NeuRA: Using Neural Networks to Improve WiFi Rate Adaptation

by

Shervin Khastoo

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2020

Author's Declaration

Abstract

Although a variety of rate adaptation algorithms have been proposed for 802.11 devices, sampling-based algorithms are preferred and used in practice because they only require frame loss information which is available on all devices. Unfortunately, sampling can impose significant overheads because it can lead to excessive frame loss or the choice of suboptimal rates. In this thesis, we design a novel neural network based rate adaptation algorithm, called NeuRA. NeuRA significantly improves the efficiency of sampling in rate adaptation algorithms by using a neural network model to predict the expected throughput of many rates, rather than sampling their throughput. Furthermore, we propose a feature selection technique to select the best set of rates to sample.

Despite decades of research on rate adaptation in 802.11 networks, there are no definitive results which determine which algorithm is the best or if any algorithm is close to optimal. We design an offline algorithm that uses information about the fate of future frames to make statistically optimal frame aggregation and rate adaptation decisions. This algorithm provides an upper bound on the throughput that can be obtained by practical online algorithms and enables us to evaluate rate adaptation algorithms with respect to this upper bound.

Our trace-based evaluations using a wide variety of real-world scenarios show that NeuRA outperforms the widely-used Minstrel HT algorithm by up to 24% (16% on average) and the Intel iwl-mvm-rs algorithm by up to 32% (13% on average). Moreover, the upper bound given by the offline optimal algorithm shows a throughput up to 58% (30% on average) higher than Minstrel HT and up to 31% (12% on average) higher than NeuRA. Hence, NeuRA reduces the gap in throughput between Minstrel HT and the offline optimal algorithm by half. Additionally, our results show that several-fold improvements over Minstrel HT shown in previous work are unlikely to be obtained in real-world scenarios. Finally, we implement NeuRA using the Linux ath9k driver to show that the neural network processing requirements are sufficiently low to be practical and that NeuRA can be used to obtain statistically significant improvements in throughput when compared with Minstrel HT.

Acknowledgements

I would like to thank my supervisor, Tim Brecht, for his guidance and support throughout my degree. I am grateful for all the time and effort he put into helping me with research skills and providing me with consistent feedback during the course of this project.

I would like to thank Ali Abedi and Ali Mashtizadeh for their valuable feedback on this thesis. I also thank Ali Abedi for helping me with the basics of WiFi networks when I was new to this area.

I appreciate the financial support provided by the David R. Cheriton School of Computer Science, Tim Brecht, and the University of Waterloo.

Finally, I thank my parents for their guidance and support. Most importantly, I thank my wonderful wife, Fatemeh Hassani, for her tremendous support during my graduate studies. I dedicate this thesis to her.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Thesis Statement: If relationships between WiFi physical rates can be leveraged, they could be used to reduce the overhead of WiFi rate adaptation algorithms and improve throughput.

1.1 Motivation

Modern wireless networking standards such as 802.11n and 802.11ac can achieve theoretical throughputs of up to 600 Mbps and 3.5 Gbps respectively. These standards use dense modulations, channel bonding, multiple-input multiple-output (MIMO), and frame aggregation to achieve such high throughput. As a result, modern WiFi devices have to choose from a large number of different configurations (i.e., up to 768 physical rates).

The best physical rate to use for transmission at any point in time depends on the channel state and a wide variety of environmental factors that can change in a fraction of a second. In order to achieve the highest possible throughput, WiFi devices use a rate adaptation algorithm to constantly choose the best physical rate for each packet transmitted. Furthermore, a frame aggregation algorithm is used to aggregate up to 64 subframes (MPDUs) into an aggregated frame (A-MPDU) to increase the MAC layer efficiency. Without frame aggregation, a 450 Mbps physical rate cannot achieve more than 50 Mbps. Rate adaptation and frame aggregation algorithms are not included in WiFi standards and device manufacturers have to choose and implement their own mechanisms.

