. The results from the full dataset multi-class classification model evaluations using the supervised and unsupervised techniques are presented. Both approaches perform relatively well with the unsupervised technique showing superior performance. The best and worst accuracies for the unsupervised technique are 84.46% (XGB) and 83.35% (RF) respectively. For the supervised technique, the best and worst classification accuracies are 83.39% (XGB) and 65.64% (RF) respectively.

4. For the batch experiments, the worst accuracies recorded are 47.74% (RF) and 72.43% (DT) for the supervised and unsupervised techniques respectively. The best accuracies recorded are 93.00% (XGB) and 99.64% (DT) for the supervised and unsupervised

techniques respectively. The unsupervised models offer better accuracies in these settings.

5. The results from the time-varying experiments show that the system responsivities achieved using the automatically labelled data varies from $\approx 65\%$ to 99% for all ML models considered.

The model classification accuracies are evaluated with the accuracy metric, M_A, computed according to Equation 4.16.

5.5.1 Justifying Automatic Labelling

To justify the evaluation of automatic labelling approach in this chapter, an analysis of the relationship between the queueing delay estimates and the parameters in the jitter time series model in Equation 5.7 is evaluated. This analysis demonstrates that automatic labelling is achievable. For both approaches, the network is in one of three states, low, medium or high congestion. Approximately 9000 observations from each state is taken and the statistics of the estimated parameters examined in each state. Fig. 5.5 and Fig. 5.6 illustrates the boxplots indicating the first, second and third quartiles for the parameters from the supervised and unsupervised techniques respectively. There is wide variation in the first to the third quartiles for the b and p parameters. The spread of the parameters estimates for the first and third quartiles in the medium and high states makes distinguishing them a challenge. The similarity in the parameter estimates is the same for the jitter traces shown in Fig. 5.6. There is clearly a distinction between the estimates of the queue parameters shown in Fig. 5.6. This is reasonable because the QD estimate is low when the congestion is low and increases as congestion builds up in the network. This makes it possible to use the QD estimates to label the corresponding jitter data. However, the boxplot for the supervised jitter model parameters reveals some overlap among the different network states for the model parameters. This presents some challenges in using these estimates to accurately label the jitter traces.

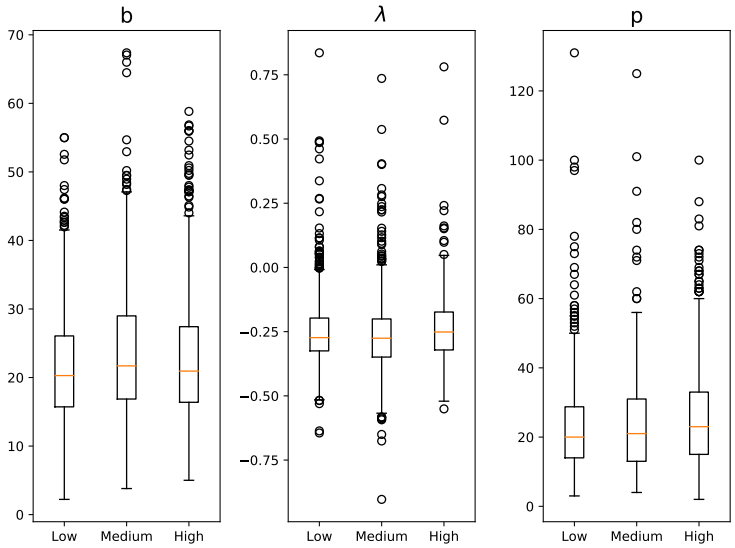

Fig. 5.5 Statistics of the jitter model estimates for the supervised approach. The boxplot reveals some overlap among the different network states for the jitter model parameters using the supervised approach. This makes it challenging to achieve accurate network congestion state detection.

5.5.2 Automatic Labelling: Constant Bitrate Traffic

Table 5.2 lists the performance of the automatic labelling technique under constant bitrate UDP background traffic. The best accuracy was recorded with the 3-flows traffic mode. The worst performance is a model accuracy of approximately 92% for the 5-flows case. As the levels of traffic for the streaming session increases, the clusters begin to overlap. This leads to inaccurate labelling of jitter traces as a result of overlapping clusters. The majority of the misses in the labelling process occur during the transition windows. The network computations struggle to consistently estimate the queue parameters as the congestion level moves from one level to another.

Let us examine what an accuracy of 95% means for the 3-flows case. Using 200 measurements from each of the three levels, Fig. 5.7, R1 depicts the labelling procedure for the 3-flows instance. It can be observed that the queue estimates match the three levels in the jitter measurements. Although, the queue estimates fail to match the jitter traces for medium

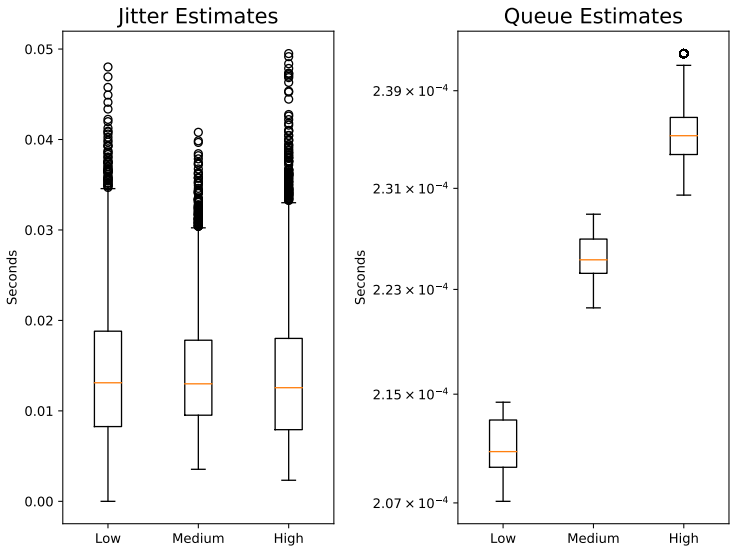

Fig. 5.6 Statistics of the estimates of the queue parameters and the associated jitter measurements. The queue estimates are significantly different which provides a mechanism to use these in labelling and classifying jitter according to the level of background traffic in the system.

Table 5.2 Accuracy of the automatic jitter labelling process under constant bitrate traffic: the best accuracy is recorded for the 3-flows scenario; the worst case is recorded with the 5-flows case.

Model	Interfering Traffic Level	Accuracy (%)
Automatic Labelling	2-flows	92.89
	3-flows	94.98
	4-flows	94.71
	5-flows	91.89

congestion range $300 < t < 410$, they differ from the low and high QD estimates. As the congestion in the network varies, the codec adaptively adjusts the video playout to match the network conditions. The feedback from the codec is captured by the network queueing model as it also shows variations during these time periods. Being able to form distinct clusters with similar characteristics enables the automatic labelling process to correctly label the jitter traces.

In Fig. 5.7, R2 a scatter-plot of the jitter traces on the y-axis and estimates of the queue parameters on the x-axis is shown. It can be observed that there are three clear distinct clusters

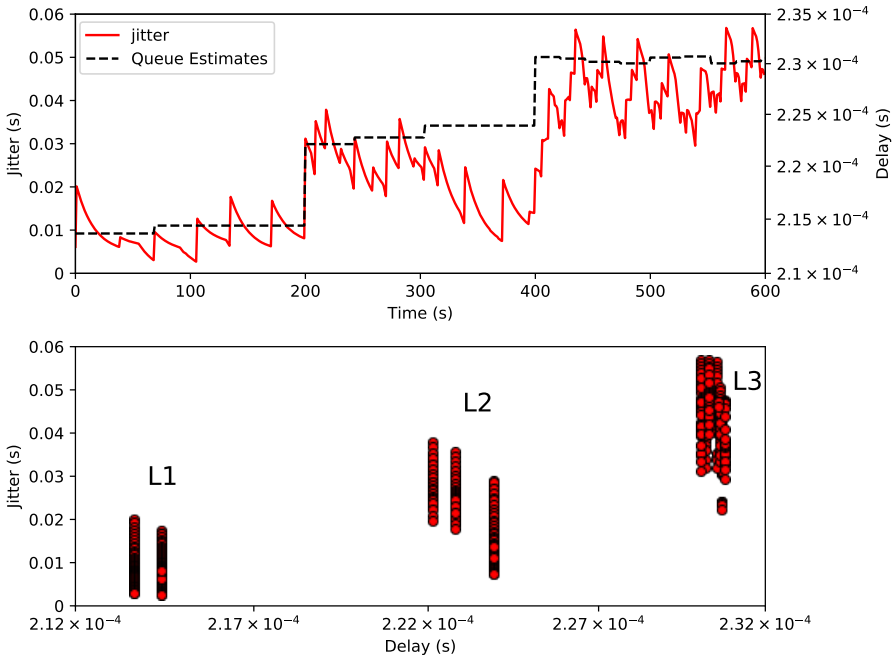

Fig. 5.7 Row 1 (R1) illustrates the automatic labelling process for the 3-flows case with 200 measurements drawn from each congestion level. The queue estimates are distinct for the three levels of congestion. This makes it possible to automatically label jitter using the queue estimates. In R2, a scatter-plot of the QD estimates and jitter depicts the formation of three distinct clusters for the data in R1. Each cluster represents the different levels of network congestion: low, medium and high (Level 1 (L1), L2, and L3). The high congestion cluster (L3) shows densely-bound clusters indicating high number of packets in the queue.

depicting the different levels of network congestion. These clusters reveal some additional information. The cluster depicting the high congestion level on the right shows some extreme network conditions. This is shown with the formation of tightly-bounded clusters. The queue build-up is at its highest level for this stage. The densely packed network estimates here indicate the high buffer occupancy levels. The variations in the queue estimates are more noticeable for the low and medium clusters.

An examination of the automatic labelling process for the 5-flows worst case scenario is conducted by plotting a scatter-plot of 10,000 data samples of jitter and queue estimates with 2000 samples drawn from each of the five levels in Fig. 5.8, Row 1. It can be seen that there are overlaps in cluster assignments as some queue estimates from the Level 5 (L5) region

were incorrectly assigned to the L1 region. There is also some evidence of data overlaps from L4 to the L5 data regions. As the levels of background traffic increases, the formation of clusters become challenging. Fig. 5.8, Row 2 depicts a kernel density plot of the data to demonstrate the effect of the mismatch between the queue estimates and the jitter traces for these different levels of congestion. There is a spread of the different congestion levels across all levels which explains the drop in accuracy. Some of the L5 estimates spread into the other clusters. The effect of this is that some of the jitter traces belonging to clusters L1 to L4 may be incorrectly assigned to L5. Similarly, there are some overlaps for some L2 estimates to L1 and L3. This is the same as some L4 estimates are assigned to some L5 jitter traces. This leads to incorrect assignment of QD estimates to the jitter traces. In these cases, achieving smoothly graduated labelling process may not be useful as the clusters increase. In these sort of cases, having the ability to automatically label the jitter traces to low, medium or high, or low to high is sufficient.

5.5.3 Automatic Labelling: Multiple Application Layer Traffic

This section evaluates the jitter automatic labelling process for a video streaming session in the presence of multiple application layer traffic. In these sets of experiments, the D-ITG load generator is configured in a multiple application mode with varying characteristics according to the profiles listed in Table 5.3. All profiles are configured to have three levels of congestion. Unless otherwise configured, the D-ITG load generator packet size is 512 bytes with a packet rate of 1000 packets per second. The streaming session lasts for 620 seconds.

In Table 5.3, Bursty (BE) refers to bursty traffic generated according to an exponential distribution. For Bursty (BE) traffic, the ON and OFF periods are set to a packet arrival rate of 1500 and 3000 respectively. For Bursty (BE*), this ON and OFF periods is set to 100 and 200. For Bursty (BC), which is bursty traffic generated with a constant inter-departure time, the ON and OFF periods are set to 500 and 1000 pps.

