Learning for Network Applications and Control

Craig Gutterman

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
under the Executive Committee
of the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2021

Abstract

Learning for Network Applications and Control

Craig Gutterman

The emergence of new Internet applications and technologies have resulted in an increased complexity as well as a need for lower latency, higher bandwidth, and increased reliability. This ultimately results in an increased complexity of network operation and management. Manual management is not sufficient to meet these new requirements.

There is a need for data driven techniques to advance from manual management to autonomous management of network systems. One such technique, Machine Learning (ML), can use data to create models from hidden patterns in the data and make autonomous modifications. This approach has shown significant improvements in other domains (e.g., image recognition and natural language processing). The use of ML, along with advances in programmable control of Software-Defined Networks (SDNs), will alleviate manual network intervention and ultimately aid in autonomous network operations. However, realizing a data driven system that can not only understand what is happening in the network but also operate autonomously requires advances in the networking domain, as well as in ML algorithms.

In this thesis, we focus on developing ML-based network architectures and data driven networking algorithms whose objective is to improve the performance and management of future networks and network applications. We focus on problems spanning across the network protocol stack from the application layer to the physical layer. We design algorithms and architectures that are motivated by measurements and observations in real world or experimental testbeds.

In Part I we focus on the challenge of monitoring and estimating user video quality of experience (QoE) of encrypted video traffic for network operators. We develop a system for **RE**al-time **QU**ality of experience metric detection for **E**ncrypted **T**raffic, **Requet**. *Requet* uses a detection algorithm to identify video and audio chunks from the IP headers of encrypted traffic. Features extracted from the chunk statistics are used as input to a random forest ML model to predict QoE

metrics. We evaluate *Requet* on a YouTube dataset we collected, consisting of diverse video assets delivered over various WiFi and LTE network conditions. We then extend *Requet*, and present a study on YouTube TV live streaming traffic behavior over WiFi and cellular networks covering a 9-month period. We observed pipelined chunk requests, a reduced buffer capacity, and a more stable chunk duration across various video resolutions compared to prior studies of on-demand streaming services. We develop a YouTube TV analysis tool using chunks statistics detected from the extracted data as input to a ML model to infer user QoE metrics.

In Part II we consider allocating end-to-end resources in cellular networks. Future cellular networks will utilize SDN and Network Function Virtualization (NFV) to offer increased flexibility for network infrastructure operators to utilize network resources. Combining these technologies with real-time network load prediction will enable efficient use of network resources. Specifically, we leverage a type of recurrent neural network, Long Short-Term Memory (LSTM) neural networks, for (i) service specific traffic load prediction for network slicing, and (ii) Baseband Unit (BBU) pool traffic load prediction in a 5G cloud Radio Access Network (RAN). We show that leveraging a system with better accuracy to predict service requirements results in a reduction of operation costs.

We focus on addressing the optical physical layer in Part III. Greater network flexibility through SDN and the growth of high bandwidth services are motivating faster service provisioning and capacity management in the optical layer. These functionalities require increased capacity along with rapid reconfiguration of network resources. Recent advances in optical hardware can enable a dramatic reduction in wavelength provisioning times in optical circuit switched networks. To support such operations, it is imperative to reconfigure the network without causing a drop in service quality to existing users. Therefore, we present a ML system that uses feedforward neural networks to predict the dynamic response of an optically circuit-switched 90-channel multi-hop Reconfigurable Optical Add-Drop Multiplexer (ROADM) network. We show that the trained deep neural network can recommend wavelength assignments for wavelength switching with minimal power excursions. We extend the performance of the ML system by implementing and testing

a Hybrid Machine Learning (HML) model, which combines an analytical model with a neural network machine learning model to achieve higher prediction accuracy.

In Part IV, we use a data-driven approach to address the challenge of wireless content delivery in crowded areas. We present the Adaptive Multicast Services (*AMuSe*) system, whose objective is to enable scalable and adaptive WiFi multicast. Specifically, we develop an algorithm for dynamic selection of a subset of the multicast receivers as feedback nodes. Further, we describe the Multicast Dynamic Rate Adaptation (*MuDRA*) algorithm that utilizes AMuSe's feedback to optimally tune the physical layer multicast rate. Our experimental evaluation of MuDRA on the ORBIT testbed shows that MuDRA outperforms other schemes and supports high throughput multicast flows to hundreds of nodes while meeting quality requirements. We leverage the lessons learned from AMuSe for WiFi and use order statistics to address the performance issues with LTE evolved Multimedia Broadcast/Multicast Service (eMBMS). We present the Dynamic Monitoring (DyMo) system which provides low-overhead and real-time feedback about eMBMS performance to be used for network optimization. We focus on the Quality of Service (QoS) Evaluation module and develop a *Two-step estimation* algorithm which can efficiently identify the SNR Threshold as a one time estimation. *DyMo* significantly outperforms alternative schemes based on the *Order-Statistics estimation* method which relies on random or periodic sampling.

Contents

List of Tables

List of Figures

Acknowledgements

I am writing this acknowledgment to recognize all those people who supported me during my Ph.D. First and foremost, I would like to thank my Ph.D. advisor, Prof. Gil Zussman. His guidance, motivation, and high standards were instrumental in shaping me as a researcher. I will be forever grateful for his mentorship. Additionally, I would like to express my appreciation to my entire thesis committee consisting of Prof. Dabisis Mitra, Prof. Ethan Katz-Bassett, Prof. Dan Kilper, and Dr. Katherine Guo.

I would like to extend a special thanks to Dr. Varun Gupta. For the first few years of my journey, Varun was a de-facto mentor to me on the AMUSE project. I also want to take this opportunity to express a deep sense of gratitude to Dr. Katherine Guo who has been a close collaborator and mentor throughout my time at Columbia.

In addition, a thank you to my fellow WiMNet research group members. I express my deepest thanks to Tingjun Chen who I began my Ph.D journey with and discussed research with throughout the process. I am thankful to work alongside Maria Gorlatova, Berk Birand, Robert Margolies, Jelena Marasevic, Saleh Soltan, Manav Kohli, Mahshid Ghasemi, Guy Grebla, and Jonathan Ostometzky. I would also like to thank other my other office mates Todd Arnold, Kunal Mahajan, and Niloofar Bayat.

My thesis would not have been possible without much successful collaborations. For my work on the *Requet* project, I had the opportunity to work with Dr. Katherine Guo, Sarthak Arora, Trey Gilliland, Dr. Xiaoyang Wang, Dr. Les Wu, and Prof. Ethan Katz-Bassett. I would like to thank Dr. Edward Grinshpun and Dr. Sameer Sharma for their collaboration at Bell Labs. For all my research related to optical networks I would like to express my deepest appreciation to Prof. Dan Kilper for his mentorship and guidance. In addition, I had the honor of collaborating with Ph.D students and Postdoctoral research scientists Artur Minakhmetov, Jiakai Yu, Weiyang Mo, Yao Li, Shengxiang Zhu, Yishen Huang, and Payman Samadi on projects related to optical networks. Throughout my work on the AMuSe project I had the opportunity to collaborate with Jaime Ferragut, Andy Xu, Bohan Wu, Hannaneh Pasandi, and Rodda John. I would like to extend

a special thanks to Dr. Yigal Bejerano for his guidance. I would also like to thank my collaborators on COSMOS including Ivan Seskar and Michael Sherman. I would also like to thank all the undergraduate and masters' students that I had the pleasure of working with.

Finally, and above all, I am grateful to my wonderful family for all of their unconditional love and support.

Financial Support: The research described in this thesis was supported in part by NSF grants: Graduate Research Fellowship Program (GRFP) DGE-1644869, NSF Integrative Graduate Education and Research Traineeship (IGERT) Fellowship From Data to Solutions, CNS-1910757, CNS-1827923, CNS-1650685, CNS-1650669, PFI-1601784, CNS-1423105, CNS-1054856, NSF CIAN ERC under grant EEC-0812072, and by the New York City Media Lab Combine program grant.

Dedication

To all my family and friends,

and specifically to my amazing parents, wonderful sister, and phenomenal grandparents.

None of this would have been possible without your love and support.

Chapter 1: Introduction

1.1 Learning for Networking

The emergence of new applications, including virtual reality, augmented reality, Internet of Things (IoT), connected cars, smart cities, and edge computing, requires increased reliability, lower latency, and higher bandwidth from wired and wireless networks. This ultimately results in an increased complexity of network operation and management. Therefore, manual management is not sufficient to meet these new requirements.

There is a need for data driven techniques to advance from manual management to autonomous management of network systems. One such technique, Machine Learning (ML), can use data to create models from hidden patterns in the data and make autonomous modifications to the network. This approach has shown significant improvements in other domains (e.g., image recognition and natural language processing) [38, 29, 83, 104]. The use of ML, along with advances in programmable control of Software-Defined Networks (SDNs), will alleviate manual network intervention and ultimately aid in autonomous network operations. Realizing a data driven system that can not only understand what is happening in the network but also operate autonomously requires advances in the networking domain, as well as in ML and Artificial Intelligence (AI) algorithms.

In this thesis, we focus on developing ML-based network architectures and data driven network algorithms whose objective is to improve the performance and management of future networks. We focus on problems spanning across the networking protocol stack from the application layer to the physical layer. Specifically, we focus on the networking domains of analyzing encrypted network traffic for video quality of experience (QoE), allocating end-to-end resources in cellular networks, enabling dynamic optical networks, and providing reliable feedback for wireless multicast.

While these domains span across the whole stack and across optical, IP, and wireless domains,

there are many commonalities in system and algorithm development. As a result, in each of these domains we explore the following: (i) Understanding the networking sub-problem, (ii) Collecting real data or simulated data, (iii) Developing an appropriate ML system, and (iv) Evaluating the proposed solution via simulations and experimentation.

The rest of this chapter is organized as follows. We first provide the background on each domain in Section 1.2. In Section 1.3 we briefly discuss the ML or data driven contribution to each domain. We conclude in Section 1.4 with an overview of the contributions to the literature.

1.2 Networking Domains

As mentioned above, we develop ML techniques for networking problems related to video streaming, end-to-end network resource allocation in cellular networks, dynamic optical systems, and adaptive wireless multicast. In this section, we provide background for each of these domains.

1.2.1 Video Streaming

The first application of ML is for real-time encrypted traffic classification. Specifically, we focus on the application of identifying video streams and user QoE. Video streaming dominates application traffic over the Internet. This has been largely attributed to the proliferation of on-demand video streaming services, with live streaming poised to take off. By 2021, video streaming traffic is expected to grow to 82% of all IP traffic [41], mobile video will represent 78% of all mobile traffic [208], while live video usage will account for 13% of global Internet traffic [30].

To ensure video streaming consumer satisfaction and reduce customer churn rate, it is essential for Internet Service Providers (ISPs) or network operators to provision and manage network resources such that streaming video end users with limited available resources maintain high QoE. A critical component of QoE-aware network management is a monitoring tool residing within the network whose objective is to infer metrics related to end user QoE.

Client-side measurement applications can accurately report QoE metrics such as player events and video quality levels [214, 144]. Traditionally, Deep Packet Inspection (DPI) enabled operators

to examine HTTP packet flows and extract video session information to infer QoE metrics [10, 32]. However, to address security and privacy concerns, content providers are increasingly adopting end-to-end encryption. A majority of YouTube traffic has been encrypted since 2016 [91] with a combination of HTTPS (HTTP/TLS/TCP) [14, 168, 56] and QUIC (HTTP/QUIC/UDP) [101, 49]. Similarly, since 2015, Netflix has been deploying HTTPS for video traffic [16].

Although the trend of end-to-end encryption does not affect client-side or server-side QoE monitoring, it renders traditional DPI-based video QoE monitoring ineffective for operators. Despite encryption occluding session information, patterns from IP headers that are still in plain text can be utilized and applied to infer video QoE metrics.

1.2.2 End-to-End Resource Allocation in Cellular Networks

The second application of ML is for end-to-end resource allocation in cellular networks. Traditional cellular networks rely on Radio-Access Networks (RANs) in which baseband signal processing is carried out at the location of the cellular antennas. As traffic has increased, cell sizes are decreasing, dramatically increasing the number of cell sites and their capacity. In order to improve the scalability of these large numbers of access points, separation of the Remote Radio Heads (RRHs) and the Base Band Processing Units (BBUs) has been proposed for 5G networks. By utilizing SDN and Network Function Virtualization (NFV), wireless RAN virtualization will offer greater flexibility for network infrastructure operators, while also adding benefits to their customers (typically called tenants [172]). By enabling RAN virtualization, Mobile Network Operators can share common RAN resources leading to reduced costs and increased energy efficiency. The concept of network virtualization will enable infrastructure as a service for end-to-end networking [51].

Another key technology to improve the performance and management of future networks is network slicing. The Next Generation Mobile Network (NGMN) Alliance defines a network slice as a set of network functions and associated resources, forming a complete virtualized end-to-end logical network meeting certain network characteristics required by the associated service [51, 2,

110]. Namely, by utilizing technologies such as SDN and NFV, slices will provide virtualized resource separation for different services, while still allowing for statistical multiplexing of network resources.

1.2.3 Dynamic Optical Systems

In recent years, the advent of network applications such as video streaming, Internet-of-Things (IoT), and cloud computing have contributed to exponential Internet traffic growth [198]. As traffic increases, the ability to manage capacity and provision new services in the optical layer in real-time becomes important. Provisioning times in commercial systems takes minutes per wavelength, and in practice, days of careful offline wavelength planning are used to mitigate unpredictable dynamic effects [96, 148]. To allow for such an increase in dynamic network traffic, optical metro and long haul networks need to be dynamic and need to utilize network resources efficiently [67, 55].

The service provider provisions optical paths for its customers, such that, in Dense Wavelength Division Multiplexing (DWDM), up to 96 wavelengths (or unique optical signals) are transmitted in a single fiber. These signals are added and dropped from the network at Reconfigurable Optical Add-Drop Multiplexer (ROADM) nodes which connect to Layer 2/3 switches. Optical switching techniques have been investigated for real time adaptation of optical layer capacity based on changing traffic conditions. A key challenge is the optical power dynamics which arise and grow in cascade in a ROADM system in the presence of optical circuit switching [184, 107, 98]. However, today's commercial ROADM systems remain 'quasi-static', with wavelengths being provisioned to meet the peak traffic requirements [202].

A key unresolved challenge to achieving dynamic ROADM systems through SDN is predicting and controlling the optical power dynamics resulting from wavelength switching operations. Power excursions can result from the interactions between the wavelength dependent gain and Automatic Gain Control (AGC) of optical amplifiers, Raman scattering in the fiber, and other wavelength dependent phenomena. AGC is commonly used in Erbium-Doped Fiber Amplifiers (EDFAs) to maintain constant average gain for varying input conditions. EDFAs are used to boost the optical

signal being transported between two end nodes. This results in power excursions, one of the main types of power dynamics in optical networks. Due to the EDFA's wavelength dependent gain, with changing input conditions, power excursions occur on active channels, and grow over multiple EDFAs along the propagation path. The AGC attempts to amplify the channel to obtain a target gain value, but adding or removing a channel can cause deviations that perturb other channels resulting in excursions. These gains cannot be corrected until slow per-channel power controls in the optical nodes are able to re-adjust the power level, which requires repeated measurements and adjustments.

Deviations of the channel power levels outside pre-allocated system margins can potentially result in service disruption due to reduced quality of transmission (QoT) [107]. For this reason, today's commercial systems take minutes and even hours to provision a wavelength through time-consuming power adjustments along an optical path [148].

1.2.4 Adaptive Wireless Multicast

The fourth domain explored is in the area of wireless networks. As video traffic continues to increase, the delivery of such traffic becomes a bottleneck for wireless networks. One solution to this problem is based on dense deployments of WiFi Access Points (APs) or cellular Base Stations (BSs), which require considerable capital and operational expenditure, may suffer from interference between APs, and may exacerbate hidden node problems [159, 80]. Wireless multicast is an attractive approach for content delivery to large groups of users interested in venue specific content.

Video delivery is an essential service for wireless networks and several solutions were proposed for crowded venues [195, 42, 231]. *Multicast* offers another approach for video delivery to large groups of users interested in venue specific content (e.g., sports arenas, entertainment centers, and lecture halls). However, WiFi networks provide *limited multicast support at a low rate (e.g.,* 1*Mbps for 802.11b/g/n,* 6*Mbps for 802.11a/n) without a feedback mechanism that guarantees service quality*. Similarly, the evolved Multicast and Broadcast Services (eMBMS) standard [4] for LTE

networks does not specify a mechanism for collecting real-time feedback from receivers which is important for tuning parameters such as transmission rates and error correction.

However, in crowded venues with tens of thousands of User Equipments (UEs) (e.g., [58]), even infrequent Quality of Service (QoS) reports by each UE may result in high signaling overhead and blocking of unicast traffic over a cellular network. Therefore, for WiFi, even if feedback is not collected continuously, a swarm of retransmission requests may be sent following an interference event thereby causing additional interruptions. Implementing a solution to this problem requires extensive understanding of packet delivery based on experimental results. In addition, a multicast system should conduct efficient rate adaptation based on only limited reports from the nodes. Therefore, there is a need to design data driven techniques for efficient feedback collection mechanisms and dynamic multicast rate adaptation mechanisms for WiFi and cellular multicast.

1.3 Contributions

In this section, we briefly describe the contributions made in the different chapters of this thesis. These include ML and data driven contributions made to the networking domains of analyzing encrypted network traffic for video QoE, allocating end-to-end network resources in cellular networks, dynamic optical networks, and wireless multicast.

1.3.1 Video Streaming

Part I focuses on systems for QoE metric detection for encrypted traffic. We design a system that works in real time and is applicable to multiple video streaming services for both WiFi and cellular networks. In Chapter 2 we develop and present a system, *Requet* for **RE**al-time **QU**ality of experience metric detection for **E**ncrypted **T**raffic, **Requet**, that is suitable for network middlebox deployment. *Requet* uses a detection algorithm we develop to identify video and audio chunks from the IP headers of encrypted traffic. Features extracted from the chunk statistics are used as input to a *random forest* ML model to predict QoE metrics, specifically, *buffer warning* (low buffer, high buffer), *video state* (buffer increase, buffer decay, steady, stall), and *video resolution*.

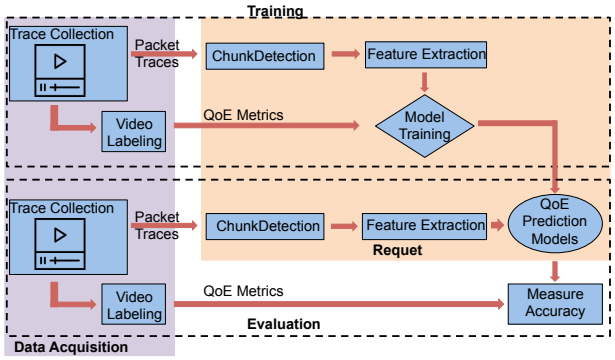


Figure 1.1: System Diagram: Data acquisition and *Requet* components: ChunkDetection, feature extraction, and QoE prediction models.

One of the main challenges is collecting enough data to accurately understand the underlying networking system and then developing a classification algorithm. We collect a large YouTube dataset consisting of diverse video assets delivered over various WiFi and LTE network conditions to evaluate the performance.

Fig. 1.1 depicts the system diagram for *Requet* and necessary components to train the QoE models as well as evaluate its performance. *Requet* consists of the ChunkDetection algorithm, chunk feature extraction, and ML QoE prediction models. The data acquisition process involves collecting YouTube traffic traces (*Trace Collection*) and generating ground truth QoE metrics as labels directly from the player (*Video Labeling*). Packet traces are fed into *Requet*'s ChunkDetection algorithm to determine audio and video chunks. The chunks are then used during the *Feature Extraction* process to obtain chunk-based features. The chunk-based features from the training data along with the corresponding QoE metrics are used to generate QoE prediction models. For evaluation, traffic traces from the testing dataset are fed into the trained QoE models to generate predicted QoE metrics. Accuracy is measured by comparing the predicted QoE metrics and the ground truth labels.

We compare *Requet* with a baseline system based on previous work and show that *Requet* outperforms the baseline system in accuracy of predicting buffer low warning, video state, and video resolution by 1.12×, 1.53×, and 3.14×, respectively.

In Chapter 3, we extend the work from Chapter 2 and present a study on YouTube TV live streaming traffic behavior over WiFi and cellular networks. YouTube TV is a television service

that offers live TV for more than 70 US television networks and is gaining popularity [234]. Our data collection spans a 9-month period leading up to Feb. 2020. We collect ground truth playback quality metrics directly from the YouTube TV application on an Android mobile device over WiFi networks and LTE cellular networks. Using the collected data, we develop a multi-chunk detection (MCD) algorithm to detect multiple video and audio chunks with concurrent transmission *in the same IP flow*. We define a multi-chunk unit (MCU) as a group of overlapping chunks. We design *random forest* ML models to infer user QoE metrics such as video resolution and buffer health from features based on MCUs extracted from IP traffic.

1.3.2 End-to-End Resource Allocation in Cellular Networks

Part II focuses on using recurrent neural networks, a type of deep neural network, to improve resource allocation in cellular networks. In Chapter 4 we focus on supporting the operation of the RAN slice broker, which maps slice requirements into the allocation of Physical Resource Blocks (PRBs). We develop a new metric, *REVA*, based on the number of PRBs available to a single Very Active bearer. *REVA* is independent of channel conditions and allows easy derivation of an individual wireless link's throughput. In order for the slice broker to efficiently utilize the RAN, there is a need for reliable and short term prediction of resource usage by a slice.

To support such prediction, we construct an LTE testbed and develop custom additions to the scheduler. Using data collected from the testbed, we compute REVA and develop a realistic time series prediction model for REVA. Specifically, we present the X-LSTM prediction model, based upon Long Short-Term Memory (LSTM) neural networks. Evaluated with data collected in the testbed, X-LSTM outperforms Autoregressive Integrated Moving Average Model (ARIMA) and LSTM neural networks by up to 31%. X-LSTM also achieves over 91% accuracy in predicting REVA. By using X-LSTM to predict future usage, a slice broker is more adept at provisioning a slice and reducing over-provisioning and Service Level Agreement (SLA) violation costs by more than 10% in comparison to LSTM and ARIMA.

In Chapter 5 we present an LSTM neural network used for BBU pool resource reallocations in

a 5G C-RAN network using a ROADM switched optical network. Traffic aggregated at different BBU pools has been shown to be comprised of different types of traffic patterns each with a temporal time component. Making accurate predictions would allow for resource reallocation before the actual demand is needed, and therefore gives enough time for optical network reconfiguration to route the traffic through the C-RAN to a BBU pool with available computing resources.

We design a discrete event simulation of traffic aggregated at different BBU pools in a 5G C-RAN ROADM optical network. We develop an accurate prediction model using an LSTM neural network to predict future network resource requirements. The LSTM network is trained with 740 samples and achieves a 7% increase in network throughput and an 18% processing resource reduction by using the predicted traffic pattern to reconfigure the ROADM optical network 30 minutes in advance.

1.3.3 Dynamic Optical Systems

Part III focuses on using ML to ensure stable performance and reliable Quality of Transmission (QoT) for dynamic optical operation. In Chapter 6 we present a *feedforward deep neural network* based-ML model to predict the power dynamics of a 90-channel ROADM system. A metro-scale multi-hop ROADM system is built to study wavelength switching using the proposed ML approach. The deep neural network is able to learn the complex optical power excursion response with 67,200 training samples and obtains a 0.1 dB RMSE for 8,400 test samples. Based on the predicted power excursions, the deep neural network can recommend valid wavelengths for wavelength switching with a precision of more than 99% over the tested samples. The deep neural network was also shown to be far more effective than regression and random forest models.

In Chapter 7 we examine an ML model, analytical model, and a hybrid ML model. We initially compare the performance of an analytical model [98] and *feedforward deep neural network* based-ML model to predict amplifier gain in an EDFA. The input to each model is the power levels of each of the 90 channel source used to generate the WDM input. The ML based model is shown to reduce the gain estimation error compared with analytical models.

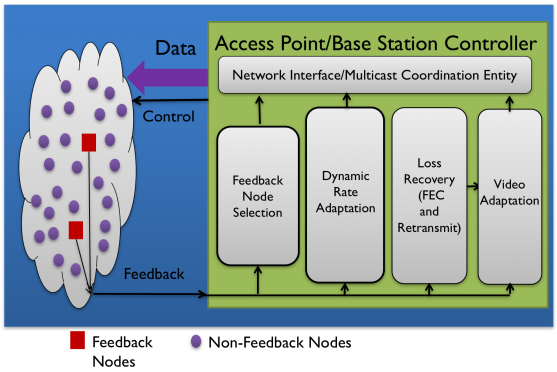


Figure 1.2: A block diagram of the contributions to adaptive wireless multicast for both WiFi and cellular networks: a light-weight feedback mechanism, multicast dynamic rate adaptation, loss recovery and Forward Error Correction (FEC), and video rate adaptation.

We then take advantage of an analytical model and use it as the input to a *feedforward deep neural network*. Based on experimental measurements, compared to the previous ML model, this hybrid ML model is shown to increase prediction accuracy of the output optical power spectrum of an EDFA by 10.5%, reduce the training sample size by 33% and reduce the training time by 37%.

1.3.4 Adaptive Wireless Multicast

Part IV focuses on the use of data-driven solutions for large scale content delivery via wireless multicast, both for WiFi and cellular networks. We address the research challenges associated with several aspects of wireless multicast. For WiFi multicast, we address challenges related to feedback and rate adaptation as part of the Adaptive Multicast Services (*AMuSe*) system [22]. For LTE, our focus is on efficient large-scale monitoring using light-weight feedback.

In Chapter 8 we study approaches for light-weight feedback for WiFi multicast. We conducted extensive experiments with over 200 WiFi nodes on the ORBIT testbed in order to better understand the performance of existing schemes. Our observations show that some nodes suffer from low *Packet Delivery Ratio (PDR)*, even when the AP is transmitting at a low bit-rate and there is no external interference (referred to as *abnormal nodes*). Furthermore, this set of abnormal nodes varies across experiments.

Next, we introduce *AMuSe*, a low-overhead feedback system which leverages the existing WiFi

standards for tuning the network parameters, i.e., optimizing the network utilization while preserving QoS requirements. The *AMuSe* feedback mechanism dynamically divides the network into clusters based on the adjacency of nodes and maximum cluster size. In each cluster, one node is selected as the feedback node that updates the Access Point (AP) about its channel quality.

In Chapter 9, we present the design and evaluation of the Multicast Dynamic Rate Adaptation (*MuDRA*) algorithm for WiFi. Our experimental evaluation on the ORBIT testbed shows that when the multicast rate exceeds an optimal rate, called the *target-rate*, numerous receivers suffer from a low packet delivery ratio and their losses cannot be easily recovered.

We experimentally demonstrate that *MuDRA* can swiftly converge to the target rate while meeting QoS requirements, e.g., ensuring that more than 85% of packets are correctly received by at least 95% of the 200 nodes in our setup. We show that in our experimental settings, *MuDRA* can deliver 3 or 4 high definition H.264 videos (each one of 4Mbps) in which over 90% of the nodes receive video quality classified as excellent or good based on user perception.

In Chapter 10, we describe the Dynamic Monitoring (*DyMo*) system designed to support efficient LTE-eMBMS[1] deployments in crowded and dynamic environments by providing accurate QoS reports with low overhead. *DyMo* identifies the maximal eMBMS SNR Threshold such that only a small number of UEs with SNR below the SNR Threshold may suffer from poor service. *DyMo* leverages eMBMS for broadcasting *stochastic group instructions* to all UEs. This simple feedback mechanism collects very limited QoS reports from the UEs. The reports are used for network optimization, thereby ensuring high QoS to the UEs.

We focus on the QoS Evaluation module and develop a *Two-step estimation* algorithm which can efficiently identify the SNR Threshold as a one time estimation. We also develop an *Iterative estimation* algorithm for estimating the SNR Threshold iteratively, when the distribution changes due to UE mobility or environmental changes, such as network component failures.

We conduct extensive at-scale simulations, based on real eMBMS radio survey measurements from a stadium and an urban area. It is shown that *DyMo* accurately infers the SNR Threshold and

[1]LTE-eMBMS enables broadcast delivery to a large number of UEs in LTE.

optimizes the eMBMS parameters with low overhead under different mobility patterns and even in the event of component failures. *DyMo* significantly outperforms alternative schemes based on the *Order-Statistics estimation* method which relies on random or periodic sampling.

1.4 Contributions to the Literature

The research about video quality of experience of YouTube in Chapter 2 appeared in the proceedings of ACM MMSys'19 [78]. An extended version appeared in ACM Transactions on Multimedia Computing, Communications and Applications (TOMM) [77]. [2]

The research results in Chapter 4 on prediction of RAN resource usage for a 5G slice broker were published in the proceedings of ACM MobiHoc'19 [76]. In addition, the results in Chapter 5 for using deep neural networks to predict BBU pool traffic in C-RAN ROADM networks appeared in the proceedings of OSA OFC'18 [140].

The deep neural network developed in Chapter 6 was published in OSA Journal of Optical Communications and Networking'18 [139]. A shorter version of this work originally appeared in ACM SIGCOMM Big DAMA'17 Workshop [79]. Chapter 7 discusses a hybrid machine learning model to improve the accuracy of power excursion prediction. The research results from this chapter are a combination of results from the proceedings of ECOC'18 [240] and OFC'20 [239]. Additional research contributions have been made to the COSMOS testbed [43] that do not appear in this thesis. These contributions include results that appeared in the proceedings of OSA OFC [136, 236, 235] and ACM MobiCom'20 [167].

The research results relating to efficient feedback collection for WiFi multicast described in Chapter 8, was published in the proceedings of IEEE ICNP'13 [20] and an extended journal version appeared in IEEE/ACM Transactions on Networking [71]. Additionally, the lessons learned from the large scale experimentation on the ORBIT testbed were summarized in an invited paper in the proceedings of GENI Research and Educational Experiment Workshop'14 (GREE) [21]. A demo of the concepts described in this work was presented at IEEE LCN'15 [75].

[2] A demo of this work was presented in the NYC Media Lab Annual Summit'19 and received the top prize in the Enabling Technology category among more than 100 demos.

The design and experimental evaluation of *MuDRA* for dynamic rate adaptation for WiFi multicast in Chapter 9 appeared in the proceedings of IEEE INFOCOM'16 [73]. An extended journal version appeared in IEEE Transactions on Wireless Communications [72]. A demo of the rate adaptation process was presented at and appeared in the proceedings of IEEE INFOCOM'16 [74][3]. The overview of the results spanning the entire *AMuSe* project appeared in the proceedings of IEEE ICCCN'16 [22] as an invited paper.

The description and evaluation of the DyMo system for efficient monitoring of large scale eMBMS deployments as described in Chapter 10 appeared in the proceedings of IEEE INFOCOM'17 [24] and was selected as the **best paper runner-up**. An extended version with additional results was fast-tracked to IEEE/ACM Transactions on Networking [23].

[3]The same demo was presented in the NYC Media Lab Annual Summit'15 and won the second prize among more than 100 demos.

Part I

Video Streaming

Chapter 2: Real-Time QoE Metric Detection for Encrypted YouTube Traffic

This part of the thesis focuses on the development of a system for real-time quality of experience metric detection for encrypted traffic, *Requet*. In this chapter, we collect a large YouTube dataset consisting of diverse video assets delivered over various WiFi and LTE network conditions to evaluate the performance. We compare *Requet* with a baseline system based on previous work and show that *Requet* outperforms the baseline system in accuracy of predicting buffer low warning, video state, and video resolution.

2.1 Introduction

Video has monopolized Internet traffic in recent years. Specifically, the portion of video over mobile data traffic is expected to be 78% by 2021 [208]. Content providers, Content Delivery Networks (CDNs), and network *operators* are all stakeholders in the Internet video sector. They want to monitor user video Quality of Experience (QoE) and improve upon it in order to ensure user engagement. Content providers and CDNs can measure client QoE metrics, such as video resolution by using server-side logs [68, 12]. Client-side measurement applications can accurately report QoE metrics such as player events and video quality levels [214, 144].

Traditionally, Deep Packet Inspection (DPI) enabled operators to examine HTTP packet flows and extract video session information to infer QoE metrics [10, 32]. However, to address security and privacy concerns, content providers are increasingly adopting end-to-end encryption. A majority of YouTube traffic has been encrypted since 2016 [91] with a combination of HTTPS (HTTP/TLS/TCP) [14, 168, 56] and QUIC (HTTP/QUIC/UDP) [101, 49]. Similarly, since 2015 Netflix has been deploying HTTPS for video traffic [16]. In general, the share of encrypted traffic is estimated to be over 80% in 2019 [57].

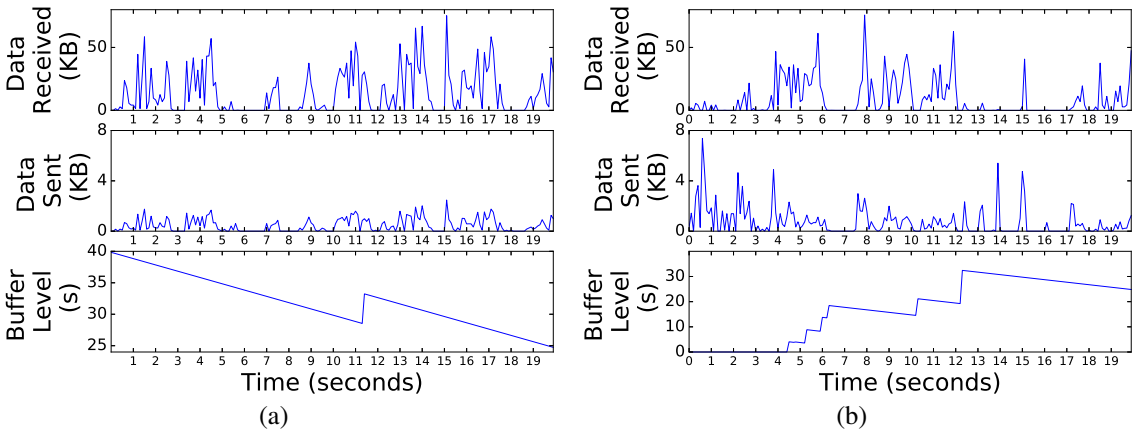


Figure 2.1: Amount of data received (KB), amount of data sent (KB), and buffer level (sec) for two sessions over a 20 sec window (100 ms granularity): (a) 720p, (b) 144p.

Although the trend of end-to-end encryption does not affect client-side or server-side QoE monitoring, it renders traditional DPI-based video QoE monitoring ineffective for operators. Encrypted traffic still allows for viewing packet headers in plain text. This has led to recent efforts to use ML and statistical analysis to derive QoE metrics for operators. These works either provide offline analysis for the entire video session [52, 152] or online analysis using both network and transport layer information with separate models for HTTPS and QUIC [133].

Previous research developed methods to derive network layer features from IP headers by capturing packet behavior in both directions: *uplink* (from the client to the server) and *downlink* (from the server to the client) [133, 109, 152]. However, determining QoE purely based on IP header information is inaccurate. To illustrate, Fig. 2.1 shows a 20 sec portion from two example sessions from our YouTube dataset, described in Section 2.5, where each data point represents 100 ms. Both examples exhibit similar patterns in the downlink direction while in the uplink direction, traffic spikes are much higher in Fig. 2.1(b) than in Fig. 2.1(a). However, Fig. 2.1(a) shows a 720p resolution with the buffer decreasing by 15 secs, whereas Fig. 2.1(b) shows a 144p resolution with the buffer increasing by 20 secs.

Given this challenge, our objective is to design features from IP header information that utilize patterns in the video streaming algorithm. In general, video clips stored on the server are divided into a number of *segments* or *chunks* at multiple resolutions. The client requests each chunk by

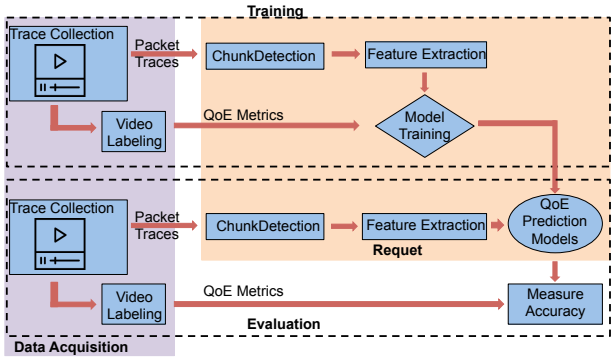


Figure 2.2: System Diagram: Data acquisition and *Requet* components: ChunkDetection, feature extraction, and QoE prediction models.

individually sending an HTTP GET request to the server. Existing work using chunks either infers QoE for the entire session [126] rather than in real-time, or lacks insight on chunk detection mechanisms from network or transport layer data [117, 177, 52].

To improve on existing approaches that use chunks, we develop **Requet**, a system for **RE**al-time **QU**ality of experience metric detection for **E**ncrypted **T**raffic designed for traffic monitoring in middleboxes by operators. *Requet* is devised for real-time QoE metric identification as chunks are delivered rather than at the end of a video session. *Requet* is designed to be memory efficient for middleboxes, where the memory requirement is a key consideration. Fig. 2.2 depicts the system diagram for *Requet* and necessary components to train the QoE models as well as evaluate its performance. *Requet* consists of the ChunkDetection algorithm, chunk feature extraction, and ML QoE prediction models. The data acquisition process involves collecting YouTube traffic traces (*Trace Collection*) and generating ground truth QoE metrics as labels directly from the player (*Video Labeling*).

Packet traces are fed into *Requet*'s ChunkDetection algorithm to determine audio and video chunks. The chunks are then used during the *Feature Extraction* process to obtain chunk-based features. The chunk-based features from the training data along with the corresponding QoE metrics are used to generate QoE prediction models. For evaluation, traffic traces from the testing dataset are fed into the trained QoE models to generate predicted QoE metrics. Accuracy is measured by comparing the predicted QoE metrics and the ground truth labels.

Recent studies have shown that (i) stall events have the largest negative impact on end user

engagement and (ii) higher average video *playback bitrate* improves user engagement [12, 54]. Motivated by these findings, *Requet* aims to predict the current video resolution and events that lead to QoE impairment *ahead of time*. This allows operators to proactively provision resources [44, 161]. *Requet* predicts low buffer level which allows operators to provision network resources to avoid stall events [109]. *Requet* predicts four video states: buffer increase, buffer decay, steady, and stall. Furthermore, *Requet* predicts current video resolution during a video session in real-time. Specifically, *Requet* predicts video resolution on a more granular scale (144p, 240p,360p, 480p, 720p, 1080p), while previous work predicts only two or three levels of video resolution for the entire video session [52, 126, 133].

We make the following contributions:

- Collect packet traces of 60 diverse YouTube video clips resulting in a mixture of HTTP/TLS/TCP and HTTP/QUIC/UDP traffic. The traces are collected in two distinct settings with the first set collected from a laptop web browser over WiFi networks from three service providers, two in the United States and one in India, and the second set collected from the YouTube App on an Android mobile device over LTE cellular networks. This is in contrast to most prior works which rely on simulation or emulation [109, 133, 203] (see Section 2.5).

- Design *Requet* components

  - Develop ChunkDetection, a heuristic algorithm to identify video and audio chunks from IP headers (see Section 2.4).

  - Analyze the correlation between audio and video chunk metrics (e.g., chunk size, duration, and download time) and various QoE metrics, and determine fundamental *chunk-based* features useful for QoE prediction. Specifically, design features based on our observation that audio chunk arrival rate correlates with the video state (see Section 2.6).

  - Develop ML models to predict QoE metrics in real-time: buffer warning, video state, and video resolution (see Section 2.7).

- Evaluate *Requet* performance
  - Demonstrate drastically improved prediction accuracy using chunk-based features versus baseline IP layer features commonly used in prior work [109, 133, 152, 203]. For the setting of a web browser over WiFi networks, *Requet* predicts low buffer warning with 92% accuracy, video state with 84% accuracy, and video resolution with 66% accuracy, representing an improvement of 1.12×, 1.53×, and 3.14×, respectively, over the existing baseline system. Furthermore, *Requet* delivers a 91% accuracy in predicting low (144p/240p/360p) or high resolution (480p/720p/1080p) in both the web browser over WiFi setting and the YouTube App over LTE setting (see Section 2.7).
  - Demonstrate that *Requet* trained in a lab environment works on unseen clips with varying lengths from different operators in multiple countries. This evaluation is more diverse than prior work [109, 133, 52, 203] (see Section 2.7).

The design and experimental evaluation of Requet appeared in ACM Transactions on Multimedia Computing, Communications and Applications (TOMM) [77]. A preliminary version appeared in the proceedings of ACM MMSys'19 [78]. The work on *Requet* started in collaboration with Dr. Katherine Guo from Bell Labs. Undergraduate students Sarthak Arora and Trey Gilliland made important contributions to the data collection.

2.2 Related Work

Traditional traffic monitoring systems rely on DPI to understand HTTP request and reply messages. The systems use meta-data to understand ABR and infer video QoE metrics. The MIMIC system estimates average bitrate, re-buffering ratio and bitrate switches for a session by examining HTTP logs [128]. Comparatively, BUFFEST builds ML classifiers to estimate buffer level based either on the content of HTTP requests in the clear or on unencrypted HTTPS requests by a trusted proxy [109]. HighSee identifies HTTP GET requests and builds a linear Support Vector Machine (SVM) [46] model to identify audio, video, and control chunks to separate audio, video and control

flows [64].

For encrypted traffic, proposals fall into two categories. The first category builds session models offline by detecting HTTP requests as in eMIMIC [126], while the second category builds ML models to predict QoE metrics either offline or online.

Offline Models: The offline approach uses entire video session traffic to generate features to classify the session into classes. YouQ classifies a session into two to three QoS classes [152]. The system in [52] builds models to roughly put a session into three categories in terms of stall events ("non-stall", "0-1 stalls", or "2-or-more stalls"), or three classes based on average quality level. The system in [153] captures IP level traffic information (which is suitable for both TLS and QUIC traffic) and feeds ML models to predict per-session Mean Opinion Score (MOS) (2 or 3 classes), longest resolution ("sd" vs. "hd"), and stalling occurrences ("yes" vs. "no"). Using simulation, [205] builds ML models to predict average bitrate, quality variation, and three levels of stall ratio (no, mid, severe) for entire sessions using post processing. Comparatively, [123] classifies a session into two categories (with or without stall events) based on cell-related information collected at the start of a video session.

Focusing on the newly proposed Network Data Analytics Function in the 5G architecture, [178] associates network QoS metrics with the MOS for each video session using ML models. Rather than using actual network traces, the evaluation of the ML models is purely based on simulation.

Online Models: The online approach uses traffic from the past time window in the session to generate features to predict QoE metrics specific to that time window. ViCrypt [182] develops ML models to predict stall events both in real-time and for the entire video session based on network level information for separate TCP and UDP flows. On the other hand, [222] builds ML models to purely predict video resolution in real-time using network level information as features. The focus of the study is on feature selection and benchmarking of different ML models. Similarly, [124] simply focuses on prediction of buffer level using features based on network level information. It only predicts two states – "buffering" and "stable" and discards any transition period in between (that is, without including these data in training or testing of ML models), while *Requet* predicts

video state based on buffer status in much finer granularity (with four exclusive states: buffer increase, buffer decay, steady state and stall) without discarding any data. The system in [133] develops features based on both network and transport level information in a 10 sec time window to build separate classifiers for HTTPS and QUIC traffic to infer startup delay (below or above a given threshold), stall event occurrence, and video quality level ("low" and "high"). This system uses features based on packet level information and collects data for time windows of 100 ms. This has a relatively large memory requirement compared to *Requet* which only requires network data collected on a per chunk basis.

The system in [28] uses network and application level features to infer startup delay and resolution. Similar to *Requet* , they also identify video chunks.

Flow Identification: Identifying video flows from encrypted traffic is orthogonal to the QoE detection problem for given ABR flows. It is an example of the broad encrypted traffic classification problem. The Silhouette system [117] detects video chunks (also named Application Data Units) from encrypted traffic in real-time for ISP middleboxes using video chunk size, payload length, download rate threshold values. The real-time system in [169] identifies Netflix videos using TCP/IP header information including TCP sequence numbers. This approach relies on a "finger print" database built from a large number of video clips hosted by Netflix. The finger print is unique for each video title, therefore it is ineffective in classifying new video titles not previously seen. The system in [203] classifies an encrypted Youtube flow every 1sec interval into HAS or non-HAS flows in real-time. For a HAS flow, it further identifies the buffer states of the video session into filling, steady, depleting and unclear. The high accuracy to predict buffer state is partly due to the fact that the entire dataset contains only 3 clips with multiple sessions for each clip. This system also uses a feature based on the standard deviation of packet size, which is not feasible for implementation in middleboxes due to the memory requirement.

2.3 Background & Problem Statement

2.3.1 Adaptive BitRate Streaming Operation

A majority of video traffic over the Internet today is delivered using HTTP Adaptive Streaming (HAS) with its dominating format being Dynamic Adaptive Streaming over HTTP (DASH) or MPEG-DASH [192, 223]. In Adaptive BitRate (ABR), a video *asset* or *clip* is encoded in multiple resolutions. Encoding is controlled by multiple parameters and a given resolution is associated with a fixed setting for quantization, which is then coarsely related to an average bandwidth determined by the source video. A clip with a given resolution is then divided into a number of *segments* or *chunks* of variable length, a few seconds in playback time [129]. Typically video clips are encoded with Variable Bitrate (VBR) encoding and are restricted by a maximum bitrate for each resolution. An audio file or the audio track of a clip is usually encoded with Constant Bitrate (CBR). For example some of the YouTube audio bitrates are 64, 128, and 192 Kbps [211].

At the start of the session, the client retrieves a manifest file which describes the location of chunks within the file containing the clip encoded with a given resolution. There are many ABR variations across and even within video providers [129]. ABR is delivered over HTTP(S) which requires either TCP or any other reliable transport [61]. The ABR algorithm can use concurrent TCP or QUIC/UDP flows to deliver multiple chunks simultaneously. A chunk can either be video or audio alone or a mixture of both.

2.3.2 Video States and Playback Regions

The client employs a *playout buffer* or *client buffer*, whose maximum value is *buffer capacity*, to temporarily store chunks to absorb network variation. To ensure smooth playback and adequate *buffer level* the client requests a video clip chunk by chunk using HTTP GET requests, and dynamically determines the resolution of the next chunk based on network condition and buffer status.[1]

[1]The field of ABR client algorithm design is an active research area [130, 92].

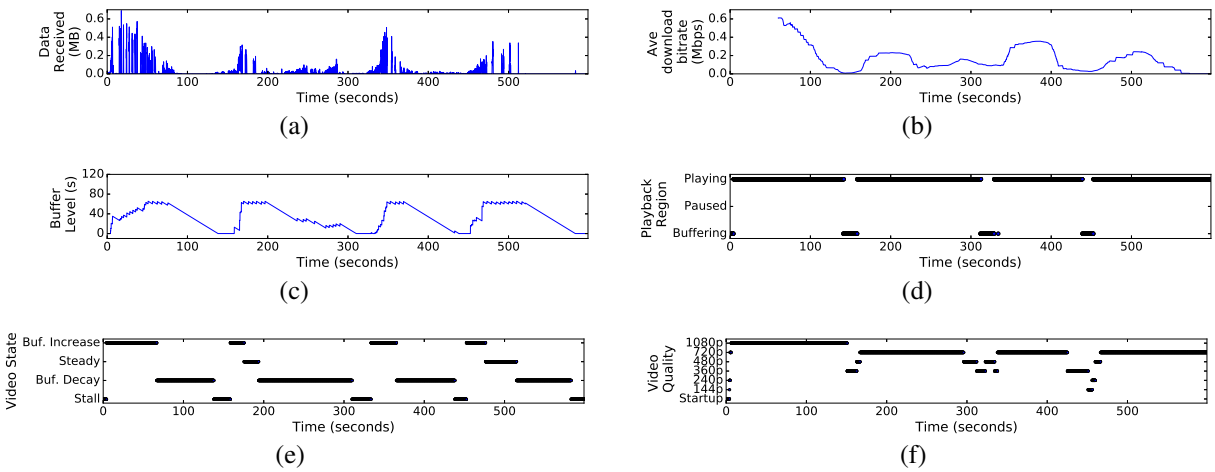


Figure 2.3: Behavior of a 10-min session in 100 ms windows: (a) amount of data received (MB), (b) average download bitrate (Mbps) over the past 60 sec, (c) buffer level, (d) playback region, (e) video state, (f) video resolution.

When the buffer level is below a low threshold, the client requests chunks as fast as the network can deliver them to increase the buffer level. We call this portion of ABR operation the *buffer filling* stage. In this stage, buffer level can increase or decrease. Once the buffer level reaches a high threshold, the client aims to maintain the buffer level in the range between the threshold and buffer capacity. One example of a client strategy is to request chunks as fast as they are consumed by the playback process, which is indicated by the video *playback bitrate* for the chunk [203]. We call this portion the *steady state* stage. The playback *stalls* when the buffer is empty before the end of the playback is reached. After all chunks have been downloaded to the client buffer, there is no additional traffic and the buffer level decreases. From the perspective of buffer level, an ABR session can experience four exclusive *video states*: *buffer increase*, *buffer decay*, *steady state*, and *stall*.

Orthogonally, from the perspective of YouTube video playback, a session can contain three exclusive *regions*: *buffering*, *playing*, and *paused*. Buffering region is defined as the period when the client is receiving data in its buffer, but video playback has not started or is stopped. Playing region is defined as the period when video playback is advancing regardless of buffer status. Paused region is defined as the period when the end user issues the command to pause video playback before the session ends. In playing region, video state can be in either buffer increase, decay, or

steady state.

Fig. 2.3 shows the behavior of a 10-min session from our dataset in Section 2.5 in each 100 ms window with (a) the amount of data received (MB), (b) download throughput (Mbps) for the past 60 sec, (c) buffer level (sec), (d) occurrence of three playback regions, (e) occurrence of four video states, and (f) video resolution. At the start of the session and after each of the three stall events, notice that video resolution slowly increases before settling at a maximum level.

2.3.3 QoE Metrics and Prediction Challenges

This subsection describes the QoE metrics that we reference and the challenges in predicting these metrics. We focus on metrics that the operator can use to infer user QoE impairments in real-time. Specifically, we use three QoE metrics: *buffer warning*, *video state* and *video quality*. We do not focus on start up delay prediction, as it has been extensively studied in [109, 133, 126].

The first QoE metric we aim to predict is the current *video state*. The four options for video state are: buffer increase, buffer decay, stall, or steady state. This metric allows for determining when the video level of the user is in the ideal situation of steady state. Video state also recognizes occurrences of buffer decay and stall events, when the operator may want to allocate more resources towards this user given that there are enough resources and the user is not limited by the data plan.

The *buffer warning* metric is a binary classifier for determining if the buffer level is below a certain threshold $BuffWarning_{\text{thresh}}$ (e.g., under 20 sec). This enables operators to provision resources in real-time to avoid potential stall events before they occur. For example, at a base station or WiFi AP, ABR traffic with buffer warning can be prioritized.

Another metric used is the current *video resolution*. Video encoders consider both resolution and target video bitrate. Therefore, it is possible to associate a lower bitrate with a higher resolution. One can argue bitrate is a more accurate indicator of video quality. However, higher resolutions for a given clip often result in higher bitrate values. The YouTube client reports in real-time resolution rather than *playback bitrate*. Therefore, we use resolution as an indicator of

video quality.

ABR allows the client to dynamically change resolution during a session. Frequent changes in resolution during a session tend to discourage user engagement. Real-time resolution prediction enables detection of resolution changes in a session. However, this prediction is challenging as *download bitrate* to video resolution does not follow a 1-to-1 mapping. In addition, a video chunk received by the client can either replace a previous chunk or be played at any point in the future. Under the assumption that playback typically begins shortly (in the order of seconds) after the user requests a clip, one can associate the average download bitrate with video quality, since higher quality requires higher bitrate for the same clip. However, this is not true in a small time scale necessary for real-time prediction. Network traffic reveals the combined effect of buffer trend (increase or decay) and video *playback bitrate* which correlates to resolution. During steady state, video's *download bitrate* is consistent with playback bitrate. However, when a client is in non-steady state, one cannot easily differentiate between the case in which a higher resolution portion is retrieved during buffer decay state (Fig. 2.1(a)), and the case in which a lower resolution portion is retrieved during buffer increase state (Fig. 2.1(b)). Both of these examples exhibit similar traffic patterns, however the behavior of QoE metrics is dramatically different.

2.4 Chunk Detection

The fundamental delivery unit of ABR is a chunk [114]. Therefore, identifying chunks instead of relying on individual packet data can capture important player events. Our approach is to explore the fundamental principle of HAS which is to transmit media in the unit of video and audio chunks. The behavior of chunks over the transmission network is directly associated with the HAS protocol and behavior of the client buffer. This method is able to derive QoE metrics as long as one can (i) detect chunks and (ii) build models associating IP level traffic information with client buffer level. Therefore, this method is immune to changes to HAS as long as chunks can be detected from IP level traffic.

Specifically, the occurrence of a chunk indicates that the client has received a complete segment

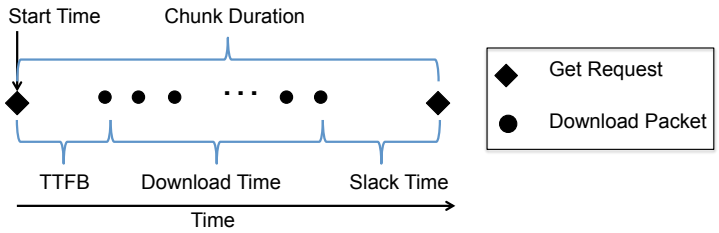


Figure 2.4: Definition of chunk metrics (video or audio).

of video or audio, resulting in increased buffer level in playback time. An essential component of *Requet* in Fig. 2.2 is its ChunkDetection algorithm to identify chunks from encrypted traffic traces. Features are extracted from the chunks and used as the input to the ML QoE prediction models.

Existing work using chunks either studies per-session QoE metrics [126] instead of predicting QoE metrics in real-time, or lacks insight in chunk detection mechanisms [117, 177, 52]. In general, there are two approaches of identifying chunks, (i) identify a packet with non-zero payload from the client to the server as an HTTP request [126] and (ii) use an idle period (e.g., 900 ms is used to separate chunks in a flow of Netflix traffic [124]).

In this section, we first describe metrics capturing chunk behavior. We then develop ChunkDetection, a heuristic algorithm using chunk metrics to identify individual audio and video chunks from IP level traces. *Requet* uses ChunkDetection to detect chunks from multiple servers simultaneously regardless of the use of encryption or transport protocol. It relies purely on source/destination IP address, port, protocol, and payload size from the IP header.

2.4.1 Chunk Metrics

We define the following metrics for a chunk based on the timestamp of events recorded on the end device (as shown in Fig. 2.4).

- **Start_Time -** The timestamp of sending the HTTP GET request for the chunk.
- **TTFB -** Time To First Byte, defined as the time duration between sending an HTTP GET request and the first packet received after the request.
- **Download_Time -** The time duration between the first received packet and the last received packet prior to the next HTTP GET request.

Algorithm 1 Audio Video Chunk Detection Algorithm

```
1: procedure CHUNKDETECTION
2:     Initialize Audio⃗ and Video⃗ for each IP flow I
3:     for each uplink packet p with IP flow I do
4:         if uplink(p) and (packetlength(p) > GET_thresh then)
5:             c ← [GetTimestamp, GetSize, DownStart,
6:             DownEnd, GetProtocol, I]
7:             AVflag ← DetectAV(c)
8:             if AVflag == 0 then
9:                 Append c to Audio⃗
10:            else if AVflag == 1 then
11:                Append c to Video⃗
12:            else
13:                Drop c
14:            GetTimestamp ← time(p)
15:            GetSize ← packetlength(p)
16:            DownFlag ← 0
17:        if downlink(p) and (packetlength(p) > Down_thresh then)
18:            if DownFlag == 0 then
19:                DownFlag = 1
20:                DownStart ← time(p)
21:            DownEnd ← time(p)
22:            DownSize+ = packetlength(p)
```

- **Slack_Time -** The time duration between the last received packet and the next HTTP GET request.

- **Chunk_Duration -** The time duration between two consecutive HTTP GET requests. The end of the last chunk in a flow is marked by the end of the flow. Note that a different concept called "segment duration" is defined in standards as playback duration of the segment [1]. For a given chunk, Chunk_Duration equals "segment duration" only during steady state.

- **Chunk_Size -** The amount of received data (sum of IP packet payload size) during Download_Time from the IP address that is the destination of the HTTP GET request marking the start of the chunk.

Note, for any chunk, the following equation holds: Chunk_Duration = sum(TTFB, Download_Time, Slack_Time).

2.4.2 Chunk Detection Algorithm

We explore characteristics of YouTube audio and video chunks. Using the web debugging proxy Fiddler [196], we discovered that audio and video are transmitted in separate chunks, and

Table 2.1: Chunk Notation

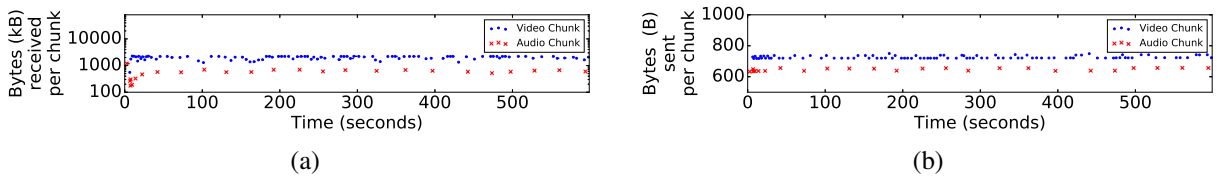
Symbol	Semantics
GET_{thresh}	pkt length threshold for request (300 B)
$Down_{\text{thresh}}$	pkt length threshold for downlink data (300 B)
$GetTimestamp$	timestamp of GET request
$GetSize$	pkt length of GET request
$DownStart$	timestamp of first downlink packet of a chunk
$DownEnd$	timestamp of last downlink packet of a chunk
$DownSize$	sum of the payload of downlink packets of a chunk
$GetProtocol$	IP header protocol field
DetectAV	sorts chunk into audio chunk, video chunk or no chunk based on $GetSize$, $DownSize$, $GetProtocol$
$\vec{Audio}$	audio chunks for an IP flow
$\vec{Video}$	video chunks for an IP flow

they do not overlap in time for either HTTPS or QUIC. For both protocols we notice at most one video or audio chunk is being downloaded at any given time. Each HTTP GET request is carried in one IP packet with IP payload size above 300 B. Smaller uplink packets are HTTP POST requests regarding YouTube log events, or TCP ACKs.

We propose a heuristic chunk detection algorithm in Algorithm 1 using notations in Table 2.1. ChunkDetection begins by initializing each IP flow with empty arrays for both audio and video chunks. This allows for the chunk detection algorithm to collect chunks from more than one server at a time.

ChunkDetection initially recognizes any uplink packet with a payload size above 300 B as an HTTP GET request (line 4). This threshold may vary depending on the content provider. For YouTube, we note that GET requests over TCP are roughly 1300 bytes, while GET requests over UDP are roughly 700 bytes. For each new GET request the $GetTimestamp$, and $GetSize$, are recorded (lines 14-16). After detecting a GET request in an IP flow, chunk size is calculated by summing up payload size of all downlink packets in the flow until the next GET is detected (lines 17-22). The last downlink packet in the group between two consecutive GET requests marks the end of a chunk download. The chunk download time then becomes the difference in timestamp between the fist and the last downlink packet in the group. [2]

[2]ChunkDetection does not flag TCP retransmission packets, therefore can overestimate chunk size when retransmission happens. ChunkDetection also assumes chunks do not overlap in time in a given IP flow. If it happens, the individual chunk size can be inaccurate, but the average chunk size over the period with overlapping chunks is still accurate.



(a) (b)

Figure 2.5: Individual video/audio chunks in a 10-min session with highest resolution (V:1080p, A:160kbps). (a) Chunk Size, (b) Get Request Size.

Once the next GET is detected, ChunkDetection records $GetTimestamp$, $GetSize$, download start time $DownStart$, download end time $DownEnd$, the protocol used $GetProtocol$, download size $DownSize$, and the IP flow I of the previous chunk (line 5). This allows for the calculation of chunk duration and slack time using the timestamp of the next GET. GET request size and chunk size are used in DetectAV (line 7) to separate data chunks into audio chunks, video chunks, or background traffic (lines 8-11). DetectAV uses the heuristic that HTTP GET request size for audio chunks is slightly smaller than request size for video chunks consistently. Figs. 2.5 and 2.6 plot the HTTP GET request size and subsequent audio/video chunk size in a high (1080p) and a low (144p) resolution session, respectively. It is evident from Figs. 2.5(b), and 2.6(b) that the HTTP GET request size for audio chunks is slightly smaller than that for video chunks. Through the examination of encrypted YouTube HTTP GET requests for video and audio using Fiddler, we discover that this difference is due to the additional fields used in HTTP GET requests for video content which do not exist for audio content. Furthermore, as can be seen in Fig. 2.5(a), the video chunk size at higher resolution levels is consistently larger than the audio chunk size. However, as can be seen in Fig. 2.6(a), the video chunk size at lower resolution levels can be similar to or even smaller than the audio chunk size. We can conservatively set the low threshold for chunk size to be 80 KB for our dataset. Furthermore, if download size is too small ($<$ 80 KB), DetectAv recognizes that the data is neither an audio or video chunk, and the data is dropped (lines 12-13). This allows ChunkDetection to ignore background YouTube traffic.

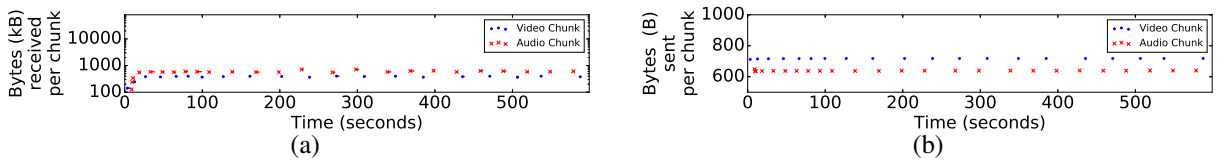


Figure 2.6: Individual video/audio chunks in a 10-min session with lowest resolution (V:144p, A:70kbps). (a) Chunk Size, (b) Get Request Size.

2.5 Data Acquisition

As shown in Fig. 2.2, data acquisition provides data for training and evaluation for *Requet* QoE prediction models, including traffic trace collection, derivation of QoE metrics as ground truth labels associated with traffic traces. We describe additional details about the publicly available dataset in Appendix 2.B. We collect data in two distinct settings, one using YouTube from a browser on a laptop over WiFi networks (*"Browser-WiFi"*), and the other using YouTube App on an Android smartphone over LTE cellular networks (*"App-LTE"*). We name the datasets Browser-WiFi and App-LTE, respectively. The data is collected over two different time periods to illustrate *Requet*'s performance, since the underlying protocol of YouTube may vary on different devices, over different networks, and over time [129].

2.5.1 Trace Collection from Browser over WiFi

For the first set of experiments, we design and implement a testbed (shown in Fig. 2.7(a)) to capture a diverse range of YouTube behavior over WiFi. We watch YouTube video clips using the Google Chrome browser on a Macbook Air laptop. We connect the laptop to the Internet via an Access Point (AP) using IEEE 802.11n. A shell script simultaneously runs Wireshark's Command Line Interface, Tshark [8], and a Javascript Node server hosting the YouTube API.

The YouTube IFrame API environment collects information displayed in the "Stats for Nerds" window. From this API we monitor: video playback region ('Playing', 'Paused', 'Buffering'), playback time since the beginning of the clip, amount of video that is loaded, and current video resolution. From these values we determine the time duration of the portion of the video clip remaining in the buffer. We collect information once every 100 ms as well as during any change

event indicating changes in video playback region or video resolution. This allows us to record any event as it occurs and to keep detailed information about playback progress.

We have two options to collect network level packet traces in our setup, on the end device or on the WiFi AP. Collecting traces at the AP would limit the test environment only to a lab setup. Therefore, we opt to collect traces via Wireshark residing on the end device. This ensures that the YouTube client data is synchronized with Wireshark traces and the data can be collected on public and private WiFi networks. Our traces record packet timestamp, size, as well as the 5-tuple for IP-header (source IP, destination IP, source port, destination port, protocol). Our dataset contains delivery over HTTPS (9% GET requests) and QUIC (91% GET requests). We do not use any transport level information. In addition, we record all data associated with a Google IP address. The IP header capture allows us to calculate total number of packets and bytes sent and received by the client in each IP flow during a given time window.

To generate traces under varying network conditions, we run two categories of experiments: *static* and *movement*. For static cases, we place the end device in a fixed location for the entire session. However, the distance from the AP varies up to 70 feet or multiple rooms away. For movement cases, we walk around the corridor (up to 100 feet) in our office building with the end device, while its only network connection is through the same AP.

We select 60 YouTube clips representing a wide variety of content types and clip lengths. Each clip is available in all 6 common resolutions from YouTube, namely 144p, 240p, 360p, 480p, 720p and 1080p. We categorize them into four groups, where groups A and B are medium length clips (8 to 12 min), C are short clips (3 to 5 min), and D are long clips (25-120 min). Table 7.1 lists the number of unique clips in the group, along with the length of each clip and the session length, that is, the duration for which we record the clip from its start.

For group A, we collect 425 sessions in both static (over 300) and movement cases (over 100) in a lab environment in our office building. All remaining experiments are conducted in static cases. For clips in group B, we collect traces in an apartment setting in the US (set B_1 with 60 sessions) and in India (set B_2 with 45 sessions) reflecting different WiFi environments. We collect

(a) (b)

Figure 2.7: Experimental setup for our trace collection. (a) WiFi experiments conducted in the lab on a laptop, (b) Cellular experiments on an android cellphone.

traces in set C and D from the lab environment, where each set contains more than 25 sessions. Overall, there are over 10 sessions for each clip in group A and B and 6 sessions for each clip in group C and D.