Sampling-based rate adaptation algorithms (e.g., Minstrel HT [12]) are highly popular in commercial devices because they make decisions based on real-time measurements and they have been shown to work in a variety of conditions. These algorithms periodically probe recently unused physical rates by transmitting data using those rates. This allows the algorithm to determine the effective throughput of physical rates empirically. Sampling, however, leads to a loss in throughput as data will be transmitted using non-optimal rates. Moreover, because the fate of each packet is unknown when sampling, data is typically sent without the use of frame aggregation, which also leads to a significant loss in channel efficiency. As a result, there is an inherent trade-off between the sampling overhead and effectiveness of the rate adaptation algorithm. In this work, we focus on improving upon practical, widely-used, sampling-based algorithms and understanding how they perform when compared to the offline statistically optimal algorithm.

With the growing number of modulation and coding schemes in newer WiFi standards, naïve sampling methods do not scale well. Recent work has found that the frame error rate (FER) of one physical rate may correlate with the frame error rate of several other physical rates [1]. To the best of our knowledge, no algorithms exist that utilize the relationships between rates to infer information about one or more rates based on the feedback obtained from sampling other rates.

Furthermore, in 802.11n and later WiFi standards, the optimal physical rate for transmission depends on the optimal number of frames to aggregate for each rate. Despite years of research on improving rate adaptation and frame aggregation algorithms, an upper bound on how much better than widely-used algorithms new algorithms can perform is unknown. An upper bound is important in order to understand how close to or how far from optimal algorithms are and whether or not more research is needed.

1.2 Overview

In this thesis, we propose a neural network model for predicting the effective throughput of all physical rates based on the effective throughput of a subset of rates that we call the sampling set. We use a feature selection method to search for and find sampling sets of different sizes (i.e., different numbers of rates) that maximize the prediction power for estimating other rates. We implement a rate adaptation algorithm called NeuRA (Neural network-based Rate Adaptation) that utilizes estimations from a neural network model to reduce the amount of sampling required for rate adaptation.

Later, we derive and compare against a statistically optimal offline algorithm that uses an oracle to make the optimal choice of the physical rate and the number of frames to aggregate. This algorithm provides an upper bound on the throughput that could be obtained using practical online algorithms (including NeuRA).

We compare the performance of NeuRA with several other state-of-the-art rate adaptation algorithms including Minstrel HT (used by hundreds of millions of devices [31] and the default rate adaptation algorithm in the Linux kernel's *mac80211* driver development framework) and the Intel iwl-mvm-rs algorithm (used by modern Intel WiFi devices included in modern laptops and computers).

When performing a trace-based evaluation across a variety of traces collected using scenarios that reflect the environments in which WiFi devices are actually used, we find that NeuRA provides significant improvements when compared with existing algorithms, even on scenarios with previously unseen client devices and movement behaviours. This demonstrates that the neural network model learns relationships between rates that are generalizable across different devices and environments. Also, in many experiments NeuRA obtains throughput surprisingly close to that of the offline optimal algorithm.

To evaluate the practicality of using NeuRA, we implement and evaluate a real-world prototype using the *ath9k* WiFi driver. We find that NeuRA effectively improves the throughput over Minstrel HT with relatively small additional requirements in processing power.

1.3 Contributions

The contributions of this thesis are as follows:

- We propose a novel rate adaptation algorithm (NeuRA) that uses a neural network model to estimate the effective throughput of the rates that are not sampled from a smaller set of sampled rates. NeuRA reduces sampling overheads and increases throughput.

- We use a recursive feature elimination (RFE) technique to recursively reduce the number of sampling rates while ensuring that remaining rates incur low overheads yet have good predictive power.

- We develop and describe an offline algorithm to calculate the statistically optimal physical rate and number of frames to aggregate. This provides an upper bound on the throughput that can be obtained using online algorithms.

- We find that NeuRA performs up to to 24% (16% on average) better than Minstrel HT and up to 32% (13% on average) better than Intel iwl-mvm-rs. NeuRA provides throughputs that are relatively close to that of the offline statistically optimal. This is despite the advantages gained by the offline algorithm from using information about the future that is not available to online algorithms like NeuRA.

- Our results show that the offline statistically optimal algorithm performs up to 58% (30% on average) better than Minstrel HT and up to 31% (12% on average) better than NeuRA. This finding suggests that even though there is some room for improving rate adaptation algorithms, it is not large. Additionally, it indicates that several-fold improvements over Minstrel HT claimed in some previous work are likely the result of algorithm misconfigurations and may not be true indicators of a fair comparison using real-world use cases.

- We implement a real-world prototype of NeuRA using the *ath9k* WiFi driver to evaluate