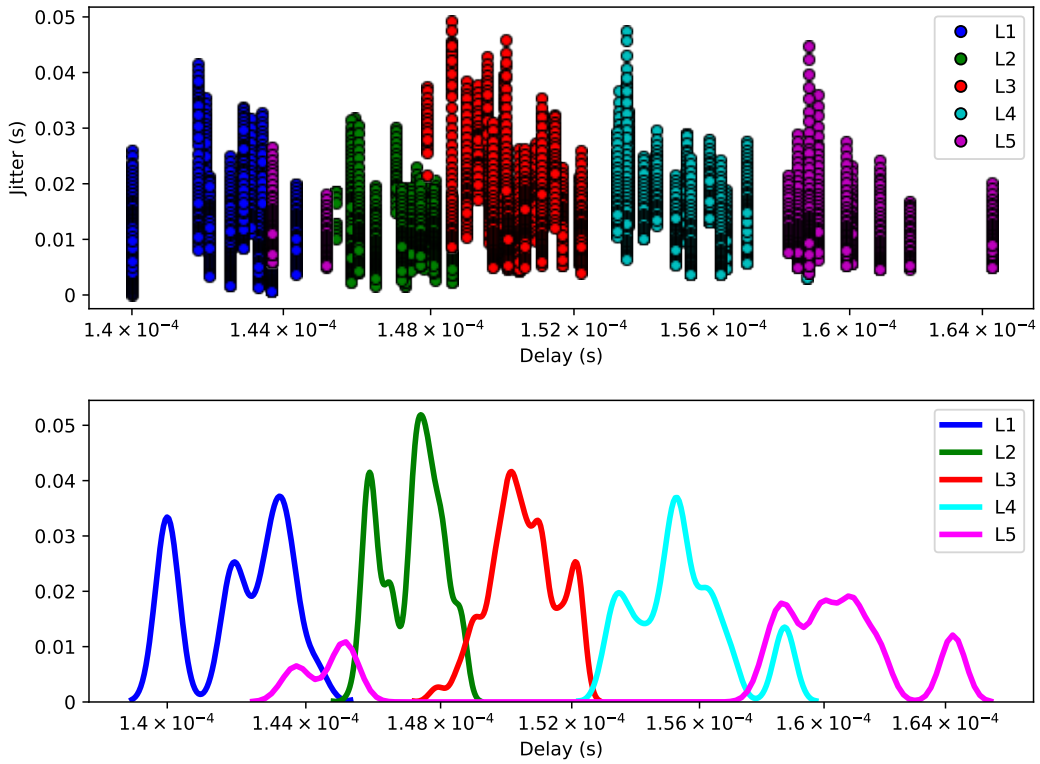

Fig. 5.8 Row 1 (R1): A scatter-plot of the 5-flows model (Level 1 (L1), L2, L3, L4 and L5) with 10,000 observations reveals some overlaps of queue estimates in the clusters formed. This results to a drop in the accuracy of the automatic labelling technique due to incorrect assignments of the QD estimates to the jitter. In R2, a kernel density plot reveals the overlaps between the estimates computed for the different clusters. This accounts for the inaccuracies in the labelling of the jitter data using the QD estimates.

Fig. 5.9 illustrates the packet counts captured for the video streaming session using the traffic profiles listed in Table 5.3. The goal of these experiments is to evaluate the automatic labelling technique under extreme network conditions. The automatic labelling process using the same configurations are repeated for these set of experiments as before. First, the queue estimates computed from the data shown in Fig. 5.9 using the k-means clustering algorithm are clustered as before. Then, the estimates for all the dataset (Profiles 1, 2, 3 and 4) are computed using a k value of 3. This is a design choice because the traffic generator was set up to generate three different levels of background traffic. Using the same data, the c is computed and the resultant analysis is used in choosing the k number of clusters. In two cases, the

Table 5.3 Background traffic profiles are listed for the multiple application traffic experiments. The bursty traffic are configured in two modes. Bursty (BE) refers to bursty traffic generated according to an exponential distribution. Bursty (BC) refers to bursty traffic generated according to a constant rate distribution.

		Packet Arrival Rates	Packet Size	Start	Stop
Profiles	Application Traffic Type	(pps)	$(bytes)$	$(seconds)$	$(seconds)$
Profile 1	ICMP, Bursty (BE), DNS	5500	2048	0	620
	ICMP, Gaming	3000	2048	198	620
	ICMP	1900	1024	396	620
Profile 2	ICMP, Bursty (BE), DNS	5500	2048	0	620
	ICMP	2000	1024	198	620
	Gaming	2000	1024	198	462
	ICMP	1900	1024	396	620
Profile 3	ICMP, Bursty (BE), DNS	5500	2048	0	620
	ICMP	2000	1024	198	620
	ICMP	1600	1024	396	620
Profile 4	ICMP, Bursty (BC)	2000	1536	0	620
	ICMP	400	1024	198	620
	Bursty (BE*)	200	512	198	462
	ICMP	200	1024	396	620

Table 5.4 The k value for the k-means algorithm number of clusters varies in some cases from the design chosen values when compared with the silhouette coefficients.

Method	Design Choice	Silhouette Analysis
Profile 1	3	4
Profile 2	3	3
Profile 3	3	4
Profile 4	3	3

ideal number of clusters as found by the silhouette coefficients differed from the three-cluster design. The silhouette coefficients suggested a k value of 4 for the data captured with Profiles 1 and 3. The issue with using cluster analysis methods such as the silhouette coefficients is that the process may produce results different from ground truth [280]. Other techniques for determining the number of clusters include Silhouette with Principal Component Analysis

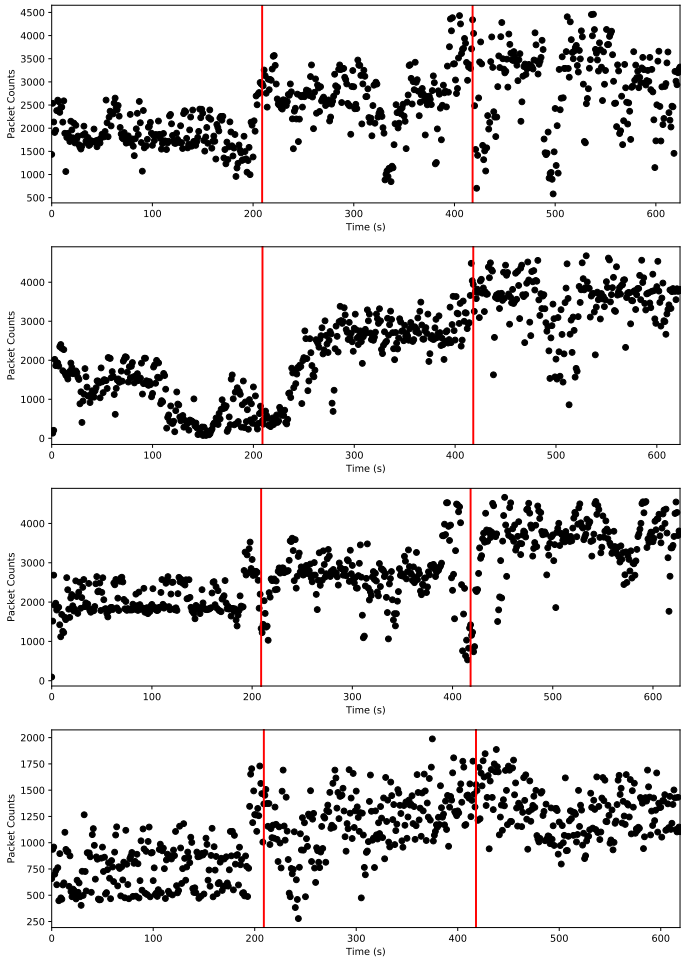

Fig. 5.9 Row 1 (R1), R2, R3 and R4 show the packet counts for the video streaming sessions with the D-ITG Profiles 1, 2, 3 and 4 respectively. The vertical red lines indicate the transition points from one congestion level to another.

(PCA) algorithm [281] and the Davies Bouldin algorithm[282, 283]. In these two instances, the queue estimates are computed using k values of 3 and 4. The k values used in these experiments are listed in Table 5.4.

Table 5.5 lists the results of the automatic labelling process under multi-application traffic types using the k values listed in Table 5.4. The best accuracy is recorded for the data with Profile 4 with an accuracy of 90.19%. The worst performance for k set to 3 is recorded with the Profile 3 dataset. This is an accuracy of approximately 78%. It can be observed that this accuracy drops to approximately 64% when evaluated with k set to 4 as suggested by the c

Table 5.5 The accuracy of the automatic jitter labelling process is listed for the multi-application traffic cases. In cases where the same k values are suggested, the queue estimates for labelling the jitter are computed with $k = 3$. For data captured using profiles 1 and 3, the queues estimates are clustered using k values of 3 and 4.

Model	Accuracy with $k = 3$ (%)	Accuracy with $k = 4$ (%)
Profile 1	84.39	71.67
Profile 2	85.89	–
Profile 3	77.95	63.69
Profile 4	90.19	–

method. This is a 14% drop in accuracy due to incorrect labelling. Similarly, the accuracy of the data from Profile 1 drops from 84% to 72% for k values of 3 and 4 respectively. This is because the formation of four clusters would lead to the assignment of the queue estimates to jitter traces from a different timestamp. The plots in Fig. 5.10, R1 and R2 depicts the automatic labelling process for the Profile 3 dataset.

In Fig. 5.10, R1 the accuracy of automatically labelling the Profile 3 dataset with a k value of 3 is illustrated. The plot shows 300 observations drawn from each of the three different levels. The queue estimates for time indices $1 < t < 550$, does not seem to be distinguishable. This leads to errors in labelling the jitter traces for three different levels as belonging to the same congestion state. The time indices between $550 < t < 700$ seem to have been inaccurately labelled accounting for the drop in accuracies. Fig. 5.10, R2 illustrates the accuracy of the automatic labelling technique for Profile 3 with a k value of 4. The data shown in this plot comprises of 200 observations each taken from three different congestion levels. The entire 600 observations shown seem to have been incorrectly labelled. This is due to the assignment of queue parameters to congestion levels that differ from network conditions from which the samples were drawn. For instance, the assignment of queue estimates captured during the first 100 seconds of the experiment to jitter values from a later time in future would lead to a mismatch. The results demonstrate the effectiveness of the automatic labelling process for constant bitrate traffic and for some severe network

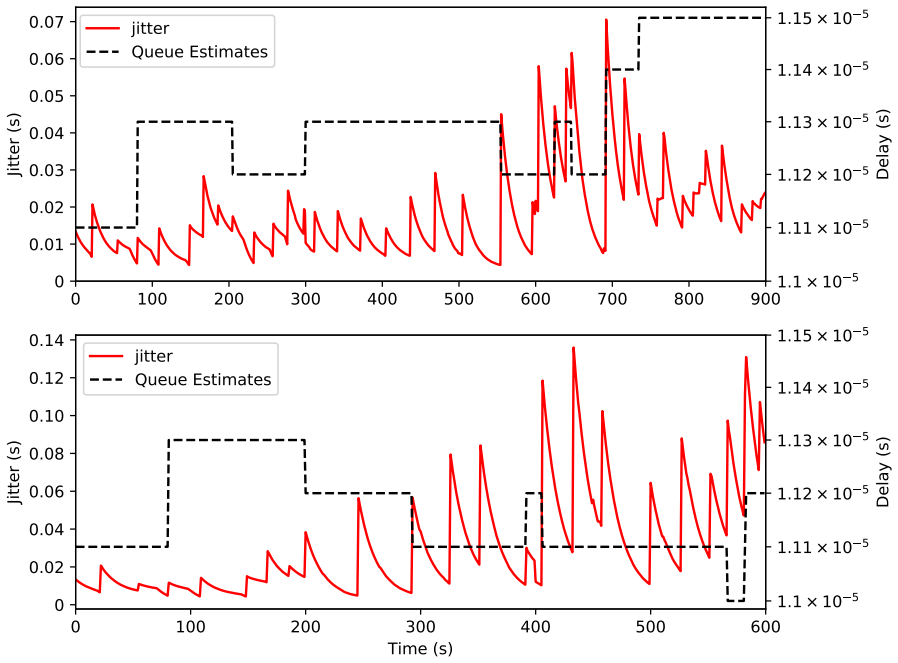

Fig. 5.10 In Row 1, R1, the automatic labelling technique for Profile 3 with a k value of 3 is illustrated. The highly congested network state with no distinct varying levels of congestion leads to a poor performance of the automatic labelling process. In Row 2 (R2), the accuracy of the automatic labelling technique for Profile 3 with a k value of 4 is shown. The plot depicts the struggle of the automatic labelling model to accurately label the jitter data using the correct queue estimates. In both cases, the QD estimates are not distinct for the three levels.

conditions. However, as the conditions in the network become extreme, there may not be clearly formed and distinct clusters which may affect the accuracy of the automatic labelling technique.