Clips in both groups A and B range from 8 to 12 min in length. In each session we play a clip and collect a 10-min trace from the moment the client sends the initial request. We choose this range of length in order for the client to experience buffer increase, decay and steady state. Shorter clips with a length close to buffer capacity (e.g., 2 min) can sometimes never enter steady state, even when given abundant network bandwidth. In general, when there is sufficient bandwidth to support the clip's requirement, a clip can be delivered in its entirety before the end of the playback happens. On the contrary, when available network bandwidth is not enough to support the clip's requirement, a clip may experience delayed startup and even stall events.

We collect traces over 6 months from Jan. through June 2018, with video resolution selection set to *"auto"*. This means the YouTube client is automatically selecting video resolution based on changes in network conditions. For each session, we set an initial resolution to ensure that all resolution levels have enough data points.

Each group includes a diverse set of clips in terms of activity level. It ranges from low activity types such as lectures to high activity types such as action sequences. This fact can be seen in the wide range of video bitrates for any given resolution. Fig. 2.8 shows the average playback bitrate for each video resolution for each clip in our dataset. All clips are shown in scatter plots, while clips in group A are also shown with box plots.[3] One can see that the average video playback bitrate spans overlapping regions. Therefore, this cannot provide a perfect indication of the video

[3]For all box plots in the chapter, the middle line is the median value. The bottom and top line of the box represents Q1 (25-percentile) and Q3 (75-percentile) of the dataset respectively. The lower extended line represents $Q1 - 1.5IQR$, where IQR is the inner quartile range (Q3-Q1). The higher extended line represents $Q3 + 1.5IQR$.

Table 2.2: Clip distribution in our dataset.

Group	Clip Length	Session Length	No. of Unique Clips
A	8 – 12 min	10 min	40
B	8 – 12 min	10 min	10
C	3 – 5 min	5 min	5
D	25 – 120 min	30 min	5

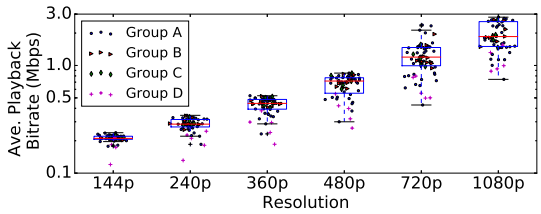


Figure 2.8: Average playback bitrate vs. video resolution for clips in our dataset. Clips in all four groups are shown in scatter plots, while clips in group *A* are also shown with box plots.

resolution even if the entire session is delivered with a fixed resolution.

In our dataset, we notice that YouTube buffer capacity varies based on video resolution. For example, it is roughly 60, and 120 sec for 1080p and 144p, respectively.

We collect data for each YouTube video session in the Chrome browser as the sole application on the end device. We record all packets between the client and any Google server. The client contacts roughly 15 to 25 different Google servers per session. We examine the download throughput (see Fig. 2.3(a) and 2.3(b) for example) further by looking at the most commonly accessed server IP addresses for each session sorted by the total bytes received. Our observation is that, during a session, the majority of traffic volume comes from a single to a few servers.

2.5.2 Trace Collection from YouTube Android App over Cellular

Testing on a mobile device connected to a laptop computer allows us to easily connect to cellular networks which enables testing outside of a lab environment over a cellular network. For the second set of experiments, we design and implement a data acquisition environment (shown in Fig. 2.7(b)) to capture YouTube video playback statistics and encrypted network packet data on an Android device over a cellular network. We use a rooted Motorola Moto G6 smartphone connected to the Internet via Google Fi's cellular networks. A shell script autonomously sets up the testing

environment using Android Debugging Bridge (ADB), collects packet data through tcpdump, and collects video playback statistics through the YouTube Android App.

The YouTube App allows for collection of video playback statistics through its "Stats for Nerds" window. This window allows us to easily monitor audio and video resolution, buffer health, and video playback region ("Playing", "Paused", and "Buffering"). We copy the information provided by that window every 1 sec to a clipboard log using ClipStack which can then be easily exported from the device.

Because we do not have access to data going through the cellular network, we opt to collect network traffic data on the phone using tcpdump for Android. We conduct tests in multiple cellular conditions such as in a car driving on the highway, on a Columbia University shuttle bus around upper Manhattan, in a backpack walking up and down the streets of New York City, and during lectures. We collect this set of cellular data over 7 months from June through Dec. 2019. Again, we use the 40 unique medium length clips in group *A* (8 to 12 min in length). The dataset consists of over 250 video sessions with resolution ranging from 144p to 1080p.

2.6 Requet ML Feature Design

We develop the ML *QoE metric prediction models* for *Requet* by using packet traces and associated ground truth labels (Section 2.5). We describe in detail in Appendix 2.A our heuristic algorithm for the video state labeling process to associate each time window with one of the four video states: buffer increase, buffer decay, steady state, and stall. As shown in Fig. 2.2, *Requet* uses its ChunkDetection component (Section 2.4) to convert traces into chunks, followed by its Feature Extraction component to extract associated features.

We develop ML models using Random Forest (RF) to predict user QoE metrics [86]. We build the RF classifier in Python using the sklearn package. We configured the model to have 200 estimators with the entropy selection criterion and the maximum number of features per tree set to auto. We choose RF for the following reasons. (i) ML classification algorithms based on decision trees have shown better results in similar problems [52, 133, 152, 203, 222, 124] with RF showing

the best performance among the class [203, 222, 124]. (ii) On our dataset, Feedforward Neural Network and RF result in roughly equal accuracy. (iii) RF can be implemented with simple rules for classification in real-time, well suited for real-time resource provisioning in middleboxes.

Each session in our dataset consists of (i) IP header trace and (ii) QoE metric ground truth labels generated by our video labeling process in data acquisition (Section 2.5). *Requet*'s ChunkDetection (Section 2.4.2) transforms the IP header trace into a sequence of chunks along with the associated chunk metrics (Section 2.4.1). The goal of *Requet* QoE models is to predict QoE metrics using chunk metrics. To train such ML models, it is critical to capture application behavior associated with QoE metrics using chunk-based features. In this section, we analyze chunk behavior in our dataset (Section 2.6.1), explore how to capture such behavior in chunk-based features (Section 2.6.2), and explain how to generate baseline features used in prior work that are oblivious to chunk information (Section 2.6.3).

2.6.1 Chunk Analysis

We apply the ChunkDetection algorithm (Algorithm 1) of *Requet* to all sessions from the 40 clips in set A in our dataset in both Browser-WiFi and App-LTE settings.

We examine the correlation between various chunk metrics (audio or video, chunk size, chunk duration, *effective rate* which we define as chunk size over chunk duration, TTFB, download time, and slack time) to QoE metrics (buffer level, video state, and resolution). In most cases of our dataset, for a given session, audio and video chunks are transmitted from one server. However, in some cases audio and video traffic comes from different servers. In other cases, the server switches during a session. These findings are consistent with existing YouTube traffic studies [144].

Chunk Analysis in Browser-WiFi Setting

We list the distribution of audio and video chunks along with video state at the end of chunk download in Table 2.3. Most of the chunks arrive during steady or buffer increase states. An extremely small fraction (4% audio and 9% video) are associated with stall or buffer decay states.

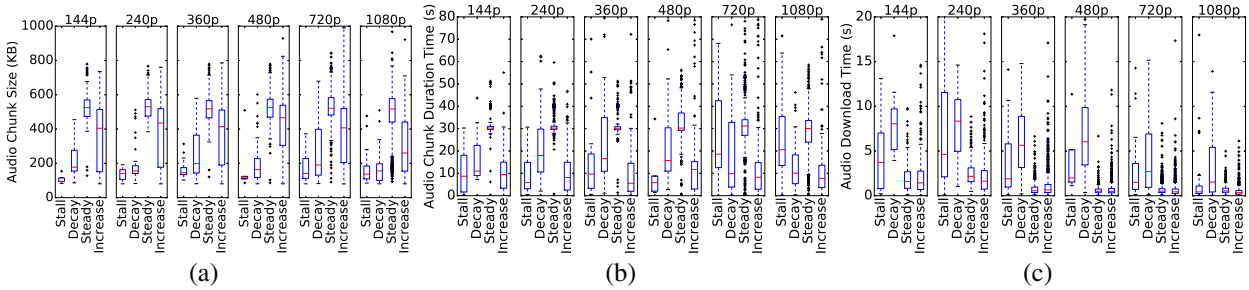


Figure 2.9: Chunk metrics for all audio chunks in set *A* in Browser-WiFi setting. (a) chunk size, (b) chunk duration, (c) download time.

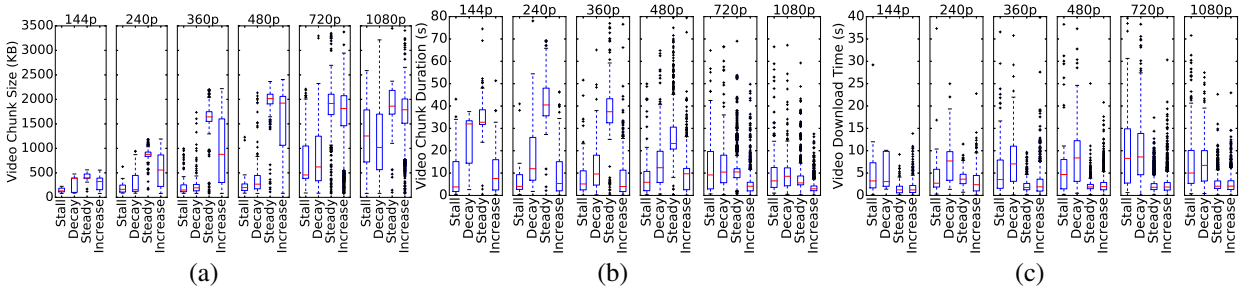


Figure 2.10: Chunk metrics for all video chunks in set *A* in Browser-WiFi setting. (a) chunk size, (b) chunk duration, (c) download time.

Table 2.3: % of chunks in each state (Set *A* Browser-WiFi).

Resolution	Video State			
	Stall	Decay	Steady	Increase
Audio	1.2	2.8	40.9	55.1
Video	3.7	5.9	47.6	42.8

They represent two possible scenarios: (i) bandwidth is limited and there are not enough chunks arriving to increase buffer level substantially or (ii) buffer is about to transition into increase state.

Figs. 2.9 and 2.10 show the box plots for chunk duration, size, and download time for audio and video chunks respectively. Each plus sign represents an outlier. TTFB reflects the round trip time from the client to the server, and has a median value of 0.05 sec. This accounts for a tiny portion of chunk duration (median value $\geq$ 5 sec). We can safely simplify the relationship between various chunk metrics to (slack time = chunk duration - download time). Notice that slack time and effective rate are derivable from chunk duration, size, and download time. The latter three are the key metrics used in our feature selection for ML models.

Audio is encoded with CBR, however our examination of HTTP requests using Fiddler [196] reveal that in the four video states (steady, buffer increase, decay and stall), audio chunk size

Table 2.4: % of chunks in each state (Set *A* App-LTE).

Resolution	Video State			
	Stall	Decay	Steady	Increase
Audio	2.7	4.0	52.3	41.0
Video	3.6	6.8	49.6	40.0

decreases in the same order. This implies that audio chunk playback time also decreases in the same order. This behavior is consistent across all resolution levels (Fig. 2.9(a)) and indicates that audio chunk size exhibits a strong correlation with video state. Across all resolution levels, Fig. 2.9(b) shows median audio chunk duration in steady and buffer increase state is roughly 30 and 10 sec respectively, but does not exhibit clear pattern in stall and buffer decay states. Fig. 2.9(c) shows audio chunk download time in steady and buffer increase states are similar in value, both smaller than that of stall state, which is smaller than that of buffer decay state. The longer download time is an indication that the network bandwidth is limited. This is a useful insight that current bandwidth alone can not reveal. For example, a specific throughput can be associated to a low resolution with the buffer increasing or a higher resolution with the buffer decreasing. All three audio chunk metrics are clearly correlated with video state.

Fig. 2.10 shows video chunk statistics. There is a large overlap across different resolutions and video states in chunk size (Fig. 2.10(a)) and chunk duration (Fig. 2.10(b)). It reveals that without knowing video state, it would be difficult to determine video resolution, chunk size, and chunk duration. For example, these statistics are very similar for a 240p chunk in buffer increase state and a 720p chunk in buffer decay. Using audio chunk statistics to identify video state is critical in separating these two cases.

For video chunks, our examination of HTTP requests using Fiddler also shows that for a clip with a given resolution, steady state chunk size is larger than that in the remaining three states. Fig. 2.10(a) further shows that median video chunk size increases as resolution increases from 144p to 480p and stays roughly the same around 2 MB from 480p to 1080p. Fig. 2.10(b) shows median chunk duration in steady state is similar for 144p, 240p, and 360p, in the range of 35 – 45 sec, and decreases from 25 sec for 480p to 5 sec for 1080p. To obtain a higher effective rate for higher

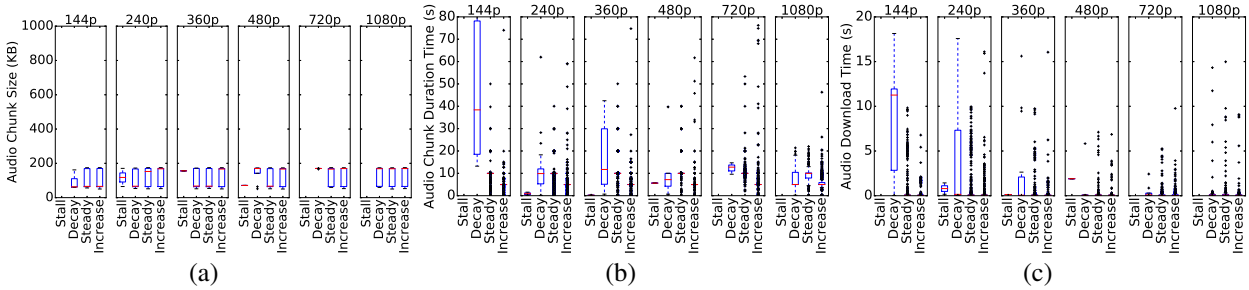


Figure 2.11: Chunk metrics for all audio chunks in set A in App-LTE setting. (a) chunk size, (b) chunk duration, (c) download time.

resolutions the chunk size levels off, but to compensate chunk duration decreases. Fig. 2.10(c) shows that median chunk download time exhibits larger values in stall or buffer decay state, smaller and similar values in steady or buffer increase state. This is expected as with limited bandwidth, a session may experience buffer decay or even stall. Both buffer decay and stall periods exhibit larger chunk download times. However, during buffer increase, retrieving smaller chunks faster than steady state results in similar download time as steady state. During steady and buffer increase state, chunk size and duration combined provide some indication of resolution levels. However, during stall and buffer decay state, no indication can be easily seen from the three metrics.

To summarize, our key observations are as follows: (i) Without knowing video state it would be difficult to differentiate between the two cases: (a) Higher resolution clip in buffer decay and (b) Lower resolution clip in buffer increase. (ii) Audio chunk statistics exhibit strong association with video state. (iii) Video chunk size increases and eventually levels off as resolution increases. At the same time, video chunk duration is higher for lower resolution levels and decreases as resolution level increases.

Chunk Analysis in App-LTE Setting

Similar to the Browser-WiFi setting, most of the chunks in the App-LTE setting arive during steady or buffer increase states.

For the App-LTE setting, Figs. 2.11 and 2.12 show the box plots for chunk duration, size, and download time for audio and video chunks respecitvely. Each plus sign represents an outlier.

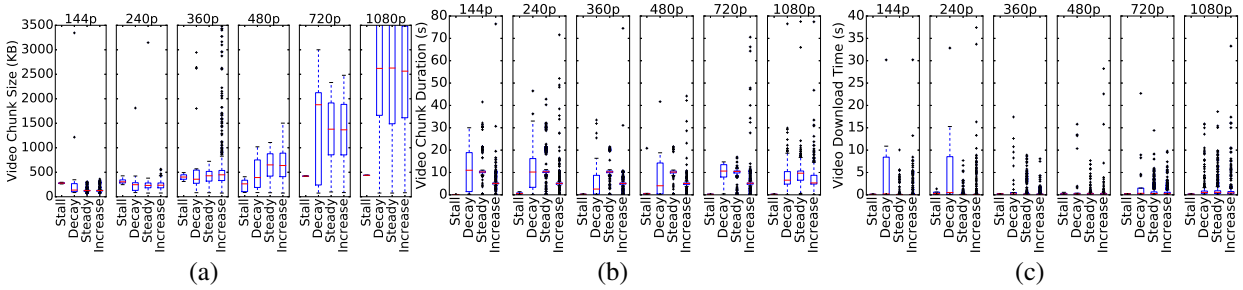


Figure 2.12: Chunk metrics for all video chunks in set *A* in App-LTE setting. (a) chunk size, (b) chunk duration, (c) download time.

Across all resolution levels, Fig. 2.11(b) shows that median audio chunk duration is roughly 10 sec in steady state and 5 sec in buffer increase state. However, there is no clear pattern in stall or buffer decay states. Audio chunk size is consistent across all resolution levels (Fig. 2.11(a)) and is usually around 70KB or 170KB. These patterns are considerably different from the Browser-WiFi setting in Fig. 2.9.

Fig. 2.12 shows video chunk statistics in the App-LTE setting. Again, the pattern is drastically different than the pattern in the Browser-WiFi setting in Fig. 2.10. Fig. 2.12(a) shows that across different states in the same resolution, the chunk size is much more consistent. In addition there is a clear pattern of increasing chunk size as the resolution increases. The video chunk duration results (Fig. 2.12(b)) show that video chunks arrive roughly every 10 sec during steady state and roughly every 5 sec during buffer increase state. This video chunk arrival behavior is similar to that of the audio chunks in the same dataset. Fig. 2.12(c) shows, with a fixed resolution, median video chunk download times exhibit larger values in stall or buffer decay state, and smaller values in steady or buffer increase state. This is expected, since with a fixed chunk size, a larger chunk duration is associated with limited bandwidth, which can cause a session to deplete its buffer (enter buffer decay state) or even stall.

2.6.2 Chunk-based Features in *Requet*

Requet identifies chunks using Algorithm 1 executed over all flows during a YouTube session. For each audio or video chunk, it records the following seven chunk metrics: protocol used to send the GET request, start time, TTFB, download time, slack time, chunk duration, and chunk size.

However, it does not record the server IP address from which the chunk is delivered to the end device as it has no relationship with our QoE metrics.

Results from Section 2.6.1 show that the most important metrics for both audio and video are chunk size, duration, and download time. Chunk arrival is not a uniform process in time and therefore, the number of chunks in a time window vary. This would require a variable number of features. Instead, *Requet* uses statistics of chunk metrics in different time windows. Specifically, for the 20 windows representing the immediate past 10, 20, ..., 200 sec, it records total number of chunks, average chunk size and download time for each time window, resulting in 60 features each for audio and video, and a total of 120 features.[4] Regarding video resolution, *Requet* only makes predictions upon receiving a video chunk. Therefore, beyond the 120 features, it further includes the 7 features associated with the video chunk. By only collecting data on a per chunk basis, *Requet* requires a minimal amount of storage of 7 fields per chunk in the middlebox. Figs. 2.11(b) and 2.12(b) show that chunks in the dataset arrive on average once every 5 sec. The sliding window based features in *Requet* make it ideal for middleboxes with a memory requirement of 1016 bytes for the 127 features (assuming each feature requires a maximum of 8 bytes).

2.6.3 Baseline Features

For the baseline system, we remove *Requet*'s ChunkDetection algorithm in Fig. 2.2 and the associated features. We replace *Requet* and design a baseline system with a set of features that are commonly used in prior work [109, 133, 152, 203]. Specifically, we select features that are used in more than one of these prior works and use time window based features. We collect basic IP level features in terms of flow duration, direction, volume (total bytes), burstiness, as well as transport protocol. For each 100 ms window, we calculate the total number of uplink and downlink packets and bytes, and include a one-hot vector representation of the transport protocols used for each IP address.[5] The five features for transport protocol are QUIC, TCP with TLS, TCP without

[4] We use the past 200sec history as YouTube buffer rarely increases beyond 3 min.

[5] In natural language processing, a one-hot vector is a $1xN$ matrix (vector) used to distinguish each word in a vocabulary from every other word in the vocabulary. The vector consists of 0s in all cells with the exception of a single 1 in a cell used uniquely to identify the word. In our case, each IP address is treated as a word.

TLS, no packets in that interval, or other. After examining the total downlink bytes of the top 20 flows in a session in our dataset, we decide to include traffic from the top 3 servers in our feature set. The remaining flows have significantly smaller traffic volume and therefore represent background traffic in a session and do not deliver video or audio traffic. By doing so, we effectively eliminate the traffic that is unrelated to our QoE metrics. In addition, we include the total number of uplink/downlink bytes and packets from the top 20 servers for the session.

We calculate the average throughput and the total number of packets in the uplink and downlink direction during a set of time intervals to capture recent traffic behavior. Specifically, we use six intervals immediately proceeding the current prediction window, and they are of length 0.1, 1, 10, 60, 120, and 200 sec.

Furthermore, during these six windows, we record the percentage of 100 ms slots with any traffic in uplink and downlink separately. These two features are added to determine how bursty the traffic is during the given time window. In addition to the four features for the total network traffic for all servers contacted during the session, the features for each of the top three servers are:

- total bytes in past 100 ms in uplink/downlink
- total number of packets in past 100 ms in uplink/downlink
- transport protocol (5 features)
- for each of the windows of length 1, 10, 60, 120, and 200 sec:
  - average throughput in uplink/downlink
  - total number of packets in uplink/downlink
  - % of 100 ms slots without any traffic in uplink/downlink

To summarize, for each time window, there are up to $4 + 3 \times (4 + 5 + 5 \times 6) = 121$ features for the baseline system.

2.7 Evaluation

We evaluate the performance of *Requet* in both the Browser-WiFi setting and the App-LTE setting. For the Browser-WiFi setting we compare the accuracy in predicting each QoE metric of *Requet* versus the baseline system. Both systems predict the current QoE metrics every 5 sec, except for *Requet* which predicts resolution every chunk. Since the collected network traffic transport payload is encrypted, we are unable to evaluate *Requet* against previous works that use deep packet inspection. Data collected as described in Section 2.5 is used for training, validation, and testing. Out of the four sets of traces in our dataset (Section 2.5.1), we use group A, the largest one to train both systems to predict each QoE metric in real-time. We follow the same testing procedure to evaluate the performance of *Requet* in the App-LTE setting. We then compare the performance differences of *Requet* in both settings.

We extend the evaluation of *Requet* in the Browser-WiFi setting by testing *Requet* on smaller groups B, C, and D. Subsequently, we use groups B_1 and B_2 to determine how training in the lab environment works on clips with similar length but with different service providers and wireless network conditions. B_1 and B_2 are experiments in residential WiFi settings in the US and India, respectively. We also use group A as the training set for evaluating shorter clips (group C) and longer clips (group D) in the same lab environment as group A.

For group A, we conduct 4-fold cross validation on the 40 clips. Specifically, we divide the 40 clips into four exclusive sets each with ten unique clips. In each fold, we first train a model for each QoE metric using RF with features from 30 clips (three of the four sets). We then test the model on the ten clips from the remaining set. We report each model's average performance over the four folds.

The buffer warning model produces two prediction possibilities. It indicates whether the buffer level is below the threshold $BuffWarning_{\text{thresh}}$ or not. The video state model produces four states and the resolution model produces six resolution levels.

We report *accuracy* of each model as the ratio of the number of correct predictions over total

number of predictions. For each label a model predicts, we further report: (i) *precision* defined as the ratio of true positives to total positives, that is, the percentage of correct predictions out of all positive predictions of a label, and (ii) *recall* defined as the ratio of correct predictions to total true occurrences of a label, that is, the percentage of a label correctly predicted.

2.7.1 Buffer Warning Prediction

The first metric we examine is buffer warning. We set the threshold for buffer level warning, $BuffWarning_{\text{thresh}}$, to be 20 secs. This provides ample time to provision enough bandwidth before an actual stall occurs.

For this metric, each time window in our dataset is labeled with either "no buffer warning" (NBfW) or "buffer warning" (BfW). In group A, significantly more chunks are labeled with NBfW (84%) than BfW (16%). The results in Table 2.5 show that in the Browser-WiFi setting both baseline and *Requet* perform well for this task, with accuracy reaching 85% and 92%, respectively. We see that precision and recall for NBfW are higher than those for BfW in both baseline and *Requet*. Given the current label is BfW, *Requet* provides significantly higher probability of predicting BfW correctly with recall of 68% over 11% for the baseline. This is because *Requet* uses chunk features to detect the case when no chunks have recently arrived. However, it is difficult for the baseline system to identify such cases due to the lack of chunk detection. For example, baseline can not differentiate packets as being part of a chunk or background traffic.

In the App-LTE setting, *Requet* shows slightly improved performance compared to Browser-WiFi. *Requet* achieves a recall of 79.9% for BfW and 99.2% for NBfW. This results in a total accuracy of 97.8%. For the Browser-WiFi dataset, the download time and TTFB of the most recent chunk, the video chunk count and the average video chunk size of a variety of windows are significant features that are used in the RF model for buffer warning prediction. For the App-LTE dataset, the download time and TTFB of the most recent chunk, the video chunk count, and the audio chunk count of a variety of windows are significant features that are used in the RF model for buffer warning prediction.

Table 2.5: Buffer warning performance with data in group *A*.

Type	Baseline Browser-WiFi		*Requet* Browser-WiFi		*Requet* App-LTE	
	Precision	Recall	Precision	Recall	Precision	Recall
BfW	51.0	11.1	79.0	68.7	88.4	79.7
NBfW	86.0	98.1	94.1	96.5	98.5	99.2
Accuracy	84.9		92.0		97.8	

Table 2.6: Video state performance with data in group *A*.

Type	Baseline Browser-WiFi		*Requet* Browser-WiFi		*Requet* App-LTE	
	Precision	Recall	Precision	Recall	Precision	Recall
Stall	31.1	7.6	70.4	51.9	92.2	86.3
Buf. Decay	32.0	16.3	78.0	78.7	65.7	25.2
Buf. Increase	64.1	57.6	80.2	84.2	88.1	95.8
Steady	57.6	80.2	90.7	92.2	89.6	90.2
Accuracy	55.4		84.2		88.2	

2.7.2 Video State Prediction

The results of video state prediction are shown in Table 2.6. In the Browser-WiFi setting, *Requet* achieves overall accuracy of 84%, compared to 55% for baseline, representing a 53% improvement. *Requet* also outperforms baseline in precision and recall for each state.

Stall, buffer decay, buffer increase and steady state appear in 3.7%, 5.9%, 42.8% and 47.6% of chunks in group *A* respectively (Table 2.3). The precision and recall for both systems increase in the same order of stall, buffer decay, buffer increase and steady.

However, baseline achieves below 40% in precision and recall for both the stall and buffer decay states. This implies that during these two states, network traffic does not have a significant pattern for baseline to discover. Furthermore, during steady state there can be gaps of 30 sec or longer. A long gap also occurs when buffer is in decay state. Baseline features cannot separate buffer decay from steady state.

Examination of the *Requet* model reveals that audio chunk count for each 20 sec window is an important feature to predict video state. For example, if there are a few audio chunks in the past 20 sec it is likely that buffer is increasing, and if there are no audio chunks in the past 120 sec it is likely to be in stall state. This explains the relatively high performance of *Requet* .

In the App-LTE setting, *Requet* achieves an overall accuracy of 88.2%. Compared to the

Table 2.7: Video resolution performance with data in group A.

Type	Baseline Browser-WiFi		*Requet* Browser-WiFi		*Requet* App-LTE	
	Precision	Recall	Precision	Recall	Precision	Recall
144p	13.0	7.6	80.6	79.9	87.8	86.2
240p	14.6	10.1	68.7	64.3	74.0	81.8
360p	14.1	9.9	49.2	64.4	74.0	79.4
480p	24.7	33.3	64.9	63.8	73.7	57.2
720p	24.5	30.3	60.6	54.5	80.3	83.4
1080p	22.2	20.1	75.0	76.9	91.9	89.4
Accuracy	21.8		66.9		80.6	

Browser-WiFi dataset, *Requet* in the App-LTE setting achieves an improved performance in predicting the stall state, but is worse in predicting buffer decay.

For the App-LTE dataset the download time and TTFB of the most recent chunk, and the number of video chunks in the time range from 60 to 200 sec are significant features that are used in the RF model for state prediction. For the Browser-WiFi dataset the download time and TTFB of the most recent chunk, the number of video chunks in the time range from 60 to 200 sec, and the average chunk size are significant features that are used in the RF model for state prediction.

2.7.3 Video Resolution Prediction

It is extremely challenging for baseline to predict video resolution even with history of up to 200 sec. Overall accuracy is only 22%, slightly better than randomly picking one out of six choices.

As seen in Fig. 2.8, there is a large overlap of average playback bitrates of video clips of different resolutions due to varying activity levels in the video content. Without any knowledge about the content of the video or the video state, it is extremely difficult if not impossible to associate a chunk given its playback bitrate with the resolution it is encoded with. Furthermore, without knowing video state there is a large overlap in video chunk size and chunk duration across resolutions as seen in Fig. 2.10.

By using both audio and video chunks, *Requet* achieves a 66% accuracy for predicting resolution (six levels) in the Browser-WiFi setting. This result demonstrates that *Requet* is able to enhance video resolution prediction. By narrowing down the options in resolution to three: small (144p/240p), medium (360p/480p), and large (720p/1080p), *Requet* achieves an accuracy of 87%.

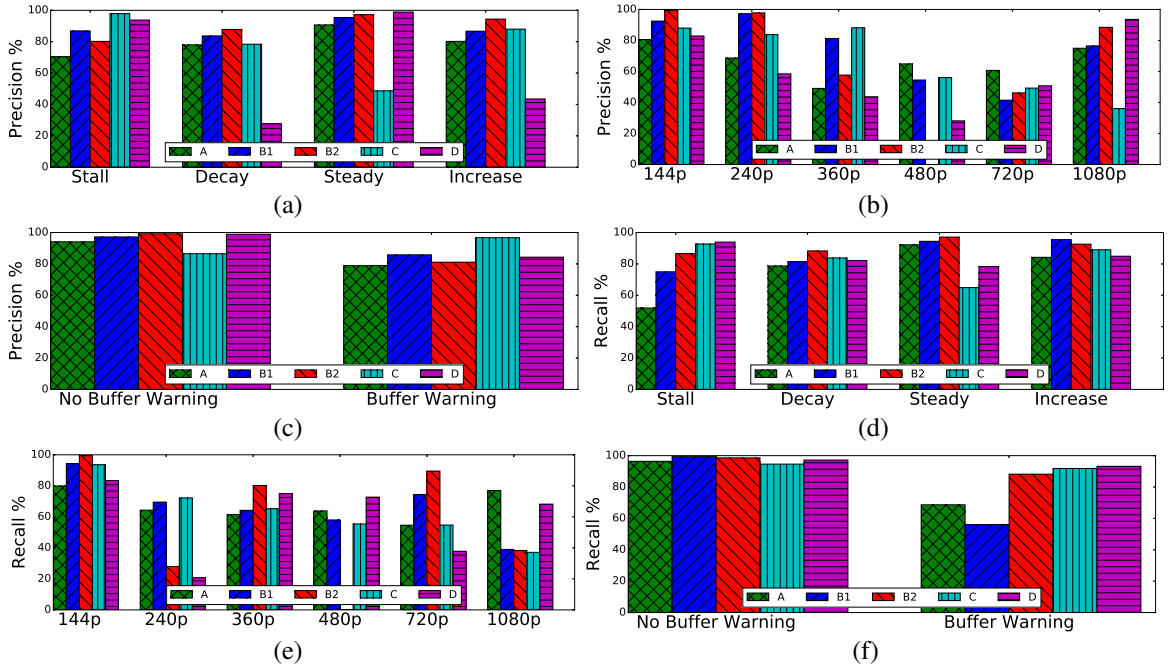


Figure 2.13: Accuracy of *Requet* models trained with group *A*. (a) Precision of video state, (b) Precision of video resolution, (c) Precision of stall warning, (d) Recall of video state, (e) Recall of video resolution, (f) Recall of stall warning.

If the number of options is reduced to two: small (144p/240p/360p) and large (480p/720p/1080p) the accuracy improves to 91%.

The accuracy of *Requet* in the App-LTE setting is 80.6%. *Requet* in the App-LTE setting has improved performance compared to in the Browser-WiFi setting in predicting all resolutions except 480p, where it has difficulties differentiating 480p from 360p. This can be caused by the dataset having more data points during 360p as well as having similar video chunk sizes for these two resolutions.

For both datasets, the most important features are those features related to the most recent chunk as well as the average video chunk size.

2.7.4 Performance Comparison of Browser-WiFi vs. App-LTE

The performance of *Requet* in the App-LTE setting is considerably greater than in the Browser-WiFi setting. As shown in Tables 2.5, 2.6, and 2.7, the accuracy for predicting buffer warning, video state, and video resolution in the Browser-WiFi setting is 92.0%, 84.2%, and 66.9%, respec-

tively. While the accuracy for predicting buffer warning, video state, and video resolution in the App-LTE setting is 97.8%, 88.2%, and 80.6%, respectively. The only exception to this is when predicting the buffer decay state, the accuracy is higher in the Browser-WiFi setting.

There are two potential reasons for the higher accuracy in the App-LTE setting. First, across different states in the same resolution, the chunk size is more consistent in the App-LTE setting. Second, the network conditions are more stable in the App-LTE setting, due to generally good service coverage in our test area. However, in the Browser-WiFi setting, artificially varying network conditions are created from movement experiments during the data collection stage. More stable network conditions naturally lead to less variation in video states once steady state is entered.

2.7.5 Extended Test over WiFi Networks

Up to this point we have reported results from our systems trained with part of group A and tested on different clips in group A in both the Browser-WiFi setting and the App-LTE setting. Next, we use group A in the Browser-WiFi setting as the training data for *Requet* and evaluate with groups B_1, B_2, C, and D. We test *Requet* on 10 clips from groups B_1 and B_2 for residential WiFi settings in the US and India, respectively, to see how they perform on unseen clips of similar length and unseen WiFi environments. In addition, we use the same lab WiFi environment in group A, to test *Requet* on 5 clips of shorter length of 5 min in group C and longer length of 25 min in group D. Fig. 2.13 reports the average precision and recall of these four tests along with the 4-fold cross validation results from group A.

Depending on the environment and QoE metric, performance of these extended sets of tests either improves or deteriorates compared with results from group A reported earlier in this section. For example, groups B_1, B_2, and C have improved precision and recall in predicting stall and buffer decay states. Group D shows lower precision in predicting buffer decay, but higher recall for both stall and buffer decay. Improved precision and recall results appear for predicting buffer threshold warning.

Accuracy for video resolution varies from experiment to experiment. Surprisingly, group B_2

has the highest overall accuracy of 70% when training with group A. This is in part due to that there were zero 480p events collected in group B_2. This resolution level has lower precision than 144p, 240p, and 1080p (see Table 2.7), and is extremely difficult for the other test sets to predict as well.

Most precision and recall results for other sets are better than group A with a few exceptions. This could be due to the fact that group A includes movement experiments, while the other groups only contain static ones. A video session naturally exhibits different behavior in different types of environments. In addition, we plan to improve our prediction models by studying how the imbalance in data samples impacts the precision and recall of each model.

2.A Appendix: Video State Labeling

A goal for predicting video QoE in real-time inside the network is to enable real-time resource provisioning to prevent stalls and decreases in video resolution. To enable this prediction, accurate labeling of video state is critical. The four exclusive video states (buffer increase, decay, stall and steady state) accurately capture the variations in buffer level. They can be used in combination with actual buffer level to predict dangerous portions of ABR operation that may lead to QoE degradation. For example, when the buffer level is close to 0, a stall event is likely to happen in the near future. Increasing network capacity for the session may prevent a stall.

As shown in Section 2.3, playback regions reported by the client ignore buffer level changes, and cannot be used to generate video states. Prior work uses manual examination which is time consuming and can be inaccurate [203]. We opt to automate the process by developing the definition of video states based on buffer level variation over time followed by our video state labeling algorithm. We define the four video states as follows:

1. **Buffer Increase:** Buffer level is increasing. It has a slope greater than ϵ per sec over time window T_{slope}.

2. **Steady State:** Buffer level is relatively flat. The slope of buffer level is between $-\epsilon$ and $+\epsilon$

Algorithm 2 Video State Labeling Algorithm

```
procedure VIDEOSTATELABELING
    Initialize δ, ϵ, T_smooth, T_slope
    for every t do
        Calculate B̂_t ← median[B_{t−T_smooth}, ..., B_{t+T_smooth}]
        Calculate m_t ← (B̂_{t+T_slope} − B̂_{t−T_slope}) / (2T_slope)
        if B̂_t ≤ δ then
            State_t ← Stall
        else if −ϵ ≤ m_t ≤ ϵ and B̂_t > Buff_SS then
            State_t = Steady State
        else if m_t < 0 then
            State_t ← Buffer Decay
        else
            State_t ← Buffer Increase
    SmoothState(State)
```

Table 2.8: Notation Summary

Symbol	Semantics	Defaults
δ	Stall threshold	0.08 sec
ϵ	Buffer slope boundary for Steady State	0.15 $\frac{sec}{sec}$
T_{smooth}	Time window for smoothing buffer	15 sec
T_{slope}	Time window to determine buffer slope	5 sec
$Buff_{\text{SS}}$	Minimum buffer level to be in steady state	10 sec
Thr_{SS}	Minimum time window to stay in steady state	15 sec
$MinTime_{\text{SS}}$	Time window to look for quick changes out of steady state	10 sec
$MinTime_{\text{stall}}$	Time window to look for quick changes out of stall state	10 sec

$\frac{sec}{sec}$ over time window T_{slope}. To be in steady state the slope needs to be in this range for greater than Thr_{SS} sec.

3. **Buffer Decay:** Buffer level is decreasing with a slope less than $-\epsilon$ $\frac{sec}{sec}$ over time window T_{slope}.

4. **Stall:** Buffer level is less than or equal to δ.

We execute our video state labeling algorithm in Algorithm 2 for each time instance t when buffer information is recorded (every 100 ms) to determine video state for a session according to our definition.

As a chunk arrives at the client, buffer level increases by chunk length in sec. During playback, buffer level decreases by 1 sec for every sec of playback. Looking at short windows or the wrong

point of a window would incorrectly determine that buffer is decreasing. We use a smoothing function to derive a more accurate buffer slope. Specifically, we use a moving median filter over a window around t defined by $[t - T_{\text{smoooth}}, t + T_{\text{smooth}}]$. We examine the rate of change of the buffer slope over a window around t defined by $[t - T_{\text{slope}}, t + T_{\text{slope}}]$.

In order to avoid rapid changes of stall state, we set δ to 0.08 sec. This value ensures that small variations in and out of stall state are consistently labeled as being in stall state. If the buffer level is above $Buff_{\text{SS}}$ and has a slope between $-\epsilon$ and $\epsilon \frac{sec}{sec}$, then we label it as steady state. If these specifications are not met and the slope is negative, we set the state to buffer decay. If the slope is positive, we set the state to buffer increase.

To ensure that video state does not change rapidly due to small fluctuations of buffer level, we use an additional heuristic of $SmoothState$: steady state has to last longer than Thr_{SS}. This allows chunks with playback time longer than this value to arrive at the client. If there are changes out of and then back into stall state that last less than $MinTime_{\text{stall}}$ we consider the entire period as stall state. Similarly, if there are changes out of and then back into steady state that last less than $MinTime_{\text{SS}}$, we consider the entire period steady state. For clarity, we list all symbols in Table 4.2, as well as the values that we find to work the best empirically for our dataset.

2.B Appendix: Dataset Info

This appendix provides a description of the dataset acquired in Section 2.5, used for *Requet* chunk detection in Section 2.4, and for evaluation in Section 2.7.

The dataset can be found in a Github Repository (https://github.com/Wimnet/RequetDataSet). The dataset is divided into 5 *group folders* for data from groups A, $B1$, $B2$, C, D, along with a summary file named 'ExperimentInfo.txt' for the entire dataset. Each line in the file describes an experiment using the following four attributes: (a) experiment number, (b) video ID, (c) initial video resolution, and (d) length of experiment in seconds.

A group folder is further divided into two subfolders, one for PCAP files, and the other for txt files. Each experiment is described by a PCAP file and a txt file. The PCAP file with name in the

form of (i) $'baseline\_\{date\}\_exp\_\{num\}.pcap'$ is for an experiment where the end device is static for the entire duration whereas a file with name in the form of (ii) $'movement\_\{date\}\_exp\_\{num\}.pcap'$ is for an experiment where the end device movement occurs during the experiment. The txt file names end with $'merged.txt'$. The txt file contains data colletect from YouTube API and summary of PCAP trace for the experiment.

In each $'merged.txt'$ file, data is recorded for each 100 ms interval. Each interval is represented as: [*Relative Time*, # Packets Sent, # Packets Received, # Bytes Sent, # Bytes Received, [Network Info 1], [Network Info 2], [Network Info 3], [Network Info 4], [Network Info 5], ... , [Network Info 25], [*Playback Info*]].

Relative Time marks the end of the interval. *Relative Time* is defined as the time since the Javascript Node server hosting the YouTube API is started. Relative Time for the 0th interval is defined as 0 sec. It is updated in intervals of 100 ms. TShark is called prior to the Javascript Node server. Therefore, the 0th interval contains Wireshark data up to the start of the Javascript Node server.

Network Info i is represented as: [IP_Src, IP_Dst, Protocol, # Packets Sent, # Packets Received, # Bytes Sent, # Bytes Received] for each interval. IP_Src is the IP address of the end device. The top 25 destination IP addresses in terms of total bytes sent and received for the entire session are recorded. For each i of the top 25 IP_Dst addresses, the Protocol associated with the higher data volume for the interval (in terms of total number of packets exchanged) is selected, and data volume in terms of packets and bytes for each interval is reported for the IP_Src, IP_Dst, Protocol tuple in [Network Info i].

Playback Info is represented as: [*Playback Event*, *Epoch Time*, *Start Time*, *Playback Progress*, *Video Length*, *Playback Quality*, *Buffer Health*, *Buffer Progress*, *Buffer Valid*]. From the perspective of video playback, a YouTube session can contain three exclusive regions: *buffering*, *playing*, and *paused*. YouTube IFrame API considers a transition from one playback region into another as an event. It also considers as an event any call to the API to collect data. The API enables the recording of an event and of detailed information about playback progress at the time the event

occurs. *Epoch Time* marks the time of the most recent collection of YouTube API data in that interval. *Playback Info* records events occurred during the 100 ms interval.

Each field of *Playback Info* is defined as follows:

- **Playback Event -** This field is a binary array with four indexes for the following states: 'buffering', 'paused', 'playing', and 'collect data'. The 'collect data' event occurs every 100 ms once the video starts playing. For example, an interval with a Playback Event [1,0,0,1] indicates that playback region has transitioned into 'buffering' during the 100 ms interval and a 'collect data' event occurred.

- **Epoch Time -** This field is the UNIX epoch time in milliseconds of the most recent YouTube API event in the 100 ms interval.

- **Start Time -** This field is the UNIX epoch time in milliseconds of the beginning of the experiment.

- **Playback Progress -** This field reports the number of seconds the playback is at epoch time from the start of the video playback.

- **Video Length -** This field reports the length of the entire video asset (in seconds).

- **Playback Quality -** This field is a binary array of size 9 with indices for the following states: unlabelled, tiny (144p), small (240p), medium (360p), large (480p), hd720, hd1080, hd1440, and hd2160. The unlabeled state occurs when the video is starting up, buffering, or paused. For example, a Playback Quality [0, 1, 1, 0, 0, 0, 0, 0, 0] indicates that during the current interval, video playback experienced two quality levels - tiny and small.

- **Buffer Health -** This field is defined as the amount of buffer in seconds ahead of current video playback. It is calculated as:

$$\text{Buffer Health} = \text{Buffer Progress} \times \text{Video Length} - \text{Playback Progress}$$

- **Buffer Progress -** This field reports the fraction of video asset that has been downloaded into the buffer.
- **Buffer Valid -** This field has two possible values: True or '-1'. True represents when data is being collected from the YouTube IFrame API. '-1' indicates when data is not being collected from the YouTube IFrame API during the current interval.

Chapter 3: Inferring Live Streaming User Experience of YouTube TV from Encrypted Traffic

In this chapter, we extend the work from Chapter 2 and present a study on YouTube TV live streaming traffic behavior over WiFi and cellular networks.

3.1 Introduction

As cord cutting has become a real trend, with twice as many subscribers abandoning their traditional TV subscriptions in 2018 versus the previous year [62], there has been an increase in live TV streaming services such as AT&T TV Now, FuboTV, Hulu, Philo, SlingTV, and YouTube TV over broadband and mobile networks [197]. By 2021, video streaming traffic is expected to grow to 82% of all IP traffic [41], while live video usage will account for 13% of global Internet traffic [30].

Popular *on-demand* streaming services have been steadily improving their ability to provide good user experience [221, 28]. In addition to network capacity improvements, a major reason that these services can provide reasonable QoE is their ability to avoid rebuffering events, due to their design decision to store tens (and even hundreds) of seconds of video in the client device buffer. Compared to on-demand services, *live streaming* services must provide much lower latency, which results in a design with a reduced client *buffer capacity*, defined as the maximum duration of the video content the buffer can hold. The question that begs to be answered is whether such design impacts the QoE of live TV streaming.

In order to gain better understanding of live streaming services, in this chapter we focus on analyzing the YouTube TV service behavior and inferring a variety of QoE metrics using machine learning (ML) techniques. YouTube TV is a television service that offers live TV for more than 70

US television networks and is gaining popularity [234]. Specifically, this chapter presents a traffic measurement study of YouTube TV service over mobile networks.

To perform the analysis, we develop a system to collect both QoE metrics as well as network level data. Our data collection period spans a 9-month period leading up to Feb. 2020. We collect ground truth playback quality metrics directly from the YouTube TV application on an Android mobile device over WiFi networks and LTE cellular networks. We extract the quality metrics from the *Stat for Nerds* window during video playback. From this window, we monitor the current video ID, video format, audio format, and buffer health. We capture packet data using *tcpdump* on the mobile device.

Our contributions are as follows: (i) Design a system to collect network traffic and client playback metrics (buffer health and video resolution). (ii) Analyze application performance of YouTube TV. We discuss how the streaming algorithm differs from on-demand video services in terms of transmission of chunks, and the design of the client buffer. We also provide insights into YouTube TV's ad replacements during commercials. (iii) Develop a multi-chunk detection (MCD) algorithm to detect multiple video and audio chunks with concurrent transmission *in the same IP flow*. We define a multi-chunk unit (MCU) as a group of overlapping chunks. (iv) Develop ML tools to infer user QoE metrics such as video resolution and buffer health from features based on MCUs extracted from IP traffic. *To the best of our knowledge, this is the first video QoE study to provide insight into the performance of a live TV streaming service over mobile networks.*

For further background on streaming algorithms and prior work on QoE inference see Sections 2.3 and 2.2 in Chapter 2, respectively. The rest of the chapter is organized as follows. Section 3.2 presents our data collection methods and measurement results to demonstrate the unique characteristics of YouTube TV traffic profiles. Section 3.3 describes our MCD algorithm and demonstrates the behavior of MCUs in our dataset. Section 3.4 presents the ML models to predict video resolution for each MCU and predict playback phase of the client buffer every 2 sec.

This research started in collaboration with Dr. Katherine Guo for Bell Labs. Trey Gilliland made important contributions to the data collection.

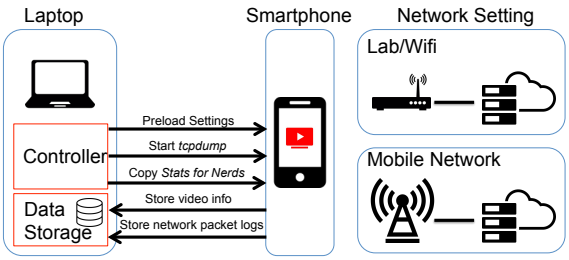


Figure 3.1: Experimental setup for our trace collection.

3.2 Measurements

In this section we describe the setup and trace collection for our YouTube TV dataset. We then describe the profile from a representative video session followed by insights that we obtained.

3.2.1 Trace Collection from YouTube Android App

We design and implement a data acquisition environment, as shown in Fig. 3.1, to capture YouTube TV video playback statistics and encrypted network packet data on an Android device. This setup contains a laptop running Ubuntu 18.04 connected via USB to a rooted Motorola Moto G6 smartphone. The phone is either connected to a WiFi Access Point (AP) or to Google Fi's LTE cellular network.

For data collection, we first run a shell script on the laptop autonomously setting up the test environment using Android Debugging Bridge (ADB). The native YouTube TV application is running on the phone. The script on the laptop uses ADB to trigger events on the phone. The initial step includes setting up the test environment by connecting to the correct network, loading a dummy startup video, and starting the *Stats for Nerds* window. This window allows us to easily monitor the audio and video codecs, current resolution, current video ID, and *buffer health*. Buffer health is defined as the time duration of the video content stored in the buffer, also known as *buffer length*. The shell script on the laptop, then initializes *tcpdump* on the phone to collect all network traffic. The network packet logs record packet timestamp, packet size, as well as the 5-tuple for IP-header (source IP, destination IP, source port, destination port, and protocol).

We then load a video for each experiment by joining a YouTube TV channel. Once the video

Table 3.1: Clip distribution in our dataset.

Channel	WiFi Experiments	Cellular Experiments
CBS	17	21
TNT	19	24
NBA TV	19	23
Food Network	17	26
BBC America	18	32
AMC	19	29
Animal Planet	18	26
BBC World	18	26
Cartoon Network	21	31
CBS Sports	19	30
CNN	17	32

is loaded, the information in the *Stats for Nerds* window is copied every 1 sec to a clipboard log using ClipStack (*Stats for Nerds* updates every 1 sec). Each experiment lasts 10 mins. After the experiment, the information in Clipstack and network packet logs, obtained via *tcpdump*, are exported from the phone to the laptop. The data is then merged into one file and uploaded to Google Cloud.

3.2.2 Typical Profile and Insights

We collect data from 11 TV channels, as seen in Table 7.1. Even though the number of channels is relatively small, since the channels carry live TV content, it is unlikely that any of the experiments contain the exact same video content. This results in the dataset having a large variation in video content. We collected this dataset over 9 months from June 2019 through Feb. 2020. The data consists of resolutions ranging from 144p to 1080p.

The WiFi dataset consists of **202** experiments. To create various network conditions, we conduct tests in the lab environment by varying the distance of the client device from the AP. The distance is up to 80 feet and multiple rooms away. The cellular dataset consists of **300** experiments. We conduct tests in various settings aiming for the client device to experience various network conditions. For example, we collect data when the device is in a car driving on the highway, in a

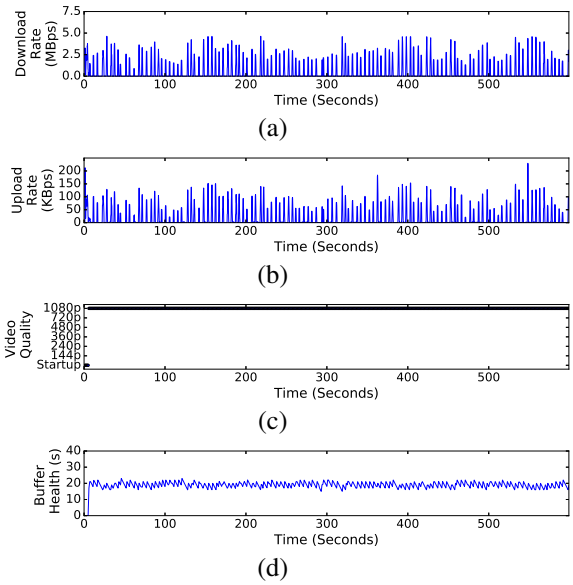


Figure 3.2: Behavior of a 10 min session: (a) Average download bitrate (MBps), (b) Average upload bitrate (KBps), (c) Video resolution, (d) Buffer health.

backpack with the owner walking on the streets, and during lectures.

For our WiFi dataset, all of the video streams are over IPv4. In addition, 96% of the data is carried over QUIC/UDP while a majority of the rest of the traffic is over TCP+TLSv1.2. For our cellular dataset, 80% is over IPv4, with the remaining over IPv6. To break it down further, 76% is QUIC/UDP with IPv4, 19% is TCP+TLSv1.2 with IPv6, and 4% is TCP+TLSv1.2 with IPv4. The IP protocol used depends on the underlying cellular service that Google Fi was using and the region of the country.

Upon close examination of the data, another interesting finding is that YouTube TV replaces some advertisements during commercial breaks with their own advertisements. We are able to identify this due to a change in Video ID. Ad replacement session times are in multiples of 15 sec, with a majority of the session time for replacement ads being 60 sec. The individual ads that YouTube inserts are 15 sec in length. In addition, YouTube inserts Slates, animated YouTube logos with the channel information, when it does not have an ad to fill the gap before starting playback of the main video.

Fig. 3.2 shows the trace from an example YouTube TV stream in our dataset collected over a 10 min period. Fig. 3.2(a) and Fig. 3.2(b) report the total downstream bitrate (MBps) and upstream

Table 3.2: Percent of Data in each Resolution

Resolution	WiFi	Cellular
Start Up	1	1
144p	23	14
240p	22	17
360p	9	15
480p	14	26
720p	12	9
1080p	19	18

bitrate (KBps), respectively. It is evident that traffic follows a periodic pattern, where short bursts of traffic are separated by periods of quiet time over the network.

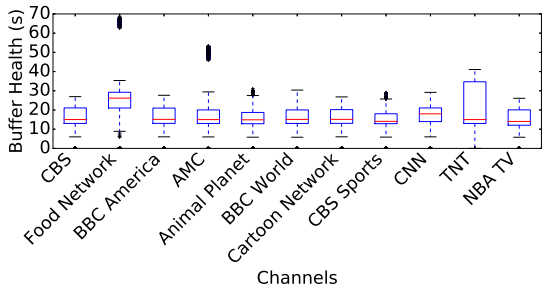
Fig. 3.2(c) and Fig. 3.2(d) show the current quality of the video frames being played and the buffer health (in seconds) respectively. Notice that buffer health increases quickly from the startup phase to over 20 sec in length when the video playback starts.

We define *buffer increase* phase as the video playback phase when buffer health is increasing faster than it is draining. For this example, this phase starts right after the client device sends the request to join the TV channel and lasts until 1 sec after playback information is collected. Once the buffer health enters a steady region determined by the ABR algorithm, it stays in a *steady* state with a buffer health between 17 and 24 sec, due to periodic draining and refilling of the buffer. Based on the proposed heuristic algorithm in Requet [78], we automatically label the playback state according to the dynamics of buffer health using four exhaustive labels: *Increase*, *Steady*, *Decay*, and *Stall*.

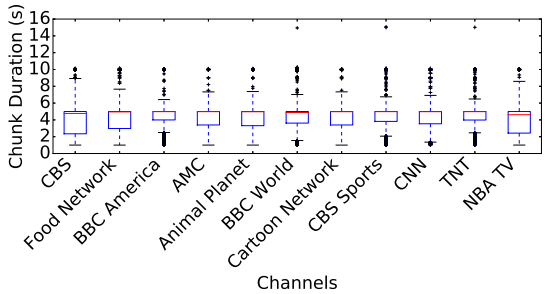
In addition to this example session, we demonstrate the typical performance of video sessions in our dataset. We focus on two important design choices for HAS: (i) buffer health during steady state and (ii) chunk duration.

Fig. 3.3(a) shows that channels in our WiFi dataset consistently have a median steady state buffer health of 15 sec. Fig. 3.4(a) shows that the median buffer health is approximately 20 sec in our cellular dataset.

In terms of buffer health during steady state, most channels consistently hover around 15 and

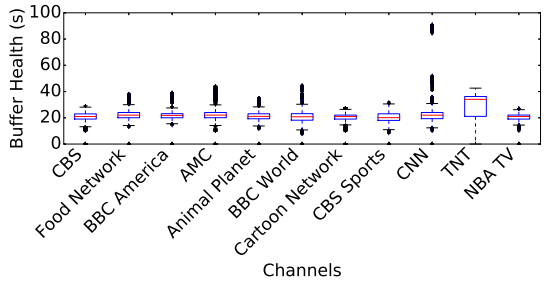


(a)

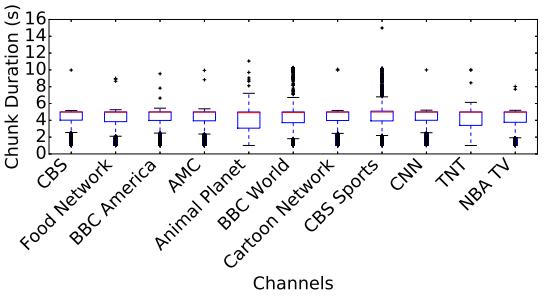


(b)

Figure 3.3: WiFi Dataset: (a) Buffer health in steady state for each channel, (b) Chunk duration in steady state for each channel.



(a)



(b)

Figure 3.4: Cellular Dataset: (a) Buffer health in steady state for each channel, (b) Chunk duration in steady state for each channel.

20 sec over WiFi and cellular networks, respectively. The outlier for the dataset is TNT which has a larger range for steady state buffer health. In comparison, the average steady state buffer health of Netflix is approximately 240 sec [124], while for YouTube on-demand streaming service, the

average buffer health during steady state can go up to 180 sec [78]. The observed buffer health for YouTube TV is different than a majority of other streaming services as YouTube TV aims for a low buffer health in an effort to maintain low latency for live streaming content.

The second important design decision for HAS is chunk duration, which is the level the buffer increases upon a chunk arrival. To calculate this, we record the amount of buffer health increase after the arrival of each chunk in the *Stats for Nerds* window, during the steady state phase. This level is relatively stable for all channels and is usually around 5 sec of video for both the WiFi (Fig. 3.3(b)) and cellular (Fig. 3.4(b)) datasets. For YouTube on-demand service, during steady state, the average chunk duration for video chunks stays between $30 - 40$ sec for 144p to 360p videos, and decreases from 40 to 5 sec when video resolution increases from 360p to 1080p. For audio chunks it is approximately 30 sec [78]. For Netflix, it has been shown to be 4 sec for video chunks, and 16 sec for audio chunks [124].

A majority of the data collected is in steady state (roughly 78% and 87% of the time for the WiFi and cellular datasets, respectively). The median time of the first data point collected after the start of a session for buffer health is 15 and 20 sec for the WiFi and cellular dataset, respectively. Due to the fact that we only collect buffer health information after the start of a session, this is an indication that once the session starts, it can reach steady state buffer health quickly and stay in steady state during most of the session.

To confirm that these observations are accurate, we investigate YouTube TV video streams on a Chrome browser on a laptop using the developer tools. Fig. 3.5 displays an example of two subsequent chunk requests along with the waterfall to show the timing of the requests. We use time to first byte (TTFB) to denote the time between the GET request and the first packet of the response. An important point to note is that HTTP requests for both chunks are pipelined and are sent at almost the same time. The first chunk has a size of 1.9 MB, a TTFB of 5 msec, and a total time of TTFB and content download of 129 msec. The second chunk has a size of 81.1 KB, a TTFB of 8 msec, and a total time of TTFB and content download of 27 msec. Information in the requested URLs show that the first chunk is a 5 sec video chunk, and the second chunk is a 5 sec

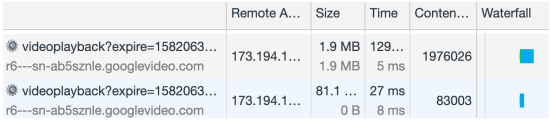


Figure 3.5: Chunks shown on Chrome browser developer tools.

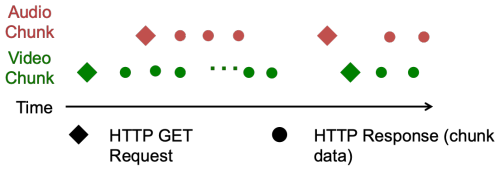


Figure 3.6: Overlapping audio and video chunks.

audio chunk. It is clear that the transmission of these two chunks overlap in time.

In practice video and audio chunks should not overlap as it may cause head-of-line blocking. Prior work based on chunk detection for QoE inference all rely on detection of chunk boundaries to derive accurate chunk size and chunk inter-arrival time [128, 109, 126, 127, 124, 78, 28]. This newly observed behavior of YouTube TV using overlapping chunks will reduce the accuracy of proposed chunk detection algorithms. Therefore, to handle this phenomenon we propose a new chunk detection algorithm in Section 3.3.1.

3.3 Methodology

3.3.1 Multi-Chunk Detection (MCD)

The fundamental delivery unit of ABR is a chunk [114]. Therefore, identifying chunks instead of relying on individual packet data can capture important player events. Specifically, the occurrence of a chunk indicates that the client has received a complete segment of video or audio, resulting in increased buffer level in playback time. Therefore, for any video QoE inference algorithm it is important to extract these chunks.

The delivery of a chunk is described with the following protocol. An HTTP GET request is sent from the client to the server. The GET request contains larger than a 300 byte payload, while smaller uplink packets are acknowledgments or HTTP POST requests. After the request is sent, the server sends an HTTP response to the client. The HTTP response are packets with payload

containing data of the chunk. The chunk size is the total amount of data received for this chunk until the next request is sent.

All existing mechanisms for chunk identification [109, 127, 124, 78] rely on the assumption that HTTP requests are not pipelined even though pipelining is feasible with multiple concurrent streams within a QUIC or HTTP/2 connection [82, 25]. In practice, video clients do not pipeline HTTP video requests as this potentially causes contention for bandwidth among video chunks, and causes head-of-line blocking [127].

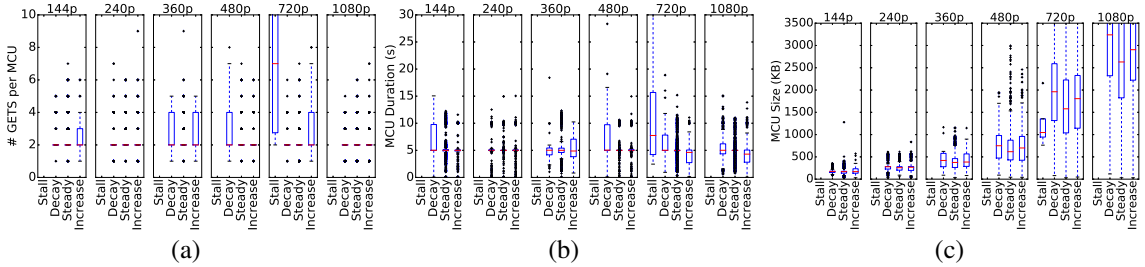


Figure 3.7: Multi-chunk metrics for MCU in the WiFi dataset: (a) Number of chunks per MCU, (b) MCU duration, (c) MCU size.

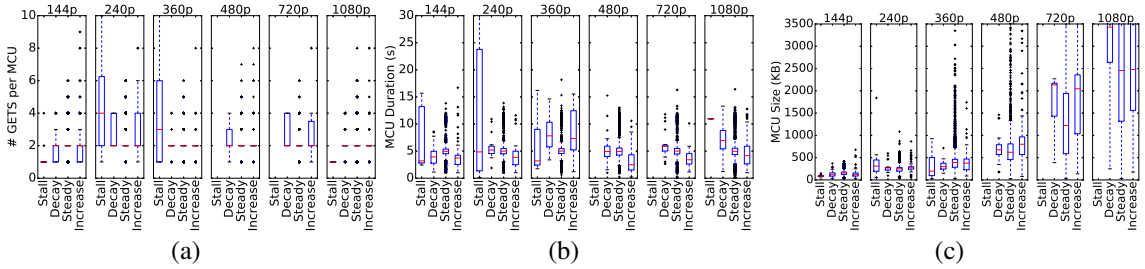


Figure 3.8: Multi-chunk metrics for MCU in the cellular dataset: (a) Number of chunks per MCU, (b) MCU duration, (c) MCU size.

However, as audio chunk size is typically significantly smaller than video chunk size for a majority of the video resolution [78], the change of head-of-line blocking for overlapping audio and video chunks is significantly smaller than overlapping video chunks. When the effect of head-of-line blocking is not significant, pipelined requests can potentially reduce latency of chunks and this is beneficial for live streaming services.

When two chunk transmission processes do not overlap in time, it is easy to see the boundary between the chunks and calculate individual chunk sizes. The problem shown in Fig. 3.5 is that for YouTube TV, the chunk requests are pipelined. The overlap of the audio chunks and the video

chunks demonstrated in Fig. 3.6 reveal that it is difficult to identify the TTFB, download time of the chunk, and the chunk size of the two pipelined chunks. For pipelined HTTP requests it is nontrivial to determine how many chunks are overlapping and the size of each of these chunks.

We propose a new algorithm *Multi-Chunk Detection* (MCD) to detect the boundary for a group of chunks and report statistics for this group. We name the group of chunks a *multi-chunk unit (MCU)*.

The algorithm operates on each IP flow. When a packet is detected in the upstream direction from a client to a server with a packet size of greater than 300 bytes, it is marked as a potential chunk request. For this MCU of this IP flow the algorithm keeps track of the number of GET requests within the next second. As soon as no GET requests are detected within a 1 sec time window, no additional GET requests are added to this MCU. The next MCU starts when a new GET request appears more than 1 sec after the previous GET request. The end of the previous MCU is marked when the next GET request appears indicating the start of the current MCU. In addition, the algorithm keeps track of all the downstream packets. The size of the download for the MCU is calculated by summing up the payload size of all the downstream packets. The time for the first GET request for the MCU is recorded, along with the first and last downstream packet of this MCU.