5.5.4 Network State Classification

This section presents the results from the network state classification tasks using the unsupervised technique introduced in this chapter and the baseline supervised approach [224]. The models are evaluated and compared using the accuracy metric introduced in Equation 4.16. In addition, the models are evaluated using two additional classification metrics, precision and recall. The precision of a model evaluates how well a model predicts positive instances

which are indeed positive. In other words, the precision of a model shows how much we can trust that the model is accurate when predicting the network congestion as high given a data sample from the high congestion class. The precision of a model is computed as

$$M_P\,(\%) = \frac{T_P}{T_P + F_P} \times 100. \tag{5.8}$$

Recall is a metric that evaluates how sensitive the ML classifier is to detecting positive instances from the observations. The recall metric is calculated by dividing the number of T_P elements by the total number of positively classified units [284]. The recall of a model is computed as

$$M_R\,(\%) = \frac{T_P}{T_P + F_N} \times 100. \tag{5.9}$$

For classification tasks, precision and recall are crucial metrics. The former being the percentage of results that are relevant and the latter being the percentage of relevant results that the ML algorithm correctly classified. The problem domain considered in this thesis chapter seeks to maximise both metrics. Here, the model accuracy and precision will be the dominant factors to consider. This is because it would be more useful for the models to be more precise and accurate in its predictions compared with the sensitivity of the models. The ability of the models to precisely detect the congestion level in the network would help the network manager proactively manage the overall network infrastructure.

5.5.4.1 Full Dataset Multi-class Classification

The results from the models' multi-class classification tasks are tabulated in Table 5.6. The model accuracies for the supervised technique range is approximately $66\% \leq M_A \leq 83\%$. The best performance for this approach is achieved with the XGB model with the RF offering the worst classification performance. For the unsupervised approach, the model accuracies range is approximately $83\% \leq M_A \leq 85\%$. It can be observed that the accuracies are better using

Table 5.6 The results of the full dataset multi-class classification tasks for the supervised and the unsupervised approach are listed. The classification results using the unsupervised approach provide superior performance compared with the supervised technique.

		Full Dataset Results		
Model	Algorithm	M_A	M_P	M_R
Supervised Approach	DT	75.70	75.50	75.67
	RF	65.64	65.30	65.74
	XGB	83.39	83.68	83.63
Unsupervised Approach	DT	84.27	84.39	84.26
	RF	83.35	84.63	83.46
	XGB	84.46	85.65	84.46

the unsupervised models. The XGB algorithm achieves the best classification accuracies with the DT models offering the worst classification accuracies. This can be attributed to the use of the gradient boosting technique and regularization of the XGB model which seeks to improve on past errors made by the model, learn from the process and improve on future performance.

The precision of the models is listed in Table 5.6. In assessing the precision of the models, the question of interest is: when a positive sample is predicted, how often is the prediction correct? The model precision using the unsupervised approach has the range $84.39\% \leq M_P \leq 85.65\%$. The M_P range for the supervised approach is $65\% \leq M_P \leq 84\%$. The variance is greater for the supervised approach. The model precision, M_P, is then evaluated for each congestion level using the XGB model for both techniques. Table 5.7 tabulates these results. Precision values of 97.47%, 80.09% and 73.48% are achieved by the supervised technique using the XGB classifier for low to high congestion. This is a difference of -1.45, 0.9 and -5.37 for low to high when compared with the M_P recorded by the unsupervised approach. The supervised approach M_P metric is only better for the medium congestion case. A similarity is observed for both techniques. The best model

Table 5.7 The precision and recall metrics for the individual congestion levels using the XGB model are shown for both approaches. The precision is better for the unsupervised models.

		XGB Results	
Model	Congestion	M_P	M_R
Supervised Approach	Low	97.47	87.17
	Medium	80.09	80.47
	High	73.48	83.25
Unsupervised Approach	Low	98.92	81.41
	Medium	79.19	72.40
	High	78.85	99.56

precision values using both approaches are recorded with the low congestion level with the lowest precision values achieved with the high congestion level. On the average, considering the worst case scenarios of both techniques, the supervised and unsupervised models are precise in detecting the correct congestion levels $\approx 75\%$ and 84% respectively.

Next, the model sensitivity, M_R, is evaluated. The XGB and the RF models produce the best and worst M_R values of 84.46% and 83.46% respectively for the combined data using the unsupervised approach. Compared with the supervised technique, the XGB and RF produce the best and worst M_R values of 83.63% and 65.74% respectively. The recall is better for the unsupervised models. The M_R for the individual congestion levels are explored for both techniques. Recall values of 81.41%, 72.40% and 99.56% are achieved by the unsupervised technique using the XGB classifier for low to high congestion. For the supervised approach, the model M_R values are 87.17%, 80.47% and 83.25% for the low to high congestion levels. The XGB with the unsupervised approach achieves the best M_R for the high congestion level. On the other hand, the XGB model with the supervised technique, achieves the best M_R of $\approx 84\%$ with the low congestion level. The model sensitivity for both the supervised and unsupervised approaches achieve a minimum of $\approx 75\%$ and 83% respectively.

Table 5.8 The results of the batch dataset classification tasks for the supervised and the unsupervised approach are listed. The unsupervised network state classification offers ≈ 4% and 25% performance gains in terms of its classification accuracies over the supervised baseline approach for the best and worst-case scenarios respectively.

		Batch Results		
Model	Congestion	DT	RF	XGB
Supervised Approach	Low	66.79	56.60	88.30
	Medium	69.55	47.74	92.18
	High	72.37	66.93	93.00
Unsupervised Approach	Low	81.41	81.41	81.41
	Medium	72.43	72.69	78.37
	High	99.64	99.17	99.60

5.5.4.2 Batch Dataset Classification

Using the pre-trained ML models, batch experiments were evaluated by computing the classification accuracy on a per class basis. Table 5.8 lists the results for the batch dataset classification for both techniques. The supervised approach records its poorest classification accuracy of 47.74% with the RF classifier. There is a challenge in distinguishing between the low and medium congestion. In this case, approximately 53% observations from the medium congestion level are mis-classified. Most of the observations here are attributed to the low congestion network state. Overall, the best and worst performance recorded by the baseline approach are 93.00% and 47.74% respectively. The unsupervised network state classification offers approximately 4% and 25% performance gains in terms of its classification accuracies over the supervised baseline approach for the best and worst-case scenarios respectively. The same trend noticed with the medium congestion state was observed as was the case before. There is a struggle to distinguish between the low and medium congestion states. For instance, with all classifiers, approximately 28% observations of the medium congestion level are mis-classified as low congestion state. The performance of the DT and RF classifiers with

both approaches are significantly boosted by including the XGB classifiers. The XGB model records classification accuracies of 81% and above. The ML classifiers for both approaches struggle in classifying the low and medium congestion states. These results suggest that there may not be much value in having the ability to detect three levels of congestion. The results suggests that the capability to detect the network state as high or low may suffice to properly monitor and manage network resources.

5.5.4.3 Time-varying Classification

In these sets of experiments, time-varying system classification was conducted using the unsupervised approach. First, a time-varying dataset is genererated by combining different numbers of samples drawn from the three congestion levels. Then, using pre-trained ML classifiers, the overall system responsivity, R_T, is computed and the system service change-points were evaluated. Time-varying datasets with 100, 200, 500, 1000, 1500, 2000 and 5000 samples drawn from each congestion level were generated. The results of the time-varying experiments are listed in Table 5.9. The samples are randomly drawn from the different regions in the order of increasing levels of congestion from low to high. The best system responsivities, R_T, of $\approx 97\%$, 98% and 99% are obtained with 500 samples for each congestion level with the DT, RF and XGB classifiers respectively. The worst R_T metrics are recorded with 2000 samples for each congestion level. Here, the DT, RF and XGB classifiers achieve system responsivities of $\approx 61.22\%$, 62.83% and 62.83% respectively.

The service change-point accuracies of some system responsivities listed in Table 5.9 were explored. Table 5.10 lists the results of the service change-point accuracies for 100, 500 and 1000. With 100 samples drawn from each congestion level, the XGB classifier achieves a system responsivity, R_T of 96.67%. In this setup, the classifier correctly detects all 100 samples for the low and high congestion regions, and is 90% accurate in predicting the medium congestion levels. The classifier mis-classifies 10 samples from the medium

Table 5.9 The results of the time-varying experiments are listed. The classification accuracy range is $61\% \leq M_A \leq 99\%$. The XGB classifier produces the best classification accuracy of 99.13% with the DT classifier producing the worst classification accuracy of 61.22%.

Model	Samples Per Level	Time-varying Results		
		DT	RF	XGB
Unsupervised Approach	100	86.00	94.67	96.67
	200	91.50	95.83	96.83
	500	96.87	98.20	99.13
	1000	92.23	91.53	88.90
	1500	94.78	93.24	95.20
	2000	61.22	62.83	62.83
	5000	95.18	96.31	96.50

congestion level as high congestion samples. The worst performance for this setup is achieved with the DT classifier. This is a R_T of 86%. This is achieved by correctly detecting all 100 samples for the low and high congestion traces, and 58 samples for the medium congestion region. The DT classifier incorrectly classifies 42 samples from the medium congestion level as high congestion samples.

The best classification accuracy is achieved with 500 observations drawn from each congestion level. The XGB classifier incorrectly classifies 13 samples from the medium congestion level as low congestion. The classifier correctly classifies all the observations from the low and high congestion levels. The RF and DT classifiers achieve 100% accuracies in detecting the low and high congestion levels while mis-classifying 27 and 47 observations respectively. From Table 5.9, it can be observed that the RF algorithm achieves the best R_T with the 3000 samples in which 1000 observations have been drawn from each congestion level. This is a R_T of 91.53%. The classifier is 100% accurate in detecting the low and high congestion levels. The model incorrectly classifies 254 medium congestion observations as high congestion samples. The RF classifier accuracy in detecting the medium congestion level is 74.60%. The XGB classifier produces the worst system responsivity for all ML classifiers in this setup. It produces a R_T of 88.90%. This is a 100% accuracy in detecting

Table 5.10 The results of the service change-point experiments using the unsupervised models are listed. The number of correctly detected samples for low, medium and high congestion states for different ML classifiers are listed.

Model	Samples Per Level(Model)	Correctly Detected Samples		
		Low	Medium	High
Unsupervised Approach	100 (XGB)	100	90	100
	100 (DT)	100	58	100
	500 (XGB)	500	487	500
	500 (RF)	500	473	500
	500 (DT)	500	453	500
	1000 (XGB)	1000	667	1000
	1000 (RF)	1000	746	1000

the low and high congestion levels. The model records a 66.7% accuracy in classifying the medium congestion observations. A total of 333 samples from the medium congestion level were incorrectly classified as high congestion samples. This is the only time that the XGB classifier accuracies were lower than the accuracies achieved by the DT and RF models.

It can be observed that in most of the scenarios evaluated, the drops in classification accuracies were as a result of the incorrect classification of the medium congestion samples as either low or high congestion levels. To boost classification accuracies, the study in [224] included deviation in the parameter estimates as additional features. There might be value in considering the deviation in the parameter estimates as a technique to improve the classification accuracies and distinguish between the estimates from different congestion levels. However, the classification accuracies are quite encouraging and can be used in the current setting to detect the state of the network. On the other hand, due to the dynamicity in network conditions, achieving clearly calibrated congestion detection may prove to be counter-productive. This is because constantly initiating corrective measures could lead to unstable network operations. The ability to classify the network congestion as either low or high may suffice to ensure stable operations.

5.6 Conclusion

This chapter presented a new approach for automatically labelling jitter using estimates of a simple queueing model in the presence of interfering traffic from other applications. When the interfering workload was constant bitrate UDP traffic, the automatic labelling procedure achieved high accuracies (> 92%). The accuracy of the labelling procedure dropped as the levels of the background congestion increased. This is due to overlapping cluster formation in the process of assigning the queue estimates to clusters. The accuracies of the automatic labelling process in the presence of multiple application traffic types range from 64% to 90% for three levels of congestion. In these extreme network conditions, the accuracy of the labelling technique was impacted by the cluster assignments of the queue estimates. Incorrect assignments of the queue estimates to the wrong clusters led to inaccurate labelling of the jitter data. When the parameters of the unsupervised labelling technique were used in a congestion classification, the XGB classifier achieved good classification accuracy and precision of 84% and 85% respectively. The unsupervised approach for network state congestion recorded $\approx$ 1.07% and 1.97% performance gains in terms of its classification accuracy and precision over the supervised approach. Generally, the XGB classifier boosted the classification performance of the DT and RF classifiers. All three classifiers recorded high classification accuracies of 99% and above in detecting the high congestion samples for the batch experiments. The labelled jitter estimates for the medium and high congestion zones had a substantial amount of overlap. As a result, network data from medium congestion regime was incorrectly classified as belonging to high congestion class. From these results, having the ability to classify the network state as being in either low or high congestion will suffice to enable automatic network management.

Chapter 6

Conclusion and Future Direction

THIS chapter concludes the thesis, provides a summary of the findings and the research limitations. It also provides some recommendations for the direction of future work.

6.1 Thesis Contributions and Conclusions

This thesis investigated the effects of network dynamics on QoD prediction and monitoring for video delivery networks. Most state-of-the-art solutions have blindly applied ML to this problem. This thesis focused on incorporating network dynamics in the ML models designed for this purpose. In comparison with alternative ML models, the techniques proposed in this thesis come at lesser computational costs making them a viable choice. The proposed methodologies in this thesis used QoD metrics, which are concerned with the quality of the data delivery process. The interest in using these statistics is based on the fact that these measurements capture the capability of the network infrastructure to reliably ensure video delivery.

The contributions of this thesis are summarized as follows:

1. The thesis extended the scope of the LA technique by contributing an adaptive technique that enables the automation of the choice of the regression parameter. This speeds up the learning process. A second thesis contribution in this area is the LA technique for intelligent resource management and video server demand profiling. This is applicable for cloud-hosted servers which may need to automate the provision of services by monitoring the health of the video network infrastructure. Using the LA technique, the thesis results demonstrated that it is achievable to accurately predict different number of users on the system. The ability to predict server loads could help a network manager to proactively manage the dynamicity of cloud provisioned resources. This can allow the network manager to accurately estimate the ideal number of resources that must be allocated to satisfy the predicted demand in order to optimize the service response time and reduce over-provisioning. These findings demonstrated that off-the-shelf ML techniques may be enhanced and automated without the need for additional data sources, which is important given the growing dynamicity of current service delivery and host infrastructures.

2. A second contribution of this thesis is a new model for estimating jitter, the Codec-aware Network Adaptation Agent (CNAA). The CNAA model achieves accurate predictions of QoD metrics, jitter by modelling the adaptive feedback loop between codecs and network congestion. The thesis results demonstrated the superior prediction performance gains of the CNAA model over some baseline models such as EWMA and ARMA. The CNAA technique achieved accuracies of approximately 93.8% and 95.8% in terms of MAE and RMSE respectively over the best performing baseline model. The superior performance of the CNAA model is due to the ability of the technique to model the dynamics present in the network. A second contribution with the CNAA model is the extension of the model to network state congestion detection and classification. The thesis results demonstrated how the model can be

used for network state classification using features of the CNAA model. The CNAA tree-based classifiers recorded approximately 80 - 97% classification accuracies. The thesis demonstrated the accuracy of the CNAA technique in detecting network service change-points. This change-point detection accuracy estimates how soon the classifier is able to detect the elevated or reduced levels of network congestion. These results are of relevance to network service practitioners. The high accuracy and response times of the CNAA classifiers demonstrate that they could be used to proactively re-route traffic or initiate an action to forestall further problems in the network.

3. The thesis contributed a novel method for automatic labelling of jitter data. This approach relies on the network to enable the data labelling method. As such, it eliminates the HITL element and removes the requirement for a network manager to label the data before training the ML models. The thesis posed the automatic data labelling as the problem of estimating the parameters of the network queue in the presence of interfering background traffic. The automatic labelling process, the unsupervised technique was compared with a baseline supervised approach which uses labelled data computed from the jitter model introduced in Chapter 4. The results found that the unsupervised technique produced better performance over the baseline approach. For instance, the findings showed that the unsupervised network state classification achieved approximately 4% and 25% performance gains in terms of its classification accuracies over the supervised baseline approach for the best and worst-case scenarios respectively using the batch datasets. In most scenarios, the results revealed that the network queue estimates from the medium congestion level overlapped with the low or high congestion samples. This led to inaccuracies in the labelled data and consequently accounted for the mismatches in network congestion level detection. The results from this analysis suggests that having the ability to classify

the network state as being in either low or high congestion would suffice to enable autonomic network management.

6.2 Research Limitations

The contributions in this thesis have demonstrated the efficacy of the proposed methods through empirical experimental evaluations. However, there are some limitations of this work which are summarized as follows:

- The experimental evaluations adopted in this work were mainly in wired settings and SDN emulated environments. The thesis did not consider wireless frameworks. It is worthy to note that there may be various sources of interference for wireless settings which may introduce additional network dynamics to consider.
- The proposed methods in this thesis do not focus on a specific video streaming method. The thesis considered a fundamental problem in video streaming network as opposed to considering a specific technology. However, there may be some other sources of network artifacts and QoD metrics to consider in a specific type of video streaming architecture such as HAS.

6.3 Future Work

This section enumerates some future work directions in the context of the methods proposed in this thesis and the research area in general.

- The LA technique for proactive resource management decisions introduced in Chapter 3 formulated the problem of estimating cloud computational requirements as an integrated framework comprising of a learning and an action stage. The learning stage proposed using ML models to predict the video QoD metric for cloud-hosted servers

and using the knowledge gained from the process to make resource management decisions during the action stage. The contribution introduced in this study is limited to the learning stage. Further research will be conducted to complete the framework by considering the action stages. This action stage will examine the efficacy of utilizing outputs from the network monitoring learning platform to initiate an alert when the video QoD falls below a specified threshold.

- This thesis examined the feasibility of the CNAA model introduced in Chapter 4 in SDN environments. The learning framework was formulated as a combination of an SDN topology discovery and statistics gathering component, learning phase and an action phase [82]. The last component was not investigated in this study. This is a future work direction in which the aim is to utilize the centralized SDN framework in initiating video flow re-routing strategies to avoid video quality impairments. The primary objective of the action stage would be to ensure that video QoD is maintained by guiding the video flows away from components that are likely to be congested. This would help preserve the quality of the video streaming session, while maintaining agreed service agreements.

Appendix A

Employability Skills and Discipline Specific Skills Training

I undertook the following training during the course of this PhD research programme.

- **Discipline Specific Skills Training**
  - Information Transmission and Management
  - Deep Learning
  - Algorithms and Approximation Theory
- **Employability Skills Training**
  - Research Integrity
  - Research Methods
  - Introduction to Statistics
  - Data Visualisation

The discipline specific modules were chosen to help equip me with the necessary technical knowledge and skills with regards to my research. The employability skills training modules were chosen to complement the discipline modules. Particularly, to be aware of the importance of objective research, research ethics, to improve my report writing skills, research data interpretation and communication of results.

References

[1] Cisco Systems 2017. Cisco Visual Networking Index: 2016—2021 White Paper. *Cisco*, 2017.

[2] Muzzafer Ali and Suchetana Chakraborty. Enabling video conferencing in low bandwidth. In *2022 IEEE 19th Annual Consumer Communications Networking Conference (CCNC)*, pages 487–488, 2022.

[3] N. Barman and M. G. Martini. QoE Modeling for HTTP Adaptive Video Streaming–A Survey and Open Challenges. *IEEE Access*, 7:30831–30859, 2019.

[4] M. Wang, Y. Cui, X. Wang, S. Xiao, and J. Jiang. Machine Learning for Networking: Workflow, Advances and Opportunities. *IEEE Network*, 32(2):92–99, 2018.

[5] R. Boutaba, M. A. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. M. Caicedo. A Comprehensive Survey on Machine Learning for Networking: Evolution, Applications and Research Opportunities. *J. Int. Ser. App.*, 9(16), June 2018.

[6] S. Ayoubi, N. Limam, M. A. Salahuddin, N. Shahriar, R. Boutaba, F. Estrada-Solano, and O. M. Caicedo. Machine Learning for Cognitive Network Management. *IEEE Comms. Mag.*, 56(1):158–165, 2018.

[7] Y. Cheng, J. Geng, Y. Wang, J. Li, D. Li, and J. Wu. Bridging Machine Learning and Computer Network Research: A Survey. *CCF Trans. on Net.*, 1(1-4):1–15, 2019.

[8] R. Yanggratoke, J. Ahmed, J. Ardelius, C. Flinta, A. Johnsson, D. Gillblad, and R. Stadler. Predicting Service Metrics for Cluster-based Services using Real-time Analytics. In *CNSM*, pages 135–143, 2015.

[9] Ruairí de Fréin. Effect of system load on video service metrics. In *2015 26th Irish Sig. and Sys. Conf. (ISSC)*, pages 1–6. IEEE, 2015.

[10] A. Bhatele, A. R. Titus, J. J. Thiagarajan, N. Jain, T. Gamblin, P. Bremer, M. Schulz, and L. V. Kale. Identifying the Culprits Behind Network Congestion. In *IEEE IPDPS*, pages 113–122, 2015.

[11] S. Fouladi, J. Emmons, E. Orbay, C. Wu, S. Riad Wahby, and K. Winstein. Salsify: Low-Latency Network Video through Tighter Integration between a Video Codec and a Transport Protocol. In *NSDI*, page 267–282, in NSDI, 2018, pp. 267–282.

[12] Xingjiao Wu, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma, and Liang He. A survey of human-in-the-loop for machine learning. *arXiv preprint arXiv:2108.00941*, 2021.

[13] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J Santamaría, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: Concepts, cnn architectures, challenges, applications, future directions. *Journal of big Data*, 8(1):1–74, 2021.

[14] What is deep learning? | ibm. https://www.ibm.com/cloud/learn/deep-learning.

[15] Asharul Islam Khan and Salim Al-Habsi. Machine learning in computer vision. *Procedia Computer Science*, 167:1444–1451, 2020.

[16] HAN Zhiyan and WANG Jian. Speech emotion recognition based on deep learning and kernel nonlinear psvm. In *CCDC*, pages 1426–1430. IEEE, 2019.

[17] Jayashree Padmanabhan and Melvin Jose Johnson Premkumar. Machine learning in automatic speech recognition: A survey. *IETE Technical Review*, 32(4):240–251, 2015.

[18] Khalid Haseeb, Irshad Ahmad, Israr Iqbal Awan, Jaime Lloret, and Ignacio Bosch. A machine learning sdn-enabled big data model for iomt systems. *Electronics*, 10(18):2228, 2021.

[19] Sherief Hashima, Basem M ElHalawany, Kohei Hatano, Kaishun Wu, and Ehab Mahmoud Mohamed. Leveraging machine-learning for d2d communications in 5g/beyond 5g networks. *Electronics*, 10(2):169, 2021.

[20] Ihab Ahmed Najm, Alaa Khalaf Hamoud, Jaime Lloret, and Ignacio Bosch. Machine learning prediction approach to enhance congestion control in 5g iot environment. *Electronics*, 8(6):607, 2019.

[21] Obinna Izima, Ruairí de Fréin, and Mark Davis. Video quality prediction under time-varying loads. In *CloudCom*, pages 129–132. IEEE, 2018.

[22] Maria Torres Vega, Cristian Perra, and Antonio Liotta. Resilience of video streaming services to network impairments. *IEEE Trans on Broad*, 64(2):220–234, 2018.

[23] Cisco visual networking index: Forecast and trends, 2017–2022, white paper.

[24] P ITU. 910. subjective video quality assessment methods for multimedia applications. *International Telecommunications Union Telecommunication Sector*, 1999.

[25] Shyamprasad Chikkerur, Vijay Sundaram, Martin Reisslein, and Lina J Karam. Objective video quality assessment methods: A classification, review, and performance comparison. *IEEE Trans. Broad.*, 57(2):165–182, 2011.

[26] ITUTG Recommendation. 1011-reference guide to quality of experience assessment methodologies, 2013.

[27] Abdelhak Bentaleb, Bayan Taani, Ali C Begen, Christian Timmerer, and Roger Zimmermann. A survey on bitrate adaptation schemes for streaming media over http. *IEEE Comm. Surveys & Tut.*, 21(1):562–585, 2018.