If the MCU is less than 30 KB in download size, it is assumed that there is no chunk information, and the MCU is removed from the flow. This approach operates at the IP level, and it therefore is applicable for both encrypted TCP and QUIC/UDP flows.

3.3.2 Multi-Chunk Unit (MCU) Behavior

We apply the MCD algorithm to all the sessions in our dataset for both WiFi and cellular networks. We investigate the characteristics of MCD to determine patterns to guide machine learning feature design for classifying the state and video resolution during the playback of a live video stream.

For both the WiFi and cellular datasets we examine the MCU related metrics under different

buffer states and video resolutions. We show the statistics of the number of chunks in each MCU, the size of each MCU, and the MCU duration (time between each MCU) in the WiFi (Fig. 3.7) and cellular (Fig. 3.8) datasets.

The first set of figures display the median number of chunks in an MCU. This median number over WiFi (Fig. 3.7(a)) and cellular (Fig. 3.8(a)) are highly centered around the median of 2 chunks. This is because the audio and video chunks usually occur together in one MCU. This indicates that for YouTube TV service, roughly one audio chunk corresponds to one video chunk, and the chunk duration should therefore be similar (5 sec for both video and audio respectively in Fig. 3.5).

The second set of figures display the MCU duration over WiFi (Fig. 3.7(b)) and cellular (Fig. 3.8(b)). During steady state the MCU are also tightly around the median of 5 sec. This is consistent with the data noted from analyzing the *Stats for Nerds* buffer health in Figs. 3.3(b), 3.4(b) and by using the Chrome developer tools (Fig. 3.5). The MCU duration decreases across all video resolutions during the buffer increase phase. This is due to the fact that during the buffer increase phase chunks are requested as fast as the network capacity allows for the client buffer to fill up.

The third set of figures show the size of MCU over WiFi (Fig. 3.7(c)) and cellular (Fig. 3.8(c)). As video resolution increases from 144p to 1080p the MCU size increases accordingly. There is a wide range for each MCU size based on the encoding of each video scene.

3.4 Evaluation

We propose solving the prediction of video resolution and video phase as a classification problem. Features based on the network level data are used as input to the system. The classes for prediction are the resolution levels available for YouTube TV: 144p, 240p, 360p, 480p, 720p, and 1080p. The classes for the prediction of the video playback phase are *steady state* or *buffering* defined to include buffer increase, decay, and stall. The three states are combined into one phase as the player is in need of data as soon as possible during all these states. We begin by describing the ML system designed for this task. We continue with the evaluation of this system using both

Table 3.3: Video resolution performance (%)

	WiFi		Cell		Both	
	Prec.	**Rec.**	**Prec.**	**Rec.**	**Prec.**	**Rec.**
144p	93	97	90	93	91	97
240p	91	93	81	88	87	89
360p	89	75	80	69	82	71
480p	80	87	80	85	80	85
720p	77	70	58	51	68	61
1080p	87	86	85	87	86	86
Accuracy	**87**		**81**		**84**	

Table 3.4: Video phase performance (%)

	WiFi		Cell		Both	
	Prec.	**Rec.**	**Prec.**	**Rec.**	**Prec.**	**Rec.**
Buffering	60	43	72	55	66	46
Steady	86	92	96	98	92	96
Accuracy	**82**		**95**		**89**	

the WiFi and cellular datasets.

3.4.1 Machine Learning Models

We develop ML models using Random Forest (RF) to infer metrics reflecting user QoE. We build the RF classifiers in Python using the sklearn package. We base the feature selection on previous works that infer QoE metrics using online models with time scale smaller than the length of a video session [182, 222, 124, 78, 133, 28]. Specifically for each prediction we compute a set of attributes based on the MCU statistics for windows of the past 10, 20, 30, 40, and 50 sec. The first attribute per window is the number of MCUs. The second attribute is the average number of GET requests per MCU. The third attribute is the average download time for each MCU. The fourth attribute is the average download size (in bytes) per MCU. The fifth attribute is the standard deviation of MCU size. The sixth attribute is the standard deviation of the inter MCU arrival time. Each one of these 6 attributes is collected for the 5 windows to create a total of 30 features.

Prediction of video playback phase is done every 2 sec, while prediction of resolution is done only after every MCU arrives. It is not beneficial to do resolution prediction more often because

resolution can only change at the boundary of video chunks. In addition to the 30 previously discussed features, for predicting resolution, we add in an additional 6 features per MCU. These 6 features include the first GET request size, the download size, the protocol used, the number of GET requests, the time between the first GET request of the MCU and first download packet of that MCU (TTFB), and the time between the first download packet of the MCU and the last packet of the MCU.

3.4.2 QoE Inference Accuracy

For classification, we divide each dataset into 5 subsets based on TV channels. From these 5 subsets we use 5-fold cross validation to determine the performance of the model on prediction of video resolution and playback phase.

Video Resolution Prediction: Table 3.3 reports the performance (precision, recall, accuracy) of resolution prediction. The prediction achieves an overall accuracy of 87%, 81%, 84%, for training on the WiFi dataset, cellular dataset, and combined datsets, respectively. For the combined dataset 86% of the errors are 1 resolution away from the actual resolution. Therefore, the classifier is able to predict the actual resolution to a 1 resolution error 98% of the time.

The most important features for the RF model for resolution prediction are the download size of the most recent MCU, and the average download size of the MCUs in each window. The next most important attribute is the standard deviation of the MCU size. The least important feature is the average number of combined chunks in each window. This is due to majority of MCUs having 2 chunks per MCU independent of video resolution.

Playback Phase Prediction: Table 3.4 shows the performance of playback phase prediction (precision, recall, accuracy). The prediction achieves an overall accuracy of 82%, 95%, 89%, for training on the WiFi dataset, cellular dataset, and combined datsets, respectively.

The most important features for the RF model for phase prediction are the average download time of each MCU in each window. The next most important features are the number of MCUs and the standard deviation of the interarrival time of the MCUs in the past 50 sec.

Part II

End-to-End Resource Allocation in Cellular Networks

Chapter 4: RAN Resource Usage Prediction for a 5G Slice Broker

Part II focuses on using recurrent neural networks, a type of deep neural network, to improve resource allocation in cellular networks. In this chapter we focus on supporting the operation of the RAN slice broker, which maps slice requirements into the allocation of Physical Resource Blocks (PRBs) in a 5G network.

4.1 Introduction

It is expected that 5G networks will support a variety of services including smart cities, autonomous and network assisted driving, augmented reality, and virtual reality. Such services will impose an extremely diverse set of requirements on the mobile network, ranging from ultra high throughput to ultra low latency at the order of milliseconds [90].

Network slicing will allow 5G operators to split a shared physical infrastructure into virtual slices to meet these diverse requirements (see Fig. 4.1). The Next Generation Mobile Network (NGMN) Alliance defines a network slice as a set of network functions and associated resources, forming a complete virtualized end-to-end logical network meeting certain network characteristics required by the associated service [51, 2, 110]. Namely, slices will provide virtualized resource separation for different services, while still allowing for statistical multiplexing of the resources.

An anticipated challenge is managing a large number of tenants, each with multiple services, resulting in separate slice instances. Each such instance may have unique Service Level Agreement (SLA) requirements in terms of bandwidth, latency, reliability, mobility, and security. Each slice can contain multiple bearers from multiple User Equipments (UEs)[1], and each bearer can have a Guaranteed Bit Rate (GBR)[2] or a non-GBR service.

[1] A bearer is defined as a path for the traffic with a common QoS from a UE to the Packet Data Network Gateway.

[2] GBR bearers have bandwidth guarantees (e.g., min throughput, max throughput, packet delay variation) from the LTE network.

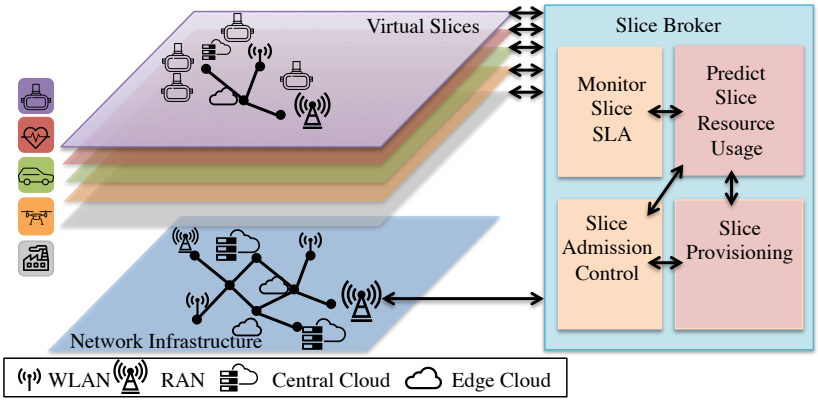


Figure 4.1: 5G network slice architecture: the network infrastructure is divided into slices for tenants. The RAN broker monitors each slice's SLA. The broker then predicts future slice resource usage. Slice provisioning is done based on the SLA and the predicted resource usage. The slice prediction and provisioning information is used by the slice broker for admission control decisions.

The complexity of management and orchestration will increase with 5G slicing. As illustrated in Fig. 4.1, a Radio Access Network (RAN) slice broker (to which we will refer to as a **broker**) is used to manage and orchestrate the slice life cycle [174]. The broker monitors each slice's SLA and predicts its future RAN resource usage. The prediction is utilized to dynamically provision resources to slices. Admission control decisions are based on the slice priority and resource requirements. For example, in Fig. 4.2(a) during the first two decision intervals four and five slices are admitted, respectively. Slice admission control algorithms that take into account given RAN usage have been developed based on solutions to the multidimensional knapsack problem [34, 179].

Efficiently utilizing the RAN resources is crucial to enable multiplexing gains and cost effectiveness for service providers [132]. Therefore, an accurate prediction model of the usage must be developed to efficiently utilize the RAN. An overestimation in the amount of allocated resources results in a decrease in revenue for the service provider, while underestimation results in SLA violations.

Specifically, this chapter has two related objectives. The first is *to develop a metric that measures the amount of Physical Resource Blocks (PRBs) available to very active bearers for each slice*. The metric could be easily used for slice provisioning. The second objective is *to develop an*

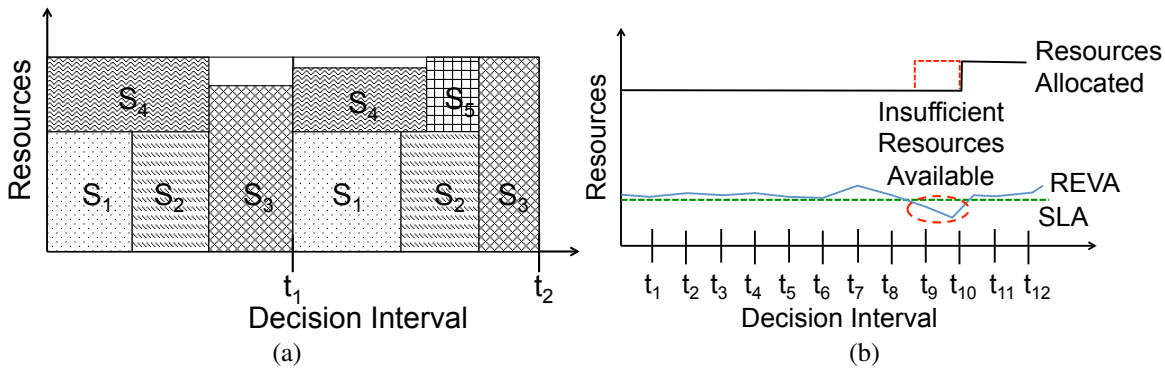


Figure 4.2: (a) An example of provisioning resources to slices which is based on the broker's admission control decisions, where in the second decision interval a 5th slice is admitted. (b) An example of monitoring REVA for a single slice and the corresponding dynamic resource provisioning.

accurate and short time scale prediction model of that metric. Once the prediction is obtained, the broker can use each slice's predicted PRB usage for slice provisioning.[3]

Accordingly we define a new metric, REVA, that precisely measures the average amount of wireless Physical Resource Blocks that the RAN scheduler can allocate to Very Active bearers (see Section 4.4). The amount of resources given to each bearer in a slice is determined by the RAN scheduler. The Very Active (VA) bearers are those that attempt to obtain more than their fair share of the PRBs that are available from the scheduler. A broker determines the amount of PRBs to allocate to each slice in order to satisfy the SLA (e.g., a slice's SLA may require a minimum video quality for a remote surveillance camera). Since the RAN scheduler reserves PRBs for each GBR bearer (e.g., voice conversation), the REVA metric focuses on measuring the PRB usage of the non-GBR bearers in a slice. For example, in Fig. 4.2(b) when REVA falls below the SLA in decision interval t_9, it reveals that there are insufficient available PRBs for VA bearers. The slice provisioning algorithm would use this information to increase the total amount of PRBs allocated to that slice.

Due to the lack of relevant data and while 5G systems are still undergoing standardization and development, we design and evaluate a model for short term (single to tens of seconds) REVA prediction **using a custom-designed experimental LTE testbed** (see Section 4.5). The scheduler

[3]Slice admission control is out of the scope of this chapter and is for future work.

in the testbed is augmented with a thin layer to compute REVA in real time[4] and we use a single Quality of Service (QoS) class identifier (QCI) per slice. The testbed was used to collect traces of hundreds of hours of RAN resource allocation under a variety of network usage patterns. We used one, two, and three overlapping periodic time patterns to emulate the temporal patterns that occur in cellular networks [215].

The data collected from the testbed is used to develop and evaluate the prediction models for REVA (see Section 4.6). Current time series models [27, 185, 88] are inadequate for multistep prediction of network resource usage over a short time scale. These models are designed for predicting one step into the future, but there is a need for higher accuracy over multiple time steps for dynamic provisioning and other network optimization techniques (e.g., VM migration). Therefore, we design a modified Long Short Term Memory (LSTM) model, X-LSTM, to improve prediction accuracy. To evaluate the performance (see Section 4.7), we use X-LSTM to predict REVA tens of seconds in advance. We show that the gains of X-LSTM over traditional models such as Autoregressive Integrated Moving Average Model (ARIMA) and LSTM neural networks increase as the number of components in the time pattern increases. Given time patterns composed of one, two, and three independent semi-periodic components, X-LSTM outperformed ARIMA and LSTM by 10%, 22%, and 31% respectively. We show that X-LSTM achieves accuracy predictions of over 91%.

To evaluate the impact of each prediction model on slice provisioning, we introduce a simple slice provisioning algorithm. The algorithm exploits the prediction models to minimize cost for service providers. The cost is measured by the amount of over-provisioned PRBs and violating the SLA. We show that X-LSTM offers the service provider a greater than 10% cost reduction compared to ARIMA and LSTM.

This work is based on collaboration with Nokia Bell Labs and has appeared previously in the proceedings of ACM Mobihoc'19 [76].

[4]We expect that the additions to the scheduler and the REVA metric will be applicable to 5G schedulers.

4.2 Related Work

Network Slicing: Architectural aspects of 5G RAN slicing are developed by the 5G NORMA project [6] within 5G-PPP, by utilizing Software Defined Networks and Network Function Virtualization. Wireless RAN virtualization will offer greater flexibility for network infrastructure operators, while also adding benefits to their customers (typically called tenants [172]). By enabling RAN virtualization, Mobile Network Operators can share common RAN resources leading to reduced costs and increased energy efficiency. The concept of network virtualization will enable infrastructure as a service for end to end networking [51]. An optimization framework was developed in [116] for resource allocation of network bandwidth and cloud processing. A network slice broker will enable mobile operators to request and lease infrastructure dynamically [174]. Orion was developed to enable dynamic virtualization of the base station [63]. In [179, 34], the authors develop an admission control decision algorithm for RAN slice requests based on the knapsack problem and propose solutions using a greedy algorithm and online-reinforcement learning. The resource efficiency and cost-effectiveness of resource management in network slicing is studied in [132]. Slice overbooking has been shown to maximize the revenue of mobile operators with minimal impact of SLAs [173].

Cellular Traffic: Recent work has characterized and modeled city wide traffic in cellular networks [237, 215, 216, 220, 238], where congestion is characterized by measuring the traffic load. In [85], the authors improve on previous congestion metrics by also including round trip times. In [112] a single cell load measure is defined as a combination of the number of connected bearers, achieved throughput, and percentage of total PRBs per bearer. In [194] congestion is characterized via skewness of measured aggregate throughput.

Network Analytics: Numerous research efforts have been devoted to improving network performance using network analytics. In [9, 187] traffic congestion and mobility is predicted across a university WLAN network. Improvements for adaptive video streaming performance over LTE are studied in [228, 229, 241] where the LTE bandwidth is estimated by monitoring the broadcast

messages or LTE bandwidth availability is given.

Time Series Modeling: Statistical and machine learning models for forecasting time series have received significant attention. Common statistical models are ARIMA and the Seasonal ARIMA (SARIMA) models [27, 185]. Neural Networks (NNs), a popular model in machine learning, are used to approximate non linear multivariate functions. Recurrent Neural Networks (RNNs) is a NN that uses feedback from previous steps. A specific type of RNN is an LSTM NN, which has memory cells to maintain information for longer periods [88]. Additional work has been done to improve the performance of LSTMs for specific applications [65, 147, 40].

4.3 Wireless RAN

In this section, we provide background on RAN resource allocation and then discuss the limitations of the existing metrics available for monitoring resource utilization. Note that we use LTE terminology while describing and evaluating our methods, but they should be applicable to 5G schedulers which are currently in development.

4.3.1 Background on RAN Resource Allocation

3GPP defines wireless resource allocation in the time and frequency domains. LTE and LTE Advanced utilize a resource allotment unit called a PRB. A PRB consists of 180 kHz in the frequency domain and one slot of 0.5 ms in the time domain. Every Transmission Time Interval (TTI) (1 ms in LTE), the scheduler distributes the available PRBs for the Downlink (DL) and Uplink (UL) among LTE bearers. The total number of PRBs assigned depends on the number of TTIs and the system bandwidth configuration. According to the LTE standard, there are 6 PRBs per TTI for 1.4 MHz configuration to 100 PRBs for 20 MHz configuration [2].

The scheduler at the base station (eNodeB) uses channel condition information received periodically from the UEs to assign Modulation and Coding Schema (MCS) to the allocated PRBs. This essentially determines the number of bits transmitted using the allocated PRB. Using a higher MCS with poor channel conditions leads to data loss and requires using more PRBs for retransmis-

sions. In good channel conditions, using a lower MCS leads to unnecessarily reduced throughput. RAN scheduling algorithms are optimized to determine the best MCS assignment for each allocated PRB. In the time domain, the schedulers also make decisions regarding how often PRBs are assigned to a specific bearer.

The LTE standard defines scheduling priorities or QCIs to address the different scheduling rules for the different classes of service [3]. Each QCI has its own associated QoS characteristics (e.g., priority, guaranteed (or not) bit rate, packet delay budget, and packet error loss rate. QCIs reserved for GBR service have the scheduler attempt to ensure certain guaranteed bitrate for the bearer. For example, QCI 1 is typically reserved for Voice over IP traffic. QCIs designed for non-GBR traffic typically use weighted or max-min fair share scheduling algorithms for PRB allocation [3]. Max-min fair share algorithms assign users with a small demand the resources that they need and distribute the remaining resources evenly to large users.

4.3.2 RAN Resource Utilization Metrics

The broker allocates PRBs based on the SLA of each slice. In order to get better insight into provisioning resources, it is important to understand how bearers in that slice are currently utilizing the resources. In addition, it would be insightful to separate PRB usage and user channel conditions. Each have an essential role in the throughput and latency of individual UEs. RAN usage has been studied by several previous research efforts [112, 237, 215, 216, 85, 194]. However, they specifically do not focus on the application of RAN slicing. Below are examples of metrics that are not adequate for a broker.

- **Aggregate percent of available PRB utilization per second by the scheduler [112] -** There is no sense of fairness and relation to per bearer SLA. A single greedy application such as FTP can utilize close to 100% of all PRBs in the LTE RAN, if there are no other bearers served by the RAN. Clearly, the RAN serving just a single client is not congested, and if a second FTP client joins, the scheduler would allocate to it roughly half of the available PRBs.

- **Aggregate throughput of all bearers [237, 215, 216] -** The metric is inadequate for the same reasons given above. A single FTP bearer in perfect channel conditions can utilize close to 100% of all PRBs.

- **Metrics based upon latency or throughput of individual or groups of bearers [85, 194] -** These metrics are inadequate due to the reasons below:

  - Bearer throughput depends on channel conditions. Low throughput or high latency of bearer(s) may not result from RAN congestion but could result from poor channel conditions of the respective bearer(s).

  - Low throughput may be a function of applications usage characteristics. Specific applications may not need a lot of network resources (e.g., Voice over IP, low resolution video, and instant messaging).

- **Number of users served by the RAN -** Such a metric does not take into account RAN resource consumption by individual bearers. A RAN serving a large number of VoIP or other low volume bearers is not necessarily congested.

4.4 RAN Resource Estimation

In this section we outline the design objective for a prediction model along with the REVA metric. We then describe the definitions needed for REVA, and the algorithm for its computation.

4.4.1 Objective

Our objective is to develop a metric, REVA, that can be used by a broker to efficiently measure, predict usage, and provision slice resources. We represent REVA as y_t throughout the remainder of the chapter. We assume the broker has a history of T decision intervals of the series: $\mathbf{y_{t-1}} = (y_{t-1}, y_{t-2}, ..., y_{t-T})$. A prediction model f uses the history $\mathbf{y_{t-1}}$ to predict s decision intervals ahead: $\hat{y_t}, \hat{y_{t+1}}, ..., \hat{y_{t+s-1}} = f(\mathbf{y_{t-1}})$. The goal is to develop a prediction model for y_t that has a minimal

prediction error ϵ_t:

$$y_t = f(\mathbf{y_{t-1}}) + \epsilon_t. \tag{4.1}$$

The REVA metric is developed as follows:

- A function of the available resources that is independent of: (i) channel conditions of the bearers; (ii) the application behavior and throughput needs of individual user bearers; (iii) transport protocol (e.g., TCP, QUIC, UDP, or raw IP); (iv) bearer throughput or roundtrip time. This would allow for scheduling slices based on the required PRBs.

- The average number of PRBs used by the bearers that attempt to obtain more than their maximal fair share of PRBs (defined as very active bearer in Definition 4.2). These are the bearers that need to be monitored to ensure SLAs.

- A method for precise and direct computation of available bearer throughput per slice. When combining the amount of average PRBs ($\overline{PRB_i}$) with individual channel bearer conditions (b_i), it allows for easy derivation of the wireless throughput available to a very active bearer i. The throughput can be computed as:

$$R(b) = \overline{PRB_i} \cdot C(b_i) \tag{4.2}$$

  where $C(b_i)$ is the average number of useful bits per PRB for bearer i [219]. The forecast of the UEs channel quality can be used to estimate the MCS [175]. Table 4.1 illustrates the average throughput given a variety of PRB rates along with the user's MCS. For each PRB range, 3 MCS values are provided along with the corresponding resulting max throughput.

4.4.2 Definitions

We introduce the following definitions prior to defining REVA. Without loss of generality, we assume one or more QCIs per slice.

Definition 1. ***Active bearers*** *for a non-GBR QCI m are bearers that use on average γ PRBs per second (e.g., $\gamma = 30$)*[5]*. Active bearers can be broken down into two groups: Very Active and Less Active.*

Definition 2. ***Very Active (VA)*** *bearers for a non-GBR QCI m are those that continuously attempt to obtain more than a maximal fair share of PRBs that are available from the scheduler for a given duration of time.*

Definition 3. ***Less Active (LA)*** *bearers for a non-GBR QCI m are the active bearers that are not VA.*

Examples of VA bearers are FTP and HTTP adaptive streaming video. Examples of LA bearers are web browsing and viewing social media. An example of a non-active bearer is a smartphone application that periodically performs keep-alive handshakes and receives push notifications. Bearers for each slice are classified into VA and LA based upon PRB resource consumption.

REVA is now formally defined as the following:

Definition 4. ***REVA for a slice*** *is defined per QCI and traffic direction (DL or UL) as: available Resource rate (in PRBs/sec) for an ideal 'Very Active' bearer.*

REVA determines the number of PRBs that a VA bearer at a given QCI can obtain. REVA specifically focuses on non-GBR bearers since, for GBR bearers a guaranteed amount of resources are allocated. The GBR service class guarantees the throughput and therefore the amount of PRBs allocated to a GBR bearer depends upon bearer channel conditions which typically varies in time. Notice that we use 1 second (1,000 TTIs) as the time interval. For low latency slices, one can use a smaller time interval (e.g., 20 to 100 TTIs).

The algorithm to calculate REVA appears in Algorithm 1. Bearers are categorized in an iterative way similar to the max-min fair share algorithm. In each iteration, LA bearers are those that use less than their fair share of the resources remaining. We define S^{total} as the total number of

[5] 30 PRBs limits the max throughput of a bearer to $\sim$ 1 KBps with channel conditions appropriate for 16QAM modulation. This parameter can vary based on the service type of the slice.

Table 4.1: Throughput for PRB rate along with the UE's MCS.

Average PRBs/sec	MCS	Max Throughput(kb/sec)
0-500	8,19,27	70,180,310
500-1000	8,19,27	140,360,630
1000-2000	8,19,27	280,720,1260
2000-3000	8,19,27	420,1080,1900
3000-5000	8,19,27	700,1800,3150

Table 4.2: Notation.

Symbol	Semantics
S^{total}	Total number of PRBs/sec available to a slice over interval Δt
δ	Fraction of control plane PRBs (typically 0.01 or 0.02)
w_m	Proportional fair weight for QCI_m
$RPRB_m$	Reserved PRBs for QCI_m
$\vec{B_m}$	Vector of PRBs of active bearers at QCI_m
$\vec{BR_m}$	Vector of PRBs of unclassified active bearers at QCI_m
$\vec{LA_m}$	Vector of PRBs of LA bearers at QCI_m
N_m	Number of active bearers at QCI_m
NR_m	Number of unclassified active bearers at QCI_m
U_m	Number of PRBs used by LA bearers at QCI_m
I_m	Fair share of PRBs at QCI_m

PRBs/sec available to a slice. For example, for a slice with 10 MHz bandwidth, $S^{\text{total}} = 50,000$ PRBs/sec. The available PRBs for the slice is $S^{\text{total}}(1-\delta)$, where δ represents a fraction of control plane PRBs.

In this section, we assume that the slice includes multiple QCIs and we compute the REVA metric for each non-GBR QCI. Algorithm 1 consists of two steps:

- Compute available PRB rate per QCI of the slice. (line 5 of Algorithm 1, Procedure 1)
- For each QCI, classify the slice bearers into VA and LA based upon their PRB consumption. Then, compute the REVA value. (lines 6-10 of Algorithm 1)

$\vec{B_m}$ is the vector of PRB rates for all the active bearers at QCI_m. Initialization is done by assigning N_m as the total number of active bearers at QCI_m. $\vec{BR_m}$ is the vector of PRB rates for all the active

Procedure 1 Aggregate available PRB Rate

1: **procedure** S_m
2: $S = S^{\text{total}}(1-\delta) - \sum_{j \leq 4} PRB_j$
3: **if** Proportional Weighted Share Scheduling **then**
4: $S_m = S - \sum_{5 \leq j \leq 9, j \neq m} min(\overline{PRB_j}, S \cdot w_j)$
5: **if** Strict Priority Scheduling **then**
6: $S_m = S - \sum_{5 \leq j \leq m} S_j - \sum_{j \geq m+1} RPRB_j$
return S_m

Algorithm 1 REVA Computation

1: Initialize δ, S^{total}
2: **for** $m = 5 : 9$ **do**
3: Initialize $\vec{B_m}$, N_m, $\vec{LA_m}$
4: $\vec{BR_m} = \vec{B_m}$, $NR_m = N_m$, $NR_m^{\text{prev}} = 0$, $U_m = 0$
5: Calculate S_m (Procedure 1)
6: **while do** $NR_m \neq NR_m^{\text{prev}}$
7: $NR_m^{\text{prev}} = NR_m$
8: $I_m = \frac{S_m - U_m}{max(1, NR_m)}$
9: Update $\vec{BR_m}$, NR_m, U_m, $\vec{LA_m}$
10: $REVA(m) = \frac{S_m - U_m}{VA_m}$

bearers that have not been classified yet, and is initially set to $\vec{B_m}$. NR_m is denoted as the number of active bearers that have not been classified yet and is initially set to N_m. U_m is the number of PRBs used by LA bearers, and is initially 0. $\vec{LA_m}$ is the vector of PRB rates for bearers that are classified as LA, and initially it is empty.

The next step of Algorithm 1 is to estimate the amount of available PRBs for non-GBR QCI_m (Procedure 1).

Procedure 1 first adjusts for control PRBs and then removes the PRB rate for all the GBR bearers in line 2. The next computation performed depends upon the scheduling schema used across QCI classes. If proportional weighted share scheduling is used, S_m is set based on the minimum of the fair share requirement or the amount of traffic required for that QCI level. In the case of priority scheduling, the amount of resources for that QCI level is calculated based on the amount of resources for higher priority QCI levels and the amount of Required PRBs (RPRB) for lower priority QCIs. The minimum number of resources for lower priority QCIs is used to ensure that even lower priority QCIs are not starved.

Table 4.3: REVA computation example.

Iteration	NR_m	U_m	I_m	LA UEs
0	20	0	2,450	12-20
1	11	1,620	4,307	8-20
2	7	16,120	4,697	5-20
3	4	25,620	4,845	3-20
4	2	39,120	4,940	2-20
5	1	44,020	4,980	2-20

Algorithm 1 continues by iteratively eliminating LA bearers. In each iteration, the amount of fair share of PRBs, I_m, is calculated. LA bearers are those that use less than I_m. The amount of PRBs used by LA bearers, U_m, is updated accordingly. After eliminating LA bearers, the amount of unclassified active bearers and the vector of PRBs of those bearers are updated, NR_m and $B\vec{R}_m$, respectively. Iterations continue until either no additional LA bearers are added ($NR_m^{\text{prev}} == NR_m$), or 0 or 1 non-LA bearers remain. Eliminated bearers are LA, and remaining bearers are classified as VA. The resulting REVA level is computed for each QCI m. Each slice will have at least 1 VA bearer by definition. Therefore, if 0 non-LA bearers remain, then the LA bearer with the largest number of PRBs becomes VA.

4.4.3 Computation Example

Consider a scenario where 20 UEs are served by a 10 MHz slice ($50,000$ PRBs/sec) with $\delta = 0.02$ and each UE has a single DL bearer at QCI 9. The PRBs over the past 1 second are assigned as follows (all units in PRBs/sec): UE1-5000, UE2-4900, UE3-4800,..., UE9-4200, UE10-3000, UE11-3000, UE12-20 each have 180 PRBS/sec. For example, in iteration 0 there are initially 20 active bearers, with $S_m = 49,000$. The fair share would be $I_m = 2,450$ PRBs, but UEs 12 to 20 use less than that amount and should be classified as LA. The amount of PRBs used by the LA bearers, U_m, would then be set to $1,620$ with 11 remaining active bearers for iteration 1. The algorithm would then operate as shown in Table 4.3. After 6 iterations there is 1 VA UE, resulting in a REVA value of 4,980 PRBs/sec.

There is one UE that is using its maximal share of the PRBs and the other UEs do not require additional resources. This can be converted into throughput based on the UE's MCS. The slice broker can then update the slice's PRB allocation accordingly.

4.5 Experimental Data Collection

Due to the lack of data from service providers, we built an experimental LTE testbed to collect data and calculate the REVA metric. PRB distribution per bearer with ≤ 1 second granularity is unavailable from deployed eNodeBs. Hence, we designed a lab LTE network with synthetic loads. The collected data is used in Section 4.6 to train forecasting models and in Section 4.7 to evaluate the impact prediction accuracy has on dynamically allocating resources. In this section we describe the experimental setup and the data collection process.

4.5.1 LTE Testbed

The experiments were performed using the lab testbed configuration depicted in Fig. 4.3. The LTE eNodeB was configured with a 10 MHz bandwidth using 700 MHz wireless spectrum (LTE band 13). All UE minicomputers were connected to LTE Remote Radio Heads (RRHs) via LTE USB modems using Radio Frequency (RF) cables and splitters. An RF impairment tool with two input and two output ports was used to emulate a variety of radio conditions for two groups of UEs.

Slices were emulated by using different QCIs, with separate Access Point Names (APNs) configured for each slice. For the purpose of this chapter we focused on predicting REVA of a non-GBR slice at QCI 9. We built an LTE load generator consisting of 15 UEs configured for QCI 9 and 3 UEs configured for QCI 3 (GBR). The operation of the load generator was controlled via scripts by a load generator controller connected to the UEs over Ethernet LAN. In each experiment, the non-GBR UEs were running FTP download over LTE in a continuous loop. The scripts also controlled GBR UEs to start/stop FTP download over LTE to emulate multiple independent patterns for the non-GBR slice.

We enhanced the LTE eNodeB scheduler by adding a thin layer to instantaneously compute

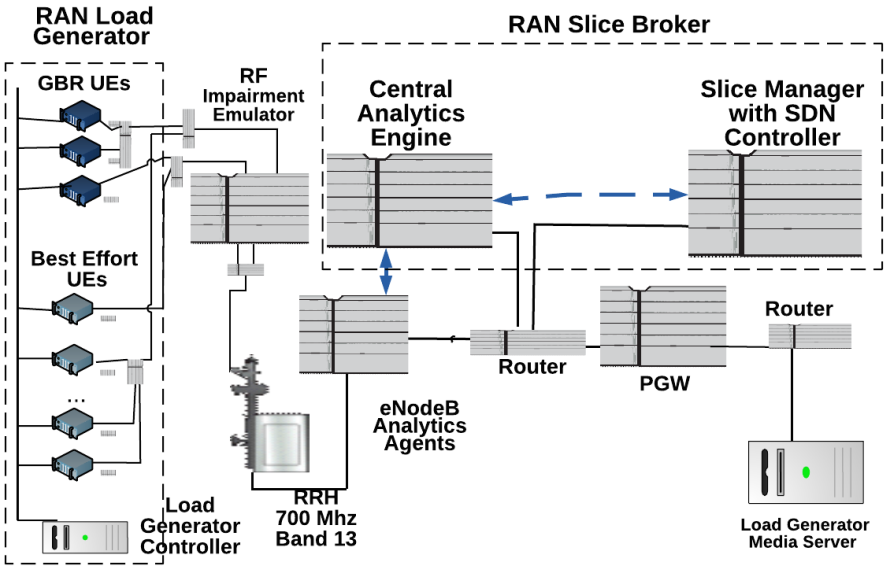


Figure 4.3: Lab Configuration Setup. The LTE eNodeB scheduler calculates REVA which is forwarded to Central Analytics Engine to compute optimal policy action. The Central Analytics Engine sends the action to the Slice Manager for enforcement. Additional components (MME, SGW, HSS, PCRF) are left out for simplicity.

REVA. The LTE eNodeB scheduler sends data digests every second in the form of the REVA metric per QCI and per slice to the Central Analytics Engine (CAE) via a dedicated out-of-band connection. The CAE performs instantaneous REVA data smoothing and further processing of the smoothed data. The predicted REVA metric is processed by the CAE policy engine to suggest an optimal policy for resource allocation. The CAE then sends this suggestion to the slice manager for enforcement.

4.5.2 Data Collection

The REVA metric (Section 4.4) is calculated at the eNodeB Scheduler. It records per bearer PRB distribution data every TTI (1 millisecond), and aggregates per bearer data into bins of time duration Δt (e.g., $\Delta t = 1$ second or $\Delta t = 100$ milliseconds). At the end of each Δt interval, it performs the computation described in Algorithm 1 (Section 4.4) and sends data digest per QCI level to the analytics engine. Our measurements indicate that executing Algorithm 1 every 1 second with approximately 500 active bearers adds less than 1% to eNodeB CPU utilization. The algorithm's complexity is $O(k^2)$, where k is the number of users. Performing preprocessing of the

data at the agent instead of sending the raw scheduler data every 1 millisecond, reduces the amount of data transfer by a factor of more than 1000.

The original REVA data has a high variance over short windows due to fluctuations in the TCP client behavior. To rectify this issue we smoothed the data to reduce the noise fluctuations. Smoothing is done by looking at a window size of 10 seconds and eliminating the minimum and maximum values to remove potential outliers. The minimum remaining value of REVA is chosen. We focus on the minimum opposed to other metrics (e.g., mean, median, max) to ensure that current slices have enough resources and to prioritize current slices over incoming new slices.

In this chapter, we discuss 3 time series sets collected from the testbed. [215] shows that there are temporal patterns of cellular traffic at time scales on an hourly, daily, and weekly basis. We vary the number of overlapping temporal patterns from 1 to 3 to emulate similar situations.

The data sets consist of the 15 non-GBR clients using FTP, and were collected for roughly 18 hours. The first data trace contained one periodic GBR client and is referenced as Set 1 and viewed in Fig. 4.4(a). This resulted in a simple square pattern with small variations and can be used for baseline evaluations of the prediction models. The second data set had two varying GBR clients that used FTP and is referenced as Set 2 and viewed in Fig. 4.4(c). Three GBR clients periodically used FTP to create a three pattern overlay profile for Set 3 (Fig. 4.4(e)).

The Autocorrelation Function (ACF) is used to analyze the potential predictability of a time series. ACF is the correlation of a signal with a time lag l version of itself. The results of ACF give insight into how much information from the past can be used to predict future values. Figs. 4.4(b), 4.4(d), 4.4(f) show ACF at a time lag of $l = 0, 1, ..., 2000$ for REVA in Sets 1,2, and 3, respectively.

The periodic nature of ACF for Set 1 shows spikes that do not decay. It also reveals that the underlying data should have a high predictability. The ACF of Set 2 shows a stong autocorrelation for the first 100 time lags with additional significant correlation at later time lags. The medium correlation values reveal a time series that should have moderate predictability. The ACF of Set 3 is not smooth like the previous two functions. In addition, there are not as many peaks in the ACF, meaning that this data set should be the most difficult of the three to predict.

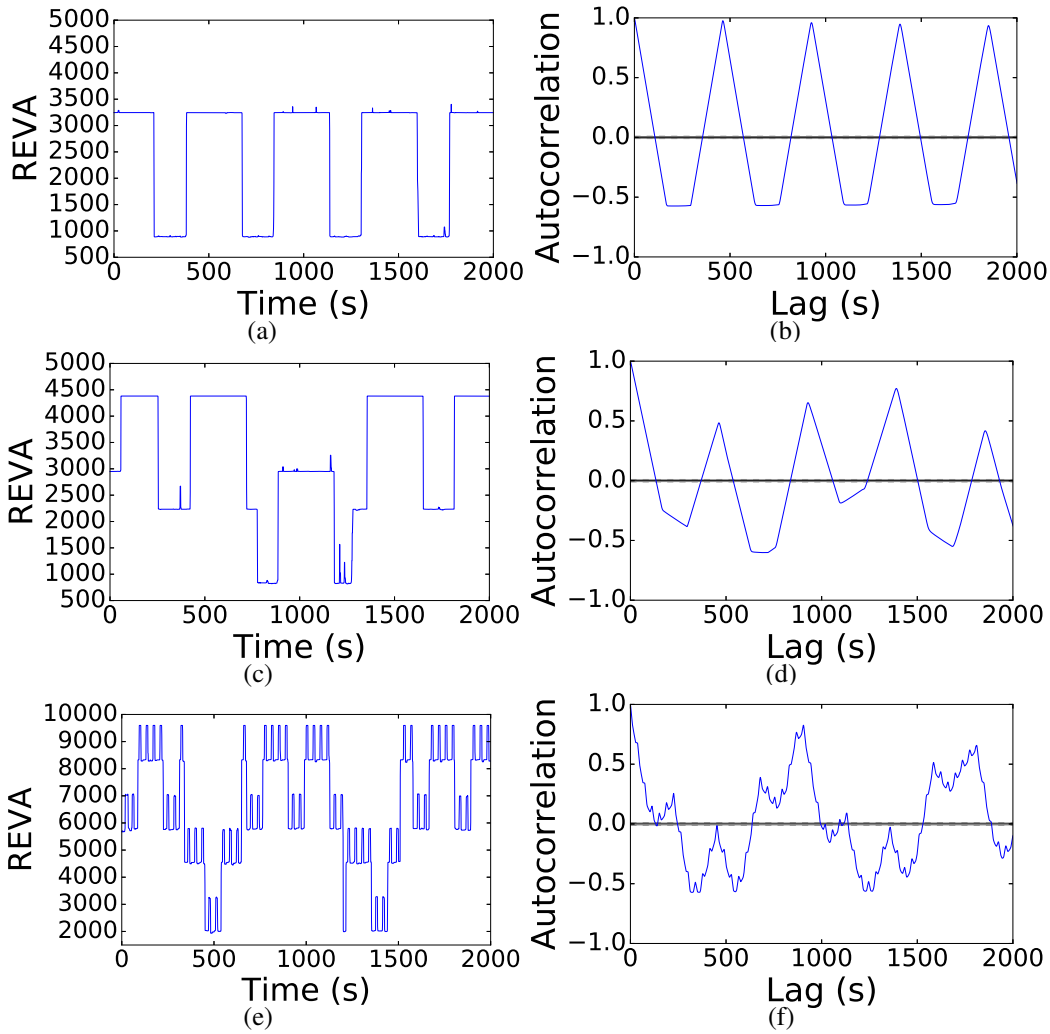


Figure 4.4: Experimental Data Collected (a) Set 1, (b) Set 1 autocorrelation, (c) Set 2, (d) Set 2 autocorrelation, (e) Set 3, (f) Set 3 autocorrelation.

4.6 Machine Learning Models

A precise prediction model allows a broker to utilize the RAN more effectively. Therefore, there is a need for reliable and short term prediction of the slice resource utilization. Current time series models are designed for predicting one step into the future and do not perform well when predicting multiple steps at a time. Our design objective is to forecast REVA for the next 30 seconds with prediction intervals of 5 seconds. The forecasting function $f(\mathbf{y_{t-1}})$, will provide a forecast for $\hat{y_{t+5}}, \hat{y_{t+10}}, \hat{y_{t+15}}, \hat{y_{t+20}}, \hat{y_{t+25}}, \hat{y_{t+30}}$. The first 60% of each time series is used as training data, the next 20% for validation, and the final 20% for testing. We begin this section with a description of the time series models used as a baseline: ARIMA and LSTM. We then describe the

architecture of the X-LSTM model designed to solve this problem.

4.6.1 ARIMA

One of the most popular statistical methods for time series analysis is the ARIMA model [27]. The ARIMA model assumes that future values are a linear function of previously observed values and random noise. The ARIMA model can be modeled as $ARIMA(p, d, q)$.

The data used to train the ARIMA model was the 5-second averages of the smoothed data. The ARIMA model was implemented in Python using the Statsmodel package. Optimizing the performance of the ARIMA prediction model requires tuning of the parameters p, q, and d. The optimal model was selected by a grid search while varying the parameters for p, q, and d between 0 and 3. The parameter setting that gave the minimum RMSE for the validation data was $(p, q, d) = (1, 0, 1)$ for both data sets. To obtain a 30 second prediction, there were 6 steps of prediction done at a time. The ARIMA model parameters are retained for every batch of multistep prediction.

4.6.2 LSTM

Recurrent Neural Networks (RNNs) differ from traditional Feedforward Neural Networks (FNNs) in that they contain feedback loops to allow information to propagate from previous steps. Feeding information from previous steps into the next step allows for a deep neural network architecture without computing and storing as many parameters. Therefore, these models are ideal neural network architectures for time series prediction. One of the major problems with traditional RNN is that it does not store information for long periods of time due to the vanishing gradient problem [87]. A type of RNN specifically designed to understand long term dependencies is an LSTM [88].

We implemented the LSTM neural network in Python on Keras using the Tensorflow backend. It contains a single recurrent layer with a linear layer. Validation data was used to tune the performance of a variety of neural network architectures. The architecture chosen for testing uses a single LSTM layer with 100 hidden neurons, and a dense output layer to predict the output congestion level. The LSTM uses a history of the previous 80 timesteps to predict the next step. The learning

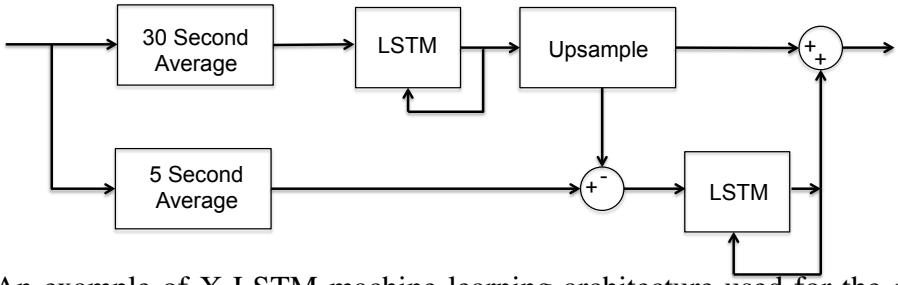


Figure 4.5: An example of X-LSTM machine learning architecture used for the experimentally collected data. This X-LSTM architecture contains two phases, one at a time scale of 30 seconds and the next at 5 seconds.

rate was set to $\alpha = 0.005$. The input data is the 5 second averages of the RAN congestion. In theory, LSTMs can store memory for an infinite number of timesteps, but, as the number of timesteps increases it creates more computation complexity to tune the parameters for back propagation. In implementation the Truncated Back Propagation Through Time (TBPTT) method is used to limit the number of memory steps [103].

For multistep prediction (Multistep LSTM), an iterative procedure is used with T set to 80. For the first prediction the real past 80 timesteps are used. For the second prediction, 79 real timesteps are used along with the previously predicted value. This continues until the 6th prediction is done with 75 real values and the previous 5 predicted values.

4.6.3 X-LSTM

We develop X-LSTM as an extension of LSTM. It is based on the idea of ARIMA and the X-11 statistical method. X-11 is an iterative process that decomposes time series data into seasonal data patterns. This method combined with the prediction of an LSTM improves results over standard methods. We break the method into two phases as can be seen in Fig. 4.5.

The X-LSTM model uses multiple LSTMs, each with a different time scale. It filters out higher order temporal patterns and uses the residual to make additional predictions on data with a shorter time scale.

We implemented each LSTM neural network of X-LSTM in Python using Keras with the Tensorflow backend. Each LSTM block contains a single recurrent layer with 100 hidden neurons with a dense output layer. For each LSTM block 80 timesteps were used. The learning rate was

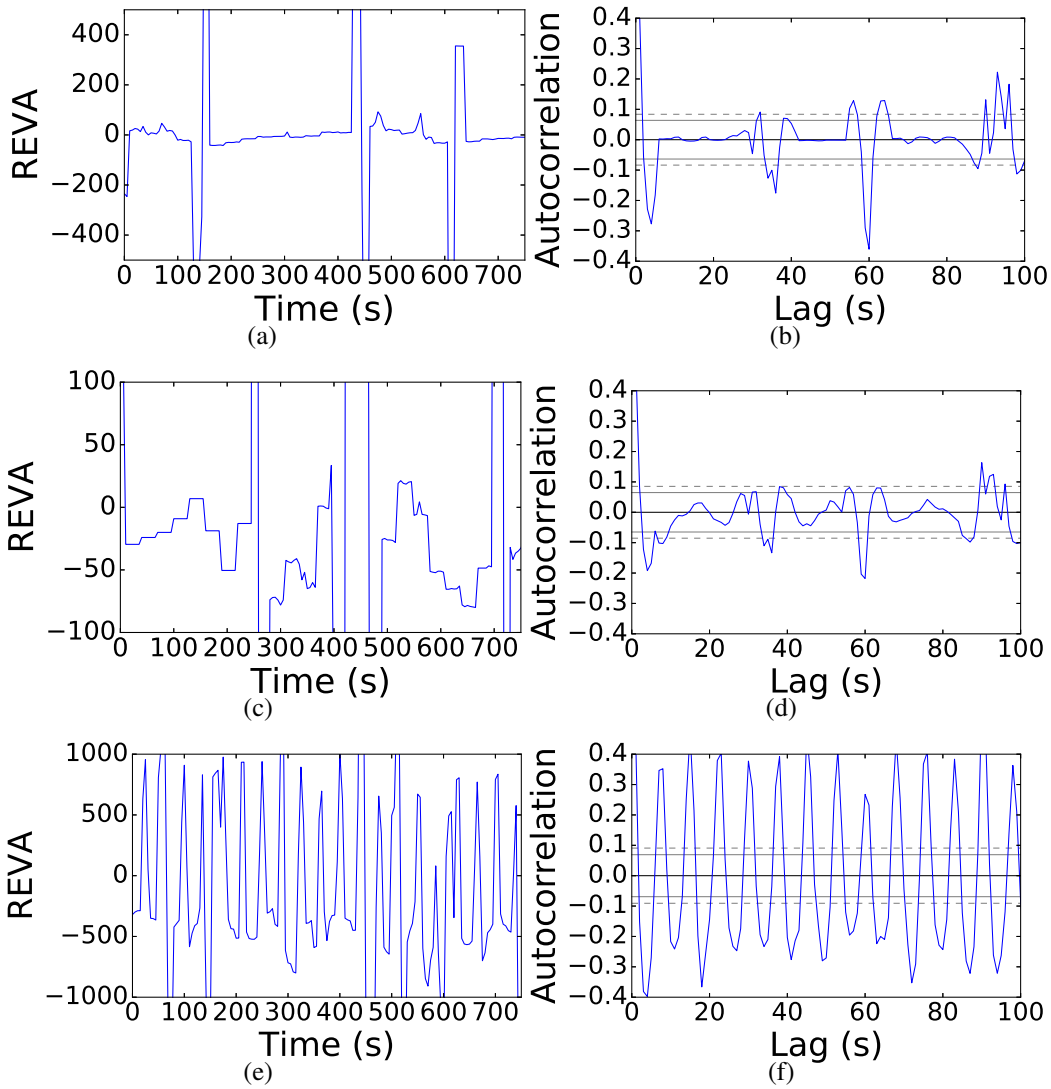
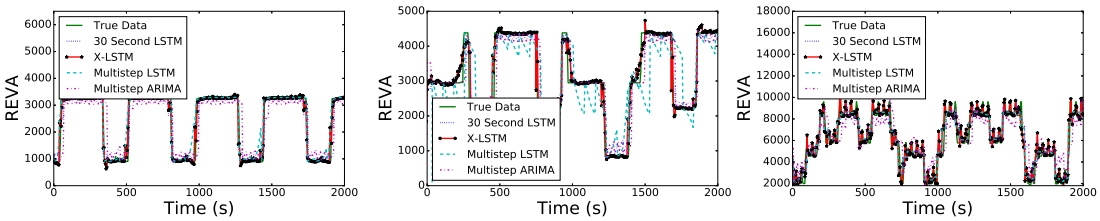


Figure 4.6: Residual of first phase prediction (a) Set 1, (b) Set 1 autocorrelation, (c) Set 2, (d) Set 2 autocorrelation, (e) Set 3, (f) Set 3 autocorrelation.

set to $\alpha = 0.005$.

For our experimental data, the first phase is done at a 30 second time level. The first LSTM block makes predictions for the average over the next 30 seconds, $\hat{y_{t+30}}$. The goal of the next phase LSTM is to make predictions for the time series of the residuals $(y_t - \hat{y_{t+30}})$ at a higher granularity. For our evaluation, a second phase with a time scale of 5 seconds was used. The residual values for Sets 1, 2 and 3 are seen in Figs. 4.6(a), 4.6(c), and 4.6(e), respectively. Looking at the autocorrelations, in 4.6(b), 4.6(d), 4.6(f), reveals that the residual data has information from previous time lags that can be used to improve the prediction accuracy.



(a) (b) (c)

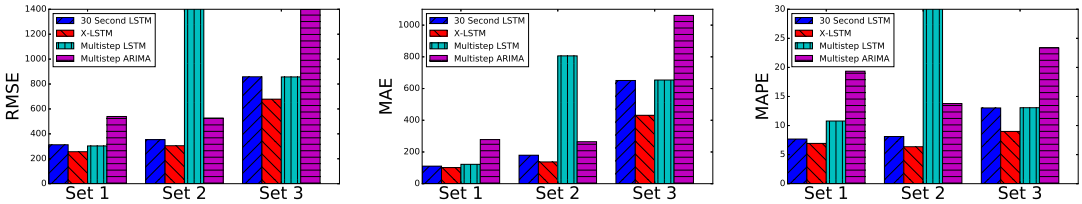
Figure 4.7: Results obtained by various prediction models: (a) Set 1, (b) Set 2, (c) Set 3.

The first step is to make a prediction for the average over the next 30 seconds. The second LSTM phase predicts the residual $(y_t - \hat{y_{t+30}})$ on a granularity of 5 seconds. It is used for multistep prediction in an iterative procedure for 6 steps. The 6 predicted values are then added back to the predicted 30 second average in order to predict REVA at $\hat{y_{t+5}}, \hat{y_{t+10}}, \hat{y_{t+15}}, \hat{y_{t+20}}, \hat{y_{t+25}}, \hat{y_{t+30}}$.

The number of phases and the time scale of each phase are the additional tuning parameters for X-LSTM neural network. While each of our data sets was only collected for roughly one day, this method can be extended to longer data sets by including additional phases. Typically networks exhibit time patterns due to natural seasonal, weekly, daily, and hourly traffic variations [215]. Data can be reduced by summarizing the data into lower granularity. For example, instead of storing per second data, data can be summarized into hourly, and daily averages. Data can be stored on the granularity of seconds for the past day, per hour bases for the past month, and daily values for the past year. This can reduce the amount of data needed for prediction by over 99%.

4.7 Evaluation

In this section we briefly discuss accuracy measures used for evaluation. We then analyze the prediction accuracy of each proposed prediction model. We continue with a description of a slice provisioning algorithm and design of a cost function based on the amount of over-provisioned PRBs and SLA violations. We conclude by showing how the algorithm exploits the prediction models to minimize cost for service providers.



(a) (b) (c)

Figure 4.8: Prediction errors for Sets 1,2,3 illustrated in Fig. 7: (a) RMSE, (b) MAE, (c) MAPE.

4.7.1 Prediction Results

Accuracy measures used for comparison between the different time series methods include: Root Mean Square Error (RMSE), Mean Absolute Error (MAE),and Mean Absolute Percentage Error (MAPE). The true output value is y_t and the predicted value is $\hat{y}_t$ for each time t. The accuracy of each proposed method is measured by splitting the experimental data collected into three components: training, validation, and test. The training data is used to tune the machine learning parameters for each method, while the validation data determines the best parameters for each method. The test data is used to compare the performance of each model.

The prediction results on test data for Sets 1, 2, and 3 can be seen in Fig. 4.7, along with the resulting error measures in Fig. 4.8. Four different prediction models are used for comparison. The first model predicts a one step 30 second mean of the time series using a vanilla LSTM model (referred to as 30 Second LSTM). The second model is determined by the X-LSTM model. The third prediction model uses a multistep LSTM to determine the congestion over the next 30 seconds in a granularity of 5 seconds. The fourth model uses a multistep ARIMA model to predict the congestion over the next 30 seconds in a granularity of 5 seconds.

Set 1 follows close to a periodic square wave with numerous time steps in between. The results show that all the models are able to follow the square wave relatively accurately. The multistep LSTM and ARIMA models have a more difficult time and incorporate errors from earlier steps leading to extrapolation. The prediction of the 30 Second LSTM is better able to determine the square pattern, as it has less information to store from previous periods and is only making a one step prediction. The resulting error measurements show that even on the simpler square

wave pattern, our X-LSTM model outperforms the other models. It is able to learn meaningful information from the residual between phase 1 and phase 2.

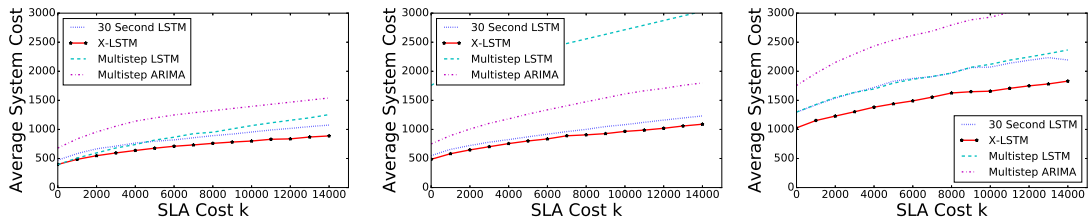
The second best model, 30 Second LSTM, gives a MAPE of 7.68% and an RMSE of 312, while the X-LSTM model gives a MAPE of 6.93% and an RMSE of 256, resulting in a 10% and 18% improvement, respectively.

Set 2 has a more complex time series which has two overlapping periodic events. All models in this data set have a higher error rate than in Set 1. The multistep LSTM has a difficult time incorporating the many variations it learned from the two overlapping patterns. When creating multistep predictions the errors continue in a positive or negative direction causing a sawtooth pattern. The multistep ARIMA model has a higher accuracy for Set 2 than the multistep LSTM model. Phase 1 of the X-LSTM model is able to track the model, however it is often slow to update when there are large changes. The X-LSTM model is able to incorporate the residual difference and make an improvement on the phase 1 predictions. The additional benefit of X-LSTM is that when the last phase LSTM needs to predict multiple steps, the bias of the prediction is reduced.

The second best model is again, 30 Second LSTM, which has a MAPE of 8.11% and an RMSE of 353. The X-LSTM model has a MAPE of 6.36% and an RMSE of 304, resulting in a 22% and 14% improvement.

The most challenging set to model due to its three overlapping periodic patterns is Set 3. The result is a lower level accuracy across all models. There is a strong residual component that has high ACF values. This allows the second phase of X-LSTM to improve upon the accuracy of the first phase and thus reduce the prediction error. The multistep ARIMA and LSTM models in this situation perform significantly worse than X-LSTM.

The MAPE of the multistep LSTM is 13.07% and for the LSTM predicting 30 second averages it is 13.03%. The X-LSTM is able to outperform these models by 31% with a MAPE of 8.99%.



(a) (b) (c)

Figure 4.9: Average system cost vs. SLA cost K for various prediction models: (a) Set 1, (b) Set 2, (c) Set 3.

4.7.2 Slice Allocation

We evaluate the impact that forecasting accuracy has on slice resource utilization. The REVA metric allows for easy adjustment of slice provisioning by using the difference between REVA and the SLA. The value can be used to estimate the amount of additional resources the slice should be allocated to satisfy the SLA or the amount of PRBs that should be removed while still satisfying the SLA.

We assume that the forecasting error ϵ_t in (1) is normally distributed with a standard deviation of σ and a mean of 0, since the exact distribution of the empirical error is unavailable. This gives a Gaussian prior for the prediction model:

$$y_t \sim \mathcal{N}(\hat{y}_t, \sigma^2) \tag{4.3}$$

We assume that assigning too few resources results in an SLA violation and incurs a penalty with cost k. Conversely, when forecasting a higher REVA value than predicted results in extra PRBs available to VA users that can otherwise be allocated to other slices. We use a one sided prediction interval h that determines the bound that should be used when assigning resources for forecast model f. We define the cost function, Γ as:

$$\Gamma(y_t) = \begin{cases} k, & \text{if } \hat{y}_t + h > y_t \\ y_t - h - \hat{y}_t, & \text{if } \hat{y}_t + h \leq y_t \end{cases}$$

Accordingly, we formulate the following optimization problem to obtain the optimal h such

that the system cost is minimized, given σ, $\hat{y}_t$, and k:

$$\underset{h}{\text{minimize}} \quad k(\hat{y}_t + h - y_t)^+ + (y_t - \hat{y}_t - h)(y_t - \hat{y}_t - h)^+. \tag{4.4}$$

Taking the expected value of the cost metric simplifies the optimization problem to:

$$\underset{h}{\text{minimize}} \quad (k + \hat{y}_t)(\Phi(\frac{h}{\sigma})) - h(1 - \Phi(\frac{h}{\sigma})) + \sigma(\frac{\phi(\frac{h}{\sigma})}{1 - \Phi(\frac{h}{\sigma})}), \tag{4.5}$$

Where $\Phi(z)$ is the CDF of the standard normal random variable Z, and $\phi(z)$ is the PDF of the standard normal random variable Z.

4.7.3 Slice Allocation Efficiency

For each forecasting model, the optimal h is used for each $\hat{y}_t$ and the cost $\Gamma(y_t)$ is calculated. Fig. 4.9 shows the average Γ per forecasting point verses the SLA violation cost k. As k increases the optimization function weighs SLA violations more. Therefore, the number of standard deviations away the prediction interval h is increases. The optimization function causes the probability of missing the SLA to decrease from 10% to 1% (depending on the forecasting algorithm's σ, $\hat{y}_t$, and the cost k).

In data set 1, with $k = 0$, Multistep LSTM slightly outperforms X-LSTM by less than 1%. For every other SLA cost k the X-LSTM outperforms the other forecasting algorithms. For $k \geq 5000$, X-LSTM provides more than 15%, 40%, and 15% reduction in average system cost over 30 Second LSTM, Multistep ARIMA, and Multistep LSTM, respectively. In data set 2, X-LSTM provides more than 11%, 35%, and 60% reduction in average system cost over 30 Second LSTM, Multistep ARIMA, and Multistep LSTM, respectively. In data set 3, X-LSTM provides more than 18%, 39%, and 18% reduction in average system cost over 30 Second LSTM, Multistep ARIMA, and Multistep LSTM, respectively. Generally across the three data sets, as SLA violations increase in cost the value of X-LSTM over the other prediction models increases.

Chapter 5: Deep Neural Network Based Dynamic Resource Reallocation of BBU Pools in 5G C-RAN ROADM Networks

In the previous chapter, we presented the X-LSTM prediction model, and showed that by using a higher accuracy prediction model a slice broker is more adept at provisioning a slice and reducing over-provisioning and Service Level Agreement (SLA) violation costs. In this chapter, we focus on allocating optical fronthaul wavelengths between the Remote Radio Heads (RRHs) and the Base Band Processing Units (BBUs).

5.1 Introduction

Traditional cellular networks rely on RANs in which baseband signal processing is carried out at the location of the cellular antennas. As traffic has increased, cell sizes are decreasing, dramatically increasing the number of cell sites and their capacity. In order to improve the scalability of these large numbers of access points, separation of the RRHs and the BBUs has been proposed for 5G networks. In centralized or cloud-RAN (C-RAN), the BBUs will be moved to a centralized location to allow for sharing of computing resources among multiple RRHs and mobile networks. By consolidating BBUs into several common locations (called BBU pools), cost and energy can be saved by sharing power and computational resources, leading to a reduction of capital and operational expenditures [145]. The high capacity required in these C-RAN fronthaul networks motivate the use of wavelength division multiplexed (WDM) optical systems.

It has been shown that traffic in different regions of a city can have different cellular network load pattern [215]. Therefore, further efficiency improvements might be possible if traffic can be reallocated among remote BBU pools based on the traffic load, using reconfigurable optical add drop multiplexers (ROADM). Previous work on assigning traffic to different BBU pools has

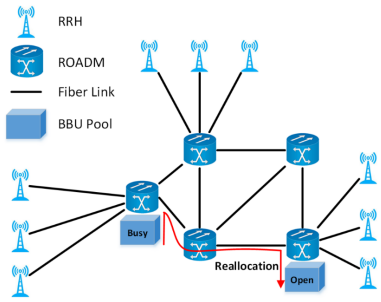


Figure 5.1: C-RAN network architecture with the capability of resource reallocation from a busy BBU to an open BBU.

relied on mixed integer linear programming (MILP) solutions [146]. Given that reconfiguration of optical networks takes minutes [148], real-time reallocation requires methods that can address this significant time dependence.

In this chapter, we develop a deep neural network based algorithm that can accurately predict future ROADM network resource requirements. Making accurate predictions 30 minutes in advance would allow for resource reallocation before the actual demand is needed, and therefore gives enough time for optical network reconfiguration to route the traffic through the C-RAN to a BBU pool with available computing resources. We compare the proposed resource reallocation approach with fixed resource allocation to evaluate the resource savings.

This research was done in collaboration with Prof. Dan Kilper's research group at the University of Arizona, with significant contributions from collaborating Postodctoral Research Scientist Dr. Yao Li, and Ph.D. student Dr. Weiyang Mo. It appeared in the proceedings of OSA OFC'18 [76].

5.2 Problem Statements and BBU Pool Resource Reallocation Approach

Fig. 5.1 shows a C-RAN architecture with 'fronthaul' WDM connections from an RRH to a remote BBU pool, which is typically via a common public radio interface (CPRI). Although fronthaul enables more efficient cloud based processing, the overall required transport capacity increases when the full digitized RF signal is used. Without the prediction of the traffic in advance, enough optical capacity and enough BBUs must be installed at each BBU pool to guarantee the

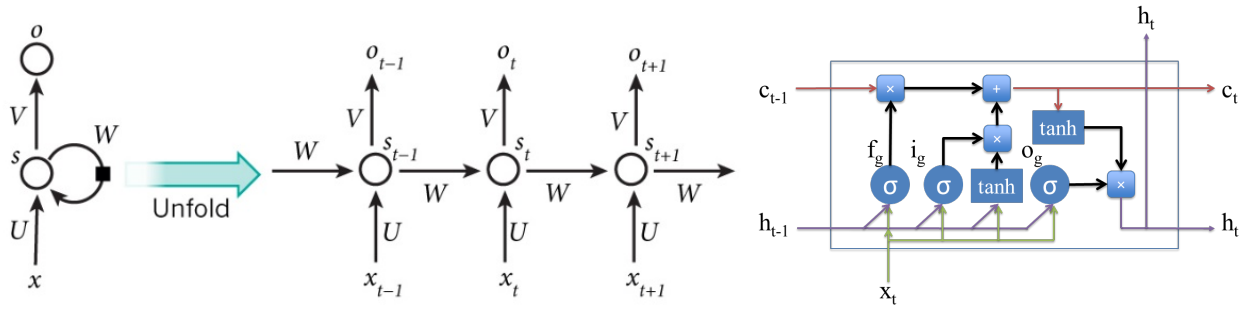


Figure 5.2: (a) Recurrent Neural architecture unrolled through time creating a deep neural network, (b) LSTM network.

peak processing requirement. On the other hand, the aggregated traffic at different BBU pools is different due to different cellular network load patterns, creating an opportunity to take advantage of sharing processing resources across BBU pools. To enable this sharing through ROADM optical network reconfiguration, which can take minutes, traffic pattern prediction must be made minutes in advance.

Recurrent neural networks (RNNs) can be used for predicting time series data through interconnected neurons based on previous time samples to make predictions for the next time series [65] as shown in Fig. 5.2(a). Unlike feedforward neural networks, RNNs do not only use information from the current input, but also use information from previous time steps. When unrolled through time, as seen in Fig. 5.2(a), it creates a deep neural network. A specific type of RNN architecture designed for long-term time series prediction is a Long Short-Term Memory (LSTM) network depicted in Fig. 5.2(b). LSTM works by storing information in the memory cell and passing it to the next time step. The inputs to the LSTM cell gates are a concatenation of the new input x_t and the previous output h_{t-1}. The LSTM cell has a forget gate fg that allows information to be forgotten from the previous cell state c_{t-1}, an input gate i_g to add information into the new cell state c_t, and an output gate o_g for passing information from the new cell state c_t to the output h_t [88].

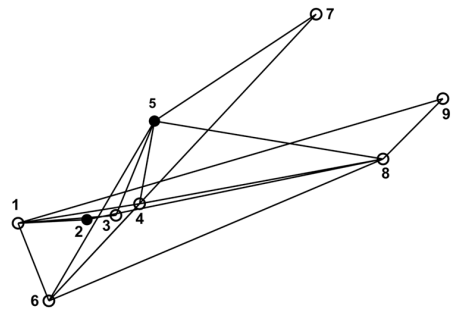
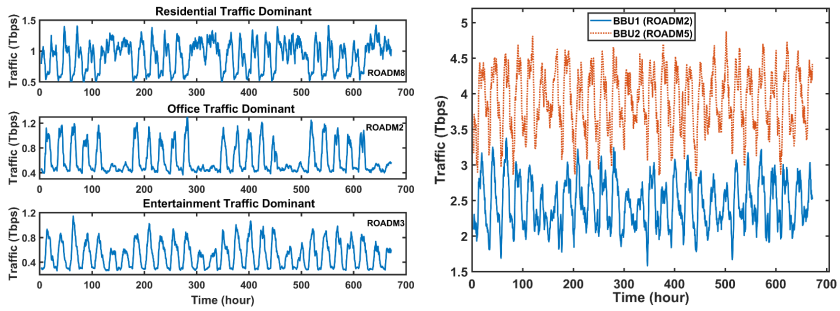


Figure 5.3: New York City regional PoP topology.

5.3 Case Study and Results

We use discrete event simulations to evaluate the resource reallocation approach, considering the traffic rejection rate and total network throughputs as two main performance metrics. A regional New York City point of presence (PoP) network is considered, where 9 ROADM nodes cover a 400km^2 region with a 3.5 average degree, as shown in Fig. 5.3. 12060 connection requests following Poisson arrivals are generated with uniformly-distributed source and destination pairs. We assume 64 small cells (SC) are directly routed to each ROADM with a maximum fronthaul link of 23Gbps per SC (i.e., 1.472Tbps peak traffic per ROADM). Residential, office, and entertainment traffic is considered in the simulations, and the distribution of the three types for each ROADM varies based on its geographical locations [65]. As a result, different ROADMs have different time-dependent traffic patterns. Fig. 5.4(a) shows the traffic pattern over 28 days for ROADM 2, ROADM 5, and ROADM 8 with residential dominant, office dominant, and entertainment dominant, respectively. Each fronthaul connection is required to pass a BBU pool for data processing before dropping at the destination ROADM. First, we evaluate the BBU pool resource reallocation approach with a highly-consolidated BBU pool placement, where only ROADM 2 and ROADM 5 (solid circle) have BBU pools. For each connection, a modified k-shortest path routing (k=5) with the first-fit wavelength assignment is used, where each routing path must pass through a BBU pool. 50GHz spaced dense WDM channels over C-band links are used with grooming and 100 Gb/s PM-QPSK modulation.

In the fixed resource allocation approach, for any connection request, the shortest path is chosen



(a) (b)

Figure 5.4: (a) Different traffic patterns (Resident, office, and entertainment dominant) of different ROADMS, (b) Traffic patterns at two BBU pools.

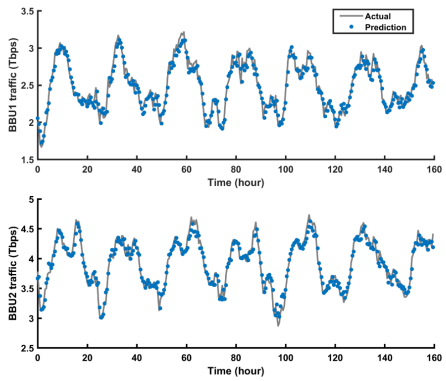
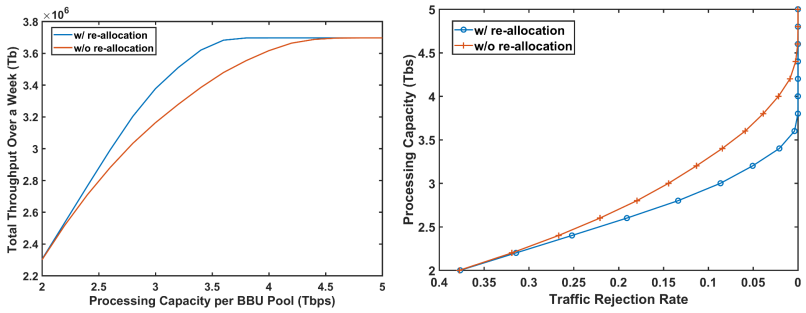


Figure 5.5: Traffic pattern prediction on two BBU pools using LSTM.

and the first BBU pool along this path is chosen to process the data. Fig. 5.4(b) shows the traffic pattern over 28 days for two BBU pools averaged every 30 minutes. The peak traffic of the two BBU pools is 4.7Tbps and 3.2Tbps, respectively. We then evaluate the LSTM network based resource reallocation. The time t of the LSTM network is in the interval of 30 minutes. Using the LSTM approach, the traffic of each BBU pool can be predicted 30 minutes in advance with high accuracy. An LSTM network is trained with 55% (740 samples) of the data in Fig. 5.4(b), and 20% (268 samples) is used for validation, and the last 25% (336 samples) is used for testing. Training is done by stochastic gradient descent where the training data is broken into batch sizes of 20 and optimized over 1000 epochs. Truncated back propagation through time (TBPTT) is used to optimize the weight parameters of the network. For the experiments, 60 time steps $X_t = \{x_{t-59}, x_{t-58}, x_{t-57}, \ldots x_t\}$ are used. Fig. 5.5 shows the prediction performance of the test data.



(a) (b)

Figure 5.6: (a) Traffic throughput improvement with resource reallocation, (b) Reduced traffic rejection rate with the resource reallocation.

The model obtains a mean absolute error (MAE) of 84.2Gbps, mean absolute percentage error of 3.1%, a root mean square error of 96.9Gbps, and a maximum error of 319.2Gbps. With the prediction of the traffic pattern in each BBU pool, the optical network can be reconfigured 30 minutes in advance so that some resources can be reallocated from one BBU pool to another for processing, and then routed to the destination ROADM.

We study the performance of resource reallocation, by varying the peak resource processing capacity of each BBU pool from 2Tbps to 5Tbps, and compare with the case with fixed resource allocation. Fig. 5.6(a) shows that with a small processing capacity, resource reallocation does not give much improvement since both BBU pools are usually overloaded and there is no additional capacity for reallocation. With the increase of BBU processing capacity, resource reallocation gives a higher 5G traffic throughput, with a 7% maximum improvement at 3.5Tbps. When the BBU processing capacity further increases, the improvement decreases because there is less of a need for resource reallocation. Fig. 5.6(b) shows the relation between the BBU processing capacity and traffic rejection rate. It is seen that with resource reallocation, zero traffic rejection rate is achieved with 3.8Tbs capacity in each BBU pool (i.e., 7.6Tbps in total). On the other hand, 4.6Tbps processing capacity is required for each BBU pool (9.2Tbps in total) to serve all the traffic. Overall, resource reallocation leads to an 18% processing resource reduction.

Part III

Dynamic Optical Systems

Chapter 6: Deep Neural network based wavelength selection and switching in ROADM systems

In this part we focus on using ML to ensure stable performance and reliable Quality of Transmission (QoT) for dynamic optical operation. Specifically, in this chapter we present a *feedforward deep neural network* based-ML model to predict the power dynamics of a 90-channel ROADM system.

6.1 Introduction

Growing dynamic traffic demands for Internet applications, including HD video rendering, cloud computing, and the Internet of things (IoT), motivate more efficient networks capable of handling a wide range of applications [106]. Dynamic ROADM systems in which connections are established through real-time wavelength switching have long been studied as a means to achieve greater scalability and increase the network resource utilization [125]. However, today's commercial ROADM systems remain 'quasi-static', with wavelengths being provisioned to meet the peak traffic requirements and left in place [202]. While ROADMs are extensively deployed in today's wavelength-division multiplexing (WDM) systems, they are primarily used for flexible wavelength provisioning without real-time switching functionality. SDN potentially provides software control capabilities that might be exploited to achieve real-time wavelength switching, but its scalability and flexibility are limited by various types of physical layer impairments [119, 230, 118].

A key unresolved challenge to achieving dynamic ROADM systems through SDN is predicting and controlling the optical power dynamics resulting from wavelength switching operations. Power excursions can result from the interactions between the wavelength dependent gain and automatic gain control of optical amplifiers, Raman scattering in the fiber and other wavelength

dependent phenomena. Deviations of the channel powers outside pre-allocated system margins can potentially result in service disruption due to reduced quality of transmission (QoT) [107]. For this reason, today's commercial systems take minutes and even hours to provision a wavelength through time-consuming power adjustments along an optical path [**nelson17**]. To realize dynamic ROADM systems, we implement a deep neural network that predicts power excursions resulting from wavelength switching operations. After training a 90-channel multi-hop ROADM system including 8 Erbium-doped fiber amplifiers (EDFAs) and 5 ROADM nodes with 67,200 training samples, the deep neural network is able to recommend wavelength assignments for wavelength switching in randomly loaded systems with over 99% precision.

The remainder of this chapter is organized as follows. In Section 6.2, we discuss the basics of power excursions and related work to address power excursions. In Section 6.3, we introduce the principle of the proposed deep neural network approach. The experimental setup is discussed in Section 6.4. In Section 6.5, we discuss the deep neural network architecture, data collection, training, and power excursion prediction. The performance of the deep neural network is evaluated against different metrics to show its effectiveness to mitigate power excursions in wavelength switching operations.

This research was done in collaboration with Prof. Dan Kilper's research group at the University of Arizona, with significant contributions from collaborating Postodctoral Research Scientist Dr. Yao Li, and Ph.D. students Shengxiang Zhu and Dr. Weiyang Mo. The hardware testbed for the evaluations was entirely built by collaborators at the University of Arizona. The research results presented in this chapter appeared in the Journal of Optical Communications and Networking [139]. A preliminary version of this chapter appeared in the proceedings of Big-DAMA Workshop at SIGCOMM'17 [79].

6.2 Problem Statement

Recent work has extensively investigated advanced modulation formats to improve the spectral efficiency and network capacity of WDM transmission systems [227]. But, these spectrally

efficient modulation formats require tighter QoT margins due to lower tolerance to both optical noise accumulation (impacting the optical signal-to-noise ratio (OSNR)) and fiber nonlinearity based impairments. As a result, the reduced transmission distances are further compromised by large margins that are needed to account for optical channel power variations or uncertainties. Thus, optical power dynamics that arise from wavelength switching operations become especially problematic in these systems. Furthermore, optical power dynamics often include phenomena that switching a wavelength on one channel causes power changes on other channels [107].

One main manifestation of optical power dynamics is the transient effect in an optical amplifier. The transient effect is fast power overshoots and undershoots that arise from sudden changes in input power due to wavelength switching operations or upstream fiber cut. For automatic gain controlled (AGC) EDFAs, a fast feedforward control loop can be implemented to augment the slower feedback control loop to effectively and rapidly suppress the transient effect. The feedforward control loop has the response time of 1 μs that can immediately adjust the pump power based on a pre-defined relationship between the pump current and input power for a target gain [199].

A different form of optical power dynamics in optical amplifiers—power excursions that result from the interactions between the wavelength dependent gain and AGC of optical amplifiers—can occur in wavelength switching operations. In the case of these power excursions, wavelength switching operations lead to persistent power differences on surviving channels, which are then corrected over long time scales using individual channel power controls in the ROADM nodes. Power excursions can grow in magnitude over cascaded amplifiers and cause substantial service disruptions. In recent work, 15 dB power excursions were reported in a WDM transmission system with recirculating loops totaling 2240 km [189]. For this reason, introducing or provisioning a new channel into a ROADM system is a time-consuming process that requires repetitive small-step power adjustments by sequentially actuating many optical components along an optical path to ensure that the powers of all surviving channels are within pre-allocated margins. In commercial-scale transmission systems, the fastest reported wavelength provisioning time is several minutes for a single 400 Gbps wavelength channel over a long-distance link [148].

There have been a number of approaches to address these amplifier-based power excursions. A fast tunable source was implemented to distribute a single optical signal over two wavelengths—one with a high gain and the other with a low gain—to equalize the mean gain and cancel out the power excursions [143]. An analytical solution was studied in [98] to mitigate power excursions based on a pre-measured EDFA gain spectrum. However, the gain spectrum does not consider the tilt change during wavelength switching operations and as a result only 5%–15% power excursion reduction is achieved. An optical probing method was also investigated to measure the EDFA gain spectrum without causing power excursions on surviving channels and thus recommend an optimal wavelength assignment with minimal power excursions [142]. Nevertheless, previous work relies on either specific system designs or specialized hardware and as a result increases the total hardware cost. Conversely, machine learning offers a more flexible solution without special hardware requirements. Particularly, machine learning has been well used to promote the development of intelligent optical communication systems [18, 230]. Through the extensive data collection of the power excursions versus changing channel loadings, a machine learning model can be trained to accurately recommend new wavelength assignments which will not cause power excursions. Previous machine learning applications examined wavelength assignment and defragmentation to minimize the channel power divergence or standard deviation of surviving channels, which primarily arises from the static gain ripple and tilt of EDFAs [94, 93]. Regression models, such as ridge regression and kernelized Bayesian regression were investigated to predict the channel power divergence in a 24-channel single-hop ROADM system, but such regression models do not consider the interactions between WDM channels and are unlikely to accurately predict the power excursions in wavelength switching operations. In order to accurately predict power excursions for WDM transmission systems including multiple ROADM hops and full C-band WDM channels, a more sophisticated machine learning model based on a deep neural network is investigated in this chapter. In this work, we extend a recent analysis of neural network based wavelength switching in [79]. We provide additional analysis on the computational complexity, overfitting reduction, and early termination using the deep neural network. The performance is also compared against ran-

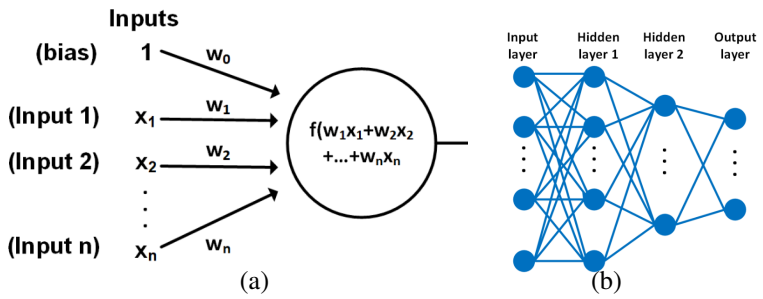


Figure 6.1: (a) The schematic diagram of a neuron, (b) Illustration of a deep neural network containing two hidden layers.

dom forest, showing the advantage of the deep neural network in learning and predicting complex power excursions.

6.3 Proposed Machine Learning Methodology

Machine learning has been developed to allow computers to learn to do a specific task without being explicitly instructed. Machine learning problems can be divided into two general categories—supervised learning problems and unsupervised learning problems. Supervised learning analyzes the training data and produces a relationship between an input object and the desired output object, which can be used for predicting the output of new input objects. Unsupervised learning problems try to draw inferences from datasets only consisting of input data. In this work, the focus is on developing a supervised machine learning model to predict power excursions based on an initial set of channels and the addition of a new set of channels. The data set includes the impact of complex interactions between channels that result in the power excursion response. A popular machine learning model for solving such complex problems is deep neural networks.

Deep neural networks are computational models that are inspired by the biological neural networks in the human brain [66]. The basic unit of a deep neural network is a neuron (also known as a node or unit) as shown in Fig. 6.1(a), which receives the input from other neurons and computes the output. In the real world, most data are nonlinear and we want these neurons to learn complex nonlinear representations. Therefore, nonlinear activation functions are introduced to the output of neurons to improve neural network approximations. Common types of nonlinear ac-

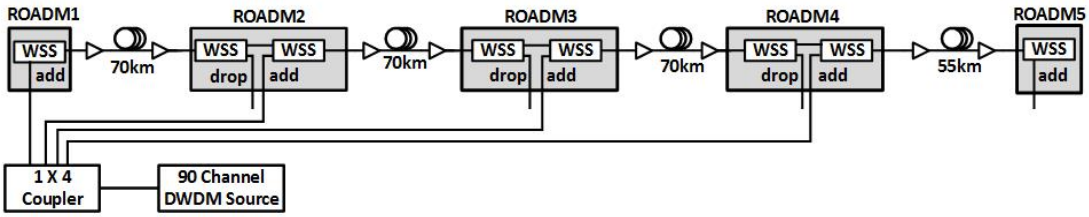


Figure 6.2: Schematic of the experiment setup including 5 ROADM nodes, 4 fiber spans and 8 EDFAs with different gain characteristics. The training, validation, and test data are collected by reconfiguring the channel loadings and measuring the power excursions.

tivation functions include tanh (hyperbolic tangent), sigmoid, and ReLU (a unit ramp function). Recent work has reported the advantages of ReLU because ReLU does not cause the "Gradient Vanishing" problem (which can completely stop the neural network from further training) [84]. However, the optimal activation functions will still depend on the particular applications and need to be determined by trial and error during the training process. Deep neural networks combine many layers of neural networks to find complex relationships and abstractions from the input data to understand and approximate the output.

A deep neural network consists of three types of layers as shown in Fig. 6.1(b): (i) Input layer: contain input neurons that provide information from the outside world. (ii) Hidden layer: contain hidden neurons that perform nonlinear transformations from the input layer to the output layer. A deep neural network may contain multiple hidden layers. (iii) Output layer: contain output neurons that predict the output to the outside world. The initial weights of the neural network are randomly set based on a probability distribution determined by the user. The first stage of training the neural network is forward propagation. The input vector is propagated through the neural network to determine the corresponding output. A cost function C is used to measure the accuracy of the predicted output y_i and the corresponding true output y_i. Common cost functions include mean square error (MSE) for regression (Eq. (6.1)) and cross entropy log loss for classification (Eq. (6.2)):

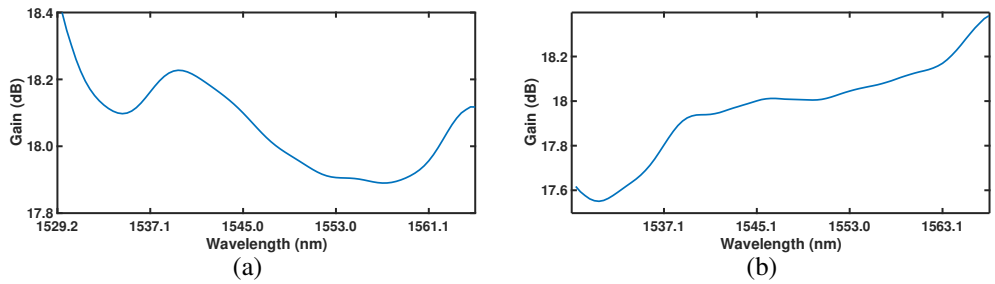


Figure 6.3: EDFAs in the first span with different gain spectra. (a) The Wavelength dependent gain spectrum of the first EDFA, (b) Wavelength dependent gain spectrum of the second EDFA.

$$C = \frac{1}{2m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{6.1}$$

$$C = \frac{1}{m}\sum_{i=1}^{m}(-y_i\log(y_i) - (1 - y_i)\log(1 - y_i)) \tag{6.2}$$

The next step is to determine how to update the network weights in order to minimize the cost function. Mini-batch (only a user-selected small subset of the training set in each training iteration) gradient descent is commonly used to determine the direction of steepest descent and how each weight in the neural network should be updated [149].

In order to optimize the performance of the deep neural network, validation data is used to compare the performance of a deep neural network with different parameters. The configuration that minimizes the specified loss function is chosen, and the test data is used to get an unbiased view of the performance of the deep neural network.

6.4 Experimental Setup

A metro-scale multi-hop ROADM system shown in Fig. 6.2 is built to study wavelength switching using the proposed machine learning approach. At the transmitter, a 90-channel comb source with spacing of 50-GHz is used to create 90 WDM channels from 191.60 THz to 196.05 THz (i.e.,

1529.2 nm to 1564.7 nm in wavelength). The power of the transmitter is then equally divided into four equal outputs using a 1×4 splitter, and each output is sent to a different ROADM (ROADM 1 to ROADM 4) to create different channel loadings. The ROADM system consists of five ROADMs which are separated by four standard single-mode fiber (SSMF) spans. Each SSMF span contains two dual-stage AGC EDFAs to compensate for the loss of the ROADMs and the transmission fiber and one variable optical attenuator (VOA) to increase the span loss to match the average 18-dB amplifier gain. Two-stage EDFAs realize the tilt-control by adjusting the attenuation of the variable optical attenuator (VOA) in the first stage, taking advantage of the fact that the tilt is dependent on the internal gains of the individual stages [142]. The tilt of each EDFA is adjusted in order to create wavelength dependent gain and study the power excursion mitigation, however, the peak to peak gain variation is kept within ±0.5 dB, which is typical for line amplifiers. Two-stage EDFAs realize the tilt-control by adjusting the attenuation of the variable optical attenuator (VOA) in the first stage, taking advantage of the fact that the tilt is dependent on the internal gains of the individual stages [142]. The VOA of the EDFA will be automatically adjusted by its internal controller based on the user-specified tilt requirement. The tilt is defined as the peak-to-peak gain variation of a least-squares-fitted line of the channel gains over the full signal band (ranging from 1529.2 nm to 1564.7 nm). Figure. 3 shows the gain spectrum of the two EDFAs in the first span. The tilt of the first EDFA is set to -0.4 dB to compensate for the stimulated Raman scattering (SRS) in the transmission fiber. The tilt of the second EDFA is set to 1.0 dB. The EDFAs in the other three spans have the same gain and tilt settings, thus giving rise to a similar gain spectrum. After cascading four transmission spans, the cumulative peak-to-peak gain variation across the C-band is measured to be 3.8 dB. Note that 4-6 dB gain variation is typically allowed between ROADM nodes depending on the system design. Such peak-to-peak gain variation along with the AGC operation results in substantial power excursions in wavelength switching operations. Each ROADM is comprised of multiple wavelength selective switches (WSSs), per-channel VOAs and per-channel optical channel monitors (OCMs). The power at the drop point is tapped to the OCM for per-channel power measurement. To measure the power excursion, we first measure the per-channel power of initial

channels at the channel drop point before the wavelength switching operation. Then, we measure the per-channel power of initial channels at the channel drop point after the wavelength switching operation (i.e., a new wavelength channel is added into the system). The power excursions are measured by taking the differences between them. Note that in general ROADMs will perform gain equalization for all the output ports to remove the power excursion accumulated in the previous link. This operation is time consuming and requires spectral analysis and therefore would come after the channel add or drop event and would be used to remove any residual power excursions. Using the deep neural network based wavelength switching, thus minimizes these gain equalization operations, resulting in more stable system operation and faster turn up times for new channels.

The effectiveness of machine learning is evaluated for power excursions that occur on top of the static power divergence due to the EDFA gain ripple and tilt. Thus, the system is initially configured to remove the static power divergence. Two types of channels are identified for wavelength switching operations—initial channels and new channels. First, the VOAs in each ROADM are initialized to ensure uniform 0-dBm launch power per-channel into the transmission fiber (i.e., 19.5-dBm total power) with the 90 initial channels (i.e., with 90-channel WDM input). These attenuation values are stored as a reference for newly added channels. Note that the VOA initialization can largely mitigate the channel power divergence due to static wavelength dependent gain in the EDFAs and static SRS in the transmission fiber. However, the initial VOA values cannot guarantee uniform 0-dBm power per-channel when the initial channel loading is changed in later experiments due to EDFA tilt change, EDFA power excursions, and dynamic SRS. Thus, VOA adjustment is executed to remove any power variations before wavelength switching, as would normally be done in system operation.

6.5 Results and Discussion

In this section, we describe the data collection process, deep neural network architecture, training process, power excursion estimation, and wavelength assignment recommendation using the

trained deep neural network. A deep neural network is first built and trained. Its performance is evaluated with regard to the number of training samples, the speed of the training process, and the accuracy in power excursion prediction and wavelength assignments. The performance of ridge regression and random forest methods are evaluated against the deep neural network for comparison purposes.

6.5.1 Data Collection

The first step in learning the complex optical power excursion response is extensive data collection of the power excursion response under a variety of channel loadings. However, such a data collection process is time-consuming due to the speed limitation of hardware actuation and software control. In this experiment, collection of each data sample takes approximately 3 seconds on average, including the latency of control signaling, WSS actuation along an optical path, VOA adjustment, and power excursion measurement. Note that the VOA adjustment is executed only once on each initial channel for each initial channel loading to remove any power variations, as would normally be done in system operation. The WSS actuation to turn on the new channels (with no additional VOA adjustments) and the power excursion measurement are executed for each wavelength switching operation. Collecting data might even take longer in commercial large-scale systems, and thus potentially imposes an obstacle to using the machine learning in practical ROADM systems. Methods to overcome to address these implementation issues will be discussed later. In this experiment, 1,680 training cases are used to train a deep neural network, each of which contains 40 power excursion measurements (i.e., 67,200 training samples in total) as the following process: 40 available wavelength positions for adding a new channel are randomly selected, and the maximal power excursion among all initial channels is measured by switching on and off these 40 wavelength positions one by one. In addition, 210 validation cases are collected for evaluating how well the deep neural network is trained and which parameters provide optimal prediction performance. Finally, another 210 testing cases are collected for evaluating the prediction accuracy and the performance of wavelength switching using the deep neural network.

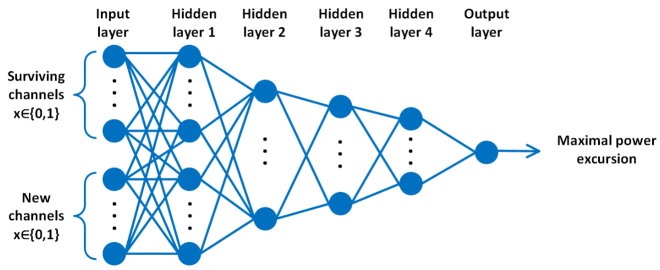


Figure 6.4: The architecture of the deep neural network. The input layer contains 180 features, representing the 'on' or 'off' state of initial channels and new channels. The output layer contains a single output, representing the maximal power excursion among all initial channels.

Table 6.1: Parameters of the optimized deep neural network.

Parameter	Value
Neurons in hidden layers	(180,120,30,15)
Activation function	(tanh, tanh, ReLU, ReLU)
L2 regularization	0.001
Dropout rate	0.1
Initial learning rate	0.005
Number of epochs	217

In total, 84000 data samples are used in this experiment, taking approximately 70 hours for collection. Note that in this experiment all channels are sent through the longest route over 4 spans (i.e., ROADM 1-2-3-4-5), but the machine learning methodology is applicable to the multi-route case by recording the added ROADM and the dropped ROADM as additional input and collecting more training samples. Strategies to minimize the number of training samples while ensuring the prediction accuracy will be detailed in the training section.

6.5.2 Deep Neural Network Architecture

A deep neural network is built to predict the power excursion that occurs when adding a new channel into the multi-hop ROADM system. The input of the deep neural network is a 180-element binary vector as shown in Eq. (6.3). The first 90 binary input features (which correspond to 90 wavelength locations) are used to represent the wavelength locations of initial channels. A '1' represents that the wavelength is initially lit or occupied and a '0' represents that the wavelength is not initially lit. The next 90 binary input features represent the wavelength locations of new channels

being added into the systems. The '1s' represent the new wavelength locations of new channels added into the system. Note that in this experiment, we focus on the channel add operation, and we will show that minimized power excursions are guaranteed for channel add operations. Since the channel drop operation is the reverse process of the channel add operation, minimized power excursions are also guaranteed for the corresponding channel drop operations. The output that we aim to predict is the maximum power excursion among all initial channels as shown in Eq. (6.4).

$$\vec{x} = [\vec{x}_{initial}, \vec{x}_{new}] = [x_1, x_2, \ldots, x_{90}, x_{91}, \ldots, x_{180}] \in \{0, 1\}^{180} \tag{6.3}$$

$$\vec{y} = \max_{j \in \text{Initial channel}} \Delta P_j \tag{6.4}$$

The optimal neural network architecture is determined by varying a number of parameters. The optimized parameters include: the number of hidden layers, the number of neurons per layer, the activation function of hidden layers, the number of iterations (or epochs), the learning rate, the L2 regularization term, and the dropout rate. The performance is determined by minimizing the root mean square error (RMSE) against the validation set. The RMSE can be interpreted as the standard deviation of the difference between observed and predicted values (in dB). A lower RMSE indicates a more accurate prediction. The architecture with the lowest validation RMSE depicted in Fig. 6.4 includes 4 hidden layers using a combination of tanh and ReLU activation functions. Other parameters are summarized in Table. I, and the details of training the deep neural network will be discussed in the next section.

6.5.3 Training

It is important to minimize the data collection time while still ensuring prediction accuracy. With a small number of available training samples, the deep neural network tends to over-fit the

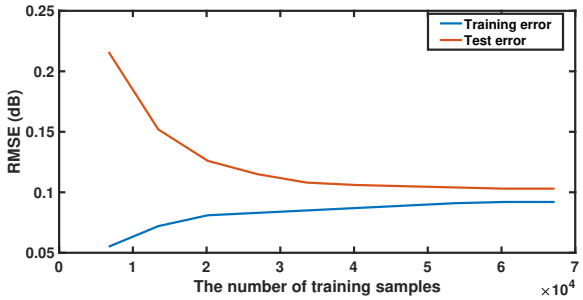


Figure 6.5: The root-mean-square error (RMSE) of the training set and the test set as a function of the number of training samples during online training.

specific training samples resulting in a low training error. However, since these training cases do not well represent the full set of behaviors in the system, this causes a high variance that generates a high prediction error in the test samples as new data samples have not been seen by the deep neural network. Getting more training samples can effectively reduce the variance and better generalize the model, but the prediction performance might saturate at some point because of the existence of a small bias that limits further improvement of learning performance. The bias in this experiment mainly arises from actual system and measurement errors such as time-varying penalties (e.g., temperature change) and measurement uncertainty (e.g., OCM inaccuracy). For example, the accuracy of OCMs in this experiment is ±0.1 dB, and it may happen that two data samples with the same features give rise to different target output.

In this experiment, online training is implemented in a control plane to determine the number of training samples that are needed as follows. 210 testing cases and 210 validation cases are first collected. The online training of the deep neural network contains repetitive processes. For each process, 168 more training cases (i.e., 6,720 more training samples) are added to the training set to train the deep neural network and calculate the root-mean-square error (RMSE) of the 210 testing cases. The online training continues until the test RMSE does not decrease with two consecutive processes. Fig. 6.5 illustrates the training RMSE and test RMSE over a varying number of training samples (also called the learning curve) during online training of the deep neural network. The training error curve shows the difference between the prediction based on the training data compared against the actual training data; whereas the test error curve shows the error in predicting the

power excursion for different sets of random data (the test set). With just 6720 training samples, there is a large difference between the training error and the test error due to large variance on different samples, indicating a poor generalization of the model. With the increased number of training samples, the training error and the test error start to converge but a gap will persist due to the inherent variance of the samples. Online training stops at 67200 training samples because there is no RMSE decrease over two consecutive processes.

The deep neural network shown in Fig. 6.4 is trained to minimize the RMSE using mini-batch stochastic gradient descent (SGD) with a mini-batch size of 64. In order to prevent overfitting, regularization techniques, including L2 regularization [201] and dropout [134] are implemented. Several combinations were tested with a varying L2 regularization value and dropout rate in each hidden layer, and we found an L2 regularization of 0.001 and a dropout rate of 0.1 can effectively prevent overfitting and achieve low RMSE. The initial learning rate is set to 0.005 and is adapted in the training stage to allow for fine weight updates. The learning rate adaptation is multiplied by 0.99 every epoch.

It is important to reduce the training time as long training time for a system can add significant cost. One important metric that decides the training time is the number of epochs in the training stage, since the total training time is linearly proportional to the number of epochs. In this experiment, the tradeoff between the prediction accuracy and the number of epochs is evaluated by comparing the accuracy of the neural network predication using the validation data set. Figure. 6 shows the RMSE (in dB) of the training set and the validation set in the training stage as a function of the number of epochs. Initially, the RMSE of both the training set and the validation set significantly decreases with the increased number of epochs. After approximate 200 epochs, although small fluctuations exist, the RMSE of the validation set shows minimal improvement. In fact, the RMSE of the validation set is 0.103 dB after 200 epochs and 0.100 dB after 900 epochs. In this experiment, the training is terminated if three consecutive epochs fail to decrease the RMSE of the validation set. By introducing this rule to the training stage, the training stage is terminated at the 217th epoch with a validation RMSE of 0.104 dB. Compared with the validation RMSE of 0.100

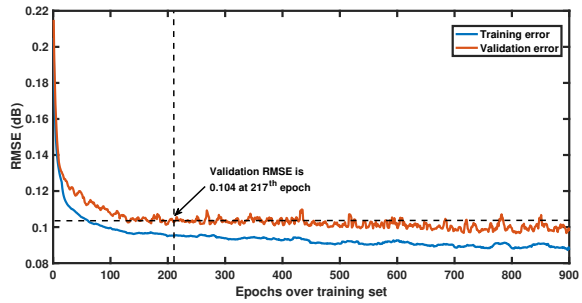
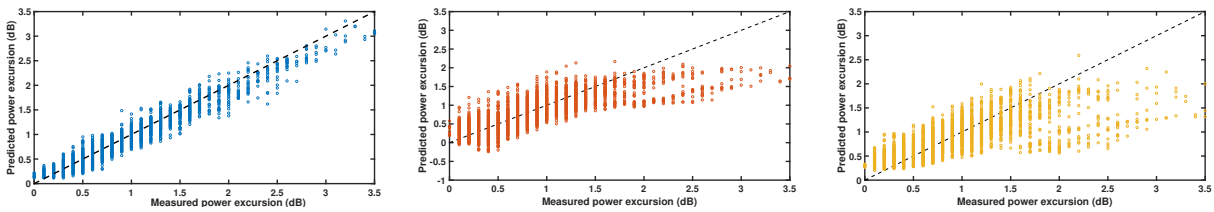


Figure 6.6: RMSE as a function of the number of epochs in the training state. The training stage is terminated at the 217th epoch with the validation set RMSE of 0.104 dB.



(a) (b) (c)

Figure 6.7: Predicted power excursion vs. measured power excursion over the test set. (a) Deep neural network, (b) Ridge regression, (c) Random forest. Both ridge regression and random forest underestimates the power excursion when the actual power excursion is above 2dB.

dB at 900th epoch, there is negligible performance difference, but the training time is reduced by more than a factor of four.

6.5.4 Performance Evaluation

After the training stage has completed, a check on the deep neural network performance is carried out against the test set using different metrics. For comparison purposes, ridge regression and random forest methods are also evaluated against the test set. For the ridge regression model, the regularization parameter was tuned to 0.01 through cross validation. For the random forest model, 200 trees (with 180 features being considered for each tree) are found to provide the best performance while ensuring minimal training time. Note that we also evaluated the support vector machine (SVM), but its computational time does not scale well to a large number of training samples [141].

First, the RMSE and the maximal prediction error of the entire test set are evaluated, and the re-

Table 6.2: Test RMSE and maximal prediction error.

Machine learning model	RMSE (dB)	Maximal error (dB)
Deep neural network	0.104	0.8
Ridge regression	0.273	2.3
Random forest	0.281	2.7

sults are summarized in Table 6.2. A lower RMSE indicates a more accurate prediction. Similarly, a lower maximal prediction error reveals a better fit under corner cases. The deep neural network outperforms ridge regression and random forest by more than a factor of two in the RMSE and the maximal error. Random forest results in a worse performance because it is not able to learn the complex nonlinear relationship among 180 features with the given 67200 training samples. Ridge regression performs worse than the deep neural network because ridge regression does not take into account the inter-dependencies between input features. The prediction errors using different machine learning models can also be viewed in Fig. 6.7. The actual power excursion ranges from 0 dB to 3.5 dB, and black dashed lines indicate the perfect prediction. Its seen that the deep neural network obtains significantly lower errors between the actual power excursions and the predicted power excursions, and its accuracy is stable over the entire power excursion range. On the other hand, both ridge regression and random forest result in high prediction errors, particularly when the actual power excursion is above 2 dB.

The second metric used to evaluate the performance is the mean square error of the channel (MSEC) at a particular wavelength. A low MSEC indicates a high prediction accuracy for the particular wavelength, while a high MSEC indicates that the particular wavelength may not be considered as a potential candidate for wavelength switching due to a substantial prediction error. Fig. 6.8 shows the MSEC of 10 different wavelengths. The deep neural network efficiently keeps the MSEC below 0.02 dB across all 10 wavelength locations, while the maximal MSEC using ridge regression and random forest can be as large as 0.11 dB and 0.13 dB, respectively. Moreover, the MSEC is stable among all 90 channel wavelength locations with a standard deviation of 0.004 (Note that only 10 wavelengths are shown in the figure), indicating that the deep neural network is

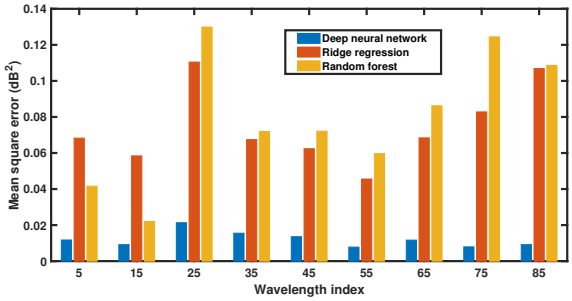


Figure 6.8: MSEC as a function of wavelength locations using different machine learning approaches. The deep neural network not only provides less prediction error but also more stable performance across the entire 90 channel spectrum.

trustworthy to make an accurate prediction over the entire spectrum. In contrast, ridge regression and random forest result in much higher standard deviations of 0.03 and 0.04, respectively.

Third, we evaluate the δ recommendation accuracy, which is the proportion of test cases in which the deep neural network is able to recommend a wavelength with a power excursion within a δ margin from the minimal power excursion (which is achieved by switching on the optimal wavelength). A higher δ recommendation accuracy indicates the model is able to accurately recommend wavelengths for wavelength switching operations within a tighter power excursion bound. Fig. 6.9 shows the δ recommendation accuracy as a function of the δ margin using the deep neural network, ridge regression, and random forest. In this experiment, the minimum δ margin is 0.1 dB, taking into account the ±0.1 dB precision in the power measurement.

When the δ margin is set to 0.1 dB, such that the system has the strictest requirement of wavelength assignments (i.e., the exact optimal wavelength must be predicted by the model), the deep neural network can recommend the optimal wavelength among 40 wavelength candidates over 79.5% of the time (i.e., 167 test cases out of 210 test cases), while ridge regression and random forest only achieves 41.4% and 56.7% recommendation accuracy. When the δ margin increases to 0.4 dB (i.e., the actual power excursion of switching on the recommended wavelength must be within 0.4 dB from the minimal power excursion), the recommendation accuracy of the deep neural network is 100%, while the recommendation accuracy of ridge regression and random forest are only 89.5% and 96.2%. We also note that although random forest demonstrates a better accu-

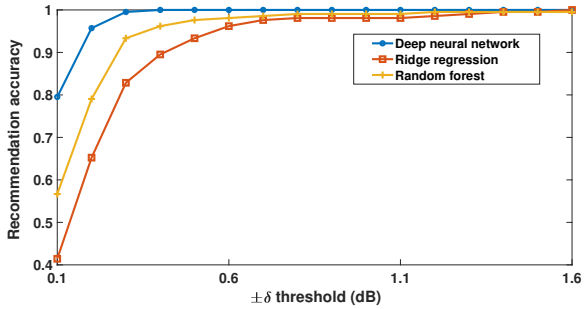
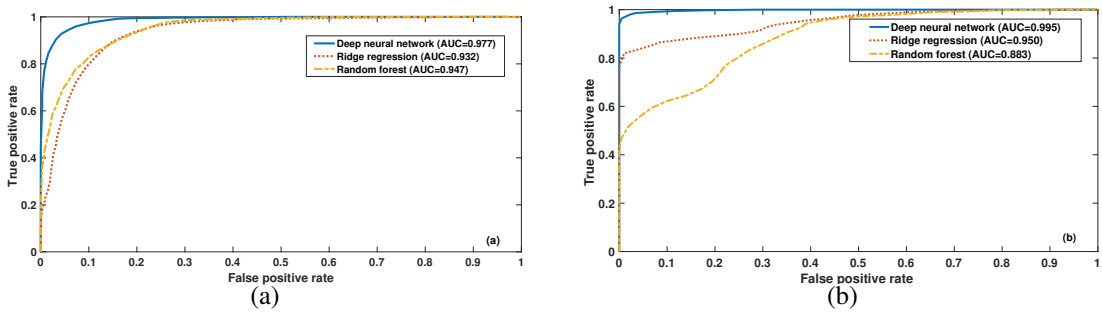


Figure 6.9: δ-recommendation accuracy as a function of δ margin from the actual minimal power excursion. The deep neural network is able to recommend the actual optimal wavelength 79.5% of 210 test cases.

racy than ridge regression under a small δ margin, its performance gets saturated after δ = 1.2 dB. This result indicates that random forest leads to a higher variance over the new data set with high power excursions (i.e., predict some test cases pretty well but others poorly). Note that these tests are conducted over a finite size, randomly generated data set within a very large space of possible values and therefore this bound does not guarantee accuracy over the full range of possible events.

Next, the classification accuracy is assessed for different power excursion thresholds using receiver operating characteristic (ROC) curves as shown in Fig. 6.10. A better classification accuracy indicates a more powerful model that is able to separate good wavelength candidates from the bad ones for a given system power excursion threshold. In this experiment, the classification accuracy is checked against two different power excursion thresholds, 0.5 dB and 1.5 dB, in according to the system QoT requirement reported in our previous work [143]. The classification is evaluated by two metrics: (i) The ability to separate positive cases from negative cases, which is quantified by the true positive rate (TPR) at a given false positive rate (FPR). A positive case means that the deep neural network recommends a channel as being within a given decision threshold, and a negative case means that the deep neural network rejects a channel as being outside a given decision threshold. Note that the decision threshold is used by the machine learning model to determine whether a potential wavelength is positive or negative, which is different from the system power excursion threshold. TPR is the ratio of correct positive predictions to all actual positives, and FPR is the ratio of incorrect positive predictions to all actual negative predictions. A perfect classification model is



(a) (b)

Figure 6.10: Receiver operating characteristic (ROC) curves to assess the classification accuracy for different system power excursion thresholds. (a) 0.5-dB threshold, (b) 1.5-dB threshold.

able to obtain 100% TPR while maintaining 0% FPR. The ROC curve is formed by connecting all TPR/FPR pairs, each of which corresponds to a different decision threshold. (ii) The area under the ROC curve (AUC). The AUC varies from 0.5 to 1, where 0.5 is the performance of a random classification model and 1 is the performance of a perfect classification model. Figure 10 shows the classification accuracy under 0.5 dB and 1.5 dB thresholds using the deep neural network, and the performance is compared to ridge regression and random forest. With a 0.5-dB power excursion threshold, the deep neural network obtains the best classification accuracy with a TPR of 80.4% while ensuring the FPR less than 1% and the AUC of 0.977. When the system power excursion threshold is increased to 1.5 dB, the deep neural network obtains the TPR of 97.1% with less than 1% FPR and the AUC is 0.995. We also note the interesting behavior of random forest for which the classification accuracy goes down (with an AUC from 0.947 to 0.883) when the system power excursion threshold is increased from 0.5 dB to 1.5 dB. This indicates that random forest tends to estimate the power excursion to be less than 1.5 dB, when the actual power excursion is above 1.5 dB.

Finally, we evaluate the PTPR, which is defined as the precision at a specific TPR under a system power excursion threshold. The precision is the ratio of true positives to the number of total positive values predicted. Keeping a high PTPR is important because minimizing the chance of adding a channel with a power excursion beyond the system margin (which may disrupt the whole transmission system) is more important than missing a possible valid channel candidate. Thus, a high PTPR guarantees reliable system operations with a minimal possibility of system disruption

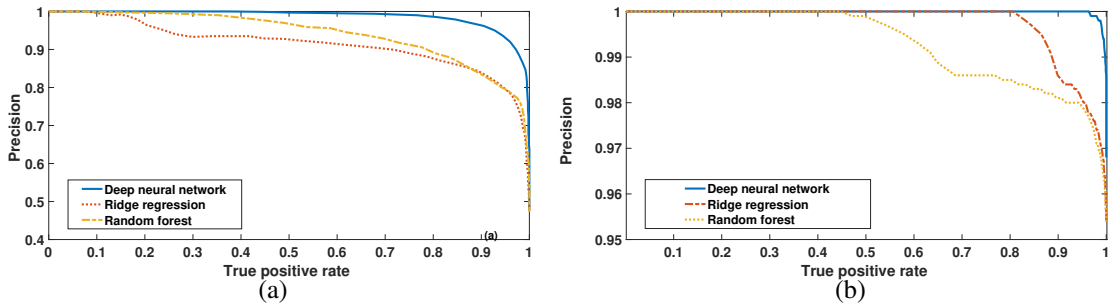


Figure 6.11: PTPR curves using different machine learning models with two different system power excursion thresholds. (a) 0.5-dB threshold, (b) 1.5-dB threshold.

due to wavelength switching operations. Fig. 6.11 shows the PTPR curve with 0.5 dB and 1.5 dB thresholds using different machine learning models. With a 0.5 dB power excursion threshold, the deep neural network obtains a precision of over 99% while ensuring a TPR of greater than 76% (i.e., ensure less than 1% false positives but also misses roughly 24% valid wavelength candidates). For comparison, ridge regression and random forest only obtain the TPR of 13.1% and 35.4% respectively in order to achieve the same precision. When the power excursion threshold is increased to 1.5 dB, the deep neural network is able to obtain a 100% precision while obtaining a 96.4% TPR (i.e., ensure zero false positives while missing only 3.6% valid wavelength candidates).

Chapter 7: Hybrid Machine Learning EDFA Model

In Chapter 6 we designed a *feedforward deep neural network* based-ML model to predict the power dynamics of a 90-channel ROADM system. In this chapter we design an improved ML model to predict the power excursion of the amplifier gain in an EDFA.

7.1 Introduction

In previous research, methods to predict to optical channel power divergence and dynamics in transmission systems include neural network based dynamic channel power estimation [79] and machine learning based power divergence prediction[94]. Furthermore, several mathematical models were introduced to predict individual channel output power under changing channel configurations, including a numerical power estimation framework used to predict EDFA output power[59], a detailed analytical model for the wavelength dependent gain impact[100] and a model designed for system applications using a simple characterization method[97], but with limited accuracy.

We examine the use of machine learning, replacing lookup tables or analytical models [100, 97, 240, 233, 142, 50], to determine the channel configuration and input power dependent EDFA gain spectrum. Deep neural networks are built and trained to predict the gain spectrum based on the input power spectrum. However, these machine learning models are built solely from a-posteriori knowledge which is trained from the experimental data, ignoring the existing a-priori knowledge. We then compare the performance of a hybrid machine learning (HML) model for EDFAs, which combines an analytical model with a neural network machine learning model to achieve higher prediction accuracy while reducing the training complexity, in both the training time and the size of the training sample data sets.

The remainder of this chapter is organized as follows. In Section 7.2, we discuss the basics of

power excursions and gain dynamics in AGC WDM line EDFAs. In Section 7.3, we discuss the experimental setup. In Section 7.4, we discuss the deep neural network model and its performance. In Section 7.5, we discuss the hybrid deep neural network model and its performance.

This research was done in collaboration with Prof. Dan Kilper's research group at the University of Arizona, with significant contributions from Postodctoral Research Scientist Dr. Yao Li, and collaborating Ph.D. students Shengxiang Zhu and Dr. Weiyang Mo. The hardware testbed for the evaluations was entirely built by collaborators at the University of Arizona. The research results presented in this chapter appeared in the proceedings of European Conference on Optical Communication'18 (ECOC) [240] and in OSA OFC'20[239].

7.2 WDM Channel Gain Models

Gain dynamics occur in automatic gain controlled (AGC) WDM line EDFAs due to wavelength dependent gain, where the gain excursion $\hat{g}$ due to a change in channel powers P_j can be written:

$$\hat{g}(\lambda_i) = g(\lambda_i)\frac{G_{TC}}{G_M}[\frac{\sum_j P_j + N_I + N_C}{\sum_j P_j g_j t_j + g_N N_R + g_I N_I}] \tag{7.1}$$

Accurate channel output power estimation therefore requires the gain ripple g_j, tilt t_j, and input noise N_I and gain g_I, amplifier noise N_R and gain g_R, and the amplifier noise compensation factor N_C. Many of these parameters such as g_j and t_j are also dependent on the input channel configuration through the internal amplifier gain. Other effects such as spectral hole burning may also play a role.

In another study, a model was proposed for calculating the gain spectrum based on measurement of the single channel ripple and WDM ripple functions [97], which can be written as:

$$\hat{g}(\lambda_i) = g(\lambda_i) + \frac{\sum_{j=1}^{n}[g_s(\lambda_j) - g(\lambda_j)]}{n} \tag{7.2}$$

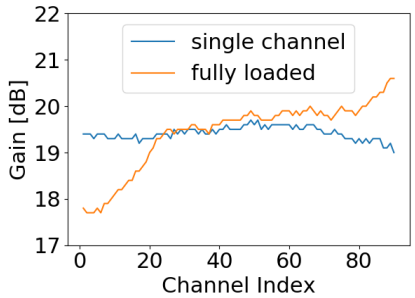


Figure 7.1: Gain spectrum in experiment and prediction by analytical model.

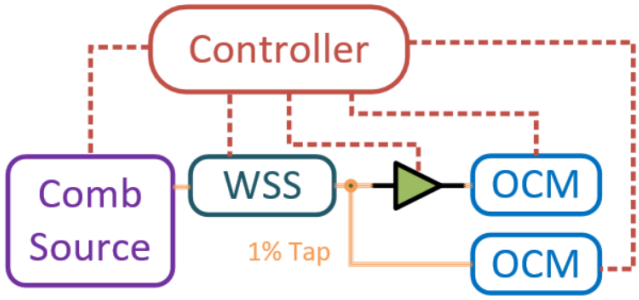


Figure 7.2: Experiment setup for data capture.

In this expression, $\hat{g}(\lambda_i)$ is the gain spectrum of wavelength λ_i when a set of wavelengths $\{\lambda_1, ..., \lambda_n\}$ is input to the EDFA. $g(\lambda_i)$ represents the characterized WDM gain spectrum, i.e., the gain spectrum when all WDM input channels are active. $g_s(\lambda_i)$ denotes the characterized single channel gain spectrum, i.e., the spectrum of the gain of each channel λ_j when no other channels are present. An example of measured $g(\lambda_j)$ and $g_s(\lambda_j)$ is shown in Fig. 7.1. This center of mass (CM) approach conveniently provides an estimate from easily measured configurations, but misses detailed effects that can be important, particularly in certain corner cases.

7.3 Experimental Setup

As shown in Fig. 7.2, a 90-channel comb source is used to generate the WDM input. The wavelength selective switch is used to control the input power spectrum. The EDFA is configured to have 3 dB tilt and work in AGC mode, with target gain set as 18 dB. Two optical channel monitors are used to monitor the input and output power spectrum. The controller is used to communicate with all of these devices and capture data.

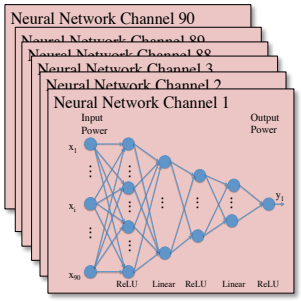


Figure 7.3: Architecture of neural networks.

First, we characterized the single channel and WDM ripple of the EDFA and built a model using (Eq. (7.2)). In Fig. 7.1, the typical single channel and WDM input gain spectra are shown. We can see that for different channel loading the gain spectra are quite different.

7.4 Machine Learning Model and Performance

7.4.1 Machine Learning Model

A supervised machine learning algorithm is designed to train a neural network model of EDFAs to predict the gain spectrum based on the input power spectrum. The Neural Network (NN) architecture is implemented with TensorFlow.

Ninety features are used as the input to the NN, representing the power levels of each of the 90 channels. A separate NN is created for each output channel. Data is divided into 3 classes: training data, validation data, and test data. The training data is used to train the NN to minimize the Mean Square Error (MSE) loss function. The validation data is used to determine which parameters provide optimal performance after using the training data. The test data is used to evaluate the trained model.

The resulting parameters for the NN architecture are described as follows. All power levels are converted into decimal power levels, normalized, and scaled by a factor of 300. Each neural network has 4 hidden layers with artificial neuron transfer function of ReLU (rectified linear unit), Linear, ReLU, Linear, and ReLU. The full NN architecture can be seen in Fig. 7.3. The model is trained by minimizing the MSE loss function using stochastic gradient descent with back-

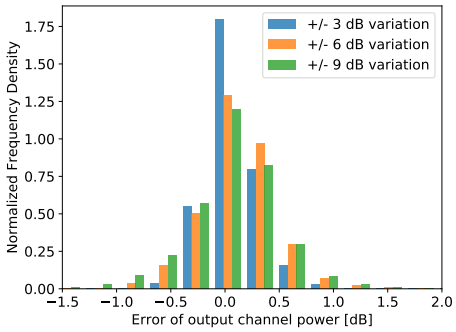


Figure 7.4: Normalized frequency desnity funciton of prediction error using the CM analytical model.

Table 7.1: RMSE of analytical and ML models.

Dynamic Range [dB]	Analytical [dB]	Machine Learning [dB]
±3	0.247	0.081
±6	0.359	0.180
±9	0.410	0.266

propagation with a mini batch size of m=60 and a learning rate, $\alpha = 0.00025$. Training is done over 15000 iterations.

7.4.2 Machine Learning Performance

We characterized the single channel and WDM ripple of the EDFA and built a model using Eq. 7.2. In Fig. 7.1, the typical single channel and WDM input gain spectra are shown. We can see that for different channel loading the gain spectra are quite different.

We used captured samples of measured input and output spectra to evaluate the error of analytical model derived from Eq. 7.2. In Fig. 7.4, the normalized frequency density (NFD) of the error of analytical model is shown. It is shown that wider dynamic range of the input power leads to higher error in the prediction of the gain spectrum.

To build the NN, we captured data samples of the input and output spectra under different channel loadings. We then compare the root mean square error (RMSE) distribution of the CM analytical model and the ML model. In Table 7.1, the dynamic range is defined as the range over which the input channel power is varied around the target power (-18 dBm) in a uniform random

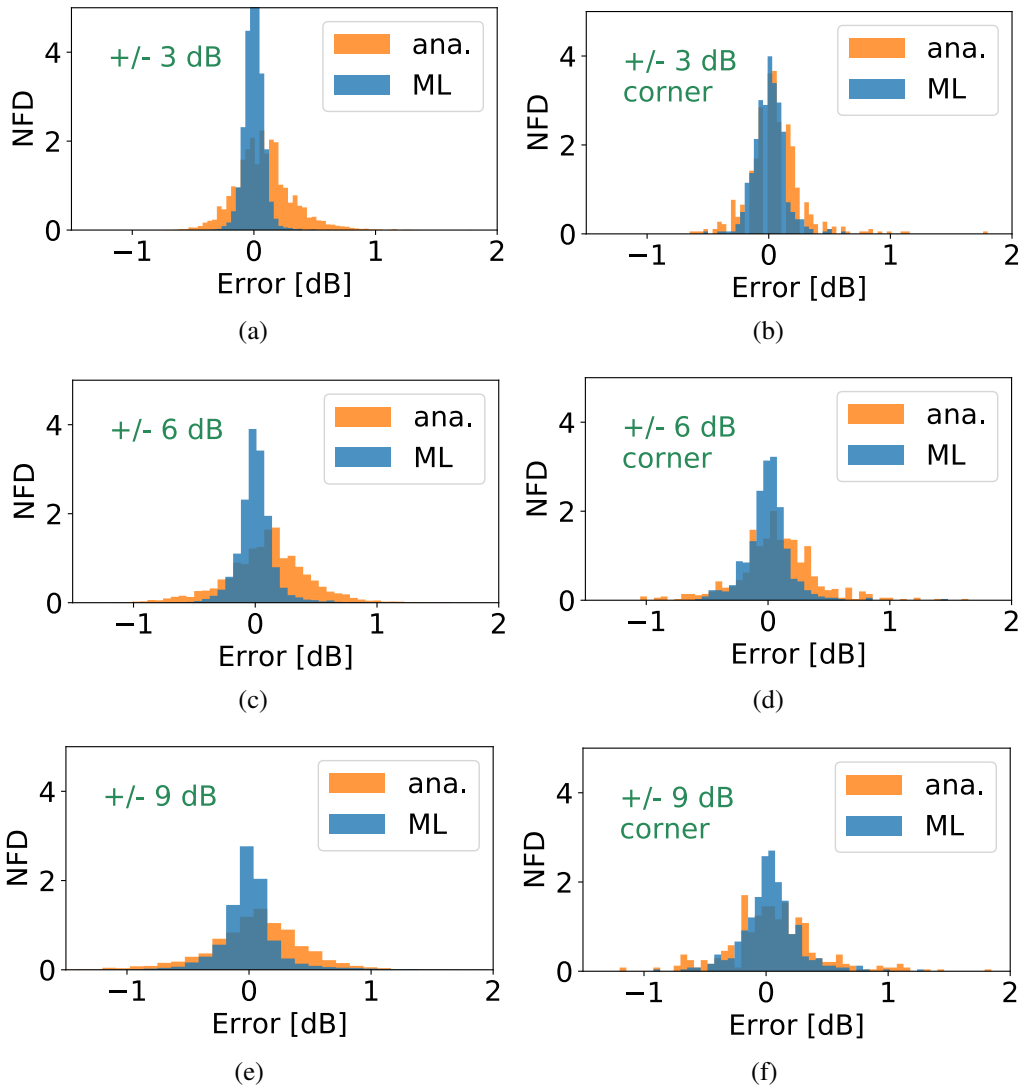


Figure 7.5: Error distribution of analytical model and ML model with dynamic range of ± 3, 6, 9 dB, (a) ±3 dB, (b) ±3 dB corner, (c) ±6 dB, (d) ±6 dB corner, (e) ±9 dB, (f) ±9 dB corner.

distribution. It is shown in the table that the ML model is able to reduce the RMSE by 67%, 50%, and 35% in scenarios where dynamic range is ±3 dB, ±6 dB, and ±9 dB.

We then analyze the error distribution of the analytical model and the ML model using the same test set, shown in Fig. 7.5(a), Fig. 7.5(c) and Fig. 7.5(e). In the case of ±3 dB dynamic range, the ratio of the errors below 0.5 dB is 99.95% for the ML model and 94.9% for the analytical model. In the ±6 dB case, the ratio is 98.37% vs. 85.11%. In the ±9 dB case, the ratio is 93.57% vs. 81.07%. The ML model shows better performance in gain spectrum prediction for all dynamic

range values.

Since the most severe channel power dynamics occur when there is a handful of open channels, it is of interest to analyze how the ML models perform in these corner cases. Here we look at the corner case with just two channels turned on. In Fig. 7.5(b), Fig. 7.5(d), and Fig. 7.5(f), the error distributions of these corner cases are shown. With dynamic ranges of ±3, 6, 9 dB, the ratios of prediction errors below 0.5 dB are 99.38% vs. 95.88%, 98.57% vs. 87.78% and 95.62% vs. 83.18%, respectively. The ML model is able to avoid high errors (over 0.5 dB) for most of the cases, outperforming the analytical model.

To verify the performance of ML modeling under different EDFA configurations, we adjusted the gain setting of the EDFA and test the error of the models. In Fig. 7.6, the EDFA with 14 dB and 22 dB gain were modeled and tested and the error distributions are shown. The overall RMSE error reduction in ±9 dB dynamic range is 10.37% (0.3535 vs. 0.3944) for 14 dB gain and 15.21% (0.3501 vs. 0.4129) for 22 dB.

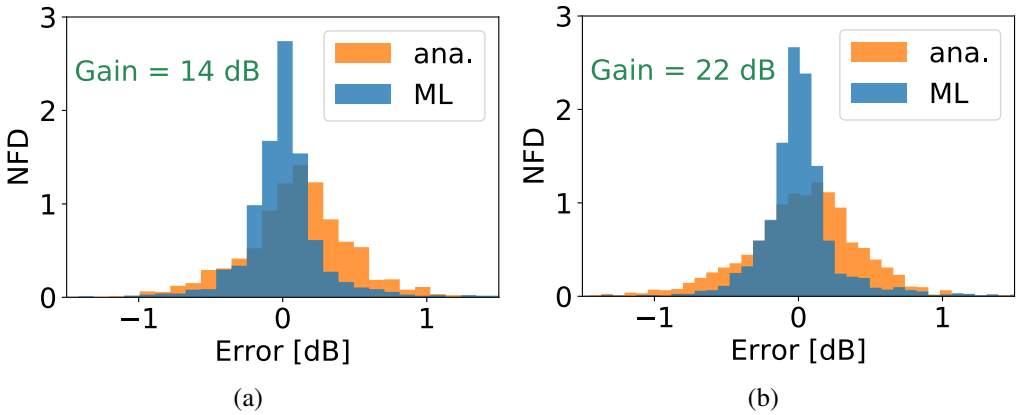
The EDFA spontaneous emission noise will show up at the downstream amplifier inputs on blocked channels if more than one amplifier is used between nodes (before being blocked again at the next node). In this ML model, noise power outside the dynamic range is ignored. Test results further show that the RMSE reduction is 51.71% (0.2368 vs. 0.4904), 33.18% (0.2346 vs. 0.3511), 9.44% (0.2840 vs. 0.3136), with blocked channel noise levels of -40, -35, -30 dBm per channel, respectively.

7.5 Hybrid Machine Learning Model and Performance

7.5.1 Hybrid Machine Learning Model

A supervised machine learning algorithm is designed to train a neural network model of EDFAs to predict the gain spectrum based on the input power spectrum. The Neural Network (NN) architecture is implemented with TensorFlow.

For the hybrid ML model, one hundred eighty features are used as the input to the NN. The first ninety features represent the power levels of each of the 90 channels. The second 90 features



(a) (b)

Figure 7.6: Error distribution of analytical model and ML model with gain value of (a) 14 dB, and (b) 22 dB.

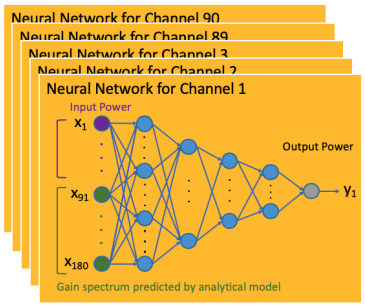


Figure 7.7: Structure of the hybrid machine learning model. 90 features ($x_1 - x_{90}$) of input channel power and another 90 features ($x_{91} - x_{180}$) of gain spectrum predicted by analytical model are used. The hidden layers have 180, 90, 90, 45 neurons.

are the predicted output power levels determined by the analytical model. All power levels are converted into decimal power levels and normalized. Each neural network has 4 hidden layers and an output layer with artificial neuron transfer functions of ReLU (rectified linear unit), Linear, ReLU, Linear, and ReLU. Each layer contains 180 neurons, 90 neurons, 90 neurons, 45 neurons, and 1 neuron, respectively. The full NN architecture can be seen in Fig. 7.7.

A separate NN is created for each output channel. Data is divided into 3 classes: training data, validation data, and test data. The training data is used to train the NN to minimize the Mean Square Error (MSE) loss function. The validation data is used to determine which parameters provide optimal performance after using the training data. The test data is used to evaluate the trained model. The model is trained by minimizing the MSE loss function using stochastic gradient

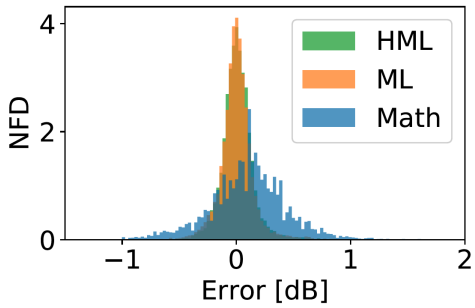
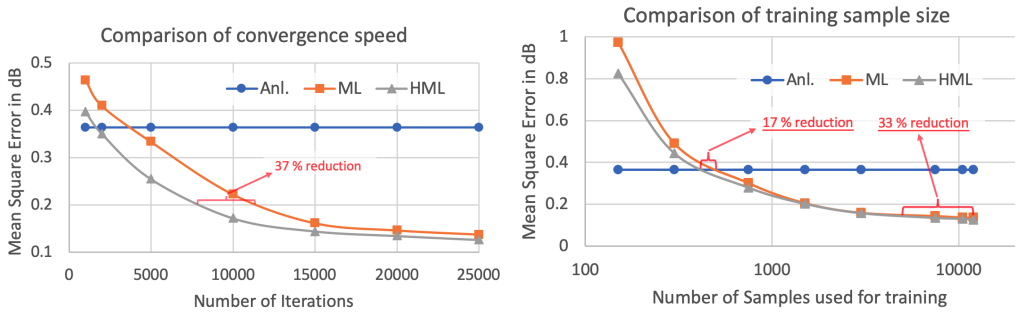


Figure 7.8: Normalized Frequency Density of Analytical, ML, and HML models after 5000 iterations.