[28] Zhengfang Duanmu, Kai Zeng, Kede Ma, Abdul Rehman, and Zhou Wang. A quality-of-experience index for streaming video. *IEEE J. of Selected Topics in Sig. Proc.*, 11(1):154–166, 2016.

[29] Quan Huynh-Thu and Mohammed Ghanbari. Scope of validity of psnr in image/video quality assessment. *Electronics letters*, 44(13):800–801, 2008.

[30] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[31] Stephen Wolf and MH Pinson. Reference algorithm for computing peak signal to noise ratio (psnr) of a video sequence with a constant delay. *ITU-T Contri. COM9-C6-E*, 2009.

[32] Martín Varela, Lea Skorin-Kapov, and Touradj Ebrahimi. Quality of service versus quality of experience. In *Quality of Experience*, pages 85–96. Springer, 2014.

[33] E.800 : Definitions of Terms Related to Quality of Service. https://www.itu.int/rec/T-REC-E.800-200809-I.

[34] ETSI TR 102 157 - V1.1.1 - Satellite Earth Stations and Systems (SES); Broadband Satellite Multimedia; IP Interworking over Satellite; Performance, Availability and Quality of Service.

[35] Tahir Nawaz Minhas. *Network Impact on Quality of Experience of Mobile Video*. PhD thesis, Blekinge Institute of Technology, 2012.

[36] Markus Fiedler, Hans-Jurgen Zepernick, Lars Lundberg, Patrik Arlos, and Mats I Pettersson. Qoe-based cross-layer design of mobile video systems: Challenges and concepts. In *2009 IEEE-RIVF International Conference on Computing and Communication Technologies*, pages 1–4. IEEE, 2009.

[37] Pramila P. Shinde and Seema Shah. A review of machine learning and deep learning applications. In *4th ICCUBEA*, pages 1–6, 2018.

[38] Oludare Isaac Abiodun, Aman Jantan, Abiodun Esther Omolara, Kemi Victoria Dada, Abubakar Malah Umar, Okafor Uchenwa Linus, Humaira Arshad, Abdullahi Aminu Kazaure, Usman Gana, and Muhammad Ubale Kiru. Comprehensive review of artificial neural network applications to pattern recognition. *IEEE Access*, 7:158820–158846, 2019.

[39] Henrik Brink, Joseph Richards, and Mark Fetherolf. *Real-world Machine Learning*. Simon and Schuster, 2016.

[40] Hilde JP Weerts, Andreas C Mueller, and Joaquin Vanschoren. Importance of tuning hyperparameters of machine learning algorithms. *arXiv preprint arXiv:2007.07588*, 2020.

[41] Mowei Wang, Yong Cui, Xin Wang, Shihan Xiao, and Junchen Jiang. Machine learning for networking: Workflow, advances and opportunities. *IEEE Network*, 32(2):92–99, 2017.

[42] George AF Seber and Alan J Lee. *Linear Regression Analysis*, volume 329. John Wiley & Sons, 2012.

[43] J Ross Quinlan. Decision trees and decision-making. *IEEE Trans. on Sys., Man, and Cybernetics*, 20(2):339–346, 1990.

[44] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[45] Beibei Wang, Dekun Zou, and Ran Ding. Support vector regression based video quality prediction. In *2011 IEEE Intl. Symp. on Multi.*, pages 476–481. IEEE, 2011.

[46] Rui Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Trans. on Neural Net.*, 16(3):645–678, 2005.

[47] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *J. of the Royal Stat. Society. series c (Applied Statistics)*, 28(1):100–108, 1979.

[48] Marc M Van Hulle. Self-organizing maps., 2012.

[49] Todd K Moon. The expectation-maximization algorithm. *IEEE Sig. Proc. Mag.*, 13(6):47–60, 1996.

[50] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Adv. in Neural Info.*, 27, 2014.

[51] Ruairí de Fréin. Source separation approach to video quality prediction in computer networks. *IEEE Comm. Ltrs.*, 20(7):1333–1336, 2016.

[52] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *J of Artificial Intelligence Research*, 4:237–285, 1996.

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

[54] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8(3-4):279–292, 1992.

[55] Yuxi Li. Deep reinforcement learning: An overview. 2017.

[56] Imran, Zeba Ghaffar, Abdullah Alshahrani, Muhammad Fayaz, Ahmed Mohammed Alghamdi, and Jeonghwan Gwak. A topical review on machine learning, software defined networking, internet of things applications: Research limitations and challenges. *Electronics*, Vol. 10(8), 2021.

[57] Mohammed Ali Al-Garadi, Amr Mohamed, Abdulla Khalid Al-Ali, Xiaojiang Du, Ihsan Ali, and Mohsen Guizani. A survey of machine and deep learning methods for internet of things (iot) security. *IEEE Comm. Surveys Tutorials*, 22(3):1646–1685, 2020.

[58] Mohammad Saeid Mahdavinejad, Mohammadreza Rezvan, Mohammadamin Barekatain, Peyman Adibi, Payam Barnaghi, and Amit P Sheth. Machine learning for internet of things data analysis: A survey. *Dig. Comms. and Net.*, 4(3):161–175, 2018.

[59] Laizhong Cui, Shu Yang, Fei Chen, Zhong Ming, Nan Lu, and Jing Qin. A survey on application of machine learning for internet of things. *Intl. J of Mach. Learn. and Cyber.*, 9(8):1399–1417, 2018.

[60] David J Miller, Zhen Xiang, and George Kesidis. Adversarial learning targeting deep neural network classification: A comprehensive review of defenses against attacks. *Proc. of the IEEE*, 108(3):402–433, 2020.

[61] Lizhen Tang and Qusay H. Mahmoud. A survey of machine learning-based solutions for phishing website detection. *Machine Learning and Knowledge Extraction*, 3(3):672–694, 2021.

[62] Ankush Meshram and Christian Haas. Anomaly detection in industrial networks using machine learning: A roadmap. In *Machine Learning for Cyber Physical Systems*, pages 65–72. Springer, 2017.

[63] Elike Hodo, Xavier Bellekens, Andrew Hamilton, Christos Tachtatzis, and Robert Atkinson. Shallow and deep networks intrusion detection system: A taxonomy and survey. 2017.

[64] Nasrin Sultana, Naveen Chilamkurti, Wei Peng, and Rabei Alhadad. Survey on sdn based network intrusion detection system using machine learning approaches. *Peer-to-Peer Net. and App.*, 12(2):493–501, 2019.

[65] Anna L. Buczak and Erhan Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Comms. Surveys Tut.*, 18(2):1153–1176, 2016.

[66] Safa Otoum, Burak Kantarci, and Hussein T Mouftah. On the feasibility of deep learning in sensor network intrusion detection. *IEEE Net. Ltrs.*, 1(2):68–71, 2019.

[67] Himanshu Sharma, Ahteshamul Haque, and Frede Blaabjerg. Machine learning in wireless sensor networks for smart cities: A survey. *Electronics*, 10(9), 2021.

[68] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Comms. surveys & Tutorials*, 21(3):2224–2287, 2019.

[69] Paulo Valente Klaine, Muhammad Ali Imran, Oluwakayode Onireti, and Richard Demo Souza. A survey of machine learning techniques applied to self-organizing cellular networks. *IEEE Comms.*, 19(4):2392–2431, 2017.

[70] Francesco Musumeci, Cristina Rottondi, Avishek Nag, Irene Macaluso, Darko Zibar, Marco Ruffini, and Massimo Tornatore. An overview on application of machine learning techniques in optical networks. *IEEE Comms. Surveys & Tutorials*, 21(2):1383–1408, 2019.

[71] Muhammad Usama, Junaid Qadir, Aunn Raza, Hunain Arif, Kok-Lim Alvin Yau, Yehia Elkhatib, Amir Hussain, and Ala Al-Fuqaha. Unsupervised machine learning for networking: Techniques, applications and research challenges. *IEEE Access*, 7:65579–65615, 2019.

[72] Zubair Md. Fadlullah, Fengxiao Tang, Bomin Mao, Nei Kato, Osamu Akashi, Takeru Inoue, and Kimihiro Mizutani. State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems. *IEEE Comms. Surveys Tutorials*, 19(4):2432–2455, 2017.

[73] Obinna Izima, Ruairí de Fréin, and Mark Davis. Evaluating load adjusted learning strategies for client service levels prediction from cloud-hosted video servers. *AICS*, 2259:198–209, 2018.

[74] M. A. Ridwan, N. A. M. Radzi, F. Abdullah, and Y. E. Jalil. Applications of machine learning in networking: A survey of current issues and future challenges. *IEEE Access*, 9:52523–52556, 2021.

[75] Sergiu Oprea, Pablo Martinez-Gonzalez, Alberto Garcia-Garcia, John Alejandro Castro-Vargas, Sergio Orts-Escolano, José García Rodríguez, and Antonis A. Argyros. A review on deep learning techniques for video prediction. *CoRR*, abs/2004.05214, 2020.

[76] Sana Aroussi and Abdelhamid Mellouk. Survey on machine learning-based qoe-qos correlation models. In *Intl. Conf. on Comp., Mgt. and Telecomm.*, pages 200–204, 2014.

[77] Muhammad Jawad Khokhar, Thibaut Ehlinger, and Chadi Barakat. From network traffic measurements to qoe for internet video. In *IFIP Net. Conf.*, pages 1–9, 2019.

[78] Donald W Marquardt and Ronald D Snee. Ridge regression in practice. *The American Statistician*, 29(1):3–20, 1975.

[79] Sunil L Kukreja, Johan Löfberg, and Martin J Brenner. A least absolute shrinkage and selection operator (lasso) for nonlinear system identification. *IFAC Proc.*, 39(1):814–819, 2006.

[80] Hui Zou and Trevor Hastie. Regularization and variable selection via the elastic net. *J. of the Royal Stat. Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

[81] Obinna Izima, Ruairí de Fréin, and Mark Davis. Predicting quality of delivery metrics for adaptive video codec sessions. In *IEEE CloudNet*, pages 1–7, 2020.

[82] Obinna Izima, Ruairí de Fréin, and Ali Malik. Codec-aware video delivery over sdns. In *2021 IFIP/IEEE Intl. Symp. on Int. Net. Managt. (IM)*, pages 732–733, 2021.

[83] Taralynn Hartsell and Steve Chi-Yin Yuen. Video streaming in online learning. *AACE journal*, 14(1):31–43, 2006.

[84] Xiangbo Li, Mahmoud Darwich, Mohsen Amini Salehi, and Magdy Bayoumi. A survey on cloud-based video streaming services. In *Advances in Computers*, volume 123, pages 193–244. Elsevier, 2021.

[85] Feng Lao, Xinggong Zhang, and Zongming Guo. Parallelizing video transcoding using map-reduce-based cloud computing. In *IEEE ISCAS*, pages 2905–2908, 2012.

[86] S. Varma. *Internet Congestion Control*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, p. 173-203, 2015.

[87] Dapeng Wu, Yiwei Thomas Hou, Wenwu Zhu, Ya-Qin Zhang, and Jon M Peha. Streaming video over the internet: Approaches and directions. *IEEE Trans. on Cir. and Sys. for Video Tech.*, 11(3):282–300, 2001.

[88] R. Pereira and E.G. Pereira. Video Streaming: Overview and Challenges in the Internet of Things. In *Pervasive Computing*, Intelligent Data-Centric Systems, pages 417–444. Academic Press, Boston, 2016.

[89] RFC 3550 - RTP: A Transport Protocol for Real-Time Applications. [Online]. Available: https://tools.ietf.org/rfcmarkup?rfc=3550&draft=&url=#section-6.4.1. (Accessed on 12/22/2021).

[90] RFC 2326 - Real Time Streaming Protocol (RTSP). https://tools.ietf.org/html/rfc2326.

[91] Mark Handley, Van Jacobson, Colin Perkins, et al. Sdp: Session description protocol, 1998.

[92] T. Friedman, R. Caceres, and A. Clark. RFC 3611 - RTP Control Protocol Extended Reports (RTCP XR). https://tools.ietf.org/html/rfc3611, 2003.

[93] RFC 7825 - A Network Address Translator (NAT) Traversal Mechanism for Media Controlled by the Real-Time Streaming Protocol (RTSP). https://tools.ietf.org/html/rfc7825.

[94] Sepandar D Kamvar, Mario T Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proc. of the 12th Intl. Conf. on World Wide Web*, pages 640–651, 2003.