(a) (b)

Figure 7.9: EComparison of models in (a) convergence speed, and (b) size of samples.

descent with backpropagation with m = 60 mini batch size and a learning rate α = 0.00025.

In the experiment, the EDFA is configured to have 18 dB gain and the input channels are randomly set to be 'on' or 'off', with -18 dBm +/- 6 dB variation in optical channel power for each 'on' channel. Three dimensions of machine learning system performance are evaluated, as shown below.

7.5.2 Hybrid Machine Learning Performance

First is the accuracy of HML model with abundant data samples (12,000 samples is used) and unlimited training time (25,000 iterations is used). The mean square error (MSE) of the predicted channel power is 0.362 dB (analytical), 0.160 dB (ML), 0.144 dB (HML), in which HML has a 10.5% reduction of error, compared with ML model. In the worst case, HML has 1.15% of high error (error > 0.5 dB), which is lower than ML (1.63%).

As a result, the ultimate performance of HML is slightly better than ML given unlimited data

and time. However, when data and time are limited, HML has a much better performance than ML. In Fig. 7.8, the normalized frequency density of error shows that HML has much narrower error distribution than ML, when the number of iterations is limited to 5,000 (originally 25,000, training sample size is the same as original). From Fig. 7.9(a), the convergence process of the models is shown, where the HML has a faster speed. In fact, considering a target MSE of 0.2 dB, HML requires 37% less time than the ML model (with the same ML configuration).

The third dimension is the number of training samples required. In Fig. 7.9(b), the MSE with different numbers of samples are shown for the three models. With fewer training samples, HML can achieve the same performance as ML. For example, to achieve the ultimate 0.134 dB MSE by the ML model, HML needs 33% fewer training samples. In another example, in order to surpass the prediction accuracy of the analytical model, HML needs 17% fewer training samples than the ML model.

Greater accuracy in the channel power prediction for EDFAs is expected to improve the accuracy of channel performance or QoT estimation, which is critical for the control and management of optical systems. In a typical optical system, the accuracy of power control and measurement is usually around 0.1 dB, which means that the 0.144 dB prediction error of the HML model is close to the system limitation. Although the HML can largely reduce the training sample size and training time relative to the ML approach, it still requires a longer data capturing process, compared to the analytical model, and further progress is needed to reduce this training process, for example through the use of transfer learning.

Part IV

Adaptive Multicast Services

Chapter 8: Light-Weight Feedback for Wireless Multicast

This part of the thesis focuses on large scale content delivery via wireless multicast for both WiFi and cellular networks. In this chapter we address the challenge of light-weight feedback for WiFi multicast.

8.1 Introduction

Current state of the art techniques using IEEE 802.11 for content delivery leverage either unicast or multicast data delivery. Commercial products [42, 231] rely on unicast for streaming the content to individual users. With standards such as 802.11ac offering total speeds up to 800 Mbps using multi-user MIMO, it is theoretically possible to serve video streams to hundreds of users. However, studies [159, 80] throw cold water on this promise. A large number of neighboring APs leads to hidden terminal problems and this coupled with increased interference sensitivity due to channel bonding, makes the entire approach highly susceptible to interference.

On the other hand, WiFi multicast services are rarely used by practical content delivery applications. Standard WiFi broadcast/multicast frames are transmitted at a fixed and low bitrate without any feedback. This raises several known reliability and efficiency issues. While some commercial products [42] are experimenting with WiFi multicast deployments for crowded environments, there remain several challenges to its widespread adoption. In particular, a published IETF Internet Draft highlights several open technical problems for WiFi multicast [160]. High packet loss due to interference and the hidden node problem can significantly degrade service quality. On the other hand, transmitting at low bitrates leads to low network utilization. As described in Section 8.2, there are numerous studies that propose solutions for overcoming these limitations from two aspects. One aims to reduce the overhead of feedback information to the multicast sender. The other aims to

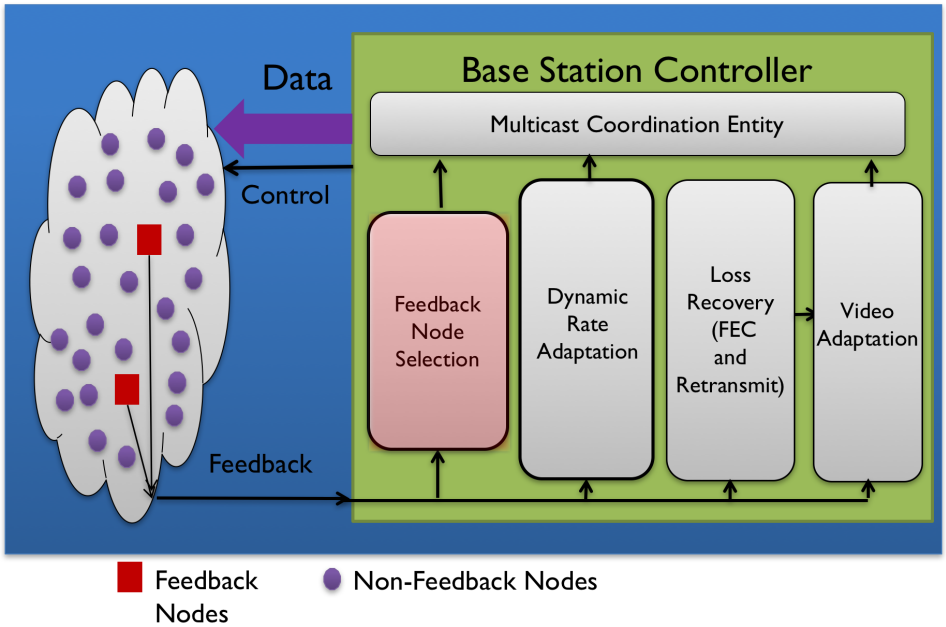


Figure 8.1: The *AMuSe* feedback mechanism (highlighted in red) as a part of the overall *AMuSe* system.

improve message reliability based on available feedback information. All the existing schemes, however, suffer from one or more issues including lack of scalability, inability to guarantee high service quality, or compliance with existing standards. Further, *none of the schemes have been tested experimentally at scale.*

We consider the use of WiFi multicast to address the challenge of providing scalable and efficient delivery of multimedia content to a very large number of WiFi nodes in a small geographical region (e.g., sport arenas, lecture halls, and transportation hubs). This is an attractive approach for delivering live video content to a dense user population that shares common interests (e.g., providing simultaneous video feeds of multiple camera angles in a sports arena).

The core challenge in providing such a service is collecting limited yet sufficient feedback from the users for optimizing the network performance. To address this challenge, *we introduce AMuSe (**A**daptive **Mu**lticast **Se**rvices), a low-overhead feedback mechanism which leverages the existing WiFi standards for tuning the network parameters, i.e., optimizing the network utilization while preserving Quality of Service (QoS) requirements. AMuSe* is based on the following hypothesis, which was reported in [11] and is validated in this chapter.

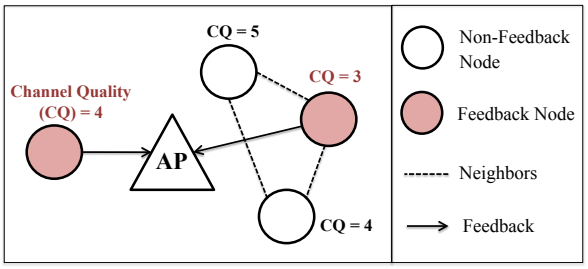


Figure 8.2: Feedback node selection by *AMuSe*. A node with the poorest channel quality in every neighborhood is selected as a Feedback node. Each feedback node periodically sends updates about the service quality to the Access Point.

Main Hypothesis: A cluster of adjacent nodes experience similar channel quality and suffer from similar interference levels. Hence, a node v with a worse channel condition than its adjacent neighbors can represent the service quality observed by the nodes in the cluster.

AMuSe dynamically divides the nodes in a network into clusters based on the adjacency of nodes and maximum cluster size (D m). In each cluster, one node is selected as a *Feedback (FB) node* and the FB node updates the AP about its service quality, e.g., channel quality (an example is shown in Fig. 8.2). The AP, in response, may take several actions such as[1]:

(i) **Rate Adaptation:** *AMuSe* can allow the APs to transmit multicast traffic at the highest possible bitrate while meeting constraints set by a network operator, i.e. ensuring high *Packet Delivery Ratio* (PDR) for a large fraction of the nodes.

(ii) **Tuning FEC:** We demonstrate in this chapter that ensuring 100% packet deliveries to all nodes is challenging. In large multicast groups, even a small amount of packet losses at nodes could lead to large packet retransmissions. In such situations, dynamically tuning application-level FEC might be a more suitable option. Feedback from *AMuSe* can be used to adjust the amount of FEC dynamically.

(iii) **Detecting Interference:** *AMuSe* collects detailed packet statistics which can be used to identify causes of packet loss in the network such as collisions and noise. For instance, packet losses that occur at the same time at multiple nodes can help pinpoint the location of the interference.

[1]The actions of the AP will require changes only at the AP side which is relatively straightforward.

AMuSe can be implemented as a light-weight application on any WiFi enabled device with minor or no modifications to the receiver devices and does not require changes to the existing 802.11 standard. The *AMuSe* feedback mechanism allows multicast service operators to balance between the number of FB nodes, the accuracy of the feedback, and the overall convergence time by controlling *AMuSe* parameters, such as the cluster radius D. *AMuSe* ensures that every node is at most D m away from an FB node with similar or weaker channel quality. To ensure sparse FB node density, any pair of FB nodes are at least D m apart which results in low communication overhead. The problem of selecting FB nodes which meet the above requirements is a variant of the well known *Minimal Independent Dominating Set* problem [131]. Although this problem is NP-hard, we prove that *AMuSe* can find a solution with a small constant approximation ratio.

We evaluated *AMuSe* on the large-scale ORBIT testbed [151] using over 200 WiFi nodes by implementing *AMuSe* on the application layer at each device. In all of our experiments, one node served as the AP and it sent a continuous multicast flow to all the other nodes, which acted as receivers. We first study the variation of channel quality metrics in different scenarios, (e.g., varying external interference levels, different transmission bit rates). The observations from these experiments serve as guiding principles for the design of *AMuSe*.

We observe that during any experiment, some nodes, which will be defined as *abnormal nodes*, suffer from low PDR, even when the AP is transmitting at a low bitrate and there is no external interference. Furthermore, this set of abnormal nodes varies across experiments.

We collected detailed channel and service statistics from all the nodes. They include the *Link Quality*[2] (LQ) reported by each node's WiFi card as representative of its observed received signal strength (RSS), its PDR, and its distance from the AP. Our preliminary evaluations show only moderate correlation between the nodes' LQ and the experienced PDR and a weak correlation between the nodes' distance from the multicast AP and the PDR values.

To validate the Main Hypothesis, we consider all the possible clusters with radius 3 and 6 m and calculate the Standard Deviation (STD) of the LQ and PDR values in the clusters at different

[2]Although LQ is not a standard measurement metric, we observed that the reported LQ by the Atheros chipsets indicates the RSS in db normalized to a reference value of -110 dBm (thermal noise).

bitrate and noise-levels. Our experiments indeed show low LQ and PDR STDs between the nodes in a cluster. However, as we increase the transmission bitrate or the noise level, we observe an increase in STD for the PDR values. We also notice that clusters with a small radius have lower LQ and PDR STDs than larger clusters.

We assess the feedback reports produced by *AMuSe* when the channel quality is evaluated according to the nodes' LQ, PDR, or a combination of them. These variants are denoted as AMuSe-LQ, AMuSe-PDR, and AMuSe-Mix respectively. We compare their performance to other feedback node selection schemes; *K-Worst* [218, 39], which selects the receivers with the worst channel condition as FB nodes, and *Random*, which selects a fixed number of random FB nodes. To evaluate the quality of an FB node selection, we compute the number of non-FB nodes that experience PDR value strictly lower than their respective FB node. We refer to these nodes as *Poorly Represented Nodes* (PRNs). We show that AMuSe-PDR and AMuSe-Mix produce a negligible number of PRNs and they outperform the other schemes when evaluated with different multicast bitrates and various noise levels. AMuSe-LQ and K-Worst have comparable performance, and are significantly better than the Random scheme.

Furthermore, we assess the performance of *AMuSe* as a service quality predictor in the event of environment changes. More specifically, we first select the FB nodes of the different variants at a given network setting. We then, compute the number of poorly represented nodes when using the same FB nodes, but after changing the multicast bitrate or the noise-level. We observe that at low bitrates AMuSe-LQ has slightly less PRNs than AMuSe-PDR, while AMuSe-PDR has similar performance to K-Worst. We notice a different trend when operating at a high multicast bitrate, in which AMuSe-PDR outperformed AMuSe-LQ and K-Worst. In all evaluations AMuSe-Mix was the best variant while Random, suffered from a very high number of PRNs. We explain these observations and provide additional results in Section 8.6.

Our experimental results demonstrate the ability of *AMuSe* to effectively provide feedback about the performance and quality of wireless multicast services. In turn, this feedback can be used for tuning the network parameters (e.g., rate adaptation, FEC configuration, and interference

classification) to optimize multimedia content delivery.

We discuss related work in Sections 8.2. We describe the network settings and our objectives in Sections 8.3 and 8.4 respectively. We present testbed evaluation of the design of *AMuSe* in Section 8.5 and the experimental results of evaluating channel quality metrics in Section 8.6. Finally, the evaluation of the performance of *AMuSe* is presented in Section 8.7 for both the static and dynamic cases.

This chapter was published in the proceedings of IEEE ICNP'13 [20] and appeared in IEEE/ACM Transactions on Networking [71]. The work on *AMuSe* project started in collaboration with Bell Labs, Nokia with Dr. Yigal Bejerano, Dr. Katherine Guo, and Dr. Thyaga Nandagopal. Collaborating Ph.D. students Dr. Jaime Ferragut and Dr. Varun Gupta made numerous important contributions in the design and data analysis of experiments.

8.2 Related work

Various methods have been proposed for multimedia content dissemination to multiple receivers. They leverage either unicast or multicast data delivery. This brief overview describes the most relevant studies Commercial products [42, 231] rely on unicast for streaming content to individual users. This approach requires deployment of numerous APs and it does not scale to crowded areas. Alternatively, the basic 802.11 multicast mechanism without any node feedback simply sets the transmission bitrate to the lowest rate. Cellular networks also operate without any node feedback and set the transmission bitrate to a low value, assuming some nodes are located near the cell edge. Any multicast mechanism without feedback results in low network utilization.

Many of the schemes to improve multicast services are based on integrating Automatic Repeat Request (ARQ) mechanisms into the protocol architecture [33, 39, 111, 193, 218], adding Forward Error Correction (FEC) packets to the multicast stream [13, 36], or both [226]. Other studies propose rate adaptation mechanisms for improved network utilization [121].

In all cases, a key requirement is having appropriate feedback from the receivers regarding their observed service quality. These feedback mechanisms can be classified as follows: (i) *In-*

dividual Feedback from multicast receivers, (ii) Leader-Based Protocol with acknowledgements (LBP-ACK), (iii) *Pseudo-Broadcast*, and (iv) *Leader-Based Protocol* with negative acknowledgements (LBP-NACK).

Individual Feedback mechanisms require all receivers to send acknowledgements of received packets either at the link layer [193, 70, 217, 191, 218], the application layer [226], or using periodic updates [13]. With *More Reliable Groupcast* (MRG) [95, 60] from IEEE 802.11 working group, each receiver transmits a bit-map of correctly received packets. Using this feedback, the sender determines lost packets and retransmits them to the group. This approach offers reliability but incurs high feedback overhead with large groups. The other three approaches reduce this overhead as follows.

The LBP-ACK approach [209, 218] provides scalability by selecting a subset of the receivers to provide feedback. The Pseudo-Broadcast approach [33, 39, 158], converts the multicast feed to a unicast flow and sends it to one leader, typically, the receiver with the weakest channel. The leader acknowledges the reception of the unicast flow. The other receivers receive packets by listening to the channel in promiscuous mode. The LBP-NACK approach [111, 120, 121] improves Pseudo-Broadcast by allowing the other receivers to send NACKs for lost packets. After receiving the ACK from the leader, the sender can infer successful transmission to all receivers since an NACK would collide with the leader's ACK.

With LBP-ACK and Pseudo-Broadcast, the selection of the leader(s) or subset of the receivers to provide feedback, can compromise service reliability. In Fig. 8.3(a), the leader v acknowledges a packet on behalf of node u, even though node u suffers from external interference that prevents correct reception of the packet. In Fig. 8.3(b), the node u might have an uplink transmission collide with the multicast packet from the AP, but since the leader correctly receives the multicast packet, the AP thinks the transmission has succeeded.

The LBP-NACK scheme requires changes to the standard and suffers from lack of reliability since a non-leader cannot reply with a NACK if it cannot identify a corrupted packet. Furthermore, due to the capture effect, the AP may be able to decode the ACK and ignore NACK messages.

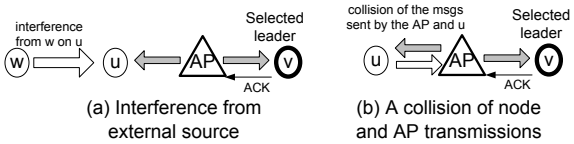


Figure 8.3: Unreliable packet delivery by the LBP and the Pseudo-Broadcast approach.

Table 8.1: Multicast: Features of related work

	Scalable (a)	**QoS Guarantees (b)**	**High Util. (c)**	**Standards Compatible (d)**	**Low Cost (e)**
Unicast	x	√	x	√	x
Basic multicast	√	x	x	√	√
Individual Feedback	x	√	x	x	√
Pseudo Broadcast	√	x	x	√	√
LBP-NACK	√	x	x	x	√
AMuSe	√	√	√	√	√

A major drawback of the LBP-NACK scheme is lack of fine-grained information about packet losses. Consider an example with 100 nodes in a multicast group, each with PDR of 99%. The expected fraction of packets for which NACK messages are received is $1 - .99^{100}$, which translates to roughly 63% of the packets. Thus, even in the case of network performing well, the AP observes poor performance.

Table 8.1 summarizes the main features of existing approaches. In summary, at least one of the following weaknesses hinders their performance: (i) requirement of feedback from a large number of receivers, (ii) ignorance of AP to interference-related packet loss, (ii) low network utilization to compensate for lack of feedback information or due to abnormal nodes, (iv) requirement of changes to standard WiFi protocol, or (v) expensive deployment of numerous APs. This motivates our desire for a scalable solution that improves reliability of multimedia delivery for WiFi deployments.

8.3 Network Setting

We consider an IEEE 802.11 WLAN and focus on a single AP serving a dense deployment of WiFi devices or *nodes*. A multicast server sends data to the AP and the AP transmits this data using multicast to all the nodes in its transmission range. There could be several sources of external interference in the network including transmissions from nodes within the network, adjacent APs, and nodes outside the network.

We follow the model where a node may report its service quality (e.g., channel quality) to an AP or multicast server. The AP or the multicast server, in response, may decide to adjust the FEC, adjust the transmission bitrate, retransmit lost packets, or execute a combination of the above. In practice, the AP and the multicast server are two separate logical entities and may reside in multiple network layers. Only the AP, however, is responsible for adjusting the network layer parameters. To simplify presentation, in the rest of the chapter we refer to AP as a representation of the combination of an AP and a multicast server.

At any given time, each node is associated with a single AP and nodes are assumed to have a quasi-static mobility pattern. In other words, nodes are free to move from place to place, but they tend to stay in the same physical locations for several minutes or more. This is a reasonable assumption for various crowded venues, such as sports arenas or transportation hubs. We assume that mobile devices can estimate their locations (e.g., by using one of the methods in [138]) with an accuracy of a few meters, and also determine if they are *static*[3] or *mobile*.

8.4 Objective

We focus on designing a light-weight feedback mechanism for supporting scalable WiFi multicast services for a very large number of nodes. This allows APs[4] to monitor the network conditions and to take appropriate actions for improving the multicast service quality while meeting various

[3] We consider a node static, if its movement is restricted to a few meters.

[4] To simplify our presentation, we assume that *AMuSe* is implemented as a software module on the APs. In practice, *AMuSe* can be realized as an independent server or even a cloud service.

service delivery constraints. We rely on the following observation reported in [11]:

Observation: *A cluster of adjacent nodes experience similar channel quality and suffer from similar interference levels. Hence, a node v with worse channel condition than its adjacent neighbors can represent the service quality observed by the nodes in the cluster.*

Based on this observation, the nodes can be grouped into clusters of adjacent nodes and a single *Feedback (FB) node* from each cluster can represent that particular cluster. The FB node can be used to report the channel quality of the cluster to the AP. Our feedback mechanism should ensure the following requirements:

(i) The FB nodes should accurately represent the network conditions in their neighborhood. This implies that the channel state experienced by non-FB nodes should not be significantly worse than the channel state reported by FB nodes.

(ii) The FB nodes should be well distributed throughout the network. In other words, the distance between the FB and non-FB nodes should be small. This ensures that the AP is informed about any interference even if it affects a small area.

(iii) The FB nodes should be responsive to changes of the service condition and should accurately report the impact of environmental changes, such as the multicast bitrate or external interference.

We now provide a formal definition of our objective. Given any FB node selection scheme and assume that every non-FB-node is represented by a single FB-node, typically the closest FB-node. A non-FB-node is considered a *Poorly Represented Node* (PRN) if its PDR is $\epsilon > 0$ below the PDR of its representing FB-node. We refer to ϵ as the *PRN Gap*. Consequently, our objective can be defined as follows;

Objective: Consider an upper bound on the number of FB nodes or their density[5] as well as a fixed PRN-Gap $\epsilon > 0$. Design a low-communication FB node selection mechanism that minimizes the following metrics:

[5]The FB node density can be enforced by requiring a minimal distance D between any two FB nodes.

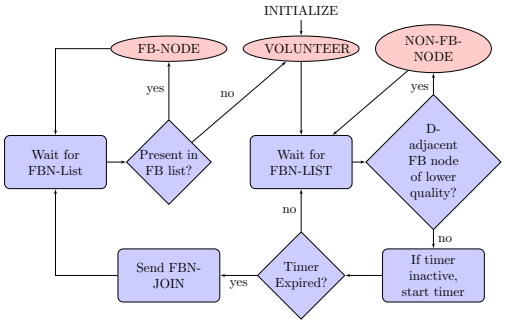


Figure 8.4: State diagram of the *AMuSe* FB node selection algorithm at each node. All nodes initialize in the VOLUNTEER state.

- Number of PRNs in normal operation as well as after environment changes, e.g. bitrate or noise level changes.
- Maximum distance between a non-FB-node and its representing FB node.

8.5 The *AMuSe* Feedback Mechanism

This section provides an overview of the *AMuSe* feedback mechanism. For any given D we define two nodes to be *D-adjacent* if they are separated by a distance of at most D. In order to find a small set of FB nodes that can provide accurate reports, *AMuSe* should satisfy the following requirements.

(i) Each node should be D-adjacent to an FB node.

(ii) An FB node must have similar or weaker channel quality than its D-adjacent nodes.

(iii) Any two FB nodes cannot be D-adjacent.

In order to evaluate the channel quality, various metrics can be considered, including *Received Signal Strength* (RSS), *Signal-to-Noise Ratio* (SNR) and *Packet Delivery Ratio* (PDR). We experimentally compare LQ^2 and PDR as channel quality metrics in Section 8.6.

8.5.1 The Feedback Node Selection Algorithm

We present a semi-distributed process for FB node selection, where some nodes volunteer to serve as FB nodes, and the AP selects the best candidates. If node location information and observed channel quality are known, then the AP can easily select the ideal set of FB nodes. Yet, this is not feasible in practice for large groups. Hence, we seek to minimize the number of nodes that send their information to the AP as part of the FB node selection process, while ensuring that a small set of FB nodes meeting the above requirements is selected.

The AP periodically (e.g., once every $\tau_{AP} = 500$ ms in our experiments) multicasts an FBN-LIST message with a list of FB nodes (these messages can be sent multiple times for reliable transmissions and do not incur overhead, since they are 1-2 packets long). Each entry in the FBN-LIST contains the node ID[6], its reported location[7], its reported channel quality, and a measure of the PDR[8].

Each node is in one of three states:

- **FB-NODE -** A node that has been selected as FB node.
- **VOLUNTEER -** A node that is not aware of any D-adjacent FB node with lower or similar channel quality and can serve as an FB node.
- **NON-FB-NODE -** A node that either is in a transient state or is aware of a D-adjacent FB node with similar or lower channel quality.

Fig. 8.4 presents the state transition diagram for each node. When a node v joins the network, it is in the VOLUNTEER state. The node waits for an FBN-LIST message, and checks if there are any D-adjacent FB nodes in this list with similar or weaker channel quality. If there are any such nodes, node v switches to the NON-FB-NODE state and records the list of D-adjacent FB nodes in the FBN-LIST message with similar or weaker channel quality.

[6]Nodes can be assigned temporary virtual IDs to maintain privacy.

[7]Relying on a user to be truthful about its location/channel quality could lead to denial-of-service attacks. Yet, we shelve this orthogonal discussion.

[8]This can be easily changed to report the last acknowledged packet sequence number to support finer granularity of message reliability.

If there are no such nodes, node v starts a random back-off timer for a period chosen in the interval $[0, T]$ (our experiments use the maximum receiver back-off timer $T = 5$ seconds). The random timer solves the problem of many nodes overwhelming the WiFi channel and AP with FBN-JOIN messages in the situation of changes in channel condition. During this countdown, if node v learns of a D-adjacent FB node from a FBN-LIST message, then it cancels its countdown, and switches to a NON-FB-NODE state. Otherwise, upon expiry of the timer, it sends a FBN-JOIN message to the AP, and waits to see if its ID appears on the next FBN-LIST. The FBN-JOIN message contains the node ID, node location, and the observed channel quality (e.g., the node PDR and LQ). If node v appears on the FBN-LIST, it switches to the FB-NODE state. If not, it repeats the back-off process again until it leaves the VOLUNTEER state. At any time, upon receipt of an FBN-LIST message, if an FB node v does not find itself on the FBN-LIST, it ceases to be in the FB-NODE state. In this case, the node returns to the VOLUNTEER state and waits for the next FBN-LIST to either (i) switch to the NON-FB-NODE state due to the existence of a D-adjacent node of lower quality, or (ii) send the FBN-JOIN message again after the back-off timer expires.

An important property of this FB node selection algorithm is that the FB node selection is done in a semi-distributed manner, since a node volunteers to serve as an FB node, only if there is no other FB node in its vicinity with weaker channel quality. Thus, the AP is only responsible to resolve conflicts when several D-adjacent nodes volunteer simultaneously and to prune unnecessary FB nodes. Consequently, after receiving FBN-JOIN messages and before sending a FBN-LIST message, the AP runs the *node pruning algorithm*, described in Section 8.5.3 to decide which nodes are FB nodes.

Each FB node periodically (e.g., once every $\tau_{FB} = 500$ ms in our experiments) sends REPORT messages to update the AP about the channel and service quality experienced by the node, and thus its representative cluster. If the AP does not receive any message from one of the FB nodes for a given duration, (for example, $3\tau_{FB}$ used in our experiments), then the AP removes it from the list of FB nodes.

A few aspects of the *AMuSe* feedback are worth pointing out.

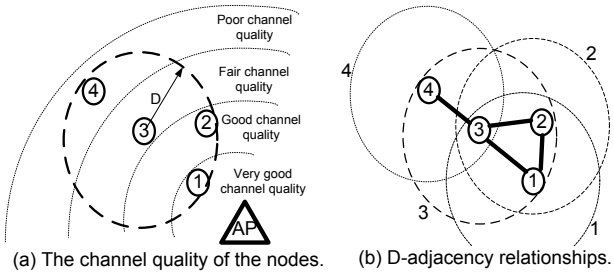


Figure 8.5: An example of a wireless network a single AP and 4 receivers. All 3 requirements described in Section 8.5 for an accurate feedback selection are important for this example.

(i) *AMuSe* does not require the nodes to listen to all the traffic on the network. All they have to do is listen to the AP on the multicast group address. This conserves energy at the receivers.

(ii) *AMuSe* does not require the location information for nodes to be very precise. As mentioned in Section 8.3, coarse granularity is acceptable, as long as the accuracy is in the order of few meters, which has been demonstrated by some studies as feasible and practical [37].

(iii) *AMuSe* provides variable levels of reliability by fine-tuning the combination of AP node selection frequency τ_{AP}, the receiver reporting frequency, τ_{FB}, the maximum receiver back-off timer T, and the node adjacency distance D. *AMuSe* can ensure more reliable and frequent reports at a cost of more overhead. Instead of a single control, *AMuSe* provides multiple control knobs, giving greater flexibility to the operator to provide different types of service for various multicast streams.

(iv) Fourth, as described above, *AMuSe* reports can be used for optimizing different aspects of WiFi multicast services, such as rate-adaptation, FEC configuration and interference classification. To this end, the REPORT messages may carry different information. For instance, in [20] we showed that PDR and LQ information is sufficient for performing rate adaptation, while reporting about received and lost packets is required for interference classification.

8.5.2 Illustrative Example

Consider the network shown in Fig. 8.5(a) with a single AP and four receivers. Assume that numbers labeling the nodes denote their IDs and the order in which they join the multicast service at this AP. There are four different channel quality levels: very good, good, fair and poor as experienced by node 1, 2, 3, and 4 respectively. Fig. 8.5(b) shows a circle with radius D around every node, say node v, where each node, u, inside the circle of v is D-adjacent to node v. Hence, nodes u and v are considered neighbors to one another.

In this example, we demonstrate the importance of all three requirements mentioned at the beginning of this section on the quality and density of the set of FB nodes. Assume first that the FB nodes have to meet only requirement (i) and (ii), but not (iii). Under these guidelines, at the moment each node joins the multicast, it has a weaker channel quality than all its neighbors, and therefore, it is selected as an FB node. At the end of the process, the network contains four FB nodes. It is easy to see that this approach does not scale for large groups.

Now, let us assume that requirement (iii) is enforced. Right after a node joins the network, the set of FB nodes is optimized. When node 1 joins, it becomes the FB node. After node 2 joins, node 2 becomes the FB node, while node 1 becomes a non-FB node because of (iii). After node 3 joins, it becomes an FB node while both node 1 and 2 become non-FB nodes because all three nodes are D-adjacent to one another. After node 4 joins, it becomes an FB node, while node 3 becomes a non-FB node. In addition, node 2 becomes an FB node again. Notice that node 2 switches state twice, after node 3 and 4 joins respectively. However, after each node joins the multicast group, the set of FB nodes is optimal.

This example shows that while *AMuSe* FB node selection algorithm satisfies all three requirements, it may cause churn as nodes enter and leave the FB-NODE state. We show next that the selected set of FB nodes is near-optimal when the set of nodes receiving the multicast do not change.

8.5.3 The Node Pruning Algorithm

As described above, the FB node selection process ensures that every receiver is D-adjacent to a *candidate* node with similar or weaker channel condition. The list of candidates at the AP contains the current FB nodes as well as the nodes in the VOLUNTEER state. Thus, the AP is responsible to trim unnecessary candidates to select a small set of FB nodes such that any pair of nodes in the set are not D-adjacent.

The problem of finding the minimum set of FB nodes that meets the three requirements above is a variant of the *minimum dominating set problem*, which is a known NP-complete problem even in the case of unit disk graph [131]. Below we present a heuristic algorithm that selects a near optimal set of candidates that meet our three requirements.

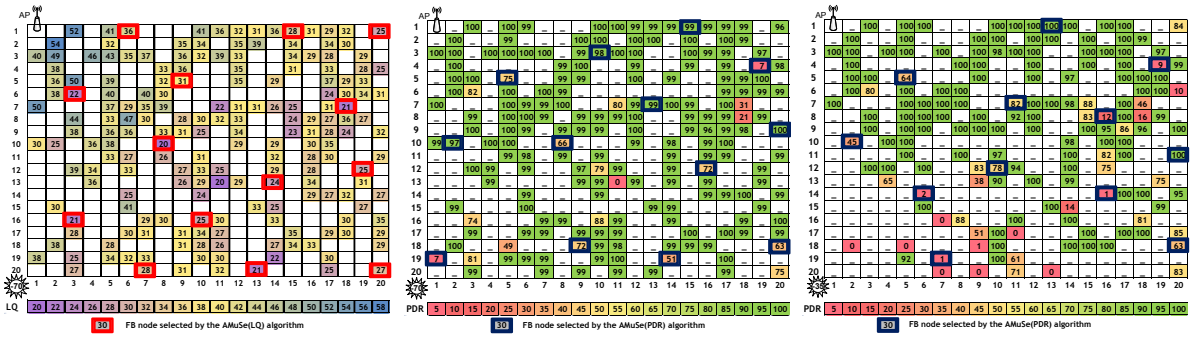
The heuristic algorithm: The AP creates a list L of the candidates sorted in increasing order according to their channel quality. Then, it iteratively selects the first candidate v in L as an FB node and remove v and all its D-adjacent nodes from L. The algorithm ends when L is empty.

Let F denote the FB nodes selected by the heuristic algorithm and OPT denote the optimal set of FB nodes among all nodes, our algorithm ensures the following property:

Proposition 1. $|F| \leq 5 \cdot |OPT|$. *If the channel quality is a monotonic decreasing function with the distance from the AP then* $|F| \leq 3 \cdot |OPT|$

For proof see Appendix 8.A.

Stability vs. optimality trade-off: As illustrated in Section 8.5.2, a naive implementation of the heuristic algorithm may cause churn of FB nodes, which obstructs system stability. Since node pruning is done by the AP, the algorithm can be easily modified to prevent churn, for instance by giving higher priorities to already selected FB nodes or relaxing the distance constraint between FB nodes. In our experiments, we also observed rapid switching of FB nodes due to minor variations in channel qualities. In this case, ensuring that the difference between channel quality of a non-FB and FB node is greater than some value greater than zero before a non-FB node volunteers is an effective solution. Although striking a proper balance between system stability and optimality of



(a) Link Quality Heatmap, noise = -70 dBm.

(b) Packet Delivery Ratio Heatmap, noise = -70 dBm.

(c) Packet Delivery Ratio Heatmap, noise = -35 dBm.

Figure 8.6: Link Quality (LQ) and Packet Delivery Ratio (PDR) heatmaps at the AP for $D = 6$ meters with transmission bitrate of 12 Mbps and noise level of -70 dBm and -35 dBm. The FB nodes are highlighted with a thick border in red in the LQ heatmap and in blue in the PDR heatmap. Empty locations represent nodes that did not produce LQ or PDR reports and they are excluded from our experiments. Nodes with $PDR = 0$ are active nodes that reported LQ values but were unable to decode packets. These nodes are excluded from the FB node selection process. Note that the minimum threshold below which a node does not become an FB node is configurable.

the FB node selection is a central topic in the design of *AMuSe*, it is beyond the scope of this thesis.

8.6 Experimental Evaluation of Testbed Environment

We validated *AMuSe* experimentally using the 400-node ORBIT testbed [151]. We describe these experiments in this section. We use the *Link Quality*[2] (LQ) metric reported by a node's WiFi card as representative of its observed RSS. We first consider the following set of auxiliary hypotheses used to validate our main hypotheses in Section 8.1.

H1: There is a correlation between the PDR and LQ values observed by a node.

H2: Clustered nodes experience similar LQ and PDR.

H3: Clustered nodes suffer from similar interference.

8.6.1 The ORBIT Testbed and Experiment Settings

The ORBIT testbed [151] consists of a dynamically configurable grid of 20×20 (400 overall) nodes each with an 802.11 radio. The grid separation between nodes is 1 meter and in addition,

Table 8.2: Evaluation Parameters

Parameter	Definition
LQ_i	Link Quality of node i with the AP.
P_i^{vec}	A vector of the packets received by node i.
(x_i, y_i)	(row, column) location of node i.
TX_{AP}	Broadcast/Multicast transmission rate at the AP.

the testbed provides a noise generator with four noise antennas at the corners of the grid whose attenuation can be independently controlled, permitting the emulation of a richer topology. In order to avoid performance artifacts stemming from a mismatch of WiFi hardware and software, we select the subset of nodes equipped with *Atheros 5212/5213* wireless cards with *ath5k* wireless driver. Furthermore, we remove unresponsive nodes (nodes with hardware issues) in the grid before every experiment. This results in approximately 200 nodes participating in each experiment.

We implemented the *AMuSe* system as an application layer program for the AP and the clients, running on all nodes. Each node is identified by its (row, column) location. The node at the corner $(1, 1)$ serves as a single multicast AP, configured in master mode, and it uses channel 40 of 802.11a[9] to send a multicast UDP flow with a transmission power of 1 mW= 0 dBm. The other nodes are the multicast receivers, configured in managed mode. This means that in practice our experiments consider *at most a quarter of the transmission range of an AP*. Each UDP packet is 1400 bytes in payload length and the payload data contains sequence number for each packet in order to identify missing packets at the nodes. While we consider a single multicast group in our experiments, *AMuSe* can allow for monitoring of several multicast groups individually. If several multicast groups should be monitored together, then a control multicast group can be setup.

Every node keeps track of the parameters described in Table 8.2, which we process off-line after each experiment. The received or dropped packets are marked by 1s or 0s respectively in a boolean vector P_i^{vec} stored at each node i. The *packet delivery ratio* (PDR) value of each node i is calculated from its P_i^{vec} vector. Note that the throughput measured at each node is a function

[9]We observed that channel 40 at the 5 Ghz band suffers from lower external interference levels on the ORBIT grid than the channels at 2.4Ghz band.

of the PDR as well as the bitrate and is different from the transmission throughput at the AP. The testbed hardware and software allows us to measure the LQ or RSS values from the user-space. The PDR values can be measured on any commodity hardware by measuring the received packets. It is possible that some environments such as iOS do not provide LQ or RSS information to the user-space. In such cases, *AMuSe* can rely on PDR measurements alone. As we show later, *AMuSe* with PDR measurements alone can provide reliable feedback.

8.6.2 Experiment Description

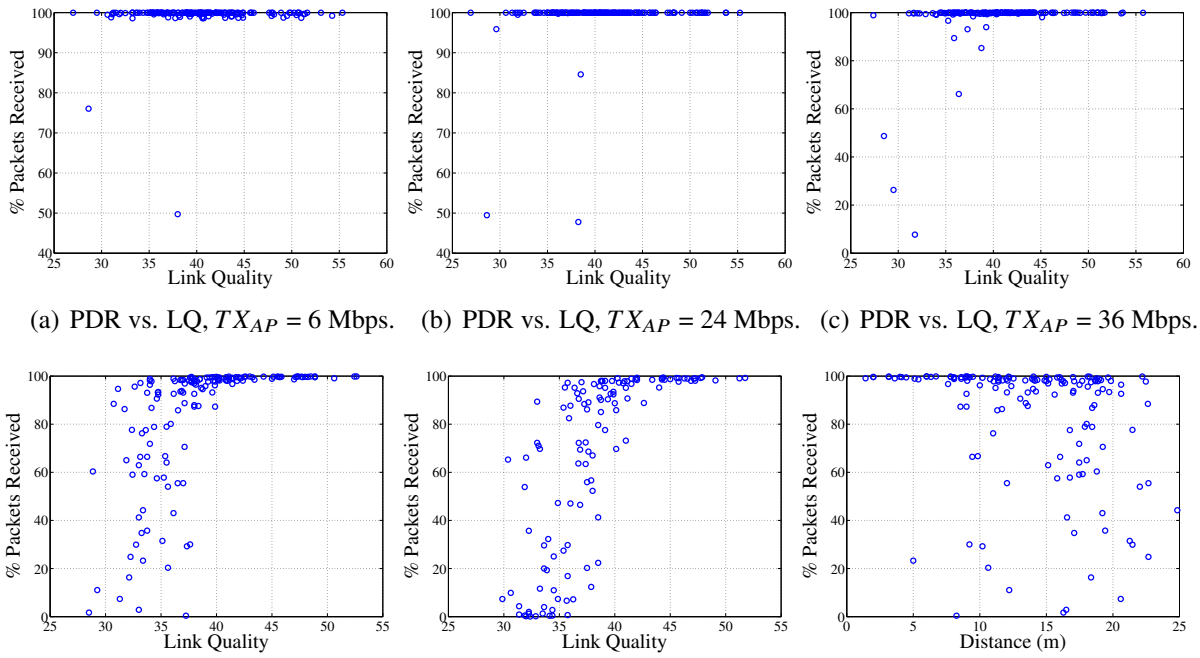
We now describe the types of experiments conducted to validate our hypotheses presented earlier in this section.

Different Bitrates: We fix the AP multicast transmission bitrate, denoted by TX_{AP}, to different values allowed by the card (6, 9, 12, 18, 24, 36, 48, 54 Mbps), each bitrate for a duration of 10 seconds. We repeat these experiments 10 times at different times of the day without any external noise.

Different Noise Levels: We fix the AP multicast transmission bitrate to 12 Mbps and turn on the noise generator near node (20, 1). The noise generator is configured to provide AWGN noise for the entire spectrum of channel 40. Starting with −70 dBm (low noise), we vary noise power in steps of 5 dBm up to −35 dBm (high noise).

Fig. 8.6 presents three sample heatmaps of one run of the experiments, when $TX_{AP} = 12$ Mbps and external noise of −70 dBm and −35 dBm generated near node (20,1). Each heatmap shows the active nodes used in the experiment and either the LQ or PDR values that they experienced, in addition to the FB nodes that the AP has selected with D-adjacency parameter of 6 meters. Nodes marked with thick red or blue border are FB nodes selected by the *AMuSe* scheme. Nodes with $PDR = 0$ are active nodes that reported LQ values but unable to decode packets in the experiment run. For example, node (13,11) with $LQ = 20$ and $PDR = 0$ in Fig. 8.6(a) and 8.6(b) for a noise level at −70 dBm. These nodes are excluded from the FB node selection algorithm.

An interesting observation is that a selected FB node v may have higher PDR (or LQ) values



(a) PDR vs. LQ, TX_{AP} = 6 Mbps. (b) PDR vs. LQ, TX_{AP} = 24 Mbps. (c) PDR vs. LQ, TX_{AP} = 36 Mbps.

(d) PDR vs. LQ, TX_{AP} = 48 Mbps. (e) PDR vs. LQ, TX_{AP} = 54 Mbps. (f) PDR vs. distance, TX_{AP} = 48 Mbps.

Figure 8.7: Experimental results for testing hypothesis H1 and verifying the presence of abnormal nodes.

than an adjacent non-FB node, say u. Such a situation results from the independent-set property of the selected FB nodes and it may occur if u is D-adjacent to another FB node with even lower PDR (or LQ) values. For instance, in Fig. 8.6(b) Node $(7, 13)$ with PDR of 99% was selected as FB node although it has a neighbor, Node $(7, 11)$, with PDR of 80%. The reason is that Node $(7, 11)$ is 6-adjacent to FB node $(10, 8)$ with PDR of 66%.

8.6.3 Hypotheses Testing

We turn to test our hypotheses based on the information collected from the experiments described in Section 8.6.2.

H1 - Correlation between PDR and LQ: Fig. 8.7(a)-8.7(e) demonstrate the correlation between the PDR of a node with respect to its LQ for different transmission bitrates without external noise, whereas, Fig. 8.7(f) shows the correlation between the PDR of a node with respect to its distance from the AP at a transmission rate of 48 Mbps. PDR values are close to 100% for almost all nodes

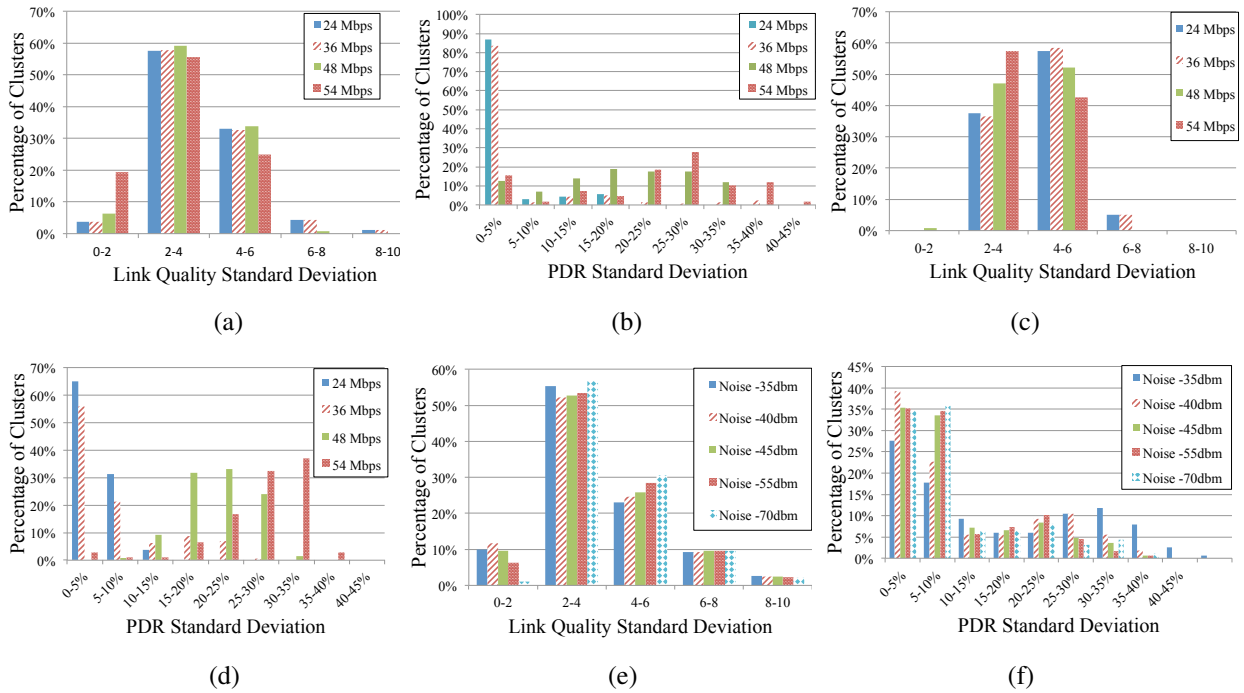
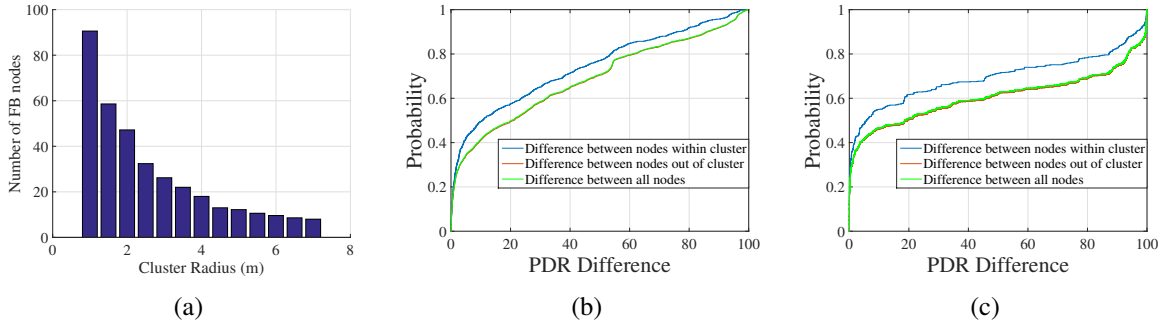


Figure 8.8: Experimental results for testing hypotheses H2——H3: (a) LQ STD: varying TX_{AP} without noise, cluster size = $3m$, (b) PDR STD: varying TX_{AP} without noise, cluster size = $3m$, (c) LQ STD: varying TX_{AP} without noise, cluster size = $6m$, (d) PDR STD: varying TX_{AP} without noise, cluster size = $6m$, (e) LQ STD: varying noise, TX_{AP} = 12 Mbps, cluster size = $3m$, and (f) PDR STD: varying noise, TX_{AP} = 12 Mbps, cluster size = $3m$.

for bitrates up to 24 Mbps (Fig. 8.7(a)-8.7(b)). Some degradation of PDR values is observed for bitrates of 36 Mbps (Fig. 8.7(c)) and even higher variance of PDR values are seen for 48 Mbps (Fig. 8.7(d)) and above.

Fig. 8.7(d) and Fig. 8.7(e) show that the correlation between the PDR and LQ is not very strong, suggesting that nodes with the same LQ value may have significantly different PDR. Fig. 8.7(f) illustrates very weak correlation between the PDR of a node and its proximity to the AP (with TX_{AP} = 48 Mbps), and some of the nodes adjacent to the AP suffer from low PDR. For instance, Fig. 8.7(f) shows that one of the nodes with distance of 5 meters from the AP suffers from PDR of 25%. This observed variation of PDR with LQ as well as variation of PDR with distance to the AP is consistent with prior work, e.g., [164],[210], [105] and [69].

H2 - Clustered nodes experience similar LQ and PDR: We measure the standard deviation (STD) of LQ and PDR without noise in each cluster radius of 3 and 6 meters on the grid, where



(a) (b) (c)

Figure 8.9: The impact of clustering: (a) the number of FB nodes for different cluster sizes, (b) CDF of PDR differences of pairs of nodes within and across clusters for no external noise and bitrate of 54Mbps, and (c) CDF of PDR differences of pairs of nodes within and across clusters for external noise of −30dBm and bitrate of 12Mbps.

each cluster contains an FB node and all its neighbors Histograms of the distribution of the LQ and PDR STD in different clusters are shown in Fig. 8.8(a)-8.8(d). We measure the same distributions in the presence of various noise levels with a cluster radius of 3 meters, and plot the results in Fig. 8.8(e) and Fig. 8.8(f). We expect the STD across clusters to be a good measurement of how similar the PDR and the LQ values are.

Fig. 8.8(a), Fig. 8.8(c), and Fig. 8.8(e) show that the LQ STD is very similar across all the bitrates regardless of the noise levels. This indicates that although adjacent nodes experience similar LQ (and similar RSS), the LQ metrics do not capture the effect of external interference and bitrate variation. By comparing Fig. 8.8(a) and Fig. 8.8(c), we see that a higher percentage of clusters report higher LQ STD for cluster size 6 m than with cluster size 3 m.

We now consider the distribution of the PDR STD values. Fig. 8.8(b) shows that with $TX_{AP} \leq$ 36 Mbps, only very few clusters show significant deviations ($>$ 5%) in PDR. This is because most nodes have PDR above 99% when $TX_{AP} \leq$ 36 Mbps as shown in Fig. 8.7. However, the variability of the PDR becomes evident at higher bitrates. By comparing Fig. 8.8(b) and Fig. 8.8(d), we observe that a higher percentage of clusters report higher PDR STD for cluster size 6 m as compared to cluster size 3 m. Further, we see in Fig. 8.8(d) that at higher bitrates, PDR STD is higher for a significant number of clusters.

As shown in Fig. 8.8(f), interference introduces noticeable deviations ($>$ 5%) in PDR across

nearly two-thirds of the clusters. To understand this, we revisit the heatmaps in Fig. 8.6(c). It is clear that the PDR values are decreasing for nodes near the bottom-left corner where the noise generator is located. The nodes which are not able to decode the AP beacons (at a bitrate of 6 Mbps) disconnect from the AP, are not shown on the heatmap, and are not included in the variance calculations. The nodes which report a 0 PDR value are the ones that fail to receive any multicast packet. These nodes are shown in the heatmap red with a 0 value. At higher noise levels, many more nodes report PDR values of 0. This explains the high levels of PDR variance observed in Fig. 8.8(f).

The increase in LQ and PDR STD with the cluster size point to the inherent tradeoff in FB node selection process using both LQ and PDR as the quality metrics. The system should ideally operate in a mode where a large fraction of the nodes experience high PDR and the PDR STD is very low. Increasing the cluster size reduces the number of FB nodes, however, leads to increased STD of quality metrics in clusters, particularly the PDR STD at higher bitrates. The average number of FB nodes for different cluster sizes is shown in Fig. 8.9(a). The FB overhead of *AMuSe* is directly proportional to the number of FB nodes. Each FB node, periodically sends an FB message which is roughly 100 bytes long. The frequency of feedback messages is application-specific e.g., for multicast rate adaptation application, 1s could be sufficient [73]. This implies that 50 FB nodes will add an overhead of 40Kbps. In our case, 50 FB nodes correspond to a cluster radius of 3m from Fig. 8.9(a). The FB overhead is much smaller than the multicast throughput measured at the AP (order of Mbps even for bitrate of 6Mbps). The above observations serve as a good motivation to carefully set the parameters for the FB node selection algorithm.

Finally, we demonstrate that clustering is not redundant by comparing the proximity of channel quality values within and across clusters. Fig. 8.9(b) shows the CDF of the PDR differences between pairs of nodes inside and across clusters for bitrate of 54Mbps and no noise for a cluster radius of 3m. We chose bitrate of 54Mbps for ease of exposition. Roughly 60% of the node pairs have PDR differences less than 20% within a cluster while fewer than 50% of pairs have differences less than 20% across clusters. Similarly, Fig. 8.9(c) shows the CDF of the PDR differ-

ences between pairs of nodes inside and across clusters for bitrate of 12Mbps and external noise of −30dBm for a cluster radius of 3m. In this case also, the differences are similar. These results show that clustering is effective in grouping nodes with similar channel qualities.

H3 - Clustered nodes suffer from similar interference: Fig. 8.6 shows that external noise has a largely local effect near the noise source. Moreoever, Fig. 8.8(f) shows that even with a small cluster size of 3 meters, the PDR STD can be high due to external interference. The above two observations validate the need for a well-distributed and non-sparse set of FB nodes to report the values of quality metrics in order to reflect the interference experienced by receivers.

Our experiments also show that increasing TX_{AP} has an impact on all nodes, and that beyond a certain bitrate, the PDR of many nodes drops below 90%, as shown in Fig. 8.7(d) and Fig. 8.7(f). Thus, it is critical to assign TX_{AP} appropriate values in order to improve the multicast service.

8.6.4 Abnormal Nodes

In general, we refer to a node with low PDR as *abnormal*. Specifically, in our experiments, a node is *abnormal* if its PDR is below the abnormal threshold $H = 90\%$. In contrast, a node is *normal* if its PDR is at least $H = 90\%$. In this section, we study the number of abnormal nodes as a function of the TX_{AP} and the link quality (LQ). Fig. 8.7(a)-8.7(d) show how PDR varies with LQ for each node in a single experiment run with TX_{AP} bitrates of $6, 24, 36$ and 48 Mbps respectively. Results from all values of TX_{AP} (including ones not shown here) show that the number of abnormal nodes increases with the increase of TX_{AP}.

In Fig. 8.7(a)-8.7(c), PDR values are close to 100% for a large fraction of the nodes for bitrates up to 36 Mbps. However, Fig. 8.7(a) demonstrates that even in the extreme case of very low TX_{AP} without any interference some of the nodes (two in this case) are abnormal and suffer from low PDR.

The set of abnormal nodes remained small when we increase TX_{AP} to higher bitrates until 36 Mbps, as shown in Fig. 8.7(b) and Fig. 8.7(c). The number of abnormal nodes increases significantly once TX_{AP} reaches 48 Mbps. Surprisingly, the set of abnormal nodes is not the same in all

experiments.

8.7 Feedback Node Selection

The primary objective of this section is to study the performance of feedback node selection schemes. We compare *AMuSe* FB node selection scheme with other schemes and in the process, validate our *main hypothesis* from Section 8.1. We consider the following schemes including the three flavors of *AMuSe* that select either the LQ, the PDR or a mix as the metric which is used by the AP for selecting FB nodes.

(i) AMuSe-LQ – *AMuSe* based on LQ.

(ii) AMuSe-PDR – *AMuSe* based on PDR.

(iii) AMuSe-Mix – *AMuSe* based on mix of LQ and PDR.

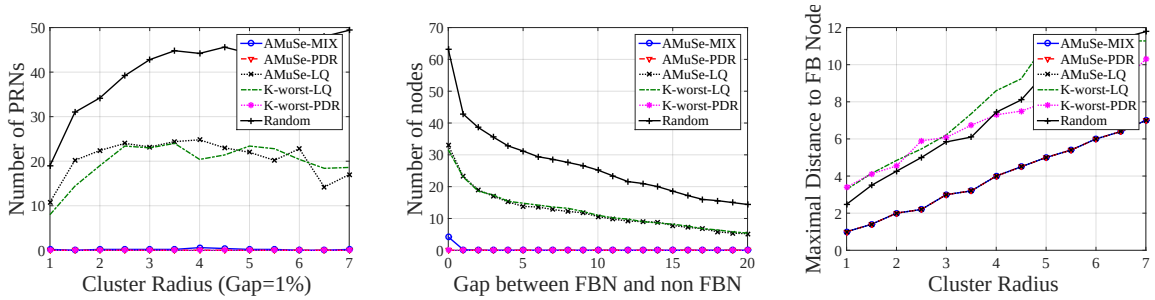
(iv) K-worst-LQ – K nodes with lowest LQ are FB nodes.

(v) K-worst-PDR – K nodes with lowest PDR are FB nodes.

(vi) K-random – K random nodes as FB nodes.

The AMuSe-Mix scheme relies on lexicographic ordering of PDR and LQ values for comparing channel quality. For nodes with PDR $> 98\%$, the ordering is based on LQ. For nodes with PDR $\leq 98\%$, the ordering is based on PDR. Thus, the channel quality is defined by the following tuple in lexicographic order: $(\min(PDR, 98), LQ)$ The motivation behind AMuSe-Mix lies in our observation that LQ is weakly correlated with PDR in Section 8.6. Very high PDR values ($> 98\%$) could result from random packet losses and small PDR variations above this value are unreliable indicators of difference in channel quality. Thus, we use AMuSe-Mix to study if LQ can be a better metric to distinguish nodes which have high PDR values.

Moreover, we study the parameter choices for cluster radius (represented by the adjacency parameter, D). When we refer to cluster radius D as a parameter for the Random, K-worst-LQ, or



(a) (b) (c)

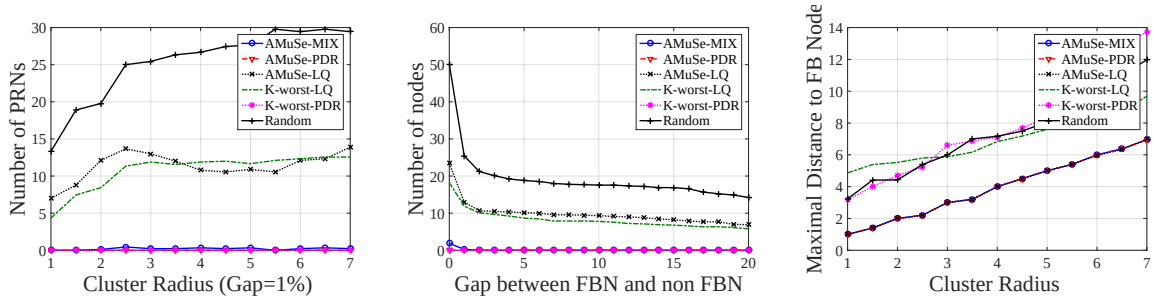
Figure 8.10: Static settings with bitrate of 48Mbps: (a) the number of Poorly Represented Nodes (PRN) vs. the cluster radius with fixed PRN-Gap of 1%, (b) PRN for different PRN-Gap and fixed cluster size of $D = 3$ m, and (c) maximal distance between an FB and non-FB node for various cluster radius.

K-worst-PDR schemes, we select as many FB nodes as *AMuSe* feedback schemes have (for a fair comparison).

We study the performance of different feedback nodes selection schemes under two network settings:

- **Static Settings:** The multicast bitrate and the external interference level are fixed.
- **Dynamic Settings:** In a dynamic environment of either (i) changing multicast bitrate, (ii) changing external interference, or (iii) emulated mobility.

For all our evaluations in both the static as well as the dynamic settings, we collected detailed packet traces at each node in the testbed for several bitrate and interference conditions. The number of nodes in the experiments was kept similar between 170 to 200 to avoid any performance mismatch. All the results for varying bitrate conditions were averaged over five runs of 10s at each bitrate. We ensured the appropriate setting of controlled interference by measuring the interference on a spectrum-analyzer on the testbed. During our experiments we observed sporadic spikes of uncontrolled interference. For mitigating their impact, we consider only time instants when there was no uncontrolled noise in our evaluations.



(a) (b) (c)

Figure 8.11: Static settings with external noise: (a) the number of Poorly Represented Nodes (PRN) vs. the cluster radius with fixed PRN-Gap of 1%, (b) PRN for different PRN-Gap and fixed cluster size of $D = 3$ m, and (c) maximal distance between an FB and non-FB node for various cluster radius.

8.7.1 Static Settings

We first study the performance of different feedback schemes while the multicast bitrate and the generated external noise level are fixed. This setting allows us to evaluate the various schemes under normal network operation in stable conditions. We repeat our experiments with different bitrates and noise levels. We present our results for 3 different cases.

(i) **Fixed bitrate of** 36 **Mbps –** The optimal bitrate at which most of the nodes experience PDR close to 100 and only a few nodes suffer from low PDRs, as shown in Fig. 8.7(c).

(ii) **Fixed bitrate of** 48 **Mbps –** Above the optimal bitrate many nodes experience low PDR, as shown in Fig. 8.7(f).

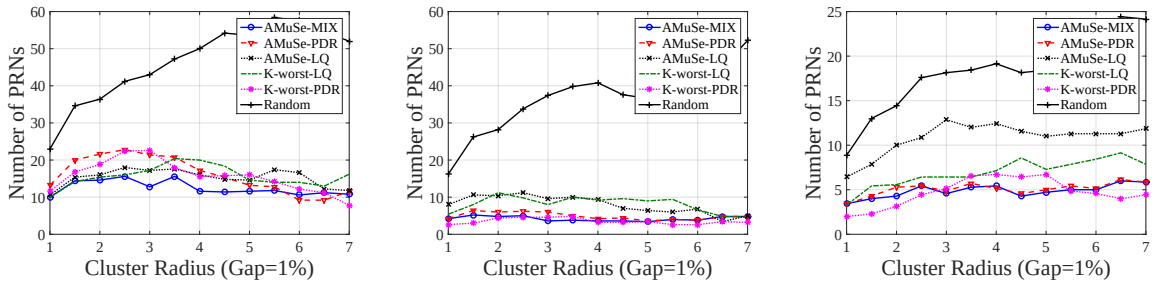
(iii) **External Noise –** The bitrate is set to 12 Mbps and the receivers suffer from different interference levels between −70dBm to −35dBm. The interference is concentrated on one corner of the grid as in Section 8.7.1.

The results of our evaluation are presented in Figs. 8.10-8.11. Figs. 8.10(a) and 8.11(a) show the number of PRNs as the cluster radius D increases at bitrate 48Mbps without external noise and at bitrate of 12Mbps wit external noise respectively. We only show the nodes with minimum PRN-Gap of 1% to avoid counting non-FB nodes with PDRs lower than their associated FB nodes by a small margin as PRN. Both AMuSe-Mix and AMuSe-PDR yield close to 0 PRNs since both

schemes select nodes with lowest PDR in each cluster. K-worst-PDR also yields 0 PRNs, since it selects nodes with overall lowest PDR values. The link quality based schemes AMuSe-LQ and K-worst-LQ have similar performance which could be explained due to the weak correlation between LQ and PDR. As expected, the Random feedback selection scheme performs the worst and as the number of feedback nodes decreases (increase in cluster size), the number of PRNs increases due to fewer feedback nodes. We omit the results at lower bitrates since they are qualitatively similar but yield fewer overall PRNs since the vast majority of the nodes experience PDR above 99%. The Random scheme yields much higher number of PRNs that increases with the cluster radius.

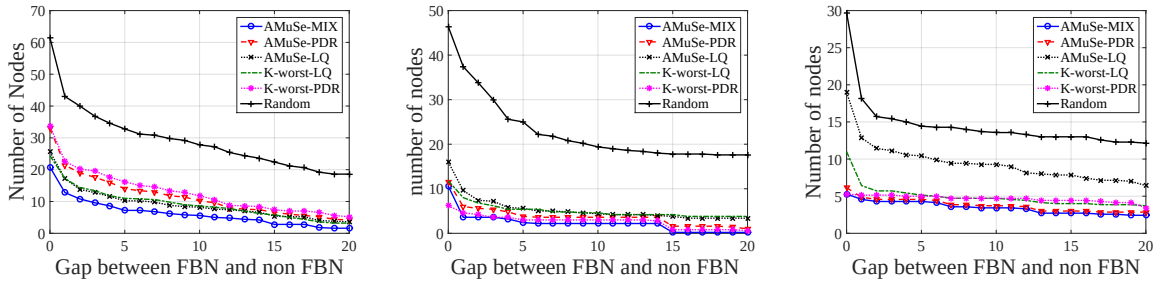
Figs. 8.10(b) and 8.11(b) present the number of PRNs at different values of PRN-Gap at bitrate 48Mbps without external noise and at bitrate of 12Mbps wit external noise respectively. The Random, K-worst-LQ, and AMuSe-LQ schemes result in a considerable number of PRNs. This number is high even for a PRN-Gap of 20% (e.g., Fig. 8.10(b) and 8.11(b) show that the K-worst-LQ and AMuSe-LQ schemes have between 5 to 10 PRNs with PRN-Gap of 20%). This means that the PDR value of each one of these nodes is at least 20% lower than its representative FB node. The situation is even worse for the Random scheme. We again omit the results at lower bitrates due to very low number of PRNs.

Finally, Figs. 8.10(c) and 8.11(c) show the maximum distance between an FB and non-FB node as D increases at bitrate 48Mbps without external noise and at bitrate of 12Mbps wit external noise respectively. As expected, for *AMuSe* schemes, this distance scales linearly with D. The maximum distance between an FB and non-FB node is significantly higher for the Random scheme and it is about twice for the K-worst-LQ and K-worst-PDR schemes. This indicates that FB nodes might be concentrated in areas of high losses. Thus, even though K-worst-PDR scheme leads to low number of PRNs, it does not obtain a well-distributed set of FB nodes. The distribution of FB nodes could be especially important in case of rapid network changes.



(a) Switching from 36 to 48 Mbps (b) Switching from 48 to 54 Mbps (c) Increasing noise by 10 dB

Figure 8.12: Dynamic Settings: The number of Poorly Represented Nodes (PRN) vs. the cluster radius with fixed PRN-Gap of 1%.



(a) Switching from 36 to 48 Mbps (b) Switching from 48 to 54 Mbps (c) Increasing noise by 10 dB

Figure 8.13: Dynamic Settings: The number of Poorly Represented Nodes (PRN) for different PRN-Gap and fixed cluster size of $D = 3$ m.

8.7.2 Dynamic Settings

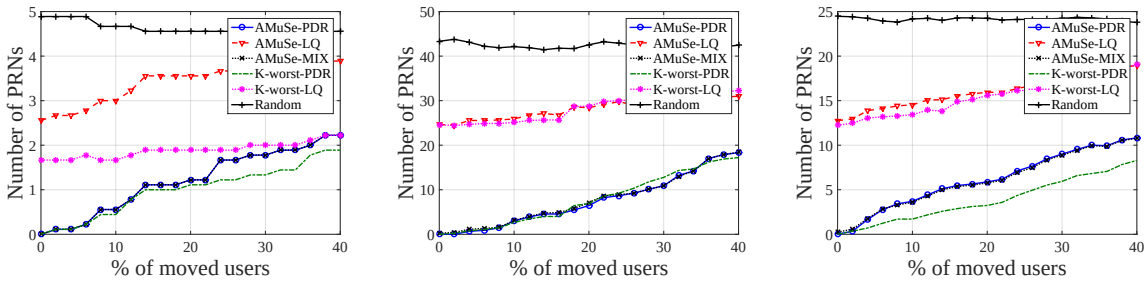
Next, we emulate a dynamic environment of either: (i) changing AP bitrate, (ii) changing external interference, (iii) emulating node mobility. The methodology of the dynamic evaluations of (i) and (ii) relies on selecting a feedback set at one bitrate or external interference value and studying the performance of that set at a different value of bitrate or interference. Since the ORBIT environment is relatively static, we emulate mobility by exchanging positions of nodes but keeping their channel quality values fixed. The FB nodes are selected at a particular setting and a fixed percentage of non-FB nodes exchange locations with each other within a certain radius. The PRNs are then evaluated with the same FB nodes and clustering as the initial conditions. The dynamic setting helps to evaluate the performance of the considered schemes under changes in the network.

Obviously, under such dynamic changes, the feedback node selection process may choose a new set of FB nodes. However, this process may require noticeable convergence time (depending

on several parameters, such as τ_{AP} and τ_{FB}) of up to a few seconds. During this time the system may not receive accurate reports about the service quality. Thus, it is essential that the selected FB nodes continue to provide accurate FB reports in the event of such changes. For instance, during any interference episode, the AP should receive the accurate feedback information without delays to take appropriate interference mitigation actions, such as adding more FEC, reducing bitrate, etc. Similarly, if the AP increases the multicast bitrate using a rate adaptation algorithm, the FB nodes should provide accurate state information about the change to the AP. For the dynamic setting we consider the following cases: (a) Switching from bitrate of 36 Mbps to 48 Mbps, (b) Switching from bitrate of 48 Mbps to 54 Mbps, (c) Increasing the noise level by 10 dB, and (d) Emulated mobility.

Fig. 8.12 presents the number of PRNs vs. the cluster radius (D) for the three cases where the PRN-Gap is 1%. Fig. 8.12(a) shows the number of PRNs when switching the bitrate from 36 to 48 Mbps. In this case, the AMuSe-LQ and K-worst-LQ have comparable performance to the static case with bitrate of 48 Mbps. This is an expected result since LQ is a measure of the received signal strength and is not affected from changing the bitrate. However, AMuSe-PDR performs significantly worse than the static case. To understand this trend, recall that at bitrate of 36 Mbps most of receivers experience PDR close to 100%, as shown in Fig. 8.7(c). Therefore, when the cluster size is small and large number of receivers are selected as FB nodes, most of the FB nodes have PDR above 99%. With such high PDR, a selected FB node may not be affected by increasing the bitrate. Observe that the number of PRNs decreases by increasing the cluster size. This is not surprising since now most of the selected FB nodes have PDR below 98%, which indicates that they experience only moderate channel quality and therefore they are more susceptible to a bitrate increase. A similar explanation holds true for the K-worst-PDR scheme. AMuSe-Mix outperforms the other schemes since it considers both the PDR and the LQ of the receivers and uses the LQ values when the PDR is very high. Like the static setting, the Random scheme suffers from very high number of PRNs.

Fig. 8.12(b) shows the number of PRNs for bitrate increases from 48 to 54Mbps. In this case



(a) (b) (c)

Figure 8.14: The number of Poorly Represented Nodes (PRNs) vs. percentage of moved nodes for (a) fixed bitrate of 36Mbps, (b) fixed bitrate of 48Mbps, and (c) bitrate of 12Mbps and noise of 5dBm.

AMuSe-Mix , AMuSe-PDR , and K-worst-PDR outperform the LQ based solutions. By revisiting Fig. 8.7(f), we see that many receivers suffer from low PDR due to a weak channel condition at a bitrate of 48 Mbps. Since these nodes are selected as FB nodes, they provide good lower bound reports of the service quality observed by the nodes in their clusters. We notice a similar situation in Fig. 8.12(c) when increasing the noise level by 10 dB.

The distribution of PRNs vs the PRN-Gap is shown in Fig. 8.13 for a cluster radius $D = 3$ m. The figure supports our observations from Fig. 8.12 and demonstrates that AMuSe-Mix outperforms the other alternatives in all cases. Since the feedback node set is not changed when increasing the bitrate or noise level, the maximum distance between an FB and non-FB node remains the same as shown in Figs. 8.10(c) or 8.11(c).

The results for emulated node mobility are shown in Fig. 8.14. Fig. 8.14(a) shows the number of PRNs vs. the percentage of moved nodes within a radius of 2m at a fixed bitrate of 36Mbps. Similar results at bitrate of 48Mbps are in Fig. 8.14(b) and with external noise in Fig. 8.14(c). The Random scheme yields the largest number of PRNs and is not affected by increasing number of moved nodes. The AMuSe-Mix, AMuSe-PDR, and K-worst-PDR schemes perform quite similarly and the PRNs for all of them increase with increase in the number of moved nodes. The LQ based schemes AMuSe-LQ and K-worst-LQ perform worse than the PDR based schemes.

We also evaluate the sensitivity of *AMuSe* to errors in node location estimation by injecting errors into reported node locations. The errors are picked from a Gaussian distribution with $\mu =$

$0, \sigma = 7$ meters. However, we observed only insignificant increases in the number of PRNs for the *AMuSe* schemes.

Our experiments on the ORBIT testbed with approximately 200 nodes validate the practicality of AMuSe-Mix as an excellent scheme for reporting the provided quality of an ongoing WiFi multicast services for both static and dynamic settings. The K-worst-PDR scheme also peforms quite well but does not yield a well-distributed set of FB nodes. Our evaluation shows that a relative small number of FB nodes is sufficient to provide accurate reports. Yet, the number of required FB nodes will also depend on the application.

8.A Proof of Proposition 1

Proposition 1: $|F| \leq 5 \cdot |OPT|$. If the channel quality is a monotonic decreasing function with the distance from the AP then $|F| \leq 3 \cdot |OPT|$

Proof of Proposition 1. We prove the general proposition of $|F| \leq 5 \cdot |OPT|$, which is based on Lemma 3.1 in [131]. The special case of $|F| \leq 3 \cdot |OPT|$, where the channel quality is a monotonic decreasing function with the distance from the AP, can be proved by using similar arguments and Lemma 3.3 in [131].

Consider a point x in the plane and let Z be an independent set of points in the circle with radius r around point x. i.e, the distance between any two points in Z is more than r. Then according to Lemma 3.1 in [131], $|Z| \leq 5$.

To prove that *AMuSe* guarantees approximation ratio of 5, we just need to show that for any given multicast group there is a mapping from F to OPT such that at most 5 nodes in F are mapped to the same node in OPT. To this end, we map every FB node $v \in F$ to its nearest node $u \in OPT$, which may be node v itself. Recall that both OPT and F are dominating independent sets such that each node has an adjacent FB node with distance at most D and the minimal distance between any pair of FB nodes is at least D. From this it is implied that any FB node v is either in OPT or it is D-adjacent to at least one node in OPT.

Now, consider an FB node $u \in OPT$ and let $W \subseteq F$ be the set of FB nodes selected by our scheme that are D-adjacent to u. Since F is an independent set it holds that W is also an independent set, i.e., the minimal distance between any pair of FB nodes $x, y \in W$ is $d_{x,y} > D$. Observe that all nodes in W are included in a disk with radius D centered at node u. Thus, according to Lemma 3.1 in [131], it follows that $|W| \leq 5$. This leads to the result that each node in OPT is associated with at most 5 nodes in F. □

Chapter 9: Multicast Dynamic Rate Adaptation

The *AMuSe* system consists of the following components: (i) an efficient feedback mechanism, and (ii) dynamic rate adaptation algorithm. In Chapter 8 we discussed the feedback node selection process. In this chapter we focus on the second component of developing a system that dynamically adapts the multicast rate.

9.1 Introduction

Multicast Rate Adaptation (RA) - Challenges: A key challenge in designing multicast RA schemes for large groups is to obtain accurate quality reports with low overhead. Some systems [226, 33, 180] experimentally demonstrated impressive ability to deliver video to a few dozen nodes by utilizing Forward Error Correction (FEC) codes and retransmissions. However, most approaches do not scale to very large groups with hundreds of nodes, due to the following:
(i) Most schemes tune the rate to satisfy the receiver with the worst channel condition. As shown in [20, 156] in crowded venues, a few unpredictable outliers, referred to as *abnormal nodes*, may suffer from low SNR and Packet Delivery Ratio (PDR) even at the lowest rate and without interference. This results from effects such as multipath and fast fading [165]. Therefore, *a multicast scheme cannot provide high rate while ensuring reliable delivery to <u>all</u> users.*
(ii) It is impractical to continuously collect status reports from all or most users without hindering performance. Even if feedback is not collected continuously, a swarm of retransmission requests may be sent following an interference event, (wireless interference is bursty [11]) thereby causing additional interruptions.

To overcome these challenges, a multicast system should *conduct efficient RA based on only limited reports from the nodes*. In the previous chapter we focused on efficient feedback collec-

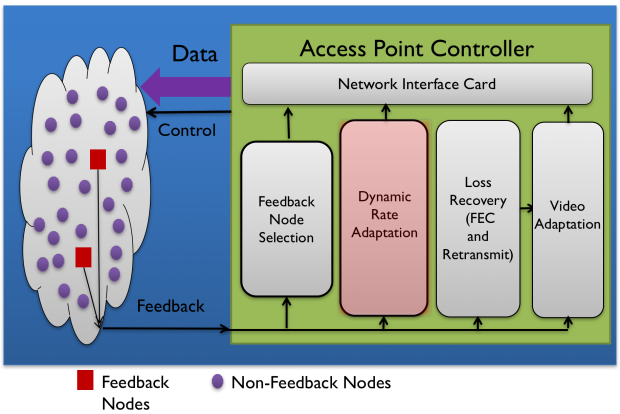


Figure 9.1: The Adaptive Multicast Services (*AMuSe*) system consisting of the Multicast Dynamic Rate Adaptation (MuDRA) algorithm and a multicast feedback mechanism.

tion mechanisms for WiFi multicast as part of the *AMuSe* system. In this chapter, we present the Multicast Dynamic Rate Adaptation (*MuDRA*) algorithm. *MuDRA* leverages the efficient multicast feedback collection of *AMuSe* and *dynamically adapts the multicast transmission rate to maximize channel utilization while meeting performance requirements*. Fig. 9.1 shows the overall *AMuSe* system composed of the *MuDRA* algorithm and the *AMuSe* feedback mechanism where the focus of this chapter is *MuDRA* .

We present a multicast rate adaptation algorithm *MuDRA* which is designed to support WiFi multicast to hundreds of users in crowded venues. *MuDRA* can provide high throughput while ensuring high *Quality of Experience* (QoE). *MuDRA* benefits from a large user population, which allows selecting a small yet sufficient number of Feedback (FB) nodes with marginal channel conditions for monitoring the quality. We address several design challenges related to appropriate configuration of the feedback level.

We note that using *MuDRA* does not require any modifications to the IEEE 802.11 standard or the mobile devices. *MuDRA* requires application layer measurements from mobile devices for multicast rate adaptation decisions. The multicast rate changes can be supported by most Access Points through changes in the driver-level code or through API calls.

We implemented *MuDRA* with the *AMuSe* system on the ORBIT testbed [151], evaluated its performance with all the operational IEEE 802.11 nodes (between 150 and 200), and compared it

to other multicast schemes. We use 802.11a to maximize the number of WiFi devices available[1]. To the best of our knowledge, this is the largest set of wireless devices available to the research community. Our key contributions are:

(i) The need for RA: We empirically demonstrate the importance of RA. Our experiments on ORBIT show that when the multicast rate exceeds an optimal rate, termed as *target-rate*, numerous receivers suffer from low PDR and their losses cannot be recovered. We also observed that even a controlled environment, such as ORBIT, can suffer from significant interference. These observations constitute the need for a stable and interference agnostic RA algorithm that *does not exceed the target-rate*.

(ii) Practical method to detect the target-rate: Pseudo-multicast schemes that rely on unicast RA [33] may occasionally sample higher rates and retreat to a lower rate after a few failures. Based on the observation above about the target rate, schemes with such sampling mechanisms will provide low QoE to many users. To overcome this, we developed a method to detect when the system operates at the target-rate, termed the **target condition**. Although the target condition is sufficient but not necessary, our experiments show that it is almost always satisfied when transmitting at the target-rate. *MuDRA* makes RA decisions based on the target condition and employs a dynamic window based mechanism to avoid rate changes due to small interference bursts.

(iii) Extensive experiments with hundreds of receivers: Our experiments demonstrate that *MuDRA* swiftly converges to the target-rate, while meeting the Service Level Agreement (SLA) requirements (e.g., ensuring PDR above 85% to at least 95% of the nodes). Losses can be recovered by using appropriate application-level FEC methods [157, 183, 36, 212, 7].

MuDRA is experimentally compared to (i) pseudo-multicast with a unicast RA [137], (ii) fixed rate, and (iii) a rate adaptation mechanism proposed in [20] which we refer to as the Simple Rate Adaptation (SRA) algorithm. *MuDRA* achieves 2x higher throughput than pseudo-multicast while sacrificing PDR only at a few poorly performing nodes. While the fixed rate and SRA schemes can obtain similar throughput as *MuDRA*, they do not meet the SLA requirements. Unlike other

[1]The ORBIT testbed supports only about 30 802.11n enabled devices

schemes, *MuDRA* preserves high throughput even in the presence of interference. Additionally, *MuDRA* can handle significant node churn. Finally, we devise a live multicast video delivery approach for *MuDRA*. We show that in our experimental settings with target rate of 24 – 36Mbps, *MuDRA* can deliver 3 or 4 high definition H.264 videos (each one of 4Mbps) where over 90% of the nodes receive video quality that is classified as excellent or good based on user perception.

To summarize, to the best of our knowledge, *MuDRA* is the first multicast RA algorithm designed to satisfy the specific needs of multimedia/video distribution in crowded venues. Moreover, *AMuSe* in conjunction with *MuDRA* is the first multicast content delivery system that has been evaluated *at scale*.

The rest of the chapter is organized as follows. Section 9.3 describes the ORBIT testbed and important observations. Section 9.4 presents the model and objectives. *MuDRA*'s design is described in Sections 9.5 and 9.6. The experimental evaluation is presented in Section 9.7.

The design and experimental evaluation of *MuDRA* appeared in the proceedings of IEEE INFOCOM'16 [73] and in IEEE Transactions on Wireless Communications [72]. Dr. Yigal Bejerano and Ph.D student Dr. Varun Gupta contributed to the design behind *MuDRA*. A demo of the rate adaptation process was presented at and appeared in the proceedings of IEEE INFOCOM'16 [74] with significant contributions from undergraduate students Raphael Norwitz and Savvas Petridis.

9.2 Related Work

Multicast rate adaptation approaches are in general closely linked to multicast feedback. In Chapter 8, we described existing approaches for multicast feedback mechanisms in detail. In this chapter, we focus on unicast and multicast te adaptation.

Unicast RA: We discuss unicast RA schemes, since they can provide insight into the design of multicast RA. In *Sampling-based algorithms*, both ACKs after successful transmissions and the relation between the rate and the success probability are used for RA after several consecutive successful or failed transmissions [102, 113, 26]. The schemes in [155, 224, 108] distinguish between losses due to poor channel conditions and collisions, and update the rate based on former.

Recently, [163, 45] propose multi-arm bandit based RA schemes with a statistical bound on the regret.

However, such schemes cannot support multicast, since multicast packets are not acknowledged. In *Measurement-based schemes* the receiver reports the channel quality to the sender which determines the rate [89, 166, 99, 213, 164, 48]. Most measurement-based schemes modify the wireless driver on the receiver end and some require changes to the standard, which we avoid.

Multicast RA: In [181, 19, 209, 33, 122] the sender uses feedback from leaders (nodes with worst channel conditions) for RA. In [121] when the channel conditions are stable, RA is conducted based on reports of a single leader. When the channel conditions are dynamic, feedback is collected from all nodes. Medusa [180] combines Pseudo-Multicast with infrequent application layer feedback reports from all nodes. The MAC layer feedback sets backoff parameters while application layer feedback is used for RA and retransmissions of video packets. Recently, in [20] we considered multicast to a large set of nodes and provided a rudimentary RA scheme which is not designed to achieve optimal rate, maintain stability, or respond to interference.

9.3 Testbed and Key Observations

We evaluate *MuDRA* on the ORBIT testbed [151], which is a dynamically configurable grid of 20×20 (400) 802.11 nodes where the separation between nodes is 1m. It is a good environment to evaluate *MuDRA*, since it provides a very large and dense population of wireless nodes, similar to the anticipated crowded venues.

Experiments: To avoid performance variability due to a mismatch of WiFi hardware and software, only nodes equipped with *Atheros 5212/5213* cards with *ath5k* driver were selected. For each experiment *we activated all the operational nodes* that meet these specifications (between 150 and 250 nodes). In all the experiments, one corner node served as a single multicast AP. The other nodes were multicast receivers. The AP used 802.11a to send a multicast UDP flow, where each packet was 1400 bytes. Most practical applications such as video streaming include a sequence number to keep track of packet delivery at the clients. We embed an artificial sequence number

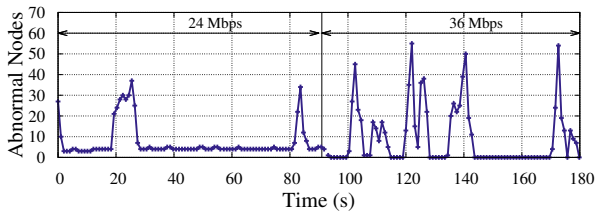


Figure 9.2: Experimental measurement of the number of abnormal nodes in time, for fixed rates of 24 and 36Mbps.

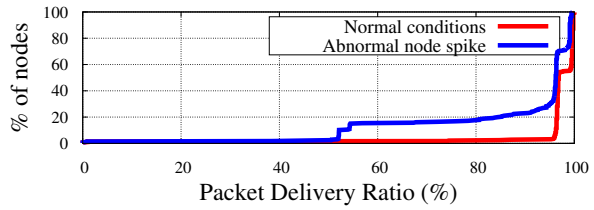


Figure 9.3: The CDF of the PDR values of 170 nodes during normal operation and during a spike at rate of 36Mbps.

for each packet in the UDP payload for measurement purposes. *The AP used the lowest supported transmission power of* $1mW = 0dBm$ *to ensure that the channel conditions of some nodes are marginal.*

Technical challenges: While analyzing the performance, we noticed that clients disconnect from the AP at high bit-rates, thereby causing performance degradation. We noticed that in several WiFi driver implementations, the beacon rate is set as the multicast rate. Increasing the bit-rate also increases the WiFi beacon bit-rate which may not be decoded at some nodes. A sustained loss of beacons leads to node disconnection. To counter this, we modified the ath5k driver to send beacons at a fixed minimum bit-rate.

Interference and Stability: We study the time variability of the channel conditions on the ORBIT testbed by measuring the number of nodes with low PDR (below a threshold of 85%). We call these nodes *abnormal nodes* (the term will be formally defined in Section 9.4). The number of abnormal nodes out of 170 nodes for rates of 24 and 36Mbps is shown in Fig. 9.2. We repeated these experiments several times and observed that even at a low rate, the channel may suffer from sporadic interference events, which cause a sharp increase in the number of abnormal nodes. These interference spikes caused by non-WiFi devices are beyond our control and their duration varies in

Table 9.1: Notation and parameter values used in experiments.

Symbol	Semantics	Exp. Val.
n	Number of nodes associated with the AP.	> 150
X	*Population threshold* - Minimal fraction of nodes that should experience high PDR.	95%
A_{max}	The maximal number of allowed abnormal nodes.	8
L	*PDR threshold* - Threshold between acceptable (normal) and low (abnormal) PDR.	85%
H	Threshold between high PDR and mid-PDR.	97%
K	Expected number of FB nodes, $K = \alpha \cdot A_{max}$.	30
R	Reporting PDR threshold.	
A_t	Number of abnormal nodes at time t.	
M_t	Number of mid-PDR FB nodes at time t.	
W_{min}	Minimal RA window size (multiples of reporting intervals).	8
W_{max}	Maximal RA window size.	32

time.

Fig. 9.3 provides the Cumulative Distribution Function (CDF) of the PDR values with and without sporadic interference. The figure shows that during a spike, over 15% of the nodes suffer from PDR around 50%. Further, the location of the nodes affected by the spikes varies with time and does not follow a known pattern. These experiments show that even in a seemingly controlled environment, *nodes may suffer from sporadic continuous interference, which may cause multicast rate fluctuations*. Users are very sensitive to changes in video quality [47, 17], and therefore, to keep a high QoE we would like to avoid rate changes due to sporadic interference.

9.4 Network Model and Objective

We consider a WiFi LAN with multiple APs and frequency planning such that the transmissions of adjacent APs do not interfere with each other. Thus, *for RA we consider a single AP with n associated users*. We assume low mobility (e.g., users watching a sports event). Although we con-

sider a controlled environment, the network may still suffer from sporadic interference, as shown in Section 9.3. The main notation used in the chapter is summarized in Table 9.1. Specifically, a *PDR-Threshold* L, is defined such that a node has high QoE if its PDR is above L. Such a node is called a *normal node*. Otherwise, it is considered an *abnormal node*.

Our *objective is to develop a practical and efficient rate control system* which satisfies the following requirements:

(R1) High throughput – Operate at the highest possible rate, i.e., the *target rate*, while preserving SLAs.

(R2) Service Level Agreements (SLAs) – Given L (e.g., $L = 85\%$), and a *Population-Threshold* X (e.g., $X = 95\%$), the selected rate should guarantee that at least $X\%$ of the nodes experience PDR above L (i.e., are normal nodes). Except for short transition periods, this provides an upper bound of $A_{max} = \lceil n \cdot (1 - X) \rceil$ on the number of permitted abnormal nodes.

(R3) Scalability – Support hundreds of nodes.

(R4) Stability – Avoid rate changes due to sporadic channel condition changes.

(R5) Fast Convergence – Converge fast to the target rate after long-lasting changes (e.g., user mobility or network changes).

(R6) Standard and Technology Compliance – No change to the IEEE 802.11 standard or operating system of the nodes.

9.5 Multicast Rate Adaptation

The overall multicast rate adaptation process of *MuDRA* as a part of the *AMuSe* system relies on three main components, as illustrated in Fig. 9.1 and discussed below. We first provide a high level description of each component and then discuss the details in the following subsections.

(i) Feedback (FB) Node Selection: Selects a small set of *FB nodes* that provide reports for making RA decisions. We describe the FB node selection process in Section 9.5.1 and calculate the reporting interval duration in Section 9.6.[2]

[2]Unlike in unicast where each packet is acknowledged, *MuDRA*'s reporting intervals are long (in the experiments

Algorithm 2 *MuDRA* Algorithm

```
1: rate ← lowestRate, window ← W_min, changeTime ← t, refTime ← t, t := current time
2: while (true) do
3:     Get PDR reports from all FB nodes
4:     Get Status of each FB node i
5:     Calc Â_t and M̂_t
6:     rate, action, changeTime ← GetRate(...)
7:     window, refTime ← GetWinSize(...)
8:     set multicast rate to rate
9:     sleep one reporting interval
```

Procedure 1 Rate Decision

```
1: procedure GETRATE(rate, window, changeTime, t)
2:     action ← Hold
3:     if (t − changeTime) > window then
4:         canDecrease ← true, canIncrease ← true
5:         for τ ← 0 To window do
6:             if Â_{t−τ} < A_max then
7:                 canDecrease ← false
8:             else if Â_{t−τ} + M̂_{t−τ} > A_max − ϵ then
9:                 canIncrease ← false
10:        if canDecrease and rate > rate_min then
11:            rate ← NextLowerRate
12:            action ← Decrease, changeTime ← t
13:        if canIncrease and rate < rate_max then
14:            rate ← NextHigherRate
15:            action ← Increase, changeTime ← t
16: return rate, action, changeTime
```

The following two components compose the *MuDRA* Algorithm (Algorithm 2). It collects the PDR values from the FB nodes, updates their status (normal or abnormal), invokes the GETRATE procedure, which calculates the desired rate, and invokes the GETWINSIZE procedure, which determines the window size of rate updates (to maintain stability).

(ii) Rate Decision (Procedure 1): Utilizes the limited and infrequent FB reports to determine the highest possible rate, termed the *target-rate*, while meeting the requirements in Section 9.4. The rate decisions (lines 5–15) rely on rate decision rules that are described in Section 9.5.2. To maintain rate stability, rate change operations are permitted, only if the conditions for rate change are satisfied for time equal to a window size (determined by the *Stability Preserving Method*).

(iii) Stability Preserving Method (Procedure 2): A window based method that maintains rate stability in the event of sporadic interference and after an RA decision. It follows the classical Ad-

we consider 2 reports per second).

Procedure 2 Window Size Determination

```
1:  procedure GETWINSIZE(Action, window, refTime, t)
2:      if Action = Decrease then
3:          window ← min(W_max, 2 · window), refTime ← t
4:      else if Action = Increase then
5:          refTime ← t
6:      else if (t − refTime) > thresholdTime
7:        and Action = Hold then
8:          window ← max(W_min, window − 1)
9:          refTime ← t
10: return window, refTime
```

ditive Increase Multiplicative Decrease (AIMD) approach. The duration of the time window varies according to the network and channel characteristics (e.g., the typical duration of interference). More details appear in Section 9.5.3.

9.5.1 Feedback Node Selection

MuDRA uses a simple and efficient mechanism based on a quasi-distributed FB node selection process, termed *K-Worst* [20], where the AP sets the number of FB nodes and their reporting rates. K nodes with the worst channel conditions are selected as FB nodes (the node's channel condition is determined by its PDR). Hence, the selection process ensures an upper bound on the number of FB messages, regardless of the multicast group size. This upper bound is required for limiting the interference from FB reports, as explained in Section 9.6. The process works as follows: At the beginning of each reporting interval the AP sends a message with a list of K or less FB nodes as well as a *reporting PDR threshold R*. R is used for adjusting the set of FB nodes to changes due to mobility or variation of the channel condition, i.e., interference[3]. Upon receiving this message, each FB node waits a short random time for avoiding collisions and then reports its measured PDR to the AP. Every other node checks if its PDR value is below R and in such situation it volunteers to serve as an FB node. To avoid a swarm of volunteering messages in the case of sporadic interference, a non FB node verifies that its PDR values are below R for three consecutive reporting intervals before volunteering. At the end of a reporting interval, the AP checks the PDR values of all the FB and volunteering nodes, it selects the K with lowest PDR values as FB nodes

[3]when the system is activated the FB list is empty and $R = L$.

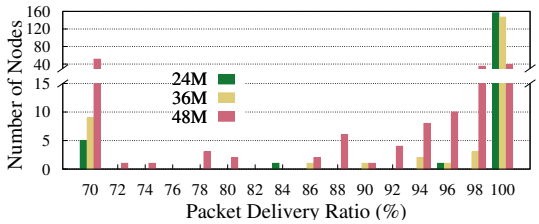


Figure 9.4: The PDR distribution of one set of experiments with TX_{AP} rates of 24, 36, and 48Mbps.

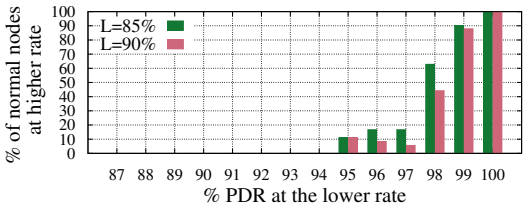


Figure 9.5: The percentage of nodes that remain normal after increasing the TX_{AP} from 36Mbps to 48Mbps vs. their PDR values at the 36Mbps for different PDR-thresholds (L).

and updates R. If the number of selected FB nodes is K then for keeping the stability of the FB list, R is set slightly below the highest PDR value of the FB nodes (e.g., 1% point below) . Otherwise, R is set slightly above the highest PDR value of the FB nodes (e.g., 0.5% point above). The AP sends a new message and the process repeats. We note that in a quasi static scenario, the values of R do not have a significant impact on the feedback or the overhead of feedback. Tuning R is a challenge only in the rare scenario when a large number of nodes with significantly different PDR values rapidly enter or leave the multicast system.

9.5.2 Rate Decision Rules and Procedure

In this subsection, we describe the *target condition* which is an essential component of the rate selection rules. Then, we describe the rules and the corresponding Procedure 1.

The Target Condition: At a given time, the FB reports are available only for the current rate. To detect the target-rate, most RA schemes occasionally sample higher rates. However, the following experiment shows that this approach may cause undesired disruption to many receivers. We evaluated the PDR distribution of $160 - 170$ nodes for different multicast transmission rates, denoted as TX_{AP} for 3 different experiment runs on different days. Fig. 9.4 shows the number of nodes in

different PDR ranges for TX_{AP} values of 24, 36, and 48Mbps for one experiment with 168 nodes. When TX_{AP} is at most 36Mbps, the number of abnormal nodes is very small (at most 5). However, when TX_{AP} exceeds 36Mbps, the PDR of many nodes drops significantly. In this experiment 47 nodes became abnormal nodes which is more than $A_{max} = 8$ (for $X = 95\%$). We observed similar results in other experiments. Thus, in this case, the *target rate* is 36Mbps which is the highest rate above which the SLA requirements will be violated.

A key challenge is to *determine if the AP operates at the target-rate, without FB reports from higher rates.* We refer to this assessment as the *target condition*. Unfortunately, the target-rate cannot be detected from RF measurements, such as SNR. As shown in [170, 81] different nodes may have different receiver sensitivities, which may result in substantial PDR gaps between nodes with similar RF measurements. However, large scale multicast environments enable us to efficiently predict the target condition as described next.

From Fig. 9.4, we obtain the following important observation.

Observation I: When operating below the target-rate, almost all the nodes have PDR close to 100%. However, when operating at the target-rate, noticeable number of receivers experience PDR below 97%. At 36Mbps, 17 nodes had PDR below 97%, which is substantially more than $A_{max} = 8$.

Fig. 9.5 shows the average percentage of nodes that remain normal vs. their initial PDR when increasing TX_{AP} from 36Mbps to 48Mbps averaged for 3 different sets of experiments. The total number of nodes in these experiments was 168. We derive the following observation from Fig. 9.5.

Observation II: There is a PDR threshold, H, such that every node with PDR between L and H becomes abnormal after the rate increase with very high probability. Typically, H can be a value slightly below 100%. In our experiments on ORBIT, we use $H = 97\%$ since 97% is the highest threshold for which this observation holds. We refer to these nodes as *mid-PDR nodes*.

Observation II is not surprising. As reported in [170, 190], each receiver has an SNR band of $2 - 5$dB, in which its PDR drops from almost 100% to almost 0%. The SNR of mid-PDR nodes lies in this band. Increasing the rate requires $2 - 3$dB higher SNR at the nodes. Hence, mid-PDR nodes with SNR in the transition band before the rate increase will be below or at the lower end of

the transition band after the increase, and therefore, become abnormal nodes.

In summary, Observations I and II imply that it is possible to assess the target condition by monitoring the nodes close to transitioning from normal to abnormal. Let A_t and M_t denote the number of abnormal and mid-PDR nodes at time t, respectively. We obtain the following empirical property.

Property 1 (Target Condition)**.** *Assume that at a given time t, the following condition holds,*

$$A_t \leq A_{max} \quad and \quad A_t + M_t > A_{max} \tag{9.1}$$

then almost surely, the AP transmits on the target-rate at time t. This is sufficient but not a necessary condition.

It is challenging to analytically predict when the target condition is satisfied with the available FB information and without a model of the receiver sensitivity of all the nodes. However, our experiments show that the target condition is typically valid when operating at the target-rate.

Adjusting the Multicast Rate: The SLA requirement (R2) and target condition (9.1) give us a clear criteria for changing the rate. The FB scheme only gives us estimates of A_t and M_t, denoted by $\hat{A}_t$ and $\hat{M}_t$ respectively. For the K-Worst scheme, if $K > A_{max} + \epsilon$ (ϵ is a small constant), then $\hat{A}_t$ and $\hat{M}_t$ are sufficient to verify if (9.1) is satisfied because of the following property:

Property 2. *If $K \geq A_{max} + \epsilon$, then, $\hat{A}_t = \min(A_t, A_{max} + \epsilon)$ and $\hat{A}_t + \hat{M}_t = min(A_t + M_t, A_{max} + \epsilon)$, where $\hat{A}_t$ and $\hat{M}_t$ are the known number of abnormal and mid-PDR known to the AP, and ϵ is a small constant. In other words, given that K is large enough, the K-worst scheme provides accurate estimates of abnormal and mid-PDR nodes.*

Proof. First consider the claim $\hat{A}_t = \min(A_t, A_{max} + \epsilon)$. Consider the case $A_t \leq A_{max} + \epsilon$, we know that the number of estimated abnormal nodes $\hat{A}_t = A_t$ since $K \geq A_{max} + \epsilon$ and all abnormal nodes must belong in the K FB nodes set. Next, if $A_t > A_{max} + \epsilon$ then all the FB nodes chosen are abnormal by the definition of the K-worst feedback scheme which implies $\hat{A}_t = A_{max} + \epsilon$.

A similar argument can be made for the claim $\hat{A}_t + \hat{M}_t = min(A_t + M_t, A_{max} + \epsilon)$. If $A_t + M_t \leq A_{max} + \epsilon$, then $\hat{A}_t + \hat{M}_t = A_t + M_t$ since the K feedback nodes will necessarily include the A_t abnormal and M_t mid-PDR nodes. If $A_t + M_t > A_{max} + \epsilon$, then $\hat{A}_t + \hat{M}_t$ which is upper bounded by $A_{max} + \epsilon$. □ □

The objective is to choose minimum K (for minimum FB overhead) that is sufficient to verify (9.1). In our experiments, we found that for $A_{max} = 8$, $K > 10$ works well (Section 9.7.1). We now derive the following *rate changing rules*:

Rule I $\hat{A}_t > A_{max}$: The system violates the SLA requirement (R2) and the rate is reduced.

Rule II $\hat{A}_t + \hat{M}_t \geq A_{max} - \epsilon$: The system satisfies the target condition.

Rule III $\hat{A}_t + \hat{M}_t < A_{max} - \epsilon$: The target condition does not hold and the rate can be increased, under the stability constraints provided in Section 9.5.3.

In our experiments we use $\epsilon = 2$ to prevent rate oscillations.

The rate change actions in Procedure 1 are based on the these rules. The flags $canIncrease$ and $canDecrease$ indicate whether the multicast rate should be increased or decreased. Rate change operations are permitted, only if the time elapsed since the last rate change is larger than the window size determined by the *Stability Preserving Method* (line 3). The for-loop checks whether the rate should be decreased according to Rule I (line 6) or increased according to Rule III (line 9) for the window duration. Finally, based on the value of the flags and the current rate, the algorithm determines the rate change operation and updates the parameters $rate$ and $action$, accordingly (lines 10–15).

9.5.3 The Stability Preserving Method

It is desirable to change the rate as soon as Rules I or III are satisfied to minimize QoE disruption (see (R5) in Section 9.4). However, as we show in Fig. 9.6 such a strategy may cause severe fluctuations of the transmission rate. These result from two main reasons: (i) the reporting mechanism not stabilizing after the last rate change, and (ii) interference causing numerous low PDR reports.

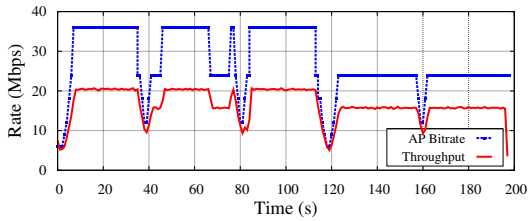


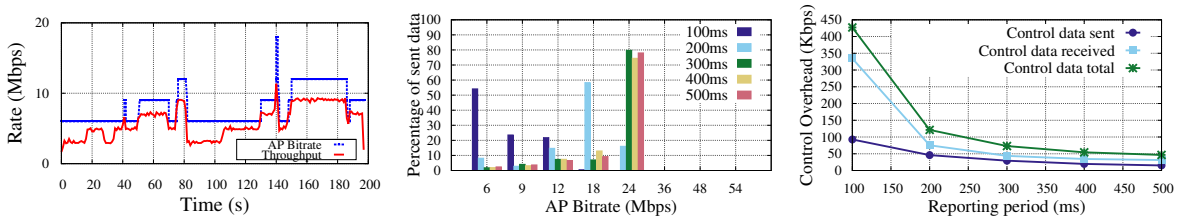
Figure 9.6: Evolution of the multicast rate over time when the delay between rate changes = 1s (2 reporting intervals).

.

To address this, we introduce in Procedure 2 a *window based RA technique* which considers the two situations and balances fast convergence with stability. In Procedure 2, the rate is changed only if the rate change conditions are satisfied over a given *time window*, after the last rate change operation (lines 5-9). To prevent oscillations due to short-term wireless channel degradation, when the rate is reduced, the window is doubled in Procedure 2 (line 3). The window size is decreased by 1 when a duration $thresholdTime$ elapses from the last rate or window size change (line 8). This allows recalibrating the window after an atypical long interference episode. The window duration varies between W_{min} and W_{max} FB reporting periods. In the experiments, $W_{min} = 8$ and $W_{max} = 32$.

9.5.4 Handling Losses

MuDRA can handle mild losses (below 15%) by adding application level FEC [212] to the multicast streams. The PDR-Threshold in our experiments ($L = 85\%$) was selected to allow nodes to handle losses in the event of short simultaneous transmission of another node. In such a situation, the collision probability is below $2/CW_{min}$, where CW_{min} is the minimal 802.11 contention window. For 802.11a/g/n $CW_{min} = 16$, which implies collision probability is below 12.5%. Therefore, nodes with high PDR (near 100%) should be able to compensate for the lost packets. If there is strong interference, other means should be used. For instance, the multicast content can be divided into high and low priority flows, augmenting the high priority flow with stronger FEC during the interference period, while postponing low priority flows.



(a) (b) (c)

Figure 9.7: (a) Rate adaptation performance for reporting intervals of 100ms, (b) Fraction of data sent at various rates with *MuDRA* for different reporting intervals, and (c) Control overhead for various reporting intervals.

9.6 Reporting Interval Duration

MuDRA relies on status reports from the FB nodes. For immediate response to changes in service quality, the status reports should be sent as frequently as possible, (i.e., minimal *reporting interval*). However, this significantly impairs the system performance as described below.

Impact of Aggressive Reporting: Figs. 9.7(a)-9.7(c) show the impact of different reporting intervals on *MuDRA*. In these experiments, the number of FB nodes (K) is 50 and the total number of nodes is 158. To focus on RA aspects, we set both W_{min} and W_{max} to 5 reporting intervals. Fig. 9.7(a) shows that when the reporting interval is too short, *MuDRA* does not converge to the target rate of 24Mbps. Fig. 9.7(b) shows that in the case of reporting interval of 100ms, more than 50% of the packets are transmitted at the lowest rate of 6 Mbps. Fig. 9.7(c) shows that the control overhead is significantly larger for short reporting intervals (shorter than 200ms). The control overhead comprises of unicast FB data sent by nodes and multicast data sent by AP to manage K FB nodes.

These phenomena result from collisions between feedback reports and multicast messages. In the event of a collision, FB reports, which are unicast messages, are retransmitted, while multicast messages are lost. Frequent reporting increases the collision probability, resulting in PDR reduction and causes the classification of many nodes as mid-PDR nodes, i.e., $PDR < H_{high} = 97\%$. Thus, due to Rule II from Section 9.5.2, the rate is kept close to the minimal rate.

Appropriate Reporting Interval Duration:

Table 9.2: The percentage of PDR loss at nodes ($\Delta PDR(T)$) as a function the reporting interval T.

T (ms)	100	200	300	400	500	700	1000
ΔPDR%	4.69	1.56	0.94	0.67	0.52	0.36	0.25

Assume a greedy AP which continuously transmits multicast messages. We now estimate the PDR reduction, denoted as ΔPDR, for a given reporting interval T and upper bound K on the number of FB nodes (both normal and abnormal), when the system operates at the low rate of 6Mbps.

Packet Transmission Duration: We denote with D and d the transmission duration of multicast and feedback report message at the rate of 6Mbps, respectively. Since the length of each multicast packet is $12Kbits$, its transmission duration is $\frac{12\text{Kbits}}{6\text{Mbps}} = 2.0$ms. Given WiFi overhead of about 30%, we assume $D = 3$ms. The feedback messages are much shorter and we assume that their transmission duration is $d = 1$ms.

Number of feedback reports and multicast messages: Consider a time interval U, say a minute. The number of feedback reports, denoted as F, is

$F = \frac{U}{T} \cdot K$

The number of multicast message B is given by,

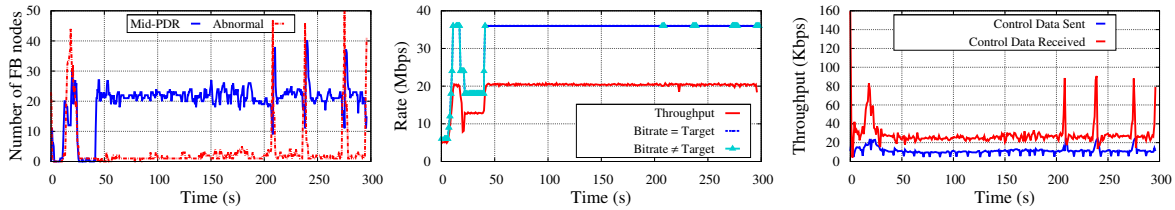
$$B = \frac{U - d \cdot F}{D} = \frac{U}{D} \cdot \left(1 - \frac{d \cdot K}{T}\right)$$

Collision probably of a multicast packet (ΔPDR): Let us first calculate the number of contention window slots, denoted by S, in which packet may be transmitted from the view point of the AP during the time interval U. Recall that between any two multicast transmissions, the AP waits an average of half of the contention window size $CW_{min}/2 = 8$. This leads to

$S = \frac{CW_{min}}{2} \cdot B$

ΔPDR is the fraction of slots in which both the AP and a FB node send a message. To simplify our estimation, we ignore collisions and retransmission of FB messages[4], and assume that in any

[4]These are second order effects of already low collision probabilities.



(a) (b) (c)

Figure 9.8: A typical sample of *MuDRA*'s operation over 300s with 162 nodes: (a) Mid-PDR and abnormal nodes, (b) Multicast rate and throughput measured at the AP, and (c) Control data sent and received.

slots only one FB node may transmit. Therefore,

$$\Delta PDR = \frac{F}{S} \cdot \frac{B}{S} = \left[\frac{2}{CW_{min}}\right]^2 \cdot \frac{F}{B}$$

With proper assignment we get,

$$\Delta PDR = \left[\frac{2}{CW_{min}}\right]^2 \cdot \frac{K \cdot D}{T - d \cdot K} \tag{9.2}$$

Equation (9.2) confirms that ΔPDR is reduced by increasing the reporting interval or by using a higher bit-rate, which reduces D. Table 9.2 provides the ΔPDR values for $K = 50$ when T varies between 0.1 to 1 second. In our experiments we wanted $\Delta PDR \leq 0.5\%$, which implies using reporting interval $T \geq 500$ms.

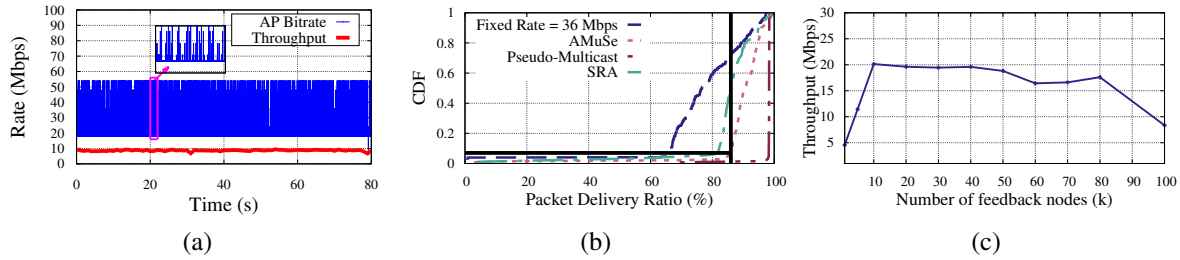
9.7 Experimental Evaluation

For evaluating the performance of *MuDRA* on the ORBIT testbed, we use the parameter values listed in Table 9.1. In all our evaluations, we consider backlogged multicast traffic. The performance metrics are described below:

(i) *Multicast rate and throughput*: The time instants when the target condition is satisfied are marked separately.

(ii) *PDR at nodes*: Measured at each node.

(iii) *Number of abnormal and mid-PDR nodes*: We monitored *all* the abnormal and mid-PDR



(a) (b) (c)

Figure 9.9: (a) Rate and throughput for the pseudo-multicast scheme, (b) CDF of PDR distributions of 162 nodes for fixed rate, *MuDRA*, Pseudo-Multicast, and SRA schemes, and (c) Multicast throughput vs. the number of feedback nodes (K).

nodes (not just the FB nodes).

(iv) *Control traffic*: The feedback overhead (this overhead is very low and is measured in Kbps).

We compared *MuDRA* to the following schemes:

(i) *Fixed rate scheme*: Transmit at a fixed rate of 36Mbps, since it is expected to be the target rate.

(ii) *Pseudo-multicast*: Unicast transmissions to the node with the lowest SNR/RSS. The unicast RA is the driver specific RA algorithm *Minstrel* [137]. The remaining nodes are configured in promiscuous mode.

(iii) *Simple Rate Adaptation (SRA) algorithm [20]*: This scheme also relies on measuring the number of abnormal nodes for making RA decisions. Yet, it is not designed to achieve the target rate, maintain stability, or respond to interference.

9.7.1 Performance Comparison

We evaluated the performance of *MuDRA* in several experiments on different days with 160 – 170 nodes. Fig. 9.8 shows one instance of such an experiment over 300s with 162 nodes. Fig. 9.8(a) shows the mid-PDR and abnormal nodes for the duration of one experiment run. Fig. 9.8(b) shows the rate determined by *MuDRA*. The AP converges to the target rate after the initial interference spike in abnormal nodes at 15s. The AP successfully ignored the interference spikes at time instants of 210, 240, and 280s to maintain a stable rate. The target-condition is satisfied except during the spikes. The overall control overhead as seen in Fig. 9.8(c) is approximately 40Kbps. The population of abnormal nodes stays around 2 – 3 for most of the time which implies that more than

Table 9.3: Average throughput (Mbps) of pseudo-multicast, *MuDRA*, and SRA schemes with and without background traffic.

	No Background traffic	Background traffic
Fixed rate = 36Mbps	20.42	13.38
Pseudo-Multicast	9.13	5.36
MuDRA	18.75	11.67
SRA	19.30	4.55

160 nodes (> 98%) have a PDR > 85%. The actual throughput is stable at around 20Mbps which after accounting for 15% FEC correction implies a goodput of 17Mbps.

Fig. 9.9(a) shows a sample of the throughput and rate performance of the pseudo-multicast scheme. The throughput achieved is close to 9Mbps. We observe that pseudo-multicast frequently samples higher rates (up to 54Mbps) leading to packet losses. The average throughput for different schemes over 3 experiments of 300s each (conducted on different days) with 162 nodes is shown in Table 9.3. *MuDRA* achieves 2x throughput than pseudo-multicast scheme. The fixed rate scheme yields approximately 10% higher throughput than *MuDRA*. SRA has similar throughput as *MuDRA*.

Fig. 9.9(b) shows the distribution of average PDR of 162 nodes for the same 3 experiments. In the pseudo-multicast scheme, more than 95% of nodes obtain a PDR close to 100% (we did not consider any retransmissions to nodes listening in promiscuous mode). *MuDRA* meets the QoS requirements of 95% nodes with at least 85% PDR. On the other hand, in SRA and the fixed rate schemes 45% and 70% of the nodes have PDR less than 85%, respectively.

In pseudo-multicast, more reliable transmissions take place at the cost of reduced throughput, since the AP communicates with the node with the poorest channel quality in unicast. The significant difference in QoS performance of the fixed rate and SRA schemes is because the target rate can change due to interference etc. In such a situation, *MuDRA* can achieve the new target rate while the fixed rate and SRA schemes lead to significant losses (we observed that exceeding the target rate even 10% of time may cause up to 20% losses and less than 5% throughput gain).

Changing number of FB nodes: We varied the number of FB nodes (K) between $1 - 100$ for *MuDRA*. Fig. 9.9(c) shows the throughput as K changes. For $K = 1$, *MuDRA* tunes to the node

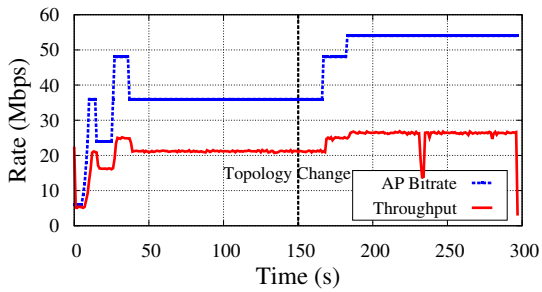


Figure 9.10: Emulating topology change by turning off FB nodes after 150s results in changing optimal rate for *MuDRA*.

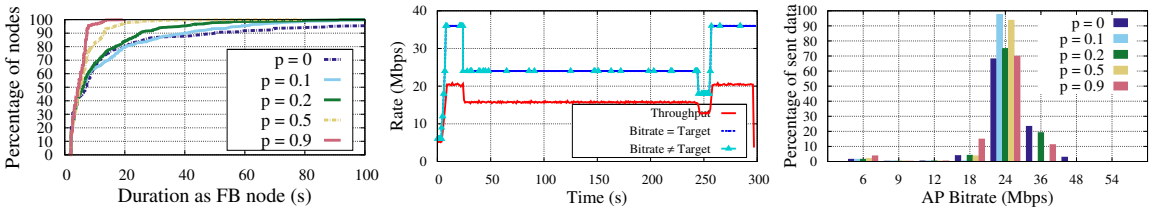
with the worst channel quality, and consequently, the throughput is very low. On the other hand, increasing K from 30 to 90 adds similar amount of FB overhead as decreasing the report interval from 500ms to 200ms in Section 9.6. Thus, the throughput decreases for a large number of FB nodes. The throughput for K between $10-50$ does not vary significantly which is aligned with our discussion in Section 9.5 that *MuDRA* needs only $K > A_{max} + \epsilon$ for small ϵ to evaluate the target rate conditions.

Impact of topology changes: To demonstrate that changes in the network may lead *MuDRA* to converge to a different rate, we devised a strategy to emulate network topology changes on the grid. During an experiment, a number of FB nodes are turned off at a given time. Since FB nodes have the lowest PDRs, it may lead to changes in the target rate as a large number of nodes with low PDR disappear from the network. Fig. 9.10 shows the scenario when 30 FB nodes are turned off after 150s during the experiment. The rate converges quickly and without oscillations to a new target rate of 54Mbps.

9.7.2 Impact of High Node Churn

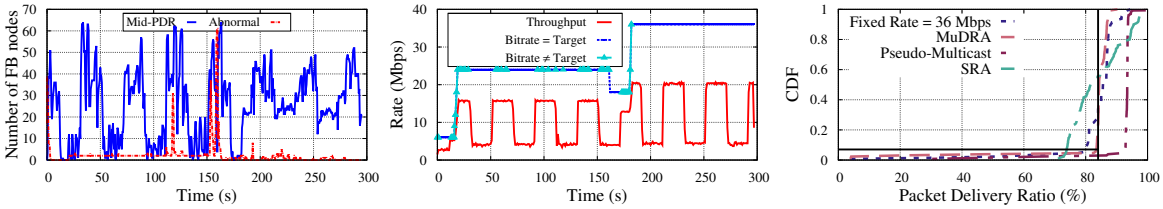
We evaluate the performance of *MuDRA* when emulating severe network changing conditions. In the experiments, each node leaves or joins the network with probability p every 6s. Thus, $p = 0.1$ implies that a node changes its state with probability of approximately 50% at least once in a minute. Initially, 50% of the nodes are randomly selected to be in the network.

We conducted 3 experiments consisting of 155 nodes (initially, 77 nodes are in on state).



(a) (b) (c)

Figure 9.11: Performance of *MuDRA* with high node churn: (a) Distribution of time duration for which a node is a FB node for different values of probability p of node switching its state on/off every 6s, (b) Multicast rate and throughput measured at the AP with $p = 0.2$, (c) Percentage of data sent at various rates for different values of p.



(a) (b) (c)

Figure 9.12: Performance of *MuDRA* with 155 nodes where an interfering AP transmits on/off traffic: (a) Mid-PDR and abnormal FB nodes, (b) Multicast rate and throughput, (c) CDF for PDR distribution with interference for fixed rate, *MuDRA*, pseudo-multicast, SRA.

Fig. 9.11(a) shows the impact of p on the distribution of time duration that the nodes remain as FB nodes. Higher values of p imply higher churn and lead to shorter periods for which nodes serve as FB nodes. The average number of changes in FB nodes per second is 2, 5, and 10 for p equal to 0, 0.2, and 0.9, respectively. Even with these changes, the average control overhead is very low (35Kbps) and is not affected by the degree of churn. Fig. 9.11(b) shows one instance of the RA process with $p = 0.2$. We see that *MuDRA* can adapt to the changing target rate at times 10, 30, and 255s. Fig. 9.11(c) shows the percentage of data sent at different rates for several values of p averaged over 3 different experiment runs. *MuDRA* achieves a similar rate distribution for all values of p. *Our experiments show that MuDRA can achieve the target rate, maintain stability, and adds low overhead, even under severe network changing conditions.*

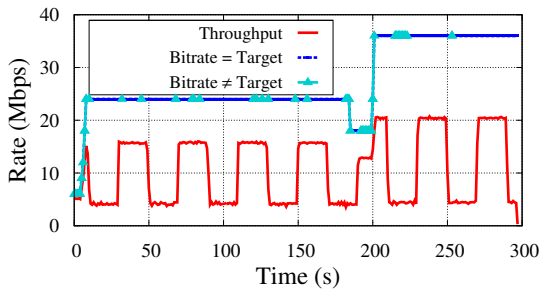


Figure 9.13: Multicast throughput with node 1-8 transmitting interfering on/off packet stream with node churn.

9.7.3 Impact of External Interference

We envision that *MuDRA* will be deployed in environments where the wireless infrastructure is centrally controlled. However, in-channel interference can arise from mobile nodes and other wireless transmissions. In addition to the uncontrolled interference spikes on ORBIT, we evaluate the impact of interference from a nearby node which transmits at the same channel as the multicast AP. We consider a scenario with two nodes near the center of the grid that exchange unicast traffic at a fixed rate of 6Mbps in a periodic on/off pattern with on and off periods 20s each. The transmission power of the interfering nodes is 0dBm which is equal to the transmission power of the multicast AP. This helps us evaluate the performance in the worst case scenario of continuous interference and study the dynamics of changing interference.

Fig. 9.12(a) shows the mid-PDR and abnormal nodes and Fig. 9.12(b) shows the rate and throughput for one experiment with 155 nodes. The number of mid-PDR nodes increases during the interference periods, due to losses from collisions. *MuDRA* converges to the target rate of 24Mbps. *Notice during interference periods, MuDRA satisfied the target-condition and that using the stability preserving method, MuDRA manages to preserve a stable rate.* The average throughput of different schemes with on/off background traffic for 3 experiments of 300s each is in Table 9.3. Pseudo-multicast achieves half while SRA has a third of the throughput of *MuDRA*. The fixed rate scheme achieves similar throughput as *MuDRA*.

The PDR distribution of nodes is in Fig. 9.12(c). *MuDRA satisfies QoS requirements while maintaining high throughput*. Pseudo-multicast scheme has 90% nodes with PDR more than 90%

since it makes backoff decisions from unicast ACKs. SRA yields 55% nodes with PDR less than 85% as it transmits at low rates. The fixed rate scheme yields 30% nodes with PDR less than 85%. The fixed rate scheme performs better than SRA since it maintains a higher rate. We also investigate the combined impact of *both interference and node churn*, where every 6s, the probability of a node switching on/off is $p = 0.2$. Fig. 9.13 shows the rate and throughput for this case. Similar to results in Section 9.7.2, *the performance of the system is not affected by node churn.*

9.7.4 Video multicast

We demonstrate the feasibility of using *MuDRA* for streaming video. The video is segmented with segment durations equal to the period of rate changes (1s) and each segment is encoded at several rates in H.264 format. For each time period, the key (I) frames are transmitted reliably at the lowest rate 6Mbps (note that transmitting the key frames can be achieved with 100% reliability even at 12Mbps on the testbed). The non-key (B and P) frames are transmitted at the rate set by *MuDRA*. At each instant, we know the expected throughput $\hat{D}_R$ for every rate R, the fraction of key frame data f_k, and the fraction of non-key frame data f_{nk}. Denote the expected throughput at 6Mbps by $\hat{D}_{min}$. The video rate can be calculated by solving linear equations $V_R = \frac{\hat{D}_{min} \cdot \hat{D}_R}{\hat{D}_{min} \cdot f_{nk} + \hat{D}_R \cdot f_k}$.

Let the multicast rate for current time period be R,the expected data throughput at this rate be $\hat{D}_R$, and the estimated throughput at the minimum rate be $\hat{D}_{min}$. Let f_k be the fraction of key frame data and f_{nk} be the fraction of non-key frame data. The video server has to determine the video rate V_R at each time t. Let the fraction of transmission time for key frames $T_k = \frac{V_R \cdot f_k}{\hat{D}_{min}}$ and fraction of transmission time for non-key frames $T_{nk} = \frac{V_R \cdot f_{nk}}{\hat{D}_R}$. We know that

$$t_k + t_{nk} = 1$$

The video rate can be calculated by solving linear equations $V_R = \frac{\hat{D}_{min} \cdot \hat{D}_R}{\hat{D}_{min} \cdot f_{nk} + \hat{D}_R \cdot f_k}$. In environments where estimates of throughput are inaccurate due to interference, techniques such as in [200] can be utilized.

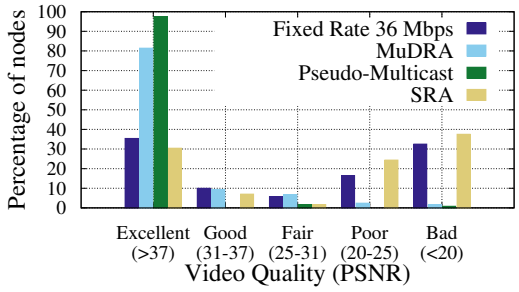


Figure 9.14: Distribution of video quality and PSNR (in brackets) measured at 160 nodes for different multicast schemes.

Experimental Results: We use raw videos from an online dataset [207] and encode the videos with H.264 standard. In our data sets, f_k is $15-20\%$. For *MuDRA* with throughput 19Mbps and FEC correction of 15%, we can support a video rate of $13-15$ Mbps, which is sufficient for 3 or 4 HD streams (each 4Mbps) on mobile devices. For each node, we generated the video streams offline by mapping the video frames to the detailed packet traces collected on ORBIT from an RA experiment. *For a fair comparison, the I frames were transmitted at* 6*Mbps for all schemes even though MuDRA can dynamically adjust the transmission rate to be much higher even for reliable transmissions.* In our experiments, we only considered a single video stream of rate V_R. We measured the PSNR of the video at each node and classified the PSNR in 5 categories based on visual perception[5].

Fig. 9.14 shows the video quality and PSNR ranges at the nodes for 3 experiments each of 300s and with $150-160$ nodes. With *MuDRA*, more than 90% of the nodes achieve excellent or good quality, 5% achieve fair quality, and less than 5% get poor or bad quality. While the pseudo-multicast scheme results in almost all nodes obtaining excellent quality, the video throughput for this scheme is significantly lower (8Mbps). SRA and the fixed rate schemes have more than 50% nodes with poor or bad video quality. The higher thorughput from *MuDRA* can allow streaming of several concurrent video streams or streams encoded at higher rates while ensuring QoS requirements.

[5]PSNR quantifies the distortion of the received as compared to the original transmitted video.

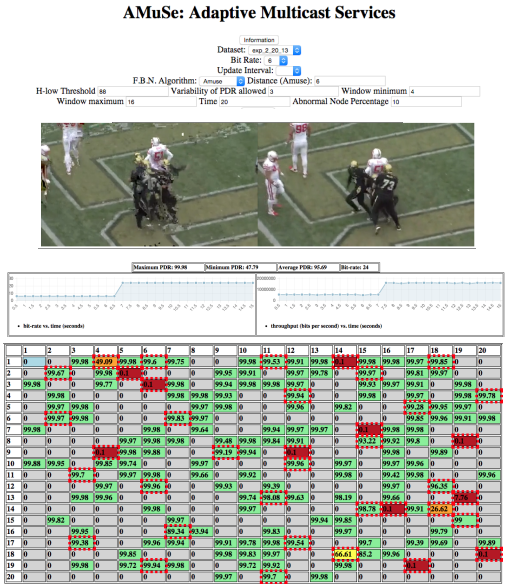


Figure 9.15: A screenshot of the web-based application for evaluating performance of *AMuSe* . The control panel for selecting the feedback and *MuDRA* algorithm parameters is on the top. The video at two selected nodes is shown below. In this example we show one node with poor quality and one with good quality video. The multicast throughput and other metrics are in the graphs. The performance of the client nodes is shown on the grid where numbers in each box indicate the PDR and the color of the box indicates the range of PDR. The nodes highlighted with a red border are FB nodes and nodes in grey are non-functional due to hardware issues.

9.8 Demonstration Application

To visually evaluate the performance of *AMuSe* and video delivery over *AMuSe* , we developed an interactive web-based application that illustrates the performance of the overall *AMuSe* system based on experimental traces collected on the ORBIT testbed. We collected the traces over several days in different experimental settings with 150-200 nodes. Each experimental trace consisted of channel measurements at each node using several metrics such Link Quality, Packet Delivery Ratio (PDR) etc.

The application allows considering different scenarios such as different channel conditions, interfering transmissions etc. For each scenario, the application shows the dynamic conditions over a period of time on the testbed from the appropriate experimental traces. The application can be used to compare the performance of several multicast schemes such as pseudo-multicast, unicast transmissions in different scenarios that have been measured on the testbed (e.g. interference, other

WiFi flows, etc.) as well as syntactic scenarios based on manipulating the measured data. We note that the application is flexible and can be used for testing even more scenarios and algorithms in the future.

The application has three main components: (i) the back-end where the experimental data is stored and managed, (ii) the front-end which provides the user interface, and (iii) a video tool for generating video streams. Both the front-end and the back-end are light weight applications. The front-end is web-based and can operate on any standard browser while the back-end requires installation of easily available open source libraries. For any experimental condition, the video tool generates the video stream received by a selected node. It maps video payload to UDP packets and discarding lost packet, according to the node's traces.

The front end is built using Angular [15] which is a JavaScript framework for rendering dynamic features on web applications. The FB node selection and *MuDRA* algorithms are built in the Django framework. The back-end utilizes a Postgres [162] database and interfaces with Django [53]. The algorithmic parameters can be tuned at any given time on the front-end. The front-end periodically relays the parameters to Django. Django utilizes the user input and system state information derived from the back-end to run the required FB and rate adaptation algorithms. The system state is then relayed to Angular, which renders the information on user's screen. Finally, the experiment can be paused at any time to allow the video tool to generate videos at the nodes for that period of time. The video tool uses ffmpeg to render and generate the videos and an nginx server [150] to transmit to the front-end.