[95] G Camarillo. Rfc 5694 peer-to-peer (p2p) architecture: Definition, taxonomies, examples, and applicability. *Network Working Group, IETF*, 2009.

[96] N. Ramzan, H. Park, and E. Izquierdo. Video Streaming over P2P Networks: Challenges and Opportunities. *Image Commun.*, 27(5):401–411, 2012.

[97] Yang-hua Chu, Sanjay G Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. *IEEE J. on Selected Areas in Comms.*, 20(8):1456–1471, 2002.

[98] DK Gifford, Kirk L Johnson, M Frans Kaashoek, and James W O'Toole Jr. Overcast: Reliable multicasting with an overlay network. In *Proc. of USENIX Symp. on OSDI*, 2000.

[99] Nazanin Magharei and Reza Rejaie. Prime: Peer-to-peer receiver-driven mesh-based streaming. *IEEE/ACM Trans. on Net.*, 17(4):1052–1065, 2009.

[100] Vinay Pai, Kapil Kumar, Karthik Tamilmani, Vinay Sambamurthy, and Alexander E Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Intl. Work. on Peer-to-Peer Systems*, pages 127–140. Springer, 2005.

[101] D. Stutzbach and R. Rejaie. Understanding Churn in Peer-to-Peer Networks. In *ACM SIGCOMM Conf. on Internet Measurement*, page 189–202, 2006.

[102] Yong Liu, Yang Guo, and Chao Liang. A survey on peer-to-peer video streaming systems. *Peer-to-Peer Networking and Applications*, 1(1):18–28, 2008.

[103] Yusuf Sani, Andreas Mauthe, and Christopher Edwards. Adaptive bitrate selection: A survey. *IEEE Comms. Surveys & Tutorials*, 19(4):2985–3014, 2017.

[104] R Pantos. Http live streaming, may 1, 2009. *Internet Engineering Task Force*.

[105] Dom Robinson. Live streaming ecosystems. *Advanced Content Delivery, Streaming, and Cloud Services*, pages 33–49, 2014.

[106] Iraj Sodagar. The mpeg-dash standard for multimedia streaming over the internet. *IEEE Multimedia*, 18(4):62–67, 2011.

[107] Jonathan Kua, Grenville Armitage, and Philip Branch. A survey of rate adaptation techniques for dynamic adaptive streaming over http. *IEEE Comms. Survs. Tut.*, 19(3):1842–1866, 2017.

[108] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A buffer-based approach to rate adaptation: Evidence from a large video streaming service. In *Proc. ACM Conf. on SIGCOMM*, pages 187–198, 2014.

[109] Kevin Spiteri, Rahul Urgaonkar, and Ramesh K Sitaraman. Bola: Near-optimal bitrate adaptation for online videos. *IEEE/ACM Trans. on Net.*, 28(4):1698–1711, 2020.

[110] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C Begen, and David Oran. Probe and adapt: Rate adaptation for http video streaming at scale. *IEEE J. on Selected Areas in Comms.*, 32(4):719–733, 2014.

[111] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. In *Proc. of Intl. Conf. on Emerg. Net. Expts. and Tech.*, pages 97–108, 2012.

[112] Hiba Yousef, Jean Le Feuvre, and Alexandre Storelli. Abr prediction using supervised learning algorithms. In *IEEE MMSP*, pages 1–6, 2020.

[113] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *Proc. of ACM Conf. on Special Interest Group on Data Comm.*, pages 325–338, 2015.

[114] Microsoft Silverlight Smooth Streaming. [Online]. Available: https://mssilverlight.azurewebsites.net/silverlight/smoothstreaming/.

[115] Live video streaming online | adobe http dynamic streaming. https://business.adobe.com/ie/products/primetime/adobe-media-server/hds-dynamic-streaming.html.

[116] Apple HTTP Live Streaming (HLS), Apple. [Online]. Available: https://developer.apple.com/streaming/.

[117] Gregory Cermak, Margaret Pinson, and Stephen Wolf. The relationship among video quality, screen resolution, and bit rate. *IEEE Trans. on Broad.*, 57(2):258–262, 2011.

[118] Zhi Li, Ali C Begen, Joshua Gahm, Yufeng Shan, Bruce Osler, and David Oran. Streaming video over http with consistent quality. In *Proc. of ACM Mulit. Sys. Conf.*, pages 248–258, 2014.

[119] Bing Wang, Jim Kurose, Prashant Shenoy, and Don Towsley. Multimedia streaming via tcp: An analytic performance study. *ACM Trans. on Multi. Comp., Comms., and App.*, 4(2):1–22, 2008.

[120] Sheng-sheng Yu, Jun Zhang, Jing-li Zhou, and Xin Zhou. A flow control scheme in video surveillance applications [j]. *Comp. Engr. & Sci.*, 9, 2005.

[121] Jaroslav Frnda, Miroslav Voznak, and Lukas Sevcik. Impact of Packet Loss and Delay Variation on the Quality of Real-Time Video Streaming. *Telecom. Syst.*, 62(2):265–275, June 2016.

[122] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hoßfeld, and Phuoc Tran-Gia. A survey on quality of experience of http adaptive streaming. *IEEE Comms. Surv. Tut.*, 17(1):469–492, 2015.

[123] M. T. Vega, D. C. Mocanu, and A. Liotta. Unsupervised Deep Learning for Real-Time Assessment of Video Streaming Services. *Multi. Tools Appl.*, 76(21):22303–22327, 2017.

[124] Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[125] Ruairí de Fréin, Cristian Olariu, Yuqian Song, Rob Brennan, Patrick McDonagh, Adriana Hava, Christina Thorpe, John Murphy, Liam Murphy, and Paul French. Integration of qos metrics, rules and semantic uplift for advanced iptv monitoring. *J. of Net. and Sys. Manag.*, 23(3):673–708, 2015.

[126] Darijo Raca, Ahmed H. Zahran, Cormac J. Sreenan, Rakesh K. Sinha, Emir Halepovic, Rittwik Jana, and Vijay Gopalakrishnan. On leveraging machine and deep learning for throughput prediction in cellular networks: Design, performance, and challenges. *IEEE Comms. Mag.*, 58(3):11–17, 2020.

[127] Abdelhak Bentaleb, Christian Timmerer, Ali C. Begen, and Roger Zimmermann. *Bandwidth Prediction in Low-Latency Chunked Streaming*, page 7–13. Asso. for Comp. Mach., 2019.

[128] Ali El Essaili, Thorsten Lohmar, and Mohamed Ibrahim. Realization and evaluation of an end-to-end low latency live dash system. In *IEEE BMSB*, pages 1–5, 2018.

[129] Yaakov Engel, Shie Mannor, and Ron Meir. The kernel recursive least-squares algorithm. *IEEE Trans. on Sig. Proc.*, 52(8):2275–2285, 2004.

[130] Video quality of service (qos) tutorial - cisco. https://www.cisco.com/c/en/us/support/docs/quality-of-service-qos/qos-video/212134-Video-Quality-of-Service-QOS-Tutorial.html#_ftn4.

[131] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. SIGCOMM '17, page 197–210. Asso. for Comp. Mach., 2017.

[132] Hongzi Mao, Shannon Chen, Drew Dimmery, Shaun Singh, Drew Blaisdell, Yuandong Tian, Mohammad Alizadeh, and Eytan Bakshy. Real-world video adaptation with reinforcement learning, 2020.

[133] Yin Zhao, Qi-Wei Shen, Wei Li, Tong Xu, Wei-Hua Niu, and Si-Ran Xu. Latency aware adaptive video streaming using ensemble deep reinforcement learning. In *Proc. of ACM Intl. Conf. on Multi.*, MM '19, page 2647–2651. Asso. for Comp. Mach., 2019.

[134] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied Logistic Regression*, volume 398. John Wiley & Sons, 2013.

[135] Trevor Hastie, Saharon Rosset, Ji Zhu, and Hui Zou. Multi-class adaboost. *Statistics and its Interface*, 2(3):349–360, 2009.

[136] Jerome H Friedman. Stochastic gradient boosting. *Compu. Stats. & data Analy.*, 38(4):367–378, 2002.

[137] Irina Rish et al. An empirical etudy of the naive bayes classifier. In *IJCAI Work. on Empirical Methods in AI*, volume 3, pages 41–46, 2001.

[138] Aman Kataria and MD Singh. A review of data classification using k-nearest neighbour algorithm. *Intl. J. of Emerg. Tech. and Adv. Eng.*, 3(6):354–360, 2013.

[139] Yusuf Sani, Darijo Raca, Jason J. Quinlan, and Cormac J. Sreenan. Smash: A supervised machine learning approach to adaptive video streaming over http. In *QoMEX*, pages 1–6, 2020.

[140] Santosh Srivastava, Maya R Gupta, and Béla A Frigyik. Bayesian quadratic discriminant analysis. *J. of ML Research*, 8(6), 2007.

[141] Selwyn Piramuthu, Michael J Shaw, and James A Gentry. A classification approach using multi-layered neural networks. *Decision Support Systems*, 11(5):509–525, 1994.

[142] Ricky KP Mok, Edmond WW Chan, and Rocky KC Chang. Measuring the quality of experience of http video streaming. In *IFIP/IEEE IM and Workshops*, pages 485–492. IEEE, 2011.

[143] Nick Feamster and Jennifer Rexford. Why (and how) networks should run themselves. *arXiv preprint arXiv:1710.11583*, 2017.

[144] David Naylor, Alessandro Finamore, Ilias Leontiadis, Yan Grunenberger, Marco Mellia, Maurizio Munafò, Konstantina Papagiannaki, and Peter Steenkiste. The cost of the "s" in https. In *Proc. of ACM Intl. on Conf. on Emerg. Net. Expts. and Tech.*, pages 133–140, 2014.

[145] Jeff Jarmoc and DSCT Unit. Ssl/tls interception proxies and transitive trust. *Black Hat Europe*, 2012.

[146] Justine Sherry, Chang Lan, Raluca Ada Popa, and Sylvia Ratnasamy. Blindbox: Deep packet inspection over encrypted traffic. In *Proc. of the ACM Conf. on Special Interest Grp. on Data Comm.*, pages 213–226, 2015.

[147] Irena Orsolic, Dario Pevec, Mirko Suznjevic, and Lea Skorin-Kapov. Youtube qoe estimation based on the analysis of encrypted network traffic using machine learning. In *IEEE Globecom*, pages 1–6, 2016.

[148] Gaya Buddhinath and Damien Derry. A simple enhancement to one rule classification. *Dept of Comp. Sci. & Soft. Eng.*, page 40, 2006.

[149] John Platt. Sequential minimal optimization: A fast algorithm for training support vector machines. 1998.

[150] Manish Mathuria. Decision tree analysis on j48 algorithm for data mining. *IntL. J. of Adv. Res. in Comp. Sci.and Soft. Eng.*, 3(6), 2013.

[151] Giorgos Dimopoulos, Ilias Leontiadis, Pere Barlet-Ros, and Konstantina Papagiannaki. Measuring video qoe from encrypted traffic. In *Proc. of IMC.*, IMC '16, page 513–526. Asso. for Comp. Mach., 2016.

[152] Pedro Casas, Michael Seufert, Florian Wamser, Bruno Gardlo, Andreas Sackl, and Raimund Schatz. Next to you: Monitoring quality of experience in cellular networks from the end-devices. *IEEE Trans. on NSM*, 13(2):181–196, 2016.

[153] Sarah Wassermann, Nikolas Wehner, and Pedro Casas. Machine learning models for youtube qoe and user engagement prediction in smartphones. *SIGMETRICS*, 46(3):155–158, January 2019.

[154] Kaushika Pal and Biraj. V. Patel. Data classification with k-fold cross validation and holdout accuracy estimation methods with 5 different machine learning techniques. In *ICCMC*, pages 83–87, 2020.

[155] Diego Didona and Paolo Romano. On bootstrapping machine learning performance predictors via analytical models. *arXiv preprint arXiv:1410.5102*, 2014.

[156] Sarah Wassermann, Michael Seufert, Pedro Casas, Li Gang, and Kuang Li. I see what you see: Real time prediction of video quality from encrypted streaming traffic. In *Proc. of Internet-QoE Workshop on QoE-Based Analy. and Manag. of Data Comm. Net.*, Internet-QoE'19, page 1–6. Asso. for Comp. Mach., 2019.