The back-end utilizes a Postgres [162] database and interfaces with Django. The database is populated using the data derived from the experimental traces. The database consists of parameters several experimental parameters at each node at different times for each experimental scenario. This allows us to characterize the performance of the testbed with evolving channel conditions. The statistics about performance at each node are derived from the detailed packet traces. The feedback algorithms are built in the Django framework. The application is very flexible and allows other feedback algorithms to be incorporated as well. The users can change the feedback algorithms and

tune the algorithm specific parameters at any given time on the front-end.

The front end is built using Angular [15] which is a JavaScript framework designed for rendering dynamic features on web applications. The front-end periodically relays the user defined parameters to Django which runs the corresponding FB algorithm and responds with information (including the state of the nodes and the system) to Angular. Angular in turn renders the information on user's screen. The period of rendering at the front-end as well as calculation of system performance parameters can be changed by the user. Typically, we use a period of 500ms.

Fig. 9.15 shows a screenshot of the application. The application allows selecting different experiment settings such as AP bit rate, feedback algorithm, number of feedback nodes on the web interface. This information is used along with data collected from the experiments to show how the performance at all the nodes on the grid. The feedback nodes are highlighted with a red border. The rate adaptation and multicast throughput measured at the AP appears below. The information on the front-end is updated periodically.

Chapter 10: Dynamic Monitoring of Large Scale LTE-eMBMS

Previously, we focused on WiFi multicast in Chapter 8 and 9. In this chapter, we consider support for efficient LTE-eMBMS deployments in crowded and dynamic environments by providing accurate QoS reports with low overhead.

10.1 Introduction

LTE-eMBMS (evolved Multimedia Broadcast/Multicast Service) [4, 115] provides an alternative method for content delivery in crowded venues which is based on broadcasting to a large population of User Equipment (UEs) (a.k.a. eMBMS receivers). As illustrated in Fig. 10.1, in order to improve the Signal-to-Noise Ratio (SNR) at the receivers, eMBMS utilizes soft signal combining techniques.[1] *Thus, a large scale Modulation and Coding Scheme (MCS) adaptation should be conducted simultaneously for all the BSs based on the Quality of Service (QoS) at the UEs*.

Unfortunately, the eMBMS standard [4] only provides a mechanism for UE QoS reporting once the communication terminates, thereby making it unsuitable for real-time traffic. Recently, the Minimization of Drive Tests (MDT) protocol [5] was extended to provide eMBMS QoS reports periodically from all the UEs or when a UE joins/leaves a BS. However, in crowded venues with tens of thousands of UEs (e.g., [58]), even infrequent QoS reports by each UE may result in high signaling overhead and blocking of unicast traffic.[2] Due to the limited ability to collect feedback, a deployment of an eMBMS system is very challenging. In particular, it is hindered by the following limitations:

[1] All the BSs in a particular venue transmit identical multicast signals in a time synchronized manner.

[2] A BS can only support a limited number of connections while the minimal duration for an LTE connection is in the order of hundreds of milliseconds.

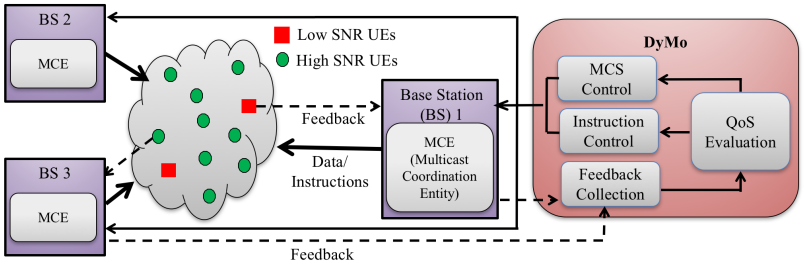


Figure 10.1: The *DyMo* system architecture: It exchanges control information with the Multicast Coordination Entity (MCE) of BSs which use soft signal combining for eMBMS. The Instruction Control module uses broadcast to dynamically partition the UEs into groups, each sending QoS reports at a different rate. The reports are sent to the Feedback Collection module and allow the QoS Evaluation module to identify an SNR Threshold. It is used by the MCS Control module to specify the optimal MCS to the MCEs.

(i) *Extensive and time consuming radio frequency surveys*: Such surveys are conducted before each new eMBMS deployment. Yet, they provide only limited information from a few monitoring nodes.

(ii) *Conservative resource allocation*: The eMBMS MCS and Forward Error Correction (FEC) codes are set conservatively to increase the decoding probability.

(iii) *Oblivious to environmental changes*: It is impossible to infer QoS degradation due to environmental changes, such as new obstacles or component failures.

Clearly, there is a need to dynamically tune the eMBMS parameters according to the feedback from UEs. However, a key challenge for eMBMS parameter tuning for large scale groups is *obtaining accurate QoS reports with low overhead*. Schemes for efficient feedback collection in wireless multicast networks have recently received considerable attention, particulalty in the context of WiFi networks (e.g., [206, 71, 217, 60]). Yet, WiFi feedback schemes cannot be easily adapted to eMBMS since unlike WiFi, where a single Access Point transmits to a node, transmissions from multiple BSs are combined in eMBMS. Efforts for optimizing eMBMS performance focus on periodically collecting QoS reports from all UEs (e.g., [31]) but such approaches rely on extensive knowledge of the user population (for more details, see Section 10.2.2).

In this chapter, we present the *Dynamic Monitoring* (*DyMo*) system designed to support ef-

ficient LTE-eMBMS deployments in crowded and dynamic environments by providing accurate QoS reports with low overhead. *DyMo* identifies the maximal eMBMS *SNR Threshold* such that only a small number of UEs with SNR below the SNR Threshold may suffer from poor service[3]. To identify the SNR Threshold accurately, *DyMo* leverages the broadcast capabilities of eMBMS for fast dissemination of instructions to a large UE population.

Each instruction is targeted at a sub-group of UEs that satisfies a given condition. It instructs the UEs in the group to send a QoS report with some probability during a reporting interval.[4] We refer to these instructions as *Stochastic Group Instructions*. For instance, as shown in Fig. 10.2, *DyMo* divides UEs into two groups. UEs with poor or moderate eMBMS SNR are requested to send a report with a higher rate during the next reporting interval. In order to improve the accuracy of the SNR Threshold, the QoS reports are analyzed and the group partitions and instructions are dynamically adapted such that the UEs whose SNR is around the SNR Threshold report more frequently. The SNR Threshold is then used for setting the eMBMS parameters, such as the MCS and FEC codes.

From a statistics perspective, *DyMo* can be viewed as a practical method for realizing *importance sampling* [154] in wireless networks. Importance sampling improves the expectation approximation of a rare event by sampling from a distribution that overweighs the important region. With limited knowledge of the SNR distribution, *DyMo* leverages Stochastic Group Instructions to narrow down the SNR sampling to UEs that suffer from poor service and consequently obtains accurate estimation of the SNR Threshold. To the best of our knowledge, this is the first realization of using broadcast instructions for importance sampling in wireless networks.

The *DyMo* system architecture is illustrated in Fig. 10.1. It operates on an independent server and exchanges control information with several BSs supporting eMBMS. The Instruction Control module instructs the different groups to send reports at different rates. The reports are sent via unicast to the Feedback Collection module and allow the QoS Evaluation module to identify an

[3]While various metrics can be used for QoS evaluation, we consider the commonly used eMBMS SNR, referred to as SNR, as a primary metric.

[4]A higher probability results in a higher reporting rate, and therefore, we will use rate and probability interchangeably.

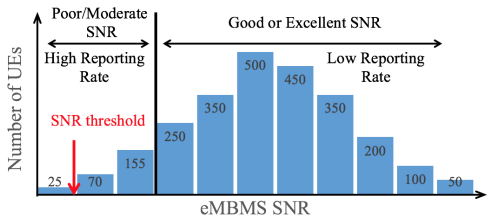


Figure 10.2: Operation of *DyMo* for a sample UE QoS distribution: UEs are partitioned into two groups based on their SNR and each group is instructed to send QoS reports at a different rate. The partitioning is dynamically adjusted based on the reports to yield more reports from UEs whose SNR is around the estimated SNR Threshold.

accurate SNR Threshold. The SNR Threshold is determined such that only a predefined number of UEs with SNR below the threshold, termed as *outliers*, may suffer from poor service. The MCS Control module utilizes the SNR Threshold to configure the eMBMS parameters (e.g., MCS) accordingly. Finally, the QoS Evaluation module continually refines group partitions based on the reports.

We focus on the QoS Evaluation module and develop a *Two-step estimation* algorithm which can efficiently identify the SNR Threshold as a one time estimation. We also develop an *Iterative estimation* algorithm for estimating the SNR Threshold iteratively, when the distribution changes due to UE mobility or environmental changes, such as network component failures. Our analysis shows that the *Two-step estimation* and *Iterative estimation* algorithms can infer the SNR Threshold with a small error and limited number of QoS reports. It is also shown that they outperform the *Order-Statistics estimation* method, a well-known statistical method, which relies on sampling UEs with a fixed probability. For instance, the *Two-step estimation* requires only 400 reports when estimating the $1th$ percentile to limit the error to 0.3% for each re-estimation. The *Iterative estimation* algorithm performs even better than the *Two-step estimation* and the maximum estimation error can be bounded according to the maximum change of SNR Threshold.

We conduct extensive at-scale simulations, based on real eMBMS radio survey measurements from a stadium and an urban area. It is shown that *DyMo* accurately infers the SNR Threshold and optimizes the eMBMS parameters with low overhead under different mobility patterns and even in the event of component failures. *DyMo* significantly outperforms alternative schemes based on the

Order-Statistics estimation method which rely on random or periodic sampling.

Our simulations show that both in a stadium-like and urban area, *DyMo* detects the *eMBMS SNR value of the* 0.1% *percentile with Root Mean Square Error (RMSE) of* 0.05% with only 5 messages per second in total across the whole network. This is at least 8 times better than *Order-Statistics estimation* based methods. *DyMo* also infers the optimal SNR Threshold with RMSE of 0.3 dB regardless of the UE population size, while the error of the best *Order-Statistics estimation* method is above 1 dB. *DyMo* violates the outlier bound (of 0.1%) with RMSE of at most 0.35 while the best *Order-Statistics estimation* method incurs RMSE of over 4 times as compared to *DyMo*. The simulations also show that after a failure, *DyMo* converges instantly (i.e., in a single reporting interval) to the optimal SNR Threshold. Thus, *DyMo* is able to infer the maximum MCS while preserving QoS constraints.

To summarize, the main contributions of this chapter are three-fold:

(i) We present the concept of Stochastic Group Instructions for efficient realization of importance sampling in wireless networks.

(ii) We present the system architecture of *DyMo* and efficient algorithms for SNR Threshold estimation.

(iii) We show via extensive simulations that *DyMo* performs well in diverse scenarios.

The principal benefit of *DyMo* is its ability to infer the system performance based on a low number of QoS reports. It converges very fast to the optimal eMBMS configuration and it reacts very fast to changes in the environment. Hence, it eliminates the need for service planning and extensive field trials. Further, *DyMo* is compatible with existing LTE-eMBMS deployments and does not need any knowledge of the UE population.

The rest of the chapter is organized as follows. We provide background information about eMBMS and a brief review of related work in Section 10.2. We introduce the model and objective in Section 10.3. We present the *DyMo* system in Section 10.4. The algorithms for SNR threshold estimation with their analysis are given in Section 10.5. The numerical evaluation results appear in Section 10.6 while some details of our analysis are given in the Appendix.

The description and evaluation of the *DyMo* system appeared in the proceedings of IEEE INFOCOM'17 [24] and was selected as a **best paper runner-up**. An extended version was fast-tracked to IEEE/ACM Transactions on Networking [23]. The design and evaluation of *DyMo* was based on significant contributions from Ph.D student Dr. Varun Gupta and co-authors at Bell Labs especially, Dr. Yigal Bejerano, Dr. Chandru Raman, and Dr. Chun-Nam Yu.

10.2 Related Work

10.2.1 eMBMS Background

LTE-Advanced networks provide broadcast services by using *evolved Multimedia Broadcast/Multicast Service* (eMBMS) [115]. eMBMS is best suited to simultaneously deliver common content like video distribution to a large number of users within a contiguous region of cells. eMBMS video distribution is offered as an unidirectional service without feedback from the UE nor retransmissions of lost packets. This is enabled by all cells acting in a coordinated Single Frequency Network (SFN) arrangement, i.e., transmitting identical signals in a time synchronized manner, called *Multicast Broadcast Single Frequency Network* (MBSFN). The identical signals combine over the air in a non-coherent manner at each of the user locations, resulting in an improved Signal-Noise Ratio (SINR). Thus, what is normally out-of-cell interference in unicast becomes useful signal in eMBMS. For avoiding further interference from cells not transmitting the same MBSFN signal, the BSs near the boundary of the MBSFN area are used as a *protection tier* and they should not include eMBMS receivers in their coverage areas.

10.2.2 Related Work

Most previous work on eMBMS (e.g., [232, 188, 135, 35]) assumes individual feedback from all the UEs and proposes various MCS selection or resource allocation techniques. Yet, extensive QoS reports impose significant overhead on LTE networks, which are already highly congested in crowded venues [58]. An efficient feedback scheme was proposed in [31] but it relies on knowledge of path loss (or block error) of the entire UE population to form the set of feedback nodes.

Table 10.1: Notation for *DyMo* model.

Symbol	Semantics
m	The number of UEs in the venue, also the number of active eMBMS receiver in static settings.
$m(t)$	The number of active eMBMS receivers at time t.
$h_v(t)$	The individual SNR value of UE v at time interval t.
$s(t)$	The SNR Threshold at time t.
p	*QoS Threshold* - The maximal portion of UEs with individual SNR value $h_v(t) < s(t)$.
r	*Overhead Threshold* - An upper bound on the average number of reports in a reporting interval.

Recently, [225] proposed a multicast-based anonymous query scheme for inferring the maximum MCS that satisfies *all UEs* without sending individual queries. However, the scheme cannot be implemented in current LTE networks, since it will require UEs to transmit simultaneous beacon messages in response to broadcast queries.

Most of the wireless multicast schemes are designed for WiFi networks and a comprehensive survey of WiFi multicast feedback approaches was described in Section 8.2. However, WiFi multicast solutions cannot easily be applied to LTE-eMBMS systems. First, in WiFi, each node is associated with an Access Point, and therefore, the Access Point is aware of every node and can specify the feedback nodes. In LTE, eMBMS UEs could be in the idle state and *the network may not be aware of the number of active UEs*. Second, eMBMS is based on simultaneous transmission from various BSs. Thus, unlike in WiFi where MCS adaptation is done at each Access Point independently, a common MCS adaptation should be done at all BSs.

10.3 Model and Objective

10.3.1 Network Model

We consider an LTE-Advanced network with multiple base stations (BSs) providing eMBMS service to a very large group of m UEs in a given large venue (e.g., sports arena, transportation hub).[5] Such venues can accommodate tens of thousands of users. The eMBMS service is managed

[5]In this chapter, we consider only the UEs subscribing to eMBMS services.

by a single *DyMo* server as shown in Fig. 10.1 and all the BSs transmit identical multicast signals in a time synchronized manner. The multicast flows contain FEC code that allows the UEs to tolerate some level of losses ℓ (e.g., up to 5% packet losses).

All UEs can detect and report the eMBMS QoS they experience. More specifically, time is divided into short *reporting intervals*, a few seconds each. We assume that the eMBMS SNR distribution of the UEs does not change during each reporting interval.[6] We define the *individual SNR value* $h_v(t)$, such that at least a given percentage $1 - \ell$ (e.g., 95%) of the eMBMS packets received by an UE v during a reporting interval t have an SNR above $h_v(t)$. For a given SNR value, $h_v(t)$, there is a one-to-one mapping to an *eMBMS MCS* such that a UE can decode all the packets whose SNR is above $h_v(t)$ [135, 35]. The remaining packets ℓ can be recovered by appropriate level of FEC assuming ℓ is not too large. A summary of the main notations used throughout the chapter are given in Table 10.1.

10.3.2 Objective

We aim to design a scalable efficient eMBMS monitoring and control system for which the objective is outlined below and that satisfies the following constraints:

(i) *QoS Constraint* – Given a *QoS Threshold* $p \ll 1$, at most a fraction p of the UEs may suffer from packet loss of more than ℓ. This implies that, *with FEC, a fraction* $1 - p$ *of the UEs should receive all of the transmitted data*. We refer to the set UEs that suffer from packet loss after FEC as *outliers* and the rest are termed *normal* UEs.

(ii) *Overhead Constraint* – The average number of UE reports during a reporting interval should be below a given *Overhead Threshold* r.

Objective: *Accurately identify at any given time* t *the maximum SNR Threshold,* $s(t)$ *that satisfies the QoS and Overhead Constraints.*

[6]The SNR of each individual eMBMS packet is a random variable selected from the UE SNR distribution. We assume that this distribution does not change significantly during the reporting interval.

Namely, the calculated $s(t)$ needs to ensure that a fraction $1 - p$ of the UEs have individual SNR values $h_v(t) \geq s(t)$.

The network performance can be maximized by using $s(t)$ to calculate the maximum eMBMS MCS that meets the QoS constraint [135, 35]. This allows reducing the resource blocks allocated to eMBMS. Alternatively for a service such as video, the video quality can be enhanced without increasing the bandwidth allocated to the video flow.

10.4 The *DyMo* System

This section introduces the *DyMo* system. It first presents the *DyMo* system architecture, which is based on the *Stochastic Group Instructions* concept. Then, it provides an illustrative example of *DyMo* operations along with some technical aspects of eMBMS parameter tuning.

10.4.1 System Overview

We now present the *DyMo* system architecture, shown in Fig. 10.1.

Feedback Collection: This module operates in the *DyMo* server and in a *DyMo Mobile-Application* on each UE. At the beginning of each reporting interval, the Feedback Collection module broadcasts *Stochastic Group Instructions* to all the UEs. These instructions specify the QoS report probability as a function of the observed QoS (i.e., eMBMS SNR). In response, each UE independently determines whether it should send a QoS report at the current reporting interval.

QoS Evaluation: The UE feedback is used to estimate the eMBMS SNR distribution, as shown in Fig. 10.2. Since the system needs to determine the SNR Threshold, $s(t)$, the estimation of the low SNR range of the distribution has to be more accurate. To achieve this goal, the QoS Evaluation module partitions the UEs into two or more groups, according to their QoS values. This allows *DyMo* to accurately infer the optimal value of $s(t)$, by obtaining more reports from UEs with low SNR. We elaborate on the algorithms for $s(t)$ estimation in Section 10.5.

MCS Control: Since the eMBMS signal is a combination of synchronized multicast transmissions from several BSs, the unicast SNR can be used as a lower bound on the eMBMS SNR. Therefore,

Table 10.2: Example of the *DyMo* feedback report overhead.

Group	No. of UEs	Report Prob.	Avg. reports per interval	Avg. per sec	Rate per min
H	250	20%	50	5	≈ 100%
L	2250	2%	45	≈ 5	≈ 12%

the initial eMBMS MCS and FEC are determined from unicast SNR values reported by the UEs during unicast connections. Then, after each reporting interval, the QoS Evaluation module infers the SNR Threshold, $s(t)$, and the MCS Control module determines the desired eMBMS settings, mainly the eMBMS MCS and FEC, according to commonly used one-to-one mappings [135, 35].

10.4.2 Illustrative Example

DyMo operations and the Stochastic Group Instructions concept are demonstrated in the following example. Consider an eMBMS system that serves $2,500$ UEs with the QoS Constraint that at most $p = 1\% = 25$ UEs may suffer from poor service. Assume a reporting interval of 10 seconds. To infer the SNR Threshold, $s(t)$, that satisfies the constraint, the UEs are divided into two groups:

- *High-Reporting-Rate* (H): 10% (250) of UEs that experience poor or moderate service quality report with probability of 20%, i.e., an expected number of 50 reports per interval.
- *Low-Reporting-Rate* (L): 90% (2250) of the UEs that experience good or excellent service quality report with probability of 2%, implying about 45 reports per interval.

Table 10.2 presents the reporting probability of each UE and the number of QoS reports per reporting interval by each group. It also shows the number of QoS reports per second and the reporting rate per minute (i.e., the expected fraction of UEs that send QoS reports in a minute). Since the QoS Constraint implies that only 25 UEs may suffer from poor service, these UEs must belong to group H. Although only 10 QoS reports are received at each second, all the UEs in group H send QoS reports at least once a minute. Thus, the SNR Threshold can be accurately detected within one minute.

10.4.3 Dynamic eMBMS Parameter Tuning

Besides the MCS, *DyMo* can leverage the UE feedback and the calculated SNR Threshold, $s(t)$, for optimizing other eMBMS parameters including FEC, video coding and protection tier. While this aspect is not the focus of this study, we briefly discuss the challenges and the solutions for dynamic tuning of the eMBMS parameters.

Once the SNR Threshold $s(t)$ is selected, *DyMo* tunes the eMBMS parameters accordingly. Every time *DyMo* changes the eMBMS parameters, the consumption of wireless resources for the service is affected as well. For instance, when the eMBMS MCS index is increased, some of the wireless resources allocated for eMBMS are not needed and can be released. Alternatively, the service provider may prefer to improve the video quality by instructing the content server to increase the video resolution. Similarly, before the eMBMS MCS index is lowered, the wireless resources should be increased or the video resolution should be reduced to match the content bandwidth requirements to the available wireless resources.

Since the eMBMS signal is a soft combination of the signals from all BSs in the venue, any change of eMBMS parameters must be synchronized at all the BSs to avoid interruption of service. The fact that all the clocks of the BSs are synchronized can be used and a scheme similar to the *two phase commit* protocol (which is commonly used in distributed databases [186]) can be used.

10.5 Algorithms for SNR Threshold Estimation

This section describes the algorithms utilized by *DyMo* for estimating the SNR Threshold, $s(t)$, for a given QoS Constraint, p and Overhead Constraint r. In particular, it addresses the challenges of partitioning the UEs into groups according to their SNR distribution as well as determining the group boundaries and the reporting rate from the UEs in each group, such that the overall estimation error of $s(t)$ is minimized. We first consider a static setting with fixed number of eMBMS receivers, m, where the SNR values of UEs are fixed. Then, we extend our solution to the case of dynamic environments and UE mobility.

10.5.1 Order Statistics

We first briefly review a known statistical method in quantile estimation, referred to as *Order-Statistics estimation*. It provides a baseline for estimating $s(t)$ and is also used by *DyMo* for determining the initial SNR distribution in its first iteration assuming a single group. Let $F(x)$ be a Cumulative Distribution Function (CDF) for a random variable X, the quantile function $F^{-1}(p)$ is given by, $\inf\{x \mid F(x) \geq p\}$.

Let $X_1, X_2, \ldots, X_r$ be a sample from the distribution F, and F_r its empirical distribution function. It is well known that the empirical quantile $F_r^{-1}(p)$ converges to the population quantile $F^{-1}(p)$ at all points p where F^{-1} is continuous [204]. Moreover, the true quantile, $\hat{p} = F(F_r^{-1}(p))$, of the empirical quantile estimate $F_r^{-1}(p)$ is asymptotically normal [204] with mean p and variance

$$\mathbb{V}\text{ar}[\hat{p}] = \frac{p(1-p)}{r} \tag{10.1}$$

For SNR Threshold estimation, F is the SNR distribution of all UEs. A direct way to estimate the SNR Threshold $s(t)$ is to collect QoS reports from r randomly selected UEs, and calculate the empirical quantile $F_r^{-1}(p)$ as an estimate.[7]

10.5.2 The Two-Step Estimation Algorithm

We now present the *Two-step estimation* algorithm that uses two groups for estimating the SNR Threshold, $s(t)$, in a static setting. We assume a fixed number of UEs, m, and a bound r on the number of expected reports. By leveraging *Stochastic Group Instructions*, *DyMo* is not restricted to collecting reports uniformly from all UEs and can use these instructions to improve the accuracy of $s(t)$. One way to realize this idea is to perform a two-step estimation that approximates the shape of the SNR distribution before focusing on the low quantile tail. The *Two-step estimation*

[7]Note that F can have at most m points of discontinuity. Therefore, we assume p is a point of continuity for F^{-1} to enable normal approximation. If the assumption does not hold, we can always perturb p by an infinitesimal amount to make it a point of continuity for F^{-1}.

algorithm works as follows:

Algorithm 1: Two-Step Estimation for the Static Case

1. Select p_1 and p_2 such that $p_1 p_2 = p$. Use p_1 as the percentile boundary for defining the two groups.

2. Select number of reports r_1 and r_2 for each step such that $r_1 + r_2 = r$.

3. Instruct all UEs to send QoS reports with probability r_1/m and use these reports to estimate the p_1 quantile $\hat{x}_1 = F_{r_1}^{-1}(p_1)$.

4. Instruct UEs with SNR value below $\hat{x}_1$ to send reports with probability $r_2/(p_1 \cdot m)$ and calculate the p_2 quantile $\hat{x}_2 = G_{r_2}^{-1}(p_2)$ as an estimation for $s(t)$ (G is the CDF of the subpopulation with SNR below $\hat{x}_1$). (G_{r_2} is the empirical CDF of the subpopulation with SNR below $\hat{x}_1$).

Upper Bound Analysis of the Two-Step Algorithm: To simplify the notation, we use r_1 and r_2 to denote the expected number of reports at each step. From (10.1) we know that

$$\hat{p}_1 = F(\hat{x}_1) \quad and \quad \hat{p}_2 = G(\hat{x}_2)$$

are unbiased estimators of p_1 and p_2 with variance

$$\mathbb{V}\text{ar}[\hat{p}_1] = \frac{p_1(1-p_1)}{r_1} \quad and \quad \mathbb{V}\text{ar}[\hat{p}_2] = \frac{p_2(1-p_2)}{r_2} \tag{10.2}$$

Our estimate $\hat{x}_2$ has true quantile $\hat{p}_1\hat{p}_2$. Assume $\hat{p}_1$ is less than $p_1 + \epsilon_1$ and $\hat{p}_2$ is less than $p_2 + \epsilon_2$ with high probability (for example, we can take ϵ_1 and ϵ_2 to be 3 times the standard deviation for $> 99.8\%$ probability). Then, the over-estimation error is bounded by

$$\begin{aligned} \epsilon &= (p_1 + \epsilon_1)(p_2 + \epsilon_2) \; - \; p \\ &= p_1 p_2 + \epsilon_1 p_2 + \epsilon_2 p_1 + \epsilon_1 \epsilon_2 - p \\ &\approx \epsilon_1 p_2 + \epsilon_2 p_1 \end{aligned} \tag{10.3}$$

after ignoring the small higher order term $\epsilon_1\epsilon_2$. The case for under-estimation is similar. As shown in the Appendix, the error is minimized by taking,

$$p_1 = p_2 = \sqrt{p} \quad and \quad r_1 = r_2 = r/2$$

so that

$$\epsilon_1 = \epsilon_2 = 3\sqrt{\sqrt{p}(1-\sqrt{p})/(r/2)}$$

This leads to proposition 2.

Proposition 2. *The distance between p and the quantile of the Two-Step estimator $\hat{x}_2$, $\hat{p} = F^{-1}(x_2)$, is bounded by*

$$6\sqrt{2}\sqrt{\frac{p\sqrt{p}(1-\sqrt{p})}{r}}$$

with probability at least $1 - 2(1 - \Phi(3)) > 99.6\%$*, where* Φ *is the normal CDF.*

We now compare this result against the bound of 3 standard deviations in the Order Statistics case, which is $3\sqrt{p(1-p)/r}$. With some simple calculations, it can be easily shown that if $p \leq 1/49 \approx 2\%$, the *Two-step estimation* has smaller error than the *Order-Statistics estimation* method. Essentially the *Order-Statistics estimation* method has an error of order $\sqrt{p}/\sqrt{r}$, while the *Two-step estimation* has an error of order $p^{3/4}/\sqrt{r}$. Since $p \ll 1$, the difference can be significant.

Example: We validated the error estimation of the *Two-step estimation* algorithm and the *Order-Statistics estimation* method by numerical analysis. We considered the cases of $p = 1\%$ and $p = 0.1\%$ of uniform distribution on $[0, 1]$ using $r = 400$ samples over population size of 10^6. The *Two-step estimation* algorithm has smaller standard error compared to the *Order-Statistics estimation*, as shown in Fig. 10.3. Its accuracy is significantly better for very small p.

The *Two-step estimation* algorithm can be generalized to 3 or more telescoping group sizes, but p will need to be much smaller for these sampling schemes in order to reduce the number of

samples.

10.5.3 The Iterative Estimation Algorithm

We now turn to the dynamic case in which *DyMo* uses the SNR Threshold estimation $s(t-1)$ from the previous reporting interval to estimate $s(t)$ at the end of reporting interval t. Assume that the total number of eMBMS receivers, m, is fixed and it is known initially.

Suppose that *DyMo* has a current estimate $\hat{x}$ of the SNR threshold, $s(t)$, and $s(t)$ changes over time. We assume that the change in SNR of each UE is bounded over a time period. Formally,

$$|h_v(t_1) - h_v(t_2)| \leq L|t_1 - t_2|$$

where L is a Lipschitz constant for SNR changes. For example, we can assume that the UEs' SNR cannot change by more than 5dB during a reporting interval. [8] This implies that within the interval, only UEs with SNR below $\hat{x}+5$dB affect the estimation of the p quantile (subject to small estimation error in $\hat{x}$).

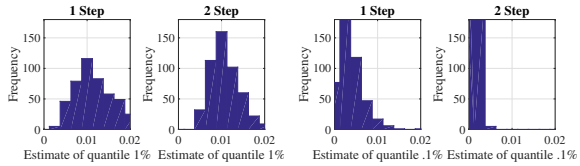
DyMo only needs to monitor UEs with SNR below $x_L = \hat{x} + L$. Denote the true quantile of x_L, defined by $F^{-1}(x_L)$, as p_L. To apply a process similar to the second step of the *Two-step estimation* algorithm by focusing on UEs with SNR below x_L, first an estimate of p_L is required. *DyMo* uses the previous SNR distribution to estimate p_L and instructs the UEs to send reports at a rate $q = r/(p_L \cdot m)$. Let Y be the number of reports received during the last reporting interval, then $Y/m \cdot q$ can be used as an updated estimator, $\hat{p_L}$, for p_L. This estimator is unbiased and has variance

$$\mathbb{V}\text{ar}[\hat{p_L}] = \mathbb{V}\text{ar}[\frac{Y}{m \cdot q}] = \frac{p_L}{m}\frac{1-q}{q} \tag{10.4}$$

As a result, the Iterative Estimation algorithm works as follows:

Algorithm 2: Iterative Estimation for the Dynamic Case

[8] In our simulations, each reporting interval has a duration of 12s.



(a) (b)

Figure 10.3: Estimates of (a) $p = 1\%$ and (b) $p = 0.1\%$ quantiles for 500 runs for the *Order-Statistics estimation* (1-step) method and the *Two-step estimation* algorithm.

1. Instruct UEs with SNR below $\hat{x} + L$ to send reports at a rate q. Construct an estimator $\hat{p}_L$ of p_L from the number of received reports Y.

2. Set $p' = p/\hat{p}_L$. Find the p' quantile $x' = G_Y^{-1}(p')$ and report it as the p quantile of the whole population (G is the CDF of the subpopulation with SNR below $\hat{x} + L$).

Upper Bound Analysis of the Iterative Algorithm: Suppose the estimation error of p_L is bounded by ϵ_1, and the estimation error of $p' = p/\hat{p}_L$ is bounded by ϵ_2 with high probability. Then, the estimation error is

$$\epsilon = (\frac{p}{\hat{p}_L} \pm \epsilon_2)p_L - p = (\frac{p}{p_L \pm \epsilon_1} \pm \epsilon_2)p_L - p.$$

The over-estimation error is bounded by

$$\frac{p}{p_L - \epsilon_1}\epsilon_1 + p_L\epsilon_2. \tag{10.5}$$

If we assume $p_L - \epsilon_1 \geq p$ (we know $p_L \geq p$ by the Lipschitz assumption), then the bound can be simplified to $\epsilon_1 + p_L\epsilon_2$. The same bound also works for the under-estimation error. If r denotes also the expected number of samples collected, $r = p_L \cdot m \cdot q$. The standard deviation of $\hat{p}_L$ can be written as:

$$\sqrt{\frac{p_L}{m}\frac{1-q}{q}} = \sqrt{\frac{p_L^2}{r}(1 - \frac{r}{p_L m})} \leq \frac{p_L}{\sqrt{r}}.$$

If we assume $\epsilon_1 = 3p_L/\sqrt{r}$, the error of $\hat{p}_L$ is less than ϵ_1 with probability at least $\Phi(3)$. Since we assume $p_L - \epsilon_1 \geq p$ above, this implies $(1 - 3/\sqrt{r})p_L \geq p$. If $r \geq 100$, then $p < 0.7p_L$ will satisfy our requirement.

The standard deviation of estimating the $p' = p/\hat{p}_L$ quantile is

$$\sqrt{\frac{1}{Y}\frac{p}{\hat{p}_L}(1-\frac{p}{\hat{p}_L})} \leq \frac{1}{2\sqrt{Y}}, \tag{10.6}$$

by using the fact that $x(1-x) \leq 1/4$ for $x \in [0,1]$ and Y is the number of reports received (a random variable). If the expected number of reports r is reasonably large (≥ 100, say), then Y can be well approximated by a normal and $Y \geq 0.7r$ with high probability $\Phi(3) = 99.8\%$. Then, (10.6) is bounded by $2/(3\sqrt{r}) \geq 1/(2\sqrt{0.7r})$ with high probability ($\Phi(3) = 99.8\%$), and we can set $\epsilon_2 = 2/\sqrt{r}$. Substituting these back into (10.5), gives us the following proposition.

Proposition 3. *The distance between p and the quantile of the estimator x, $\hat{p} = F^{-1}(x)$, is approximately bounded by*

$$5\frac{p_L}{\sqrt{r}}$$

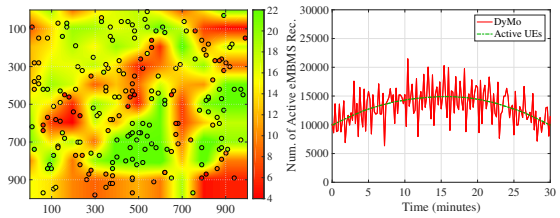
with probability at least $1 - 2(1 - \Phi(3)) > 99.6\%$, if the expected sample size $r \geq 100$ and $p \leq 0.7p_L$.

This shows that the error is of order $p_L/\sqrt{r}$. We can see that the estimation error can be smaller compared to the error of order $p^{3/4}/\sqrt{r}$ in the static *Two-step estimation* if p_L is small (i.e., the SNR of individual users does not change much during a reporting interval).

Exponential Smoothing: *DyMo* applies exponential smoothing by weighing past and current reports to smooth the estimates of the SNR Threshold, $s(t)$, and take older reports into account. It estimates the SNR Threshold as

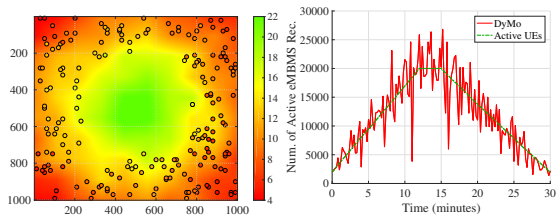
$$s(t) = \alpha\hat{x}(t) + (1-\alpha)s(t-1)$$

where $\hat{x}(t)$ is the new raw SNR Threshold estimate using the *Iterative estimation* above and $s(t-1)$ is the SNR Threshold from the previous reporting interval. We set $\alpha = 0.5$ to allow some re-use of past reports without letting them have too strong an effect on the estimates (e.g., samples older than 7 reporting intervals have less than 1% weight). *DyMo* also uses the exponential smoothing



(a) (b)

Figure 10.4: (a) The heatmap of SNR distribution of UEs (b) the evolution of the number of active UEs over time compared to the number estimated by *DyMo* for a homogeneous environment.



(a) (b)

Figure 10.5: (a) The heatmap of UE SNR distribution in a stadium area of $1000 \times 1000m^2$ and (b) the evolution of the number of active UEs over time compared to the number estimated by *DyMo* for a stadium environment.

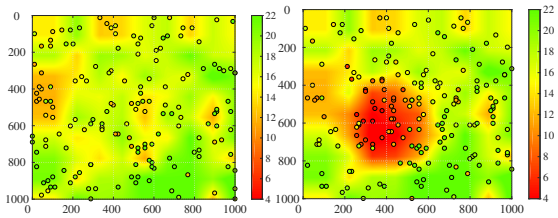
method for estimating the SNR distribution while taking into account QoS reports from previous reporting intervals.

Dynamic and Unknown Number of eMBMS Receivers: If the total number of UEs, $m(t)$, is unknown or changes dynamically, *DyMo* can estimate $m(t)$ by requiring UEs above the threshold $\hat{x} + L$ to send reports. These UEs can send reports at a lower rate, since $m(t)$ is not expected to change rapidly. Similar to the *Two-step estimation* algorithm, *DyMo* allocates $r_1 = r_2 = r/2$ reports to each group. The errors in estimating the total number of UEs $m(t)$ will contribute to the error ϵ_1 in the estimation of p_L in (10.5). The error analysis in this case is largely similar.

10.6 Performance Evaluation

10.6.1 Methodology

We perform extensive simulations to evaluate the performance of *DyMo* with various values of QoS Constraint, p, Overhead Constraint, r, and number of UEs, m. Our evaluation considers dy-



(a) (b)

Figure 10.6: The heatmap of the SNR distribution of UEs (a) before a failure and (b) after a failure.

namic environments with UE mobility and a changing number of *active eMBMS receivers* denoted by $m(t)$, dynamically selected from the given set of m UEs in the considered venue. In this chapter, we present a few sets of simulation results, which capture various levels of variability of the SNR threshold, $s(t)$, over time.

We consider a variant of *DyMo* where the number of active UEs is unknown and is estimated from its measurements. We compare the performance of *DyMo* to four other schemes. To demonstrate the advantages of *DyMo*, we augment each scheme with additional information, which is hard to obtain in practice. The evaluated benchmarks are the following:

• ***Optimal*** – Full knowledge of SNR values of the UEs at any time and consequently accurate information of the SNR distribution. *This is the best possible benchmark although impractical, due to its high overhead.*

• ***Uniform*** – Full knowledge of the SNR characteristics at any location while assuming uniform UE distribution and static eMBMS settings. In practice, *this knowledge cannot be obtained even with rigorous field trial measurements.*

• ***Order-Statistics*** – It is based estimation of the SNR Threshold using random sampling. The active UEs send reports with a fixed probability of $r/\mathbb{E}[m(t)]$ per second, assuming that the expected number of active UEs, $\mathbb{E}[m(t)]$, is known. We assume that the UEs are configured with this reporting rate during initialization. In practice, $\mathbb{E}[m(t)]$ *is not available. We also ignore initial configuration overhead in our evaluation*. *Order-Statistics* is the best possible approach when not using broadcast messages for UE configuration. We consider two variants of *Order-Statistics*. The first is ***Order-Statistics w.o. History*** which ignores SNR measurements from earlier reporting

intervals. The second variant ***Order-Statistics w. History*** considers the history of reports.

Both *DyMo* and *Order-Statistics w. History* perform the same exponential smoothing process for assigning weights to the measurements from previous reporting intervals with a smoothing factor of $\alpha = 0.5$. We use the following metrics to evaluate the performance of the schemes:

(i) *Accuracy* – The accuracy of the *SNR Threshold* estimation, $s(t)$. After calculating $s(t)$ at each reporting interval, we check the actual *SNR Threshold Percentile* in the accurate SNR distribution of the considered scheme. This metric provides the percentile of active UEs with individual SNR values below $s(t)$.

(ii) *QoS Constraint violation* – The number of outliers above the QoS Constraint p. The number of outliers of a scheme in a given reporting interval t is defined as the actual SNR Threshold Percentile of the scheme times the number of active eMBMS receivers, $m(t)$, at time t.

(iii) *Overhead Constraint violation* – The number of reports above the Overhead Threshold, r, at each reporting interval.

The total simulation time for each instance is 30mins with 5 reporting intervals per minute (each is 12s). During each reporting interval, an active UE may send its SNR value at most once. The accuracy of each SNR report is 0.1dB.

10.6.2 Simulated Environments

We simulated a variety of environments with different SNR distributions and UE mobility patterns. Although the simulated environments are artificial, their SNR distributions mimic those of real eMBMS networks obtained through field trial measurements. To capture the SNR characteristics of an environment, we divide its geographical area into rectangles of $10m \times 10m$. For each reporting interval, each UE draws its individual SNR value, $h_v(t)$, from a Gaussian-like distribution which is a characteristic of the rectangle in which its located. The rectangles have different mean SNR, but the same standard deviation of roughly 5dB (as observed in real measurements).

Thus, the SNR characteristics of each environment are determined by the mean SNR values of the rectangles at any reporting interval.

To demonstrate the performance of the different schemes, we discuss three types of environments.

• **Homogeneous:** In the homogeneous[9] setting the mean SNR value of each rectangle is fixed and it is uniformly selected in the range of $5-25$dB. Fig. 10.4(a) provides an example of the mean SNR values of such a venue as well as typical UE location distribution. In such instances, we assume random mobility pattern, in which each UE moves back and forth between two uniformly selected points. During the simulation, 50% of the UEs are always active, while the other 50% join and leave at some random time, as illustrated by Fig. 10.4(b). As we show later in such setting $s(t)$ barely change over time.

• **Stadiums:** In a stadium, the eMBMS service quality is typically significantly better inside the stadium than in the surrounding vicinity (e.g., the parking lots). To capture this, we simulate several stadium-like environments, in which the stadium, in the center of the venue, has high eMBMS SNR with mean values in the range of $15-25$dB. On the other hand, the vicinity has significantly lower SNR with means values of $5-10$dB. An example of a stadium is shown in Fig. 10.5(a).

We assume a mobility pattern in which, the UEs move from the edges to the inside of the stadium in 12mins, stay there for 3mins, and then go back to the edges.[10] As shown in Fig. 10.5(b), as the UEs move toward the center, the number of active UEs gradually increases from 10% of the UEs to 100%, and then declines again as they move away.

• **Failures:** Such an environment is similar to the homogeneous setting with a sudden event of a component failure. In the case of a malfunctioning component, the QoS in some parts of a venue can degrade significantly. To simulate failures, we consider cases in which the eMBMS SNR is high with a mean between $15-25$dB. During the simulation, (around the 10th minute), we mimic a failure by reducing the mean SNR values of some of the rectangles by over 10dB to the

[9] We use the term homogeneous since the term uniform is already used to denote the *Uniform* scheme.

[10] While significant effort has been dedicated to modeling mobility (e.g., [171, 176] and references therein), we use a *simplistic mobility model* since our focus is on the multicast aspects rather than the specific mobility patterns.

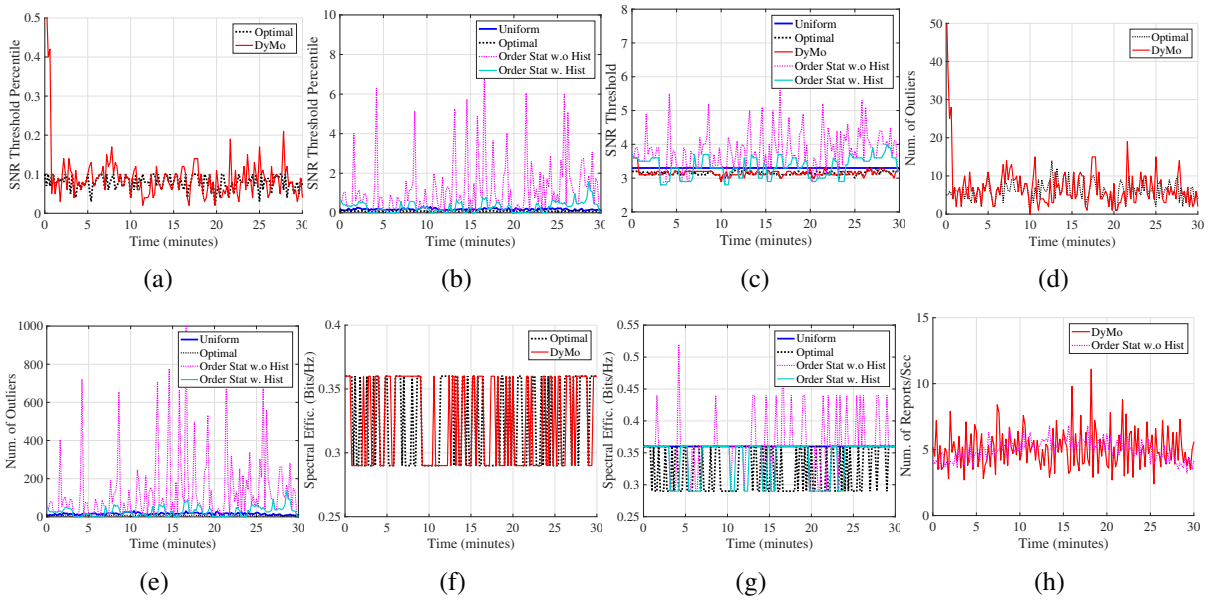


Figure 10.7: Simulation results from a single simulation instance lasting for 30mins in a component homogeneous environment with 20,000 UEs moving side to side between two random points, with $p = 0.1$ and $r = 5$ messages/sec. (a) The actual percentile of the SNR Threshold estimated by *DyMo*, (b) the actual percentile of the SNR Threshold estimated by *Order-Statistics*, (c) the SNR Threshold estimation, (d) spectral Efficiency of *Optimal* vs. *DyMo*, (e) spectral Efficiency of *Optimal* vs. *Order-Statistics*, (f) the number of Outliers by using *DyMo*, (g) the number of outliers by using *Uniform* and *Order-Statistics*, and (h) the QoS report overhead.

range of $5 - 10$dB. The mean SNR values are restored to their original values after a few minutes. Figs. 10.6(a) and 10.6(b) provide an example of the mean SNR values of such a venue before and after a failure, respectively. We assume the same mobility pattern like the homogeneous setting, as shown by Fig. 10.4(b).

10.6.3 Performance over time

We first illustrate the performance of the different schemes over time for three given instances, a homogeneous, a stadium and a failure scenarios, with $m = 20,000$ UEs, QoS Constraint $p = 0.1\%$, and Overhead constraint $r = 5$ reports/sec, i.e., 60 messages per reporting interval. The *number of permitted outliers depends on the number of active UEs at the current reporting interval. In the three considered scenarios, it can be at most* 20 *at any given time*. The key difference between the different instances is the rate at which the SNR Threshold changes. In the homogeneous environ-

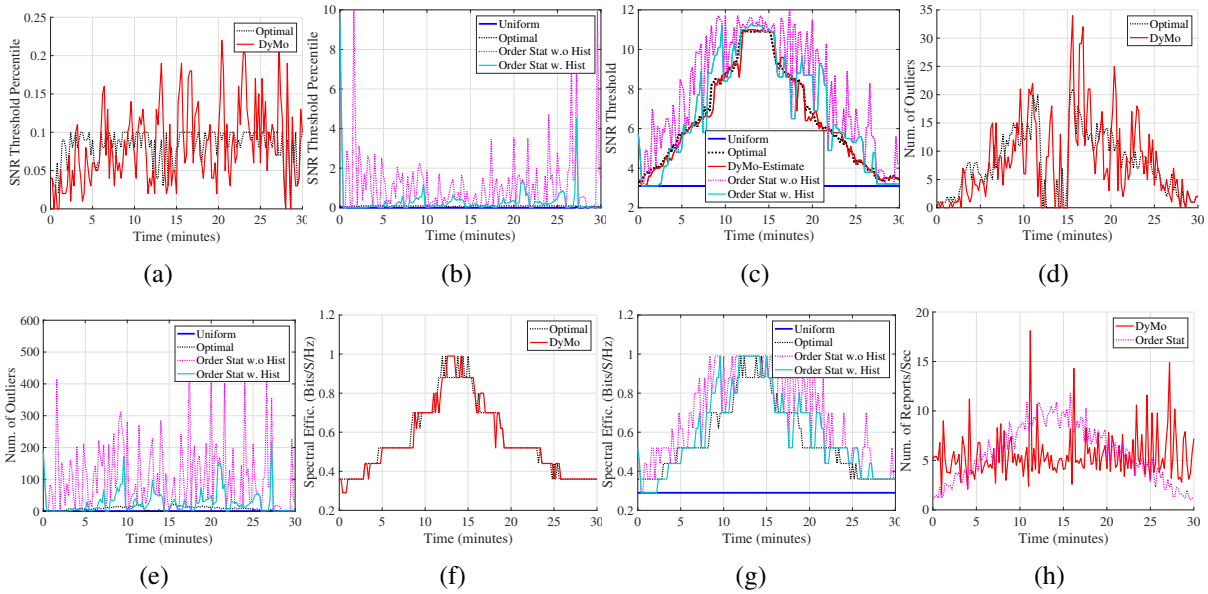


Figure 10.8: Simulation results from a single simulation instance lasting for 30mins in a stadium environment with 20,000 UEs moving from the edges to the center and back, with $p = 0.1$ and $r = 5$ messages/sec. (a) The actual percentile of the SNR Threshold estimated by *DyMo*, (b) the actual percentile of the SNR Threshold estimated by *Order-Statistics*, (c) the SNR Threshold estimation, (d) spectral efficiency of *Optimal* vs. *DyMo*, (e) spectral efficiency of *Optimal* vs. *Order-Statistics*, (f) the number of Outliers by using *DyMo*, (g) the number of Outliers by using *Uniform* and *Order-Statistics*, and (h) the QoS report overhead.

ment the SNR Threshold is almost fixed with very limited variability. In the case of the stadium, the SNR Threshold gradually changes as the UEs change their locations. In the failure scenario, the SNR Threshold is roughly fixed but it drops instantly by 10dBs for the duration of the failure.

The results of the homogeneous, stadium and failure cases are shown in Figs. 10.7, 10.8 and 10.9, respectively. Figs. 10.7(a), 10.7(b), 10.8(a), 10.8(b), 10.9(a), and 10.9(b) show the actual SNR Threshold percentile over time. From Figs. 10.7(a), 10.8(a) and 10.9(a), we observe that *DyMo* can accurately infer the SNR Threshold with an estimation error of at most 0.1%. Fig. 10.9(a) shows slightly higher error of 0.25% at the time of the failure (at the 7th minute). The *Order-Statistics* variants suffer from much higher estimation error to the order of a few percentage points, as shown by Figs. 10.8(b), 10.8(b) and 10.8(b)[11]. This performance gap results in different estimation accuracy of the SNR Threshold for *DyMo* and *Order-Statistics* schemes as illustrated in

[11]Notice that the pairs (i) Figs. 10.7(a) and 10.7(b), (ii) Figs. 10.8(a) and 10.8(b) as well as (iii) Figs. 10.9(a) and 10.9(b) use different scales for the Y axes.

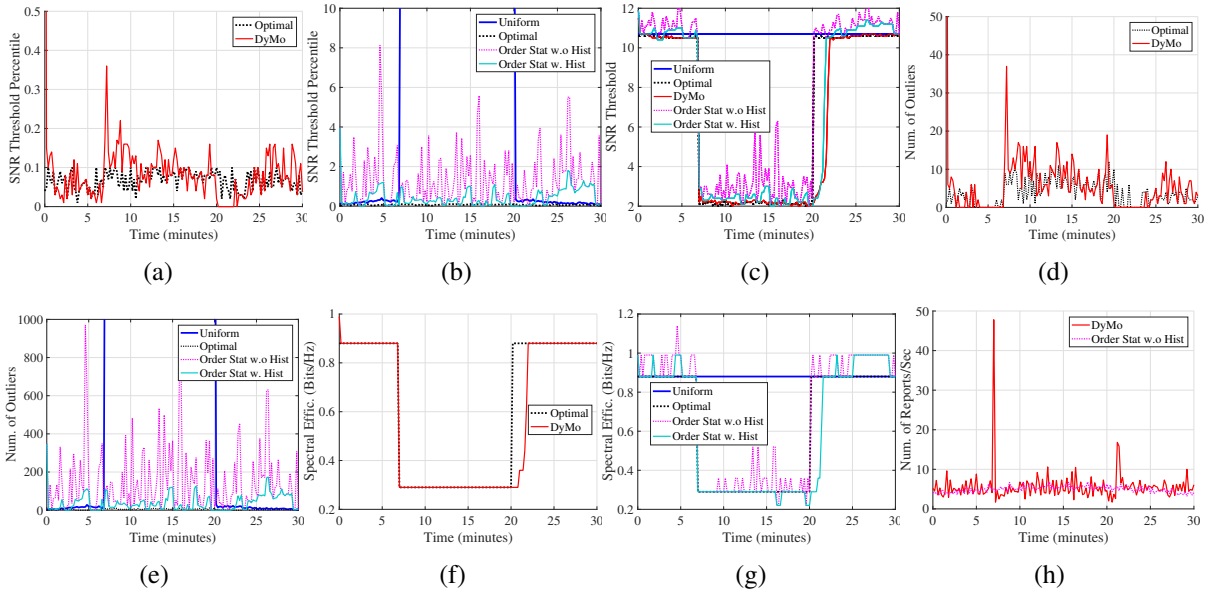


Figure 10.9: Simulation results from a single simulation instance lasting for 30mins in a component failure environment with 20,000 UEs moving side to side between two random points, with $p = 0.1$ and $r = 5$ messages/sec. (a) The actual percentile of the SNR Threshold estimated by *DyMo*, (b) the actual percentile of the SNR Threshold estimated by *Order-Statistics*, (c) the SNR Threshold estimation, (d) spectral Efficiency of *Optimal* vs. *DyMo*, (e) spectral Efficiency of *Optimal* vs. *Order-Statistics*, (f) the number of Outliers by using *DyMo*, (g) the number of outliers by using *Uniform* and *Order-Statistics*, and (h) the QoS report overhead.

Figs. 10.7(c), 10.8(c) and 10.9(c), respectively. These figures show that the performance of *DyMo* and *Optimal* is almost identical. Even in the event of a failure, *DyMo* reacts immediately and detects the SNR Threshold accurately. The *Order-Statistics* variants react quickly to a failure but not as accurately as *DyMo*. After the recovery, both *DyMo* and *Order-Statistics w. History* gradually increase their SNR Threshold estimates, due to the exponential smoothing process.

The SNR Threshold estimation gap directly impacts the number of outliers as well as the network utilization, i.e., the spectral efficiency. Figs. 10.7(d) and 10.7(e) show the number of outliers of *DyMo* and *Order-Statistics* variants for the homogeneous environment, respectively[12], while Figs. 10.7(f) and 10.7(g) show the spectral efficiency of the schemes. Figs. 10.7(d) and 10.7(f) reveal that after a short adaptation phase *DyMo* converges to the optimal performance, i.e., spectral efficiency, while preserving the QoS constraint. Fig. 10.7(f) show that both *Optimal* and *DyMo*

[12]Notice that the figure pairs, (i) Figs. 10.7(d) and 10.7(e), (ii) Figs. 10.8(d) and 10.8(e) as well as (iii) Figs. 10.9(d) and 10.9(e), use different scales for the Y axes.

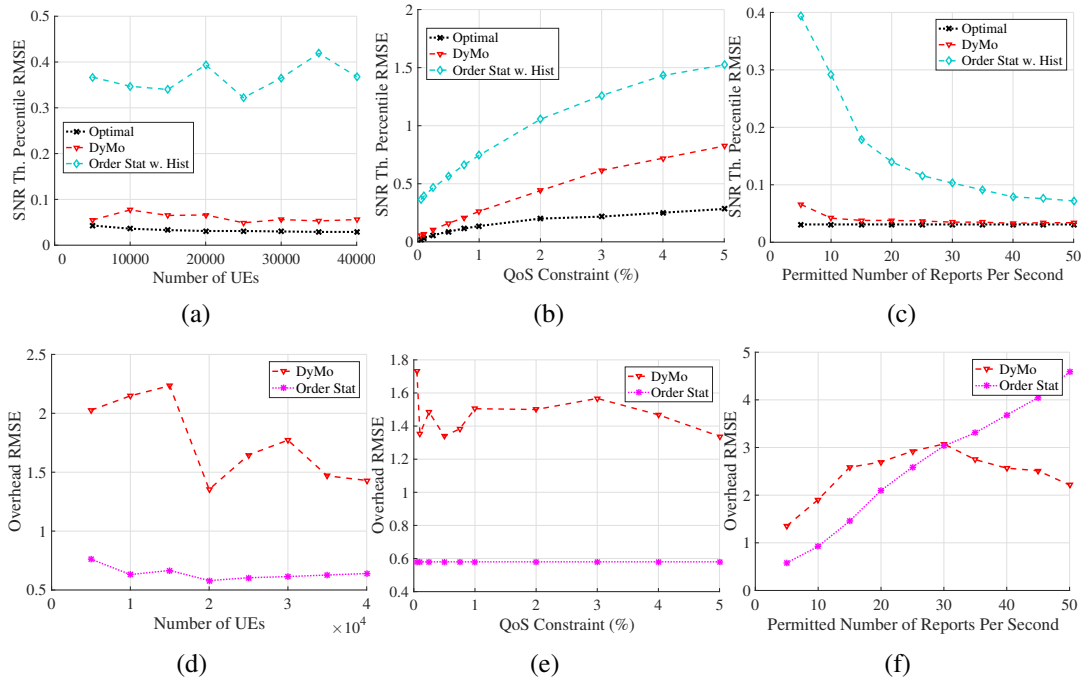


Figure 10.10: The Root Mean Square Error (RMSE) of different parameters averaged over 5 different simulation instances lasting for 30mins each in homogeneous scenario with different SNR characteristics and UE mobility patterns. (a) SNR Threshold percentile RMSE vs. the total number of UEs in the system, (b) SNR Threshold percentile RMSE vs. the QoS Constraint p, (c) SNR Threshold percentile RMSE vs. the number of permitted reports , (d) Overhead RMSE vs. the number of UEs, (e) Overhead RMSE vs. the QoS constraint p, and (f) Overhead RMSE vs. the number of permitted reports.

fluctuate between two spectral efficiency levels, 0.29 and 0.36 bit/sec/Hz, which results from oscillatation between two MCS levels 3 and 4. Such oscillations can be easily suppressed by enforcing some delay between MCS increase operations. The *Order-Statistics* variants over estimate the SNR threshold and suffer from higher number of outliers, as shown by Fig. 10.7(e). The homogeneous setting represents quasi-static environments with minor variation of the SNR threshold, $s(t)$. In such settings, the *Uniform* scheme provides a good estimation[13] of $s(t)$ and its number of outliers as well as the obtained spectral efficiency are comparable to *DyMo*. However, this is not the situation when $s(t)$ is time varying.

The number of outliers of *DyMo* and *Order-Statistics* variants for the stadium environment is shown in Figs. 10.8(d) and 10.8(e), respectively, while Figs. 10.9(d) and 10.9(e) illustrate the

[13] Assuming rigorous field trial measurements.

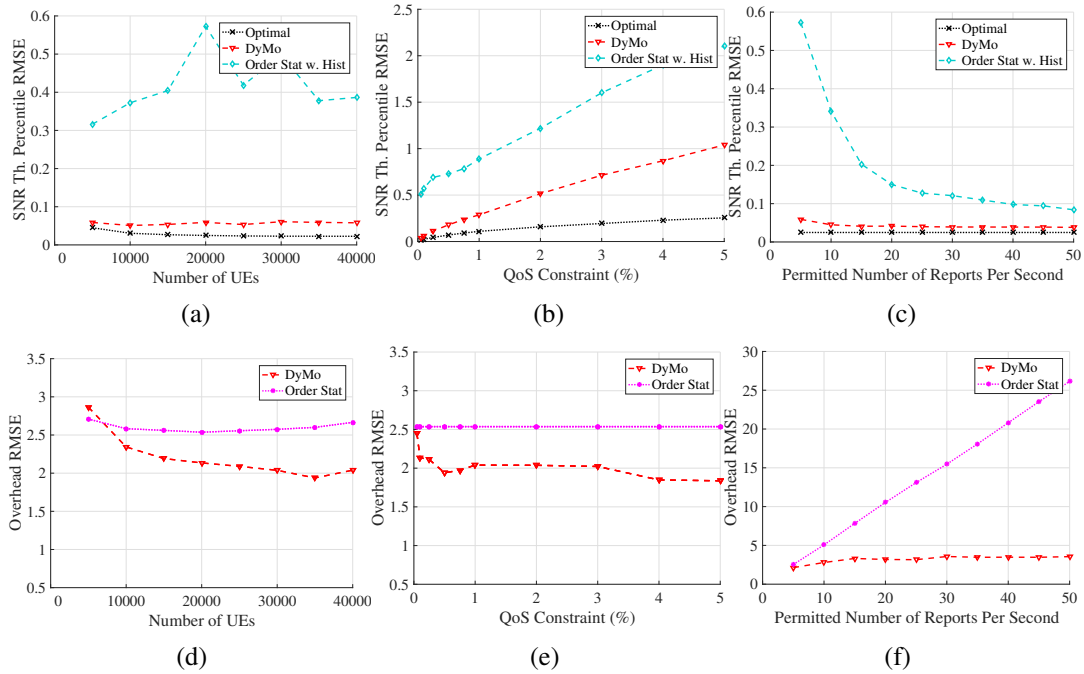


Figure 10.11: The Root Mean Square Error (RMSE) of different parameters averaged over 5 different simulation instances lasting for 30mins each in a stadium environment with different SNR characteristics and UE mobility patterns. (a) SNR Threshold percentile RMSE vs. the total number of UEs in the system, (b) SNR Threshold percentile RMSE vs. the QoS Constraint p, (c) SNR Threshold percentile RMSE vs. the number of permitted reports, (d) Overhead RMSE vs. the number of UEs, (e) Overhead RMSE vs. the QoS constraint p, and (f) Overhead RMSE vs. the number of permitted reports.

number of outliers of *DyMo* and *Order-Statistics* variants for the failure scenario, in this order. These figures show that the number of outliers that results from the *Order-Statistics w. History* and *Order-Statistics w.o. History* variants are occasionally over 200 and 800, respectively. Whereas, *DyMo* ensures that the number of outliers at any time is comparable to *Optimal* and in the worst case it exceeds the permitted number by less than a factor of 2.

Figs. 10.8(f) and 10.8(g) show the spectral efficiency for the stadium environment, whereas Figs. 10.9(f) and 10.9(g) show the spectral efficiency for the component failure case. The spectral efficiency for each case is correlated to the SNR Threshold. For the stadium environment, *DyMo* has spectral efficiency close to *Optimal* while *Uniform* has the lowest spectral efficiency. In the event of a failure, the spectral efficiency of *DyMo* follows the *Optimal* as expected from the SNR Threshold estimations. Since *Order-Statistics* variants typically over estimate the SNR Thresh-

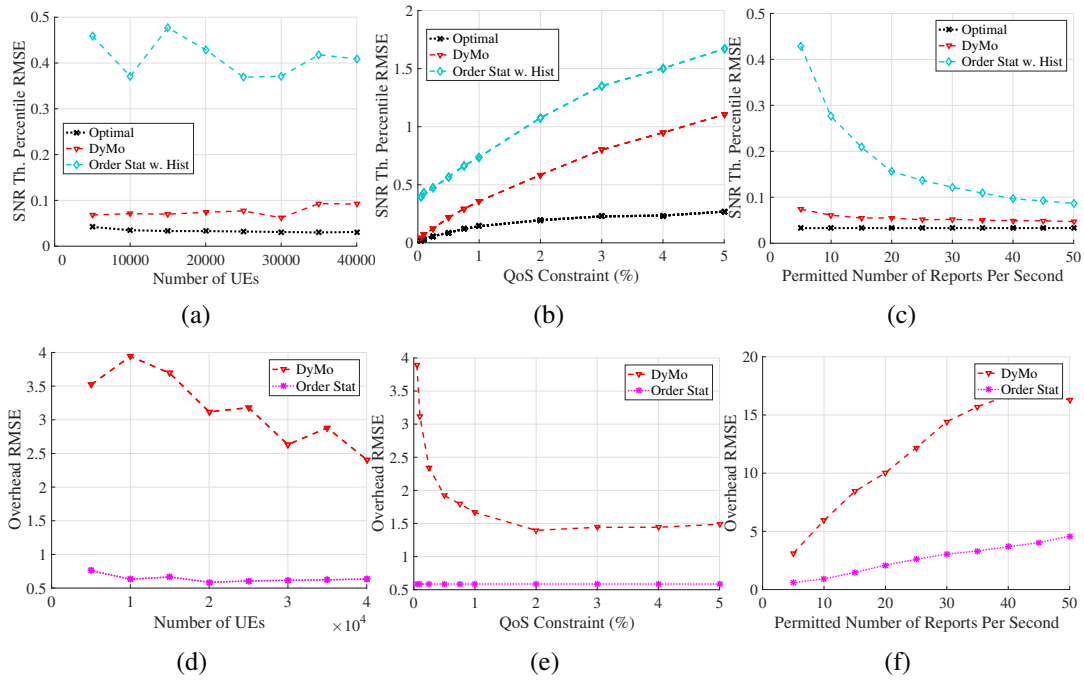


Figure 10.12: The Root Mean Square Error (RMSE) of different parameters averaged over 5 different simulation instances lasting for 30mins each in failure scenario with different SNR characteristics and UE mobility patterns. (a) SNR Threshold percentile RMSE vs. the total number of UEs in the system, (b) SNR Threshold percentile RMSE vs. the QoS Constraint p, (c) SNR Threshold percentile RMSE vs. the number of permitted reports , (d) Overhead RMSE vs. the number of UEs, (e) Overhead RMSE vs. the QoS constraint p, and (f) Overhead RMSE vs. the number of permitted reports.

old, they frequently determine MCS and consequently spectral efficiency that exceed the optimal settings. Such inaccuracy leads to a high number of outliers.

Figs. 10.7(h), 10.8(h) and 10.9(h) indicate only mild violation of the Overhead Constraint by both the *DyMo* and *Order-Statistics* variants. We observe that accurate SNR Threshold estimation allows *DyMo* to achieve near optimal spectral efficiency with negligible violation of the QoS Constraint. The other schemes suffer from sub-optimal spectral efficiency, excessive number of outliers, or both. Given that the permitted number of outliers is at most 20, the *Order-Statistics w. History* and *Order-Statistics w.o. History* schemes exceed this value sometimes by a factor of 10 and 40, respectively. Among these two alternatives, *Order-Statistics w. History* leads to lower number of outliers. While *Uniform* provides accurate estimation of $s(t)$ for the homogeneous environment, we observe that it yields a very conservative eMBMS MCS setting in the stadium

example, which causes low network utilization. In the failure scenario, the conservative eMBMS MCS of *Uniform* is not sufficient to cope with the low SNR Threshold and it leads to excessive number of outliers.

10.6.4 Impact of Various Parameters

We now turn to evaluate the quality of the SNR Threshold estimation and the schemes' ability to preserve the QoS and Overhead Constraints under various settings. We use the same configuration of $m = 20,000$ UEs, $p = 0.1\%$ and $r = 5$ reports/sec and we evaluate the impact of changing the values of one of the parameters. The results for the homogeneous, stadium and failure scenarios are shown in Figs. 10.10, 10.11 and 10.12, respectively. Each point in the figures is the average of 5 different simulation instances of 30mins each with different SNR characteristics and UE mobility patterns. The error bars are small and not shown. In these examples, we compare *DyMo* only with *Optimal* and *Order-Statistics w. History* which is the best performing alternative. We omit the *Uniform* scheme since it does not adapt to variation of $s(t)$.

First, we consider the impact of changing these parameters on the accuracy of the SNR Threshold estimation. Figs. 10.10(a), 10.11(a), and 10.12(a) show the Root Mean Square Error (RMSE) in SNR Threshold percentile estimation vs. m, for homogeneous, stadium and failure scenarios, respectively. The non-zero values of RMSE in *Optimal* are due to quantization of SNR reports. The RMSE in the SNR Threshold estimation of *DyMo* is close to that of *Optimal* regardless of the number of UEs, while *Order-Statistics w. History* suffers from order of magnitude higher RMSE.

Figs. 10.10(b), 10.11(b), and 10.12(b) show the RMSE in SNR Threshold estimation as the QoS Constraint p changes, for homogeneous, stadium and failure scenarios. *DyMo* outperforms the alternative schemes as p increases. As p increases, we observe an increasing quantization error, which impacts the RMSE of all the schemes including the *Optimal*. Recall that the SNR distribution is represented by a histogram where each bar has a width of $0.1dB$. As p increases, the number UEs in the bar that contains the p percentile UE increases as well. Since $s(t)$ should be below the SNR value of this bar, we notice a higher quantization error.

Figs. 10.10(c), 10.11(c), and 10.12(c) illustrate the SNR Threshold percentile RMSE as the Overhead Constraint is relaxed, for homogeneous, stadium and failure cases, respectively. The SNR Threshold percentile RMSE of *DyMo* is 0.05% even with Overhead Constraint of 5 reports/sec, while *Optimal* RMSE due to quantization is 0.025%. *DyMo* error slightly reduces by relaxing the Overhead Constraint (*Optimal* error stays 0.25%). Even with 10 times higher reporting rate, *DyMo* significantly outperforms the *Order-Statistics* alternatives. The RMSE in SNR Threshold percentile for *Order-Statistics* is in the order of the required average value of 0.1 even with a permitted overhead of 50 reports/sec, i.e,. 3000 reports per reporting interval. This is a very high overhead on the unicast traffic, since in LTE networks the number of simultaneously open unicast connections is limited, i.e., several hundreds per base station and each connection lasts several hundred msecs even for sending a short update. Unlike the downlink, uplink resources are not reserved for eMBMS systems and utilize the unicast resources. The RMSE of number of outliers is qualitatively similar to the SNR Threshold percentile results.

We also compute the overhead RMSE for different UE population sizes, m, QoS Constraint p, and Overhead Constraints r. The results are shown is sub-figures (d), (e) and (f) of Figs. 10.10, 10.11 and 10.12, respectively. In most cases, the overhead RMSE of *DyMo* is between $1-4$ reports even when the system parameters change. We observe an increase in the overhead RMSE only in failure scenarios when the permitted overhead is relaxed, as shown in Fig. 10.12(f). This is expected immediately after a failure because many more UEs suffer from poor service than *DyMo* estimated. Thus, as the permitted overhead increases also the spike in the number of reports during the first reporting interval after the failure also increases, which results in a gradual increase of the Overhead RMSE[14].

Figs. 10.10 and 10.12 show that the *Order-Statistics* variants experience very low violation of the Overhead Constraint in the homogeneous and Failure scenarios. This is not surprising, since in these scenarios the variation in the number of active eMBMS receivers is very small and this number is roughly $\mathbb{E}[m(t)]$ (the expected number of active eMBMS receivers). As mentioned in

[14]Notice that the RMSE metric is sensitive to sporadic but very high error.

Section 10.6.1, this observation is misleading, since we assume that $\mathbb{E}[m(t)]$ is known and we ignore the overhead of configuring the UEs with the proper reporting rate. Obviously, *the exact number of active receivers,* $\mathbb{E}[m(t)]$*, is unknown in practice.* Furthermore, Fig. 10.11(f) shows that in scenarios with high variation in the number of active receivers, $m(t)$, (like the case in the stadium simulations) the violation of the Overhead Constraint is high and it is amplified as the permitted number of reports, r, increases. This is due to the static reporting rate of *Order-Statistics* despite dynamic changes of the number of active eMBMS receivers. Fig. 10.11(f) confirms that the overhead violation of *Order-Statistics* is very sensitive to the estimation of $\mathbb{E}[m(t)]$ and its variance.

Given that the number of active eMBMS receivers, $m(t)$*, is unknown and may change significantly over time, Order-Statistics cannot practically preserve the Overhead Constraint without keeping track of the active UEs and sending individual messages to a subset of the active UEs.* However, keeping track of $m(t)$ requires each UE to report when it starts and stops receiving eMBMS services, which may incur much higher overhead than permitted. For instance, in our simulations with $m = 20,000$ UEs, even if such switching occurs at most once (start and stop) by each UE, the total number of reports is $40,000$. When dividing this number by the simulation duration of 30 minutes ($1,800$ sec) we get 22 messages/second, which is much higher than the permitted overhead.

Summary: Our simulations show that *DyMo* achieves accurate, close to optimal, estimation of the SNR Threshold even when the number of active eMBMS receivers is unknown. It can improve the spectral efficiency for eMBMS operation, while adding a very low reporting overhead. *DyMo* can predict the SNR Threshold with lower errors than other alternatives under a wide range of the SNR Threshold requirement p and reporting Overhead Constraint r. These observations show that *DyMo* exceeds the expectations of our analysis in Section 10.5.

10..5 Analysis of the Two-Step Estimation Algorithm

We now extend the analysis of the *Two-step estimation* algorithm given in Section 10.5.2. We show that the optimal settings for minimizing the error ϵ of Equation (10.3) is obtained by taking

$$p_1 = p_2 = \sqrt{p} \quad and \quad r_1 = r_2 = r/2$$

Notice that the settings should satisfy the following two constraints:

$$p = p_1 \cdot p_2 \tag{10.7}$$

and

$$r = r_1 + r_2 \tag{10.8}$$

From Equation (10.2) and by taking 3 times the standard deviation, we get that the errors ϵ_1 and ϵ_2 are

$$\epsilon_1 = 3\sqrt{\frac{p_1(1-p_1)}{r_1}} \quad and \quad \epsilon_2 = 3\sqrt{\frac{p_2(1-p_2)}{r_2}}$$

By combining with Equation (10.3), we get

$$\epsilon = 3\sqrt{\frac{p_1(1-p_1)}{r_1}}\, p_2 + 3\sqrt{\frac{p_2(1-p_2)}{r_2}}\, p_1 \tag{10.9}$$

By using the two constraints (10.7) and (10.8), we assign $p_2 = p/p_1$ and $r_2 = r - r_1$. Consequently,

$$\begin{aligned} \epsilon &= 3\sqrt{\frac{p_1(1-p_1)}{r_1}}\,\frac{p}{p_1} + 3\sqrt{\frac{(p/p_1)(1-p/p_1)}{r-r_1}}\, p_1 \\ &= 3\,p\left[\sqrt{\left(\frac{1}{p_1}-1\right)\frac{1}{r_1}} + \sqrt{\left(\frac{p_1}{p}-1\right)\frac{1}{r-r_1}}\right] \end{aligned} \tag{10.10}$$

By taking the partial derivative $\frac{\partial \epsilon}{\partial p_1}$ we get,

$$\frac{\partial \epsilon}{\partial p_1} = 3\,p \left[-\left(2\,p_1^2 \sqrt{\left(\frac{1}{p_1} - 1\right)\,\frac{1}{r_1}} \right)^{-1} + \left(2\,p \sqrt{\left(\frac{p_1}{p} - 1\right)\,\frac{1}{r - r_1}} \right)^{-1} \right] \tag{10.11}$$

For minimizing the error we calculate $\frac{\partial \epsilon}{\partial p_1} = 0$ and get that

$$p_1^2 \sqrt{\left(\frac{1}{p_1} - 1\right)\frac{1}{r_1}} = p \sqrt{\left(\frac{p_1}{p} - 1\right)\,\frac{1}{r - r_1}} \tag{10.12}$$

By simple mathematical manipulations we get

$$p_1^4 \left(\frac{1}{p_1} - 1\right)(r - r_1) = p^2 \left(\frac{p_1}{p} - 1\right)\,r_1 \tag{10.13}$$

Similarly, from the partial derivative $\frac{\partial \epsilon}{\partial r_1}$ we get

$$\frac{\partial \epsilon}{\partial r_1} = 3\,p \left[\sqrt{\left(\frac{1}{p_1} - 1\right)}\,\frac{-1}{2\,r_1^{3/2}} + \sqrt{\left(\frac{p_1}{p} - 1\right)}\,\frac{1}{2\,(r - r_1)^{3/2}} \right] \tag{10.14}$$

For minimizing the error we calculate $\frac{\partial \epsilon}{\partial r_1} = 0$ and get that

$$\sqrt{\left(\frac{1}{p_1} - 1\right)}\,\frac{1}{2\,r_1^{3/2}} = \sqrt{\left(\frac{p_1}{p} - 1\right)}\,\frac{1}{2\,(r - r_1)^{3/2}} \tag{10.15}$$

By simple mathematical manipulations we get

$$\left(\frac{1}{p_1} - 1\right)\,(r - r_1)^3 = \left(\frac{p_1}{p} - 1\right)\,r_1^3 \tag{10.16}$$

Noticed that Equations (10.13) and (10.16) together provide two simple conditions to optimize p_1 and r_1. By dividing Equation (10.13) by Equation (10.16) we obtain,

$$(r - r_1) = \frac{r_1\, p_1^2}{p} \tag{10.17}$$

Using Equation (10.17) in Equation (10.16) results that

$$\begin{aligned} \left(\frac{1}{p_1} - 1\right) \left(\frac{r_1\, p_1^2}{p}\right)^3 &= \left(\frac{p_1}{p} - 1\right) r_1^3 \\ \left(\frac{1}{p_1} - 1\right) \frac{p_1^6}{p^2} &= \left(\frac{p_1}{p} - 1\right) \end{aligned} \tag{10.18}$$

From this we get the following relation

$$p_1^6 - p_1^5 + p_1\, p^2 - p^3 = 0 \tag{10.19}$$

The only real solutions are $p_1 = \pm\sqrt{p}$. Since p_1 must be positive we get that $p = \sqrt{p}$. From this solution and Equation (10.17), it is implies that the optimal setting is

$$p_1 = p_2 = \sqrt{p}, \;\; and \;\; r_1 = r_2 = r/2$$

Consequently, the errors of the two steps are

$$\epsilon_1 = \epsilon_2 = 3\sqrt{\sqrt{p}(1 - \sqrt{p})/(r/2)}$$

From this we obtain Proposition 2 and a bound on the error of,

$$6\sqrt{2}\sqrt{\frac{p\sqrt{p}(1 - \sqrt{p})}{r}}$$

This concludes our analysis of the *Two-step estimation* algorithm.

Part V

Conclusions

This thesis presented ML-based network architectures and data driven network algorithms whose objective is to improve the performance and management of future networks. Below we highlight general conclusions and possible future directions.

Video Streaming

In Part I we focused on developing systems for QoE metric detection for encrypted traffic. In Chapter 2, we presented a system, *Requet*, for Real-time Quality of experience metric detection for Encrypted traffic. *Requet* consists of the ChunkDetection algorithm, chunk feature extraction, and ML QoE prediction models. Our evaluation using YouTube traffic collected over WiFi networks demonstrates that *Requet*, which uses chunk-based features, exhibits significantly improved prediction power over the baseline system, that uses IP-layer features.

In Chapter 3, we extended the work from Chapter 2 and presented a study on YouTube TV live streaming traffic behavior over WiFi and cellular networks. We presented our study which spanned a 9-month period. Using the collected data, we developed a multi-chunk detection algorithm to detect multiple video and audio chunks with concurrent transmission in the same IP flow.

A current limitation of *Requet* is that it is based on specific services and needs to be trained separately for each streaming algorithm. Therefore, one direction of future work includes building a generic model for a wide range of networks and client algorithms for ABR. Another direction of future work includes using SDN to utilize *Requet* and investigate the QoE improvements achieved via resource scheduling.

End-to-End Resource Allocation in Cellular Networks

In Part II we focused on using deep neural networks to improve resource allocation in cellular networks. In Chapter 4 we presented a new metric, *REVA*, that precisely measures the amount of PRBs that the RAN scheduler can allocate to VA bearers. An experimental LTE testbed was developed to collect *REVA* for time series analysis. In addition, we proposed an evaluated a new time series prediction model, X-LSTM, for *REVA*. We showed that X-LSTM provides a higher

degree of accuracy over other time series models such as ARIMA and LSTM.

In Chapter 5 we presented an LSTM neural network used for BBU pool resource reallocations in a 5G C-RAN network using a ROADM switched optical network. Through simulations our analysis shows that using the predicted traffic pattern to reconfigure the ROADM network can improve network throughput and reduce the amount of BBU processing resources required.

Dynamic Optical Systems

In Part III we focused on using ML to ensure stable performacne and reliable QoT for dynamic optical operation. In Chapter 6 we presented a *feedforward deep neural network* based-ML model to predict the dynamic power excursions of a 90-channel ROADM system. Based on the predicted power excursions, the deep neural network can recommend valid wavelengths for wavelength switching with a precision over 99% over the tested samples. The deep neural network was also shown to be far more effective than regression and random forest models.

In Chapter 7 we extended the work from Chapter 6 and examine an ML model, an analytical model, and a hybrid ML model. The hybrid ML model takes advantage of an analytical model as input to the ML model. Based on experimental measurements, the hybrid ML model is shown to increase prediction accuracy of the output optical power spectrum of an EDFA.

For future work we will investigate the deep neural network approach in large-scale networks along with transfer and online learning techniques for practical implementations. We plan on using the optical-wireless infrastructure of the COSMOS testbed to evaluate the ML models from Part III of this thesis, along with extending the works in [136, 236, 235, 167].

Adaptive Multicast Services

In Part IV we focused on the use of data-driven solutions for large scale content delivery via wireless multicast, both for WiFi and cellular networks.

In Chapters 8 and 9 we addressed challenges related to feedback and rate adaptation as part of the Adaptive Multicast Services (*AMuSe*) system [22] for WiFi multicast. In Chapter 8 we studied

approaches for light-weight feedback for WiFi multicast. We presented the design and large-scale experimental evaluation of the AMuSe system for providing scalable and efficient multicast services for a large group of users in a small geographical region. In Chapter 9, we presented the design and evaluation of the Multicast Dynamic Rate Adaptation (*MuDRA*) algorithm for WiFi. *MuDRA* balances fast adaptation to channel conditions and stability, which is essential for multimedia applications. Our experimental evaluation of *MuDRA* on the ORBIT testbed with over 150 nodes shows that *MuDRA* outperforms other schemes and supports high throughput multicast flows to hundreds of receivers while meeting quality requirements.

In Chapter 10, we described the Dynamic Monitoring (*DyMo*) system designed to support efficient LTE-eMBMS, based on the concept of Stochastic Group Instructions. Our extensive simulations show that DyMo achieves accurate, close to optimal, estimation of the SNR Threshold even when the number of active UEs is unknown. It can improve the spectral efficiency for eMBMS operation while adding a low reporting overhead.

In summary, this thesis contributed to the development ML and data driven network algorithms spanning across the networking protocol stack from the application layer to the physical layer. In each of the networking domains we explored the networking sub-problem and presented ML algorithms and architectures that were motivated by measurements and observations in the real world or from experimental testbeds. While we are able to show promising results, there remain a variety of open challenges to advance ML for networking. Finally, to make a fully autonomous network that is practically deployable there is a still a need to improve the accuracy, scalability, and integration of ML and AI network solutions.

References

[1] 3GPP, "Transparent end-to-end Packet-switched Streaming Service (PSS)", 3rd Generation Partnership Project (3GPP), TS 26.234, Jun. 2010.

[2] ——, "Evolved Universal Terrestrial Radio Access Physical channels and modulation", 3rd Generation Partnership Project (3GPP), TS 36.211, Jan. 2011.

[3] ——, "Policy and charging control architecture (Release 15)", 3rd Generation Partnership Project (3GPP), TS 23.203, 2018.

[4] *3GPP TS 26.346 V13.1.0, 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Multimedia Broadcast/Multicast Service (MBMS); Protocols and codecs (Release 13)*, 2015. [Online]. Available: `http://www.3gpp.org/DynaReport/26346.htm`.

[5] *3GPP TS 37.320 V12.2.0 , 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Universal Terrestrial Radio Access (UTRA) and Evolved Universal Terrestrial Radio Access (E-UTRA); Radio measurement collection for Minimization of Drive Tests (MDT); Overall description; Stage 2 (Release 12)*, 2014. [Online]. Available: `http://www.3gpp.org/DynaReport/37320.htm`.

[6] 5G Infrastructure Association and others, "Deliverable d3.2 5G NORMA network architecture intermediate report", *January*, 2017.

[7] J. Ababneh and O. Almomani, "Survey of error correction mechanisms for video streaming over the internet", in *IJACSA*, vol. 5, 2014, pp. 155–161.

[8] *About wireshark*, `https://www.wireshark.org/about.html`.

[9] H. Abu-Ghazaleh and A. S. Alfa, "Application of mobility prediction in wireless networks using markov renewal theory", *IEEE Transactions on Vehicular Technology*, vol. 59, no. 2, pp. 788–802, 2010.

[10] V. Aggarwal, E. Halepovic, J. Pang, S. Venkataraman, and H. Yan, "Prometheus: Toward quality-of-experience estimation for mobile apps from passive network measurements", in *Proc. ACM HotMobile*, Feb. 2014.

[11] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network", *Proc. ACM SIGCOMM*, 2004.

[12] A. Ahmed, Z. Shafiq, H. Bedi, and A. R. Khakpour, "Suffering from buffering? detecting QoE impairments in live video streams", in *Proc. IEEE ICNP*, 2017.

[13] O. Alay, T. Korakis, Y. Wang, and S. Panwar, "Dynamic rate and FEC adaptation for video multicast in multi-rate wireless networks", *ACM/Springer Mobile Netw. and Appl.*, vol. 15, no. 3, pp. 425–434, 2010.

[14] J. Amann, O. Gasser, Q. Scheitle, L. Brent, G. Carle, and R. Holz, "Mission accomplished?: HTTPS security after diginotar", in *Proc. ACM IMC*, Nov. 2017.

[15] *AngularJS*. [Online]. Available: `https://angularjs.org/`.

[16] L. Armasu, *Netflix adopts efficient HTTPS encryption for its video streams*, `https://www.tomshardware.com/news/netflix-efficient-https-video-streams,32420.html`, Aug. 2016.

[17] A. Balachandran, V. Sekar, A. Akella, S. Seshan, I. Stoica, and H. Zhang, "A quest for an internet video quality-of-experience metric", in *Proc. ACM HotNets*, 2012.

[18] L. Barletta, A. Giusti, C. Rottondi, and M. Tornatore, "QoT estimation for unestablished lightpaths using machine learning", in *Proc. OSA OFC*, 2017, Th1J–1.

[19] A. Basalamah, H. Sugimoto, and T. Sato, "Rate adaptive reliable multicast MAC protocol for WLANs", in *Proc. IEEE VTC*, 2006.

[20] Y. Bejerano, J. Ferragut, K. Guo, V. Gupta, C. Gutterman, T. Nandagopal, and G. Zussman, "Scalable WiFi Multicast Services for Very Large Groups", in *Proc. IEEE ICNP*, 2013.

[21] ——, "Experimental evaluation of a scalable WiFi multicast scheme in the ORBIT testbed", in *2014 Third GENI Research and Educational Experiment Workshop*, IEEE, 2014, pp. 36–42.

[22] Y. Bejerano, V. Gupta, C. Gutterman, and G. Zussman, "AMuSe: Adaptive multicast services to very large groups-project overview", in *Proc. IEEE ICCCN*, 2016.

[23] Y. Bejerano, C. Raman, C.-N. Yu, V. Gupta, C. Gutterman, T. Young, H. A. Infante, Y. M. Abdelmalek, and G. Zussman, "DyMo: Dynamic monitoring of large-scale LTE-multicast systems", *IEEE/ACM Transactions on Networking*, vol. 27, no. 1, pp. 258–271, 2019.

[24] Y. Bejerano, C. Raman, C.-N. Yu, V. Gupta, C. Gutterman, T. Young, H. Infante, Y. Abdelmalek, and G. Zussman, "DyMo: Dynamic monitoring of large scale LTE-multicast systems", in *Proc. IEEE INFOCOM*, 2017.

[25] M. Belshe, R. Peon, and M. Thomson, *Hypertext transfer protocol version 2 (http/2)*, 2015.

[26] J. Bicket, “Bit-rate selection in wireless networks”, *PhD thesis, MIT*, 2005.

[27] G. E. Box, G. M. Jenkins, G. C. Reinsel, and G. M. Ljung, *Time Series analysis: forecasting and control*. John Wiley & Sons, 2015.

[28] F. Bronzino, P. Schmitt, S. Ayoubi, G. Martins, R. Teixeira, and N. Feamster, “Inferring streaming video quality from encrypted traffic: Practical models and deployment experience”, in *Proc. ACM SIGMETRICS*, 2020.

[29] N. Brown and T. Sandholm, “Superhuman AI for heads-up no-limit poker: Libratus beats top professionals”, *Science*, vol. 359, no. 6374, pp. 418–424, 2018.

[30] *Cable live video usage will increase 15-fold by 2021, Cisco predicts*, `https://www.fiercevideo.com/cable/live-video-usage-will-increase-15-fold-by-2021-cisco-predicts`.

[31] Y. Cai, S. Lu, L. Zhang, C. Wang, P. Skov, Z. He, and K. Niu, “Reduced feedback schemes for LTE MBMS”, in *Proc. IEEE VTC*, 2009.

[32] P. Casas, M. Seufert, and R. Schatz, “YOUQMON: a system for on-line monitoring of YouTube QoE in operational 3G networks”, *SIGMETRICS Performance Evaluation Review*, vol. 41, no. 2, pp. 44–46, 2013.

[33] R. Chandra, S. Karanth, T. Moscibroda, V. Navda, J. Padhye, R. Ramjee, and L. Ravindranath, “DirCast: a practical and efficient Wi-Fi multicast system”, in *Proc. IEEE ICNP*, 2009.

[34] C.-Y. Chang, N. Nikaein, and T. Spyropoulos, “Radio access network resource slicing for flexible service execution”, in *Proc. IEEE INFOCOM Workshop on RS-FCN*, 2018.

[35] J. Chen, M. Chiang, J. Erman, G. Li, K. Ramakrishnan, and R. K. Sinha, “Fair and optimal resource allocation for LTE multicast (eMBMS): Group partitioning and dynamics”, in *Proc. IEEE INFOCOM*, 2015.

[36] H.-T. Chiao, S.-Y. Chang, K.-M. Li, Y.-T. Kuo, and M.-C. Tseng, “WiFi multicast streaming using AL-FEC inside the trains of high-speed rails”, in *Proc. IEEE BMSB*, 2012.

[37] K. Chintalapudi, A. Iyer, and V. Padmanabhan, “Indoor localization without the pain”, in *Proc. ACM MobiCom*, 2010.

[38] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using RNN encoder-decoder for statistical machine translation”, *arXiv preprint arXiv:1406.1078*, 2014.

[39] N. Choi, Y. Seok, T. Kwon, and Y. Choi, "Leader-based multicast service in IEEE 802.11v networks", in *Proc. IEEE CCNC*, 2010.

[40] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling", *arXiv preprint arXiv:1412.3555*, 2014.

[41] *Cisco visual networking index: Forecast and methodology, 2016–2021*, `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html`.

[42] *Cisco, white-paper, Cisco connected stadium Wi-Fi solution*, 2011. [Online]. Available: `http://www.cisco.com/web/strategy/docs/sports/c78-675064_svcs.pdf`.

[43] *Cloud enhanced open software defined mobile wireless testbed for city-scale deployment (cosmos)*, `https://cosmos-lab.org/`, 2020.

[44] G. Cofano, L. De Cicco, T. Zinner, A. Nguyen-Ngoc, P. Tran-Gia, and S. Mascolo, "Design and experimental evaluation of network-assisted strategies for HTTP adaptive streaming", in *Proc. ACM MMSys*, 2016.

[45] R. Combes, A. Proutiere, D. Yun, J. Ok, and Y. Yi, "Optimal rate sampling in 802.11 systems", in *Proc. IEEE INFOCOM*, 2014.

[46] C. Cortes and V. Vapnik, "Support-vector networks", *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[47] N. Cranley, P. Perry, and L. Murphy, "User perception of adapting video quality", *Int. J. Hum. Comput. St.*, vol. 64, no. 8, pp. 637–647, 2006.

[48] R. Crepaldi, J. Lee, R. Etkin, S.-J. Lee, and R. Kravets, "CSI-SF: Estimating wireless channel state using CSI sampling and fusion", in *Proc. IEEE INFOCOM*, 2012.

[49] Y. Cui, T. Li, C. Liu, X. Wang, and M. Kühlewind, "Innovating transport with QUIC: design approaches and research challenges", *IEEE Internet Computing*, vol. 21, no. 2, pp. 72–76, 2017.

[50] A. D'amico, S. Straullu, A. Nespola, I. Khan, S. Abdelfattah, E. Virgillito, S. Piciaccia, A. Tanzi, G. Galimberti, S. Abrate, and V. Curri, "Machine-learning aided OSNR prediction in optical line systems", in *Proc. ECOC*, 2019.

[51] "Description of network slicing concept", NGMN 5G Alliance, Tech. Rep. 1, 2016.

[52] G. Dimopoulos, I. Leontiadis, P. Barlet-Ros, and K. Papagiannaki, "Measuring video QoE from encrypted traffic", in *Proc. ACM IMC*, 2016.

[53] *Django project*. [Online]. Available: `https://www.djangoproject.com/`.

[54] F. Dobrian, V. Sekar, A. Awan, I. Stoica, D. Joseph, A. Ganjam, J. Zhan, and H. Zhang, "Understanding the impact of video quality on user engagement", in *Proc. ACM SIGCOMM*, 2011.

[55] R. D. Doverspike and J. Yates, "Optical network management and control", *Proc. IEEE*, vol. 100, no. 5, pp. 1092–1104, 2012.

[56] Z. Durumeric, Z. Ma, D. Springall, R. Barnes, N. Sullivan, E. Bursztein, M. Bailey, J. A. Halderman, and V. Paxson, "The security impact of HTTPS interception", in *Proc. NDSS*, Feb. 2017.

[57] *Encrypted traffic analytics, Cisco white paper*, `https://www.cisco.com/c/dam/en/us/solutions/collateral/enterprise-networks/enterprise-network-security/nb-09-encrytd-traf-anlytcs-wp-cte-en.pdf`, 2019.

[58] J. Erman and K. K. Ramakrishnan, "Understanding the super-sized traffic of the super bowl.", in *Proc. ACM IMC*, 2013.

[59] Y. Fei, A. Fumagalli, M. Garrich, B. Sarti, U. Moura, N. G. González, and J. Oliveira, "Estimating EDFA output power with an efficient numerical modeling framework", in *Proc. IEEE ICC*, 2015.

[60] Z. Feng, G. Wen, C. Yin, and H. Liu, "Video stream groupcast optimization in WLAN", in *Proc. IEEE ITA*, 2010.

[61] R. T. Fielding and J. F. Reschke, "Hypertext transfer protocol (HTTP/1.1): message syntax and routing", *RFC*, vol. 7230, pp. 1–89, 2014.

[62] T. Fitzgerald, *Why cord cutting doubled in 2018 and 10 million have left since 2012*, `https://www.forbes.com/sites/tonifitzgerald/2019/03/06/why-cord-cutting-doubled-in-2018-and-10-million-have-left-since-2012/2607cf49409f`, 2019.

[63] X. Foukas, M. K. Marina, and K. Kontovasilis, "Orion: Ran slicing for a flexible and cost-effective multi-service mobile network architecture", in *Proc. ACM MobiCom*, 2017.

[64] S. Galetto, P. Bottaro, C. Carrara, F. Secco, A. Guidolin, E. Targa, C. Narduzzi, and G. Giorgi, "Detection of video/audio streaming packet flows for non-intrusive QoS/QoE monitoring", in *IEEE Int. Workshop on Measurement and Networking*, Sep. 2017.

[65] F. A. Gers, N. N. Schraudolph, and J. Schmidhuber, “Learning precise timing with LSTM recurrent networks”, *Journal of Machine Learning Research*, vol. 3, no. 8, pp. 115–143, 2002.

[66] X. Glorot, A. Bordes, and Y. Bengio, “Deep sparse rectifier neural networks”, in *Proc. of the fourteenth international conference on artificial intelligence and statistics*, 2011.

[67] S. Gringeri, N. Bitar, and T. J. Xia, “Extending software defined network principles to include optical transport”, *IEEE Commun. Mag.*, vol. 51, no. 3, pp. 32–40, 2013.

[68] T. A. Guarnieri, I. Drago, A. B. Vieira, Í. Cunha, and J. M. Almeida, “Characterizing QoE in large-scale live streaming”, in *Proc. IEEE GLOBECOM*, 2017.

[69] H. Gudmundsdottir, E. I. Ásgeirsson, M. H. L. Bodlaender, J. T. Foley, M. M. Halldórsson, and Y. Vigfusson, “Wireless scheduling algorithms in complex environments”, in *Proc. ACM MSWiM*, 2014.

[70] S. K. S. Gupta, V. Shankar, and S. Lalwani, “Reliable multicast MAC protocol for wireless LANs”, in *Proc. IEEE ICC*, 2003.

[71] V. Gupta, Y. Bejerano, C. Gutterman, J. Ferragut, K. Guo, T. Nandagopal, and G. Zussman, “Light-weight feedback mechanism for WiFi multicast to very large groups-experimental evaluation”, *IEEE/ACM TON*, vol. 24, no. 6, pp. 3826–3840, 2016.

[72] V. Gupta, C Gutterman, Y. Bejerano, and G. Zussman, “Experimental evaluation of large scale WiFi multicast rate control”, *IEEE Transactions on Wireless Communications*, vol. 17, no. 4, pp. 2319–2332, 2018.

[73] V. Gupta, C. Gutterman, Y. Bejerano, and G. Zussman, “Dynamic rate adaptation for WiFi multicast to very large groups – design and experimental evaluation”, in *Proc. IEEE INFOCOM*, 2016.

[74] V. Gupta, R. Norwitz, S. Petridis, C Gutterman, G. Zussman, and Y. Bejerano, “AMuSe: Large-scale wifi video distribution-experimentation on the ORBIT testbed”, in *Proc. IEEE INFOCOM WKSHPS*, 2016.

[75] V. Gupta, R. Norwitz, S. Petridis, C. Gutterman, G. Zussman, and Y. Bejerano, “WiFi multicast to very large groups-experimentation on the ORBIT testbed”, in *Demo IEEE LCN*, 2015.

[76] C. Gutterman, E. Grinshpun, S. Sharma, and G. Zussman, “RAN resource usage prediction for a 5G slice broker”, in *Proc. ACM MobiHoc*, 2019.

[77] C. Gutterman, K. Guo, S. Arora, T. Gilliland, X. Wang, L. Wu, E. Katz-Bassett, and G. Zussman, "Requet: Real-time QoE detection for encrypted YouTube traffic", *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 16, no. 2s, 2020.

[78] C. Gutterman, K. Guo, S. Arora, X. Wang, L. Wu, E. Katz-Bassett, and G. Zussman, "Requet: Real-time QoE detection for encrypted YouTube traffic", in *Proc. ACM MMSys*, 2019.

[79] C. L. Gutterman, W. Mo, S. Zhu, Y. Li, D. C. Kilper, and G. Zussman, "Neural network based wavelength assignment in optical switching", in *Proceedings of the Workshop on Big Data Analytics and Machine Learning for Data Communication Networks*, 2017, pp. 37–42.

[80] N. Hajlaoui and I. Jabri, "On the performance of IEEE 802.11n protocol", in *Proc. ACM WiNTECH*, 2012.

[81] D. Halperin, W. Hu, A. Sheth, and D. Wetherall, "Predictable 802.11 packet delivery from wireless channel measurements", in *Proc. ACM SIGCOMM*, 2010.

[82] R. Hamilton, J. Iyengar, I. Swett, and A. Wilk, "QUIC: a UDP-based secure and reliable transport for HTTP/2", *Internet Draft, Network Working Group*,

[83] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proc. IEEE CVPR*, 2016.

[84] R. Hecht-Nielsen, "Theory of the backpropagation neural network", in *Neural networks for perception*, Elsevier, 1992, pp. 65–93.

[85] B. Héder, P. Szilágyi, and C. Vulkán, "Dynamic and adaptive QoE management for OTT application sessions in LTE", in *Proc. IEEE PIMRC*, 2016.

[86] T. K. Ho, "Random decision forests", in *Proc. IEEE Conf. Document analysis and recognition*, 1995.

[87] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, *Gradient flow in recurrent nets: The difficulty of learning long-term dependencies*, 2001.

[88] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[89] G. Holland, N. Vaidya, and P. Bahl, "A rate-adaptive MAC protocol for multi-hop wireless networks", in *Proc. ACM MobiCom*, 2001.

[90] E. Hossain and M. Hasan, "5G cellular: Key enabling technologies and research challenges", *IEEE Instrumentation & Measurement Magazine*, vol. 18, no. 3, pp. 11–21, 2015.

[91] *How Google is making YouTube safer for its users, Fortune*, `http://fortune.com/2016/08/02/google-youtube-encryption-https/`, Aug. 2016.

[92] T. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation: Evidence from a large video streaming service", in *Proc. ACM SIGCOMM*, 2014.

[93] Y. Huang, P. B. Cho, P. Samadi, and K. Bergman, "Power excursion mitigation for flexgrid defragmentation with machine learning", *Journal of Optical Communications and Networking*, vol. 10, no. 1, A69–A76, 2018.

[94] Y. Huang, C. L. Gutterman, P. Samadi, P. B. Cho, W. Samoud, C. Ware, M. Lourdiane, G. Zussman, and K. Bergman, "Dynamic mitigation of edfa power excursions with machine learning", *Optics express*, vol. 25, no. 3, pp. 2245–2258, 2017.

[95] *IEEE draft standard for information technology telecommunications and information exchange between systems local and metropolitan area networks - specific requirements, part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications - amendment: MAC enhancements for robust audio video streaming*, IEEE, 2011.

[96] "Infinera and DANTE Set Guinness World Record on GÉANT Network Record set for fastest provisioning of long haul optical transmission capacity", Infinera, Tech. Rep., 2013.

[97] K. Ishii, J. Kurumida, and S. Namiki, "Wavelength assignment dependency of AGC EDFA gain offset under dynamic optical circuit switching", in *Proc. OSA OFC*, 2014, W3E–4.

[98] ——, "Experimental investigation of gain offset behavior of feedforward-controlled WDM AGC EDFA under various dynamic wavelength allocations", *IEEE Photon. J.*, vol. 8, no. 1, pp. 1–13, 2016.

[99] G. Judd, X. Wang, and P. Steenkiste, "Efficient channel-aware rate adaptation in dynamic environments", in *Proc. ACM MobiSys*, 2008.

[100] J. Junio, D. C. Kilper, and V. W. Chan, "Channel power excursions from single-step channel provisioning", *IEEE/OSA JOCN*, vol. 4, no. 9, A1–A7, 2012.

[101] A. M. Kakhki, S. Jero, D. R. Choffnes, C. Nita-Rotaru, and A. Mislove, "Taking a long look at QUIC: an approach for rigorous evaluation of rapidly evolving transport protocols", in *Proc. ACM IMC*, Nov. 2017.

[102] A. Kamerman and L. Montebani, "WaveLAN-ii: A high-performance wireless LAN for the unlicensed band", *Bell Labs technical journal*, vol. 2, no. 3, 118–133, 1997.

[103] A. Karpathy, J. Johnson, and L. Fei-Fei, "Visualizing and understanding recurrent networks", *arXiv preprint arXiv:1506.02078*, 2015.

[104] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks", in *Proc. IEEE CVPR*, 2014.

[105] S. Kaul, M. Gruteser, and I. Seskar, "Creating wireless multi-hop topologies on space-constrained indoor testbeds through noise injection", in *Proc. IEEE TRIDENTCOM*, 2006.

[106] D. Kilper, K. Bergman, V. W. Chan, I. Monga, G. Porter, and K. Rauschenbach, "Optical networks come of age", *Optics and Photonics News*, vol. 25, no. 9, pp. 50–57, 2014.

[107] D. C. Kilper, M Bhopalwala, H Rastegarfar, and W Mo, "Optical power dynamics in wavelength layer software defined networking", in *Proc. OSA Photonic Networks and Devices*, 2015.

[108] J. Kim, S. Kim, S. Choi, and D. Qiao, "CARA: collision-aware rate adaptation for IEEE 802.11 WLANs", in *Proc. IEEE INFOCOM*, 2006.

[109] V. Krishnamoorthi, N. Carlsson, E. Halepovic, and E. Petajan, "BUFFEST: predicting buffer conditions and real-time requirements of HTTP(S) adaptive streaming clients", in *Proc. ACM MMSys*, 2017.

[110] G. Ku and J. M. Walsh, "Resource allocation and link adaptation in LTE and LTE advanced: A tutorial", *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1605–1633, 2015.

[111] J. K. Kuri and S. Kumar, "Reliable multicast in multi-access wireless LANs", *ACM/Kluwer Wirel. Netw.*, vol. 7, pp. 359–369, 4 2001.

[112] R. Kwan, R. Arnott, R. Trivisonno, and M. Kubota, "On pre-emption and congestion control for LTE systems", in *Proc. IEEE VTC*, 2010.

[113] M. Lacage, M. Manshaei, and T. Turletti, "IEEE 802.11 rate adaptation: A practical approach", in *ACM MSWiM*, 2004.

[114] W. Law, "Ultra-Low-Latency Streaming Using Chunked-Encoded and Chunked-Transferred CMAF", Akamai, Tech. Rep., Oct. 2018.

[115] D. Lecompte and F. Gabin, "Evolved multimedia broadcast/multicast service (eMBMS) in LTE-advanced: Overview and rel-11 enhancements", *IEEE Comm. Mag.*, vol. 50, no. 11, pp. 68–74, 2012.

[116] M. Leconte, G. Paschos, P. Mertikopoulos, and U. Kozat, "A resource allocation framework for network slicing", in *Proc. IEEE INFOCOM*, 2017.

[117] F. Li, J. W. Chung, and M. Claypool, "Silhouette: Identifying YouTube video flows from encrypted traffic", in *Proc. ACM NOSSDAV*, 2018.

[118] Y. Li and D. C. Kilper, “Optical physical layer SDN”, *JOCN*, vol. 10, no. 1, A110–A121, 2018.

[119] Y. Li, W. Mo, S. Zhu, Y. Shen, J. Yu, P. Samadi, K. Bergman, and D. C. Kilper, “Tsdx: Enabling impairment-aware all-optical inter-domain exchange”, *Journal of Lightwave Technology*, vol. 36, no. 1, pp. 142–154, 2017.

[120] Z. Li and T. Herfet, “HLBP: A hybrid leader based protocol for MAC layer multicast error control in wireless LANs”, in *Proc. IEEE GLOBECOM*, 2008.

[121] W.-S. Lim, D.-W. Kim, and Y.-J. Suh, “Design of efficient multicast protocol for IEEE 802.11n WLANs and cross-layer optimization for scalable video streaming”, *IEEE Trans. Mobile Comput.*, vol. 11, no. 5, pp. 780–792, 2012.

[122] K. Lin, W. Shen, C. Hsu, and C. Chou, “Quality-differentiated video multicast in multi-rate wireless networks”, *IEEE Trans. Mobile Comput.*, vol. PP, no. 99, p. 1, 2011.

[123] Y. Lin, E. M. R. Oliveira, S. B. Jemaa, and S. Elayoubi, “Machine learning for predicting QoE of video streaming in mobile networks”, in *Proc. IEEE ICC*, 2017.

[124] S. C. Madanapalli, H. H. Gharakheili, and V. Sivaraman, “Inferring Netflix user experience from broadband network measurement”, in *Proc. IEEE TMA*, 2019.

[125] A. Mahimkar, A. Chiu, R. Doverspike, M. D. Feuer, P. Magill, E. Mavrogiorgis, J. Pastor, S. L. Woodward, and J. Yates, “Bandwidth on demand for inter-data center communication”, in *Proc. ACM HotNets*, 2011.

[126] T. Mangla, E. Halepovic, M. Ammar, and E. Zegura, “eMIMIC: estimating HTTP-based video QoE metrics from encrypted network traffic”, in *Proc. IEEE TMA*, 2018.

[127] ——, “Using session modeling to estimate HTTP-based video QoE metrics from encrypted network traffic”, *IEEE Transactions on Network and Service Management*, vol. 16, no. 3, pp. 1086–1099, 2019.

[128] T. Mangla, E. Halepovic, M. H. Ammar, and E. W. Zegura, “MIMIC: using passive network measurements to estimate HTTP-based adaptive video QoE metrics”, in *Proc. IEEE TMA*, 2017.

[129] A. Mansy, M. H. Ammar, J. Chandrashekar, and A. Sheth, “Characterizing client behavior of commercial mobile video streaming services”, in *Proc. ACM MoVid*, 2014.

[130] H. Mao, R. Netravali, and M. Alizadeh, “Neural adaptive video streaming with pensieve”, in *Proc. ACM SIGCOMM*, 2017.

[131] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz, "Simple heuristics for unit disk graphs", *Networks*, vol. 25, pp. 59–68, 1995.

[132] C. Marquez, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "How should i slice my network?: A multi-service empirical evaluation of resource sharing efficiency", in *Proc. ACM MobiCom*, 2018.

[133] M. H. Mazhar and Z. Shafiq, "Real-time video quality of experience monitoring for HTTPS and QUIC", in *Proc. IEEE INFOCOM*, 2018.

[134] F. Melgani and L. Bruzzone, "Classification of hyperspectral remote sensing images with support vector machines", *IEEE Transactions on geoscience and remote sensing*, vol. 42, no. 8, pp. 1778–1790, 2004.

[135] L. Militano, D. Niyato, M. Condoluci, G. Araniti, A. Iera, and G. M. Bisci, "Radio resource management for group-oriented services in LTE-A", *IEEE Trans. Veh. Technol.*, vol. 64, no. 8, pp. 3725–3739, 2015.

[136] A. Minakhmetov, C. Gutterman, T. Chen, J. Yu, C. Ware, L. Iannone, D. Kilper, and G. Zussman, "Experiments on cloud-RAN wireless handover using optical switching in a dense urban testbed", in *Proc. OSA OFC*, 2020, Th2A–25.

[137] *Minstrel*, `https://wireless.wiki.kernel.org/en/developers/documentation/mac80211/ratecontrol/minstrel`.

[138] P. Mirowski, H. Steck, P. Whiting, R. Palaniappan, M. MacDonald, and T. K. Ho, "KL-divergence kernel regression for non gaussian fingerprint based localization", in *Proc. IPIN*, 2011.

[139] W. Mo, C. L. Gutterman, Y. Li, S. Zhu, G. Zussman, and D. C. Kilper, "Deep-neural-network-based wavelength selection and switching in ROADM systems", *Journal of optical communications and networking*, vol. 10, no. 10, pp. D1–D11, 2018.

[140] W. Mo, C. L. Gutterman, Y. Li, G. Zussman, and D. C. Kilper, "Deep neural network based dynamic resource reallocation of BBU pools in 5G C-RAN ROADM networks", in *Proc. OSA OFC*, 2018, Th1B–4.

[141] W. Mo, Y.-K. Huang, S. Zhang, E. Ip, D. C. Kilper, Y. Aono, and T. Tajima, "ANN-based transfer learning for QoT prediction in real-time mixed line-rate systems", in *Proc. OSA OFC*, 2018.

[142] W. Mo, S. Zhu, Y. Li, and D. Kilper, "EDFA wavelength dependent gain spectrum measurement using weak optical probe sampling", *IEEE Photonics Technology Letters*, vol. 30, no. 2, pp. 177–180, 2017.

[143] W. Mo, S. Zhu, Y. Li, and D. C. Kilper, "Dual-wavelength source based optical circuit switching and wavelength reconfiguration in multi-hop ROADM systems", *Optics express*, vol. 25, no. 22, pp. 27 736–27 749, 2017.

[144] A. Mondal, S. Sengupta, B. R. Reddy, M. J. V. Koundinya, C. Govindarajan, P. De, N. Ganguly, and S. Chakraborty, "Candid with YouTube: adaptive streaming behavior and implications on data consumption", in *Proc. NOSSDAV*, 2017.

[145] F. Musumeci, C. Bellanzon, N. Carapellese, M. Tornatore, A. Pattavina, and S. Gosselin, "Optimal BBU placement for 5G C-RAN deployment over WDM aggregation networks", *Journal of Lightwave Technology*, vol. 34, no. 8, pp. 1963–1970, 2015.

[146] A. Nag, Y. Zhang, L. A. DaSilva, L. Doyle, and M. Ruffini, "Integrating wireless BBUs with optical OFDM flexible-grid transponders in a C-RAN architecture", in *Proc. OSA OFC*, 2017, M2G–2.

[147] D. Neil, M. Pfeiffer, and S.-C. Liu, "Phased LSTM: Accelerating recurrent network training for long or event-based sequences", in *Advances in Neural Information Processing Systems*, 2016, pp. 3882–3890.

[148] L. E. Nelson, G. Zhang, N. Padi, C. Skolnick, K. Benson, T. Kaylor, S. Iwamatsu, R. Inderst, F. Marques, D. Fonseca, M. Du, T. Downs, T. Scherer, C. Cole, Y Zhou, P. Brooks, and A. Schubert, "SDN-controlled 400GbE end-to-end service using a CFP8 client over a deployed, commercial flexible ROADM system", in *Proc. OSA OFC*, 2017.

[149] A. Y. Ng, "Feature selection, l 1 vs. l 2 regularization, and rotational invariance", in *Proc. ICML*, 2004.

[150] *NGINX*. [Online]. Available: `https://www.nginx.com/resources/wiki/`.

[151] *ORBIT testbed*, http://orbit-lab.org/.

[152] I. Orsolic, D. Pevec, M. Suznjevic, and L. Skorin-Kapov, "YouTube QoE estimation based on the analysis of encrypted network traffic using machine learning", in *Proc. IEEE Globecom Workshops*, 2016.

[153] I. Orsolic, M. Suznjevic, and L. Skorin-Kapov, "YouTube QoE estimation from encrypted traffic: Comparison of test methodologies and machine learning based models", in *Proc. IEEE QoMEX*, 2018.

[154] A. B. Owen, *Monte Carlo theory, methods and examples*. 2013.

[155] Q. Pang, V. Leung, and S. Liew, "A rate adaptation algorithm for IEEE 802.11 WLANs based on MAC-layer loss differentiation", in *IEEE BroadNets*, 2006.

[156] K. Papagiannaki, M. Yarvis, and W. S. Conner, "Experimental characterization of home wireless networks and design implications", in *Proc. IEEE INFOCOM*, 2006.

[157] E. Park, S. Han, H. Kim, K. Son, and L. Jing, "Efficient multicast video streaming for IPTV service over WLAN using CC-FEC", in *Proc. IEEE ICICSE*, 2008.

[158] Y. Park, C. Jo, S. Yun, and H. Kim, "Multi-room IPTV delivery through pseudo-broadcast over IEEE 802.11 links", in *Proc. IEEE VTC*, 2010.

[159] K. Pelechrinis, T. Salonidis, H. Lundgren, and N. Vaidya, "Experimental characterization of 802.11n link quality at high rates", in *Proc. ACM WiNTECH*, 2010.

[160] C. Perkins and M. McBride, "Multicast WiFi problem statement", IETF Internet-Draft, 2015, `https://tools.ietf.org/html/draft-mcbride-mboned-wifi-mcast-problem-statement-00`.

[161] S. Petrangeli, T. Wu, T. Wauters, R. Huysegems, T. Bostoen, and F. De Turck, "A machine learning-based framework for preventing video freezes in HTTP adaptive streaming", *Journal of Network and Computer Applications*, 2017.

[162] *PostgreSQL*. [Online]. Available: `http://www.postgresql.org/`.

[163] B. Radunovic, A. Proutiere, D. Gunawardena, and P. Key, "Dynamic channel, rate selection and scheduling for white spaces", in *Proc. ACM CoNEXT*, 2011.

[164] H. Rahul, F. Edalat, D. Katabi, and C. Sodinii, "Frequency-aware rate adaptation and MAC protocols", in *Proc. ACM MobiCom*, 2009.

[165] T. S. Rappaport, *Wireless Communication Principle and Practice, 2nd edition*. Prentice Hall, 2002.

[166] S. Rayanchu, A. Mishra, D. Agrawal, S. Saha, and S. Banerjee, "Diagnosing wireless packet losses in 802.11: Separating collision from weak signal", in *Proc. IEEE INFOCOM*, 2008.

[167] D. Raychaudhuri, I. Seskar, G. Zussman, T. Korakis, D. Kilper, T. Chen, J. Kolodziejski, M. Sherman, Z. Kostic, X. Gu, *et al.*, "Challenge: COSMOS: A city-scale programmable testbed for experimentation with advanced wireless", in *Proc. ACM MobiCom*, 2020.

[168] A. Razaghpanah, A. A. Niaki, N. Vallina-Rodriguez, S. Sundaresan, J. Amann, and P. Gill, "Studying TLS usage in android apps", in *Proc. ACM CoNEXT*, Dec. 2017.

[169] A. Reed and M. Kranch, "Identifying HTTPS-protected Netflix videos in real-time", in *Proc. CODASPY*, 2017.

[170] C. Reis, R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Measurement-based models of delivery and interference in static wireless networks", in *Proc. ACM SIGCOMM*, 2006.

[171] I. Rhee, M. Shin, S. Hong, K. Lee, S. J. Kim, and S. Chong, "On the levy-walk nature of human mobility", *IEEE Trans. Netw.*, vol. 19, no. 3, pp. 630–643, 2011.

[172] P. Rost, C. Mannweiler, D. S. Michalopoulos, C. Sartori, V. Sciancalepore, N. Sastry, O. Holland, S. Tayade, B. Han, D. Bega, *et al.*, "Network slicing to enable scalability and flexibility in 5G mobile networks", *IEEE Communications Magazine*, vol. 55, no. 5, pp. 72–79, 2017.

[173] J. X. Salvat, L. Zanzi, A. Garcia-Saavedra, V. Sciancalepore, and X. Costa-Perez, "Overbooking network slices through yield-driven end-to-end orchestration", in *Proc. ACM CoNEXT*, 2018.

[174] K. Samdanis, X. Costa-Perez, and V. Sciancalepore, "From network sharing to multi-tenancy: The 5G network slice broker", *IEEE Commun. Mag.*, vol. 54, no. 7, pp. 32–39, 2016.

[175] Z. Sayeed, Q. Liao, D. Faucher, E. Grinshpun, and S. Sharma, "Cloud analytics for wireless metric prediction-framework and performance", in *Proc. IEEE CLOUD*, 2015.

[176] S. Scellato, I. Leontiadis, C. Mascolo, P. Basu, and M. Zafer, "Evaluating temporal robustness of mobile networks", *IEEE Trans. Mobile Comput.*, vol. 12, no. 1, pp. 105–117, 2013.

[177] P. Schmitt, F. Bronzino, R. Teixeira, T. Chattopadhyay, and N. Feamster, "Enhancing transparency: Internet video quality inference from network traffic", in *Proc. TPRC46*, 2018.

[178] S. Schwarzmann, C. C. Marquezan, M. Bosk, H. Liu, R. Trivisonno, and T. Zinner, "Estimating video streaming QoE in the 5G architecture using machine learning", in *Proc. ACM MobiCom Internet-QoE Workshop*, 2019.

[179] V. Sciancalepore, K. Samdanis, X. Costa-Perez, D. Bega, M. Gramaglia, and A. Banchs, "Mobile traffic forecasting for maximizing 5G network slicing resource utilization", in *Proc. IEEE INFOCOM*, 2017.

[180] S. Sen, N. K. Madabhushi, and S. Banerjee, "Scalable WiFi media delivery through adaptive broadcasts", in *Proc. USENIX NSDI*, 2010.

[181] Y. Seok and Y. Choi, "Efficient multicast supporting in multi-rate wireless local area networks", in *Proc. IEEE ICOIN*, 2003.

[182] M. Seufert, P. Casas, N. Wehner, L. Gang, and K. Li, "Features that matter: Feature selection for on-line stalling prediction in encrypted video streaming", in *Proc. IEEE INFOCOM Network Intelligence: Machine Learning for Networking Workshop*, 2019.

[183] V. Sgardoni, M. Sarafianou, P. Ferre, A. Nix, and D. Bull, "Robust video broadcasting over 802.11a/g in time-correlated fading channels", *IEEE Trans. Consum. Electron.*, vol. 55, no. 1, pp. 69–76, 2009.

[184] M. Shiraiwa, H. Furukawa, T. Miyazawa, Y. Awaji, and N. Wada, "Concurrently establishing and removing multi-wavelength channels reconfiguration system: Implementation for a dynamic and agile next-generation optical switching network", in *Proc. IEEE PS*, 2015.

[185] J. Shiskin, *The X-11 variant of the census method II seasonal adjustment program*, 15. US Government Printing Office, 1965.

[186] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts, Sixth Edition*. McGraw-Hill, 2010, ISBN: 0-07-352332-1.

[187] R Sivakumar, E. A. Kumar, and G Sivaradje, "Prediction of traffic load in wireless network using time series model", in *Proc. IEEE PACC*, 2011.

[188] R. Sivaraj, A. Pande, and P. Mohapatra, "Spectrum-aware radio resource management for scalable video multicast in LTE-advanced systems", in *Proc. IFIP Networking*, 2013.

[189] F. Smyth, D. C. Kilper, S. Chandrasekhar, and L. P. Barry, "Applied constant gain amplification in circulating loop experiments", *Journal of lightwave technology*, vol. 27, no. 21, pp. 4686–4696, 2009.

[190] M. R. Souryal, L. Klein-Berndt, L. E. Miller, and N. Moayeri, "Link assessment in an indoor 802.11 network", in *Proc. IEEE WCNC*, 2006.

[191] V. Srinivas and L. Ruan, "An efficient reliable multicast protocol for 802.11-based wireless LANs", in *Proc. IEEE WoWMoM*, 2009.

[192] T. Stockhammer, "Dynamic adaptive streaming over HTTP -: Standards and design principles", in *Proc. ACM MMSys*, 2011.

[193] M.-T. Sun, L. Huang, A. Arora, and T.-H. Lai, "Reliable MAC layer multicast in IEEE 802.11 wireless networks", in *Proc. IEEE ICPP*, 2002.

[194] P. Szilágyi and C. Vulkán, "LTE user plane congestion detection and analysis", in *Proc. IEEE PIMRC*, 2015.

[195] Y. Tanigawa, K. Yasukawa, and K. Yamaoka, "Transparent unicast translation to improve quality of multicast over wireless LAN", in *Proc. IEEE CCNC*, 2010.

[196] *Telerik fiddler, the free web debugging proxy*, https://www.telerik.com/fiddler.

[197] *The best live TV streaming services*, https://www.pcmag.com/picks/the-best-live-tv-streaming-services.

[198] "The Zettabyte Era - Trends and Analysis - Cisco", Cisco, Tech. Rep., 2016.

[199] C. Tian and S. Kinoshita, "Analysis and control of transient dynamics of EDFA pumped by 1480-and 980-nm lasers", *Journal of Lightwave Technology*, vol. 21, no. 8, p. 1728, 2003.

[200] G. Tian and Y. Liu, "Towards agile and smooth video adaptation in dynamic http streaming", in *Proc. ACM CoNEXT*, 2012.

[201] V. Tinto, "Dropout from higher education: A theoretical synthesis of recent research", *Review of educational research*, vol. 45, no. 1, pp. 89–125, 1975.

[202] I. Tomkos, S. Azodolmolky, J. Sole-Pareta, D. Careglio, and E. Palkopoulou, "A tutorial on the flexible optical networking paradigm: State of the art, trends, and research challenges", *Proc. IEEE*, vol. 102, no. 9, pp. 1317–1337, 2014.

[203] D. Tsilimantos, T. Karagkioules, and S. Valentin, "Classifying flows and buffer state for YouTube's HTTP adaptive streaming service in mobile networks", in *Proc. ACM MMSys*, 2018.

[204] A. W. Van der Vaart, *Asymptotic statistics*. Cambridge university press, 2000.

[205] V. Vasilev, J. Leguay, S. Paris, L. Maggi, and M. Debbah, "Predicting QoE factors with machine learning", in *Proc. IEEE ICC*, 2018.

[206] J Vella and S. Zammit, "A survey of multicasting over wireless access networks", *IEEE Commun. Surveys Tuts.*, vol. 15, no. 2, pp. 718–753, 2013.

[207] *Video dataset*, https://media.xiph.org/video/derf/.

[208] *Video streaming will account for 78% of all mobile traffic by 2021*, https://www.technative.io/video-streaming-will-account-for-78-of-all-mobile-traffic-by-2021/.

[209] J. Villalon, P. Cuenca, L. Orozco-Barbosa, Y Seok, and T. Turletti, "Cross-layer architecture for adaptive video multicast streaming over multirate wireless LANs", *IEEE J. Sel. Areas Commun.*, vol. 25, no. 4, pp. 699–711, 2007.

[210] A. Vlavianos, L. Law, I. Broustis, S. Krishnamurthy, and M. Faloutsos, "Assessing link quality in IEEE 802.11 wireless networks: Which is the right metric?", in *Proc. IEEE PIMRC*, 2008.

[211] N. Vogt, *Youtube audio quality bitrate used for 360p, 480p, 720p, 1080p, 1440p, 2160p*, https://www.h3xed.com/web-and-internet/youtube-audio-quality-bitrate-240p-360p-480p-720p-1080p, 2015.

[212] K. N. D. Vukobratovic, "A survey on application layer forward error correction codes for IP datacasting in DVB-H", in *3rd COST 2100 MCM*, 2007.

[213] M. Vutukuru, H. Balakrishnan, and K. Jamieson, "Cross-layer wireless bit rate adaptation", in *Proc. ACM SIGCOMM*, 2009.

[214] F. Wamser, M. Seufert, P. Casas, R. Irmer, P. Tran-Gia, and R. Schatz, "YoMoApp: A tool for analyzing QoE of YouTube HTTP adaptive streaming in mobile networks", in *Proc. European Conf. on Networks and Communications (EuCNC)*, 2015.

[215] H. Wang, F. Xu, Y. Li, P. Zhang, and D. Jin, "Understanding mobile traffic patterns of large scale cellular towers in urban environment", in *Proc. ACM IMC*, 2015.

[216] J. Wang, J. Tang, Z. Xu, Y. Wang, G. Xue, X. Zhang, and D. Yang, "Spatiotemporal modeling and prediction in cellular networks: A big data enabled deep learning approach", in *Proc. IEEE INFOCOM*, 2017.

[217] X. Wang, L. Wang, and D. Wang Y.and Gu, "Reliable multicast mechanism in WLAN with extended implicit MAC acknowledgment", in *Proc. IEEE VTC*, 2008.

[218] X. Wang, L. Wang, Y. Wang, Y. Zhang, and A. Yamada, "Supporting MAC layer multicast in IEEE 802.11n: Issues and solutions", in *Proc. IEEE WCNC*, 2009.

[219] X. Wang, E. Grinshpun, D. Faucher, and S. Sharma, "On medium and long term channel conditions prediction for mobile devices", in *Proc. IEEE WCNC*, 2017.

[220] X. Wang, Z. Zhou, Z. Yang, Y. Liu, and C. Peng, "Spatio-temporal analysis and prediction of cellular traffic in metropolis", in *Proc. IEEE ICNP*, 2017.

[221] S. Wassermann, P. Casas, M. Seufert, and F. Wamser, "On the analysis of YouTube QoE in cellular networks through in-smartphone measurements", in *Proc. IFIP WMNC*, 2019.

[222] S. Wassermann, M. Seufert, P. Casas, L. Gang, and K. Li, "I see what you see: Real time prediction of video quality from encrypted streaming traffic", in *Proc. ACM MobiCom Internet-QoE Workshop*, 2019.

[223] N. Weil, *The state of MPEG-DASH 2016*, http://www.streamingmedia.com/Articles/Articles/Editorial/Featured-Articles/The-State-of-MPEG-DASH-2016-110099.aspx.

[224] S. Wong, H. Yang, S. Lu, and V. Bharghavan, "Robust rate adaptation for 802.11 wireless networks", in *Proc. ACM MobiCom*, 2006.

[225] F. Wu, Y. Yang, O. Zhang, K. Srinivasan, and N. B. Shroff, "Anonymous-query based rate control for wireless multicast: Approaching optimality with constant feedback", in *Proc. ACM MobiHoc*, 2016.

[226] M Wu, S. Makharia, H. Liu, D. Li, and S. Mathur, "IPTV multicast over wireless LAN using merged hybrid ARQ with staggered adaptive FEC", *IEEE Trans. Broadcast.*, vol. 55, no. 2, pp. 363 –374, 2009.

[227] T. J. Xia, G. A. Wellbrock, M.-F. Huang, S. Zhang, Y.-K. Huang, D.-i. Chang, S. Burtsev, W. Pelouch, E. Zak, H. de Pedro, *et al.*, "Transmission of 400G PM-16QAM channels over long-haul distance with commercial all-distributed raman amplification system and aged standard SMF in field", in *Proc. OSA OFC*, 2014.

[228] X. Xie, X. Zhang, S. Kumar, and L. E. Li, "piStream: Physical layer informed adaptive video streaming over LTE", in *Proc. ACM MobiCom*, 2015.

[229] Q. Xu, S. Mehrotra, Z. Mao, and J. Li, "Proteus: Network performance forecast for real-time, interactive mobile applications", in *Proc. ACM MobiSys*, 2013.

[230] S. Yan, F. N. Khan, A. Mavromatis, D. Gkounis, Q. Fan, F. Ntavou, K. Nikolovgenis, F. Meng, E. H. Salas, C. Guo, *et al.*, "Field trial of machine-learning-assisted and SDN-based optical network planning with network-scale monitoring database", in *Proc. ECOC*, 2017.

[231] *Yinzcam*, http://www.yinzcam.com/.

[232] J. Yoon, H. Zhang, S. Banerjee, and S. Rangarajan, "MuVi: A multicast video delivery scheme for 4G cellular networks", in *Proc. ACM MobiCom*, 2012.

[233] Y. You, Z. Jiang, and C. Janz, "OSNR prediction using machine learning-based EDFA models", in *Proc. ECOC*, 2019.

[234] *YouTube TV review: The best premium live-TV streaming service*, `https://www.cnet.com/news/youtube-tv-review-the-best-premium-live-tv-streaming-service/`.

[235] J. Yu, T. Chen, C. Gutterman, S. Zhu, G. Zussman, I. Seskar, and D. Kilper, "COSMOS: Optical architecture and prototyping", in *Proc. OSA OFC*, 2019, M3G–3.

[236] J. Yu, C. Gutterman, A. Minakhmetov, M. Sherman, T. Chen, S. Zhu, G. Zussman, I. Seskar, and D. Kilper, "Dual use SDN controller for management and experimentation in a field deployed testbed", in *Proc. OSA OFC*, 2020, T3J–3.

[237] C. Zhang, X. Ouyang, and P. Patras, "ZipNet-GAN: Inferring fine-grained mobile traffic patterns via a generative adversarial neural network", in *Proc. ACM CoNEXT*, 2017.

[238] C. Zhang and P. Patras, "Long-term mobile traffic forecasting using deep spatio-temporal neural networks", in *Proc. ACM Mobihoc*, 2018.

[239] S. Zhu, C. Gutterman, A. D. Montiel, J. Yu, M. Ruffini, G. Zussman, and D. Kilper, "Hybrid machine learning EDFA model", in *Proc. OSA OFC*, 2020, T4B–4.

[240] S. Zhu, C. L. Gutterman, W. Mo, Y. Li, G. Zussman, and D. C. Kilper, "Machine learning based prediction of erbium-doped fiber WDM line amplifier gain spectra", in *Proc. ECOC*, 2018.

[241] X. K. Zou, J. Erman, V. Gopalakrishnan, E. Halepovic, R. Jana, X. Jin, J. Rexford, and R. K. Sinha, "Can accurate predictions improve video streaming in cellular networks?", in *Proc. ACM HotMobile*, 2015.