[157] Umer Saeed, Sana Ullah Jan, Young-Doo Lee, and Insoo Koo. Fault diagnosis based on extremely randomized trees in wireless sensor networks. *Reliability Eng. & Sys. Safety*, 205:107284, 2021.

[158] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[159] Adithi D. Chakravarthy, Sindhura Bonthu, Zhengxin Chen, and Qiuming Zhu. Predictive models with resampling: A comparative study of machine learning algorithms and their performances on handling imbalanced datasets. In *IEEE ICMLA*, pages 1492–1495, 2019.

[160] Sarah Wassermann, Michael Seufert, Pedro Casas, Li Gang, and Kuang Li. Let me decrypt your beauty: Real-time prediction of video resolution and bitrate for encrypted video streaming. In *TMA*, pages 199–200. IEEE, 2019.

[161] Craig Gutterman, Katherine Guo, Sarthak Arora, Xiaoyang Wang, Les Wu, Ethan Katz-Bassett, and Gil Zussman. *Requet: Real-Time QoE Detection for Encrypted YouTube Traffic*, page 48–59. Asso. for Comp. Mach., 2019.

[162] Craig Gutterman, Katherine Guo, Sarthak Arora, Trey Gilliland, Xiaoyang Wang, Les Wu, Ethan Katz-Bassett, and Gil Zussman. Requet: Real-time qoe metric detection for encrypted youtube traffic. *ACM Trans. Multi. Comput. Comm. Appl.*, 16(2s), 2020.

[163] Michael Seufert, Pedro Casas, Nikolas Wehner, Li Gang, and Kuang Li. Stream-based machine learning for real-time qoe analysis of encrypted video streaming traffic. In *ICIN*, pages 76–81. IEEE, 2019.

[164] Michael Seufert, Pedro Casas, Nikolas Wehner, Li Gang, and Kuang Li. Features that matter: Feature selection for on-line stalling prediction in encrypted video streaming. In *IEEE INFOCOM*, pages 688–695. IEEE, 2019.

[165] Vengatanathan Krishnamoorthi, Niklas Carlsson, Emir Halepovic, and Eric Petajan. *BUFFEST: Predicting Buffer Conditions and Real-Time Requirements of HTTP(S) Adaptive Streaming Clients*, page 76–87. Asso. for Comp. Mach., 2017.

[166] M. Hammad Mazhar and Zubair Shafiq. Real-time video quality of experience monitoring for https and quic. In *IEEE INFOCOM*, pages 1331–1339, 2018.

[167] Francesco Bronzino, Paul Schmitt, Sara Ayoubi, Guilherme Martins, Renata Teixeira, and Nick Feamster. Inferring streaming video quality from encrypted traffic: Practical models and deployment experience. *Proc. ACM Meas. Anal. Comput. Syst.*, 3(3):1–25, 2019.

[168] Suman Pandey, Mi Jung Choi, Jae-Hyung Yoo, and James Won-Ki Hong. Streaming pattern based feature extraction for training neural network classifier to predict quality of vod services. In *IFIP(IM)*, pages 551–557. IEEE, 2021.

[169] Susanna Schwarzmann, Clarissa Cassales Marquezan, Marcin Bosk, Huiran Liu, Riccardo Trivisonno, and Thomas Zinner. Estimating video streaming qoe in the 5g architecture using machine learning. In *Proc. of Internet-QoE Work. on QoE-based Analy. & Mgt. of Data Comm. Net.*, pages 7–12, 2019.

[170] Sabina Baraković and Lea Skorin-Kapov. Survey and challenges of qoe management issues in wireless networks. *J. of Comp.r Net. and Comm.*, 2013, 2013.

[171] Ivan Bartolec, Irena Orsolic, and Lea Skorin-Kapov. In-network youtube performance estimation in light of end user playback-related interactions. In *QoMEX*, pages 1–3. IEEE, 2019.

[172] Irena Orsolic, Mirko Suznjevic, and Lea Skorin-Kapov. Youtube qoe estimation from encrypted traffic: Comparison of test methodologies and machine learning based models. In *QoMEX*, pages 1–6. IEEE, 2018.

[173] Irena Oršolić, Petra Rebernjak, Mirko Sužnjević, and Lea Skorin-Kapov. In-network qoe and kpi monitoring of mobile youtube traffic: Insights for encrypted ios flows. In *CNSM*, pages 233–239. IEEE, 2018.

[174] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli. CS2P: Improving Video Bitrate Selection and Adaptation with Data-Driven Throughput Prediction. In *ACM SIGCOMM*, page 272–285, 2016.

[175] M. Claeys, S. Latré, J. Famaey, and F. De Turck. Design and Evaluation of a Self-Learning HTTP Adaptive Video Streaming Client. *IEEE Comm. Ltrs*, 18(4):716–719, 2014.

[176] Ruairí de Fréin. Take off a load: Load-adjusted video quality prediction and measurement. In *IEEE ICCIT*, pages 1886–1894. IEEE, 2015.

[177] Lorenzo Rossi, Jacob Chakareski, Pascal Frossard, and Stefania Colonnese. A poisson hidden markov model for multiview video traffic. *IEEE/ACM Trans. on Net.*, 23(2):547–558, 2014.

[178] Christos G. Bampis and Alan C. Bovik. Feature-based prediction of streaming video qoe: Distortions, stalling and memory. *Sig. Proc.: Image Comm.*, 68:218–228, 2018.

[179] Huyen T. T. Tran, Duc V. Nguyen, Nam Pham Ngoc, and Truong Cong Thang. Overall quality prediction for http adaptive streaming using lstm network. *IEEE Trans. on CSVT.*, 31(8):3212–3226, 2021.

[180] Yunbo Wang, Lu Jiang, Ming-Hsuan Yang, Li-Jia Li, Mingsheng Long, and Li Fei-Fei. Eidetic 3d lstm: A model for video prediction and beyond. In *ICLR*, 2018.

[181] Hossein Ebrahimi Dinaki, Shervin Shirmohammadi, Emil Janulewicz, and David Côté. Forecasting video qoe with deep learning from multivariate time-series. *IEEE Open J. of Sig. Proc.*, 2:512–521, 2021.

[182] Keith Kirkpatrick. Software-defined networking. *Comms. of the ACM*, 56(9):16–19, 2013.

[183] J. Xie, F. R. Yu, T. Huang, R. Xie, J. Liu, C. Wang, and Y. Liu. A Survey of Machine Learning Techniques Applied to Software Defined Networking (SDN): Research Issues and Challenges. *IEEE Comms. Surveys Tutorials*, 21(1):393–430, 2019.

[184] J. Carner, A. Mestres, E. Alarcón, and A. Cabellos. Machine Learning-based Network Modeling: An Artificial Neural Network Model vs a Theoretical Inspired Model. In *ICUFN*, pages 522–524, 2017.

[185] S. Jain, M. Khandelwal, A. Katkar, and J. Nygate. Applying Big Data Technologies to Manage QoS in an SDN. In *CNSM*, pages 302–306, 2016.

[186] Ali Malik, Ruairí de Fréin, and Benjamin Aziz. Rapid restoration techniques for software-defined networks. *Applied Sciences*, 10(10):3411, 2020.

[187] R. Pasquini and R. Stadler. Learning End-to-End Application QoS from Openflow Switch Statistics. In *IEEE NetSoft*, pages 1–9, 2017.

[188] A. Ben Letaifa. Adaptive QoE Monitoring Architecture in SDN Networks: Video Streaming Services Case. In *IWCMC*, pages 1383–1388, 2017.

[189] Stefano Petrangeli, Tingyao Wu, Tim Wauters, Rafael Huysegems, Tom Bostoen, and Filip De Turck. A machine learning-based framework for preventing video freezes in http adaptive streaming. *J. of Net. and Comp. App.*, 94:78–92, 2017.

[190] Chris Seiffert, Taghi M Khoshgoftaar, Jason Van Hulse, and Amri Napolitano. Rusboost: A hybrid approach to alleviating class imbalance. *IEEE Trans. on Sys., Man, and Cyber.-Part A: Systems and Humans*, 40(1):185–197, 2009.

[191] Diego Da Hora, Karel Van Doorselaer, Koen Van Oost, and Renata Teixeira. Predicting the effect of home wi-fi quality on qoe. In *IEEE INFOCOM*, pages 944–952. IEEE, 2018.

[192] Florian Wamser, Pedro Casas, Michael Seufert, Christian Moldovan, Phuoc Tran-Gia, and Tobias Hossfeld. Modeling the youtube stack: From packets to quality of experience. *Comp. Net.*, 109:211–224, 2016.

[193] Thomas Zinner, Oliver Hohlfeld, Osama Abboud, and Tobias Hoßfeld. Impact of frame rate and resolution on objective qoe metrics. In *QoMEX*, pages 29–34. IEEE, 2010.

[194] Amir Ligata, Erma Perenda, and Haris Gacanin. Quality of experience inference for video services in home wifi networks. *IEEE Comm. Mag.*, 56(3):187–193, 2018.

[195] Rajarshi Bhattacharyya, Bainan Xia, Desik Rengarajan, Srinivas Shakkottai, and Dileep Kalathil. Flowbazaar: A market-mediated software defined communications ecosystem at the wireless edge. *arXiv preprint arXiv:1801.00825*, 2018.

[196] Doreid Ammar, Katrien De Moor, Lea Skorin-Kapov, Markus Fiedler, and Poul E Heegaard. Exploring the usefulness of machine learning in the context of webrtc performance estimation. In *LCN*, pages 406–413. IEEE, 2019.

[197] Suying Yan, Yuchun Guo, Yishuai Chen, and Feng Xie. Predicting freezing of webrtc videos in wifi networks. In *Intl. Conf. on Ad Hoc Net.*, pages 292–301. Springer, 2018.

[198] Yanjiao Chen, Kaishun Wu, and Qian Zhang. From qos to qoe: A tutorial on video quality assessment. *IEEE Comms. Surveys & Tutorials*, 17(2):1126–1165, 2014.

[199] Part 3: How to compete with broadcast latency using current adaptive bitrate technologies | aws media blog. https://aws.amazon.com/blogs/media/part-3-how-to-compete-with-broadcast-latency-using-current-adaptive-bitrate-technologies/.

[200] Athula Balachandran, Vyas Sekar, Aditya Akella, Srinivasan Seshan, Ion Stoica, and Hui Zhang. Developing a predictive model of quality of experience for internet video. *ACM SIGCOMM Review*, 43(4):339–350, 2013.

[201] David Hands and Miles Wilkins. A study of the impact of network loss and burst size on video streaming quality and acceptability. In *International Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, pages 45–57. Springer, 1999.

[202] Tiia Ojanperä, Markus Luoto, Mikko Majanen, Petteri Mannersalo, and Pekka T Savolainen. Cognitive network management framework and approach for video streaming optimization in heterogeneous networks. *Wireless Personal Communications*, 84(3):1739–1769, 2015.

[203] Ruairí de Fréin. Load-adjusted video quality prediction methods for missing data. In *2015 10th International Conference for Internet Technology and Secured Transactions (ICITST)*, pages 314–319. IEEE, 2015.

[204] Sifan Liu and Edgar Dobriban. Ridge regression: Structure, cross-validation, and sketching. *arXiv preprint arXiv:1910.02373*, 2019.

[205] LE Melkumova and S Ya Shatskikh. Comparing ridge and lasso estimators for data analysis. *Procedia engineering*, 201:746–755, 2017.

[206] Cun-Hui Zhang and Jian Huang. The sparsity and bias of the lasso selection in high-dimensional linear regression. *The Annals of Statistics*, 36(4):1567–1594, 2008.

[207] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, pp. 43-56, 2009.

[208] Hui Zou and Trevor Hastie. Regression shrinkage and selection via the elastic net, with applications to microarrays. *JR Stat Soc Ser B*, 67:301–20, 2003.

[209] Mark R Segal. Machine learning benchmarks and random forest regression. 2004.

[210] Alan Miller. *Subset selection in regression*. CRC Press, 2002.

[211] Yuichi Takano and Ryuhei Miyashiro. Best subset selection via cross-validation criterion. *Top*, pages 1–14, 2020.

[212] Amy R Reibman, Vinay A Vaishampayan, and Yegnaswamy Sermadevi. Quality monitoring of video over a packet network. *IEEE transactions on multimedia*, 6(2):327–334, 2004.

[213] Rafael Moreno-Vozmediano, Rubén S Montero, Eduardo Huedo, and Ignacio M Llorente. Efficient resource provisioning for elastic cloud services based on machine learning techniques. *Journal of Cloud Computing*, 8(1):1–18, 2019.

[214] Sebastian Orlowski, Roland Wessäly, Michal Pióro, and Artur Tomaszewski. Sndlib 1.0—survivable network design library. *Networks: An International Journal*, 55(3):276–286, 2010.

[215] Mahmoud Abbasi, Amin Shahraki, and Amir Taherkordi. Deep learning for network traffic monitoring and analysis (ntma): a survey. *Computer Communications*, 170:19–41, 2021.

[216] James F. Kurose and Keith W. Ross. *Computer Networking: A Top-Down Approach, 6th Edition*. Publisher: Pearson, pp. 755-786, 2012.

[217] Chavoosh Ghasemi, Hamed Yousefi, and Beichuan Zhang. Internet-scale video streaming over ndn. *IEEE Network*, 35(5):174–180, 2021.

[218] rfc4689. https://datatracker.ietf.org/doc/html/rfc4689.

[219] Jim Gettys and Kathleen Nichols. Bufferbloat: dark buffers in the internet. *Communications of the ACM*, 55(1):57–65, 2012.

[220] Songyang Zhang, Weimin Lei, Wei Zhang, and Yunchong Guan. Congestion control for rtp media: A comparison on simulated environment. In *International Conference on Simulation Tools and Techniques*, pages 43–52. Springer, 2019.

[221] Anfu Zhou, Huanhuan Zhang, Guangyuan Su, Leilei Wu, Ruoxuan Ma, Zhen Meng, Xinyu Zhang, Xiufeng Xie, Huadong Ma, and Xiaojiang Chen. Learning to Coordinate Video Codec with Transport Protocol for Mobile Video Telephony. In *ICMCN*, pages 1–16, 2019.

[222] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, pages 1–6, 2010.

[223] Ostinato traffic generator for network engineers. https://ostinato.org/.

[224] Ruairí de Fréin, Obinna Izima, and Ali Malik. Detecting network state in the presence of varying levels of congestion. In *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6. IEEE, 2021.

[225] Colin Perkins. *RTP: Audio and Video for the Internet*. Addison-Wesley Professional, 2003.

[226] Ali Malik and Ruairí de Fréin. A proactive-restoration technique for sdns. In *ISCC*, pages 1–6. IEEE, 2020.

[227] Ali Malik and Ruairí de Fréin. Sla-aware routing strategy for multi-tenant software-defined networks. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7. IEEE, 2020.

[228] Stefano Avallone, S Guadagno, Donato Emma, Antonio Pescapè, and Giorgio Ventre. D-itg distributed internet traffic generator. In *First International Conference on the Quantitative Evaluation of Systems, 2004. QEST 2004. Proceedings.*, pages 316–317. IEEE, 2004.

[229] POX Controller Network Software Platform. https://github.com/noxrepo/pox. (Accessed on 12/22/2021).

[230] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM computer communication review*, 38(2):69–74, 2008.

[231] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[232] Lawrence R Rabiner and Bernard Gold. Theory and application of digital signal processing. *Englewood Cliffs: Prentice-Hall*, 1975.

[233] Li Hui, Bei-qian Dai, and Lu Wei. A pitch detection algorithm based on amdf and acf. In *2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings*, volume 1, pages I–I. IEEE, 2006.

[234] FR Johnston, JE Boyland, Maureen Meadows, and E Shale. Some properties of a simple moving average when applied to forecasting a time series. *Journal of the Operational Research Society*, 50(12):1267–1271, 1999.

[235] J Stuart Hunter. The exponentially weighted moving average. *Journal of quality technology*, 18(4):203–210, 1986.

[236] D Graupe, DJ Krause, and J Moore. Identification of autoregressive moving-average parameters of time series. *IEEE Transactions on Automatic Control*, 20(1):104–107, 1975.

[237] 14.1 - autoregressive models | stat 501. https://online.stat.psu.edu/stat501/lesson/14/14.1.

[238] 8.1 stationarity and differencing | forecasting: Principles and practice (2nd ed). https://otexts.com/fpp2/stationarity.html. (Accessed on 03/15/2022).

[239] 14.5.1 - arima models | stat 501. https://online.stat.psu.edu/stat501/lesson/14/14.5/14.5.1. (Accessed on 03/15/2022).

[240] pmdarima · pypi. https://pypi.org/project/pmdarima/.

[241] Chapter 48 time series modeling with arima in r | community contributions for edav fall 2019. https://jtr13.github.io/cc19/time-series-modeling-with-arima-in-r.html#acfpacf. (Accessed on 03/15/2022).

[242] Duke University Statistical Forecasting. Identifying the orders of AR and MA terms in an ARIMA model. https://people.duke.edu/~rnau/411arim3.htm.

[243] L. Yang, Y. Dong, W. Tian, and Z. Wang. The Study of New Features for Video Traffic Classification. *Multimedia Tools and Applications*, 78(12):15839–15859, 2019.

[244] Dipali Shah, Purvang Dalal, Mohanchur Sarkar, and Sejal Dalal. Evaluation of rtt-based techniques for network state classification. In *Proceedings of the International Conference on Intelligent Systems and Signal Processing*, pages 79–91. Springer, 2018.

[245] Mohanchur Sarkar, KK Shukla, and KS Dasgupta. Network state classification based on the statistical properties of rtt for an adaptive multi state proactive transport protocol for satellite based networks. *International Journal of Computer Networks and Communication (IJCNC)*, 2(6):155–174, 2010.

[246] Purvang Dalal, Mohanchur Sarkar, Kankar Dasgupta, Nikhil Kothari, et al. Adaptive tcp: a sender side mechanism with dynamic adjustment of congestion control parameters for performance improvement in wlan. *International Journal of Communications, Network and System Sciences*, 8(05):130, 2015.

[247] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning*. Springer, pp. 311-323, 2013.

[248] L. Breiman. Random Forests, Machine Learning. *Springer*, 45:5–32, 2001.

[249] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[250] J. But, U. Keller, D. Kennedy, and G. Armitage. Passive TCP Stream Estimation of RTT and Jitter Parameters. In *IEEE LCN*, pages 8 pp.–441, 2005.

[251] A. Mukhopadhyay, T. Chakraborty, S. Bhunia, I. Misra, and S. Sanyal. An Adaptive Jitter Buffer Playout Algorithm for Enhanced VoIP Performance. In *ICACIT*, pages 219–230. Springer, 2011.

[252] Obinna Izima, Ruairí de Fréin, and Mark Davis. Video quality prediction under time-varying loads. In *2018 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 129–132, 2018.

[253] Obinna Izima, Ruairí de Fréin, and Ali Malik. A survey of machine learning techniques for video quality prediction from quality of delivery metrics. *Electronics*, 10(22):2851, 2021.

[254] Nabajeet Barman, Emmanuel Jammeh, Seyed Ali Ghorashi, and Maria G Martini. No-reference Video Quality Estimation Based on Machine Learning for Passive Gaming Video Streaming Applications. *IEEE Access*, 7:74511–74527, 2019.

[255] Georgios Kougioumtzidis, Vladimir Poulkov, Zaharias D Zaharis, and Pavlos I Lazaridis. A Survey on Multimedia Services QoE Assessment and Machine Learning-Based Prediction. *IEEE Access*, 10:19507–19538, 2022.

[256] Obinna Izima, Ruairí de Fréin, and Mark Davis. Predicting quality of delivery metrics for adaptive video codec sessions. In *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*, pages 1–7. IEEE, 2020.

[257] Robert Munro Monarch. *Human-in-the-Loop Machine Learning: Active learning and annotation for human-centered AI*. Simon and Schuster, 2021.

[258] Data engineering, preparation, and labeling for ai 2019 - cognilytica. https://www.cognilytica.com/document/report-data-engineering-preparation-and-labeling-for-ai-2019/. (Accessed on 02/07/2022).

[259] Nir Kshetri. Data Labeling for the Artificial Intelligence Industry: Economic Impacts in Developing Countries. *IEEE IT Professional*, 23(2):96–99, 2021.

[260] Data-labelling Startups Want to Help Improve Corporate AI. https://tinyurl.com/yc52b2xd. (Accessed on 06/24/2022).

[261] Avoiding Garbage in Machine Learning. https://tinyurl.com/437puu6t. (Accessed on 06/24/2022).

[262] Robert B Cooper. Queueing theory. In *Proceedings of the ACM'81 conference*, pages 119–122, 1981.

[263] Thomas G Robertazzi. *Computer networks and systems: queueing theory and performance evaluation*. Springer Science & Business Media, 2000.

[264] Jordan Ansell, Winston KG Seah, Bryan Ng, and Stuart Marshall. Making queueing theory more palatable to sdn/openflow-based network practitioners. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pages 1119–1124. IEEE, 2016.

[265] Sanjit K Kaul and Roy D Yates. Timely Updates by Multiple Sources: The M/M/1 Queue Revisited. In *IEEE CISS*, pages 1–6, 2020.

[266] Obinna Izima, Ruairí de Fréin, and Ali Malik. Codec-aware video delivery over sdns. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 732–733. IEEE, 2021.

[267] Ryu sdn framework. https://ryu-sdn.org/. (Accessed on 02/10/2022).

[268] Wolfgang Fischer and Kathleen Meier-Hellstern. The markov-modulated poisson process (mmpp) cookbook. *Performance evaluation*, 18(2):149–171, 1993.

[269] Leonard Kleinrock. Queueing systems, volume 2: Computer applications. johnwiley and sons, 1976.

[270] Adrian Popescu and Doru Constantinescu. On kleinrock's independence assumption. In *Network Performance Engineering*, pages 1–13. Springer, 2011.

[271] John DC Little and Stephen C Graves. Little's law. In *Building intuition*, pages 81–100. Springer, 2008.

[272] Angela Orebaugh, Gilbert Ramirez, and Jay Beale. *Wireshark & Ethereal network protocol analyzer toolkit*. Elsevier, 2006.

[273] Aristidis Likas, Nikos Vlassis, and Jakob J Verbeek. The global k-means clustering algorithm. *Pattern recognition*, 36(2):451–461, 2003.

[274] Computer Networks—Wireshark Labs. https://tinyurl.com/3kzd6jch. (Accessed on 05/27/2022).

[275] Rogério Leão Santos De Oliveira, Christiane Marie Schweitzer, Ailton Akira Shinoda, and Ligia Rodrigues Prete. Using mininet for emulation and prototyping software-defined networks. In *2014 IEEE Colombian conference on communications and computing (COLCOM)*, pages 1–6. Ieee, 2014.

[276] Henning Schulzrinne, Steven Casner, R Frederick, and Van Jacobson. Rfc3550: Rtp: A transport protocol for real-time applications, 2003.

[277] Ketan Rajshekhar Shahapure and Charles Nicholas. Cluster Quality Analysis Using Silhouette Score. In *IEEE DSAA*, pages 747–748, 2020.

[278] L. Hui, B. Dai, and L. Wei. A Pitch Detection Algorithm Based on AMDF and ACF. In *IEEE ICASSP*, volume 1, 2006.

[279] Tianqi Chen and Carlos Guestrin. Xgboost: Reliable large-scale tree boosting system. In *Proceedings of the 22nd SIGKDD Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA*, pages 13–17, 2015.

[280] HANA Řezanková. Different Approaches to the Silhouette Coefficient Calculation in Cluster Evaluation. In *Intl. Conf. AMSE*, pages 1–10, 2018.

[281] V Divya and K Nirmala Devi. An Efficient Approach to Determine Number of Clusters Using Principal Component Analysis. In *IEEE ICCTCT*, pages 1–6, 2018.

[282] Junwei Xiao, Jianfeng Lu, and Xiangyu Li. Davies Bouldin Index Based Hierarchical Initialization K-Means. *Intelligent Data Analysis*, 21(6):1327–1338, 2017.

[283] Akhilesh Kumar Singh, Shantanu Mittal, Prashant Malhotra, and Yash Vardhan Srivastava. Clustering Evaluation by Davies-Bouldin Index (DBI) in Cereal Data using K-Means. In *IEEE ICCMC*, pages 306–310, 2020.

[284] Margherita Grandini, Enrico Bagli, and Giorgio Visani. Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756*, 2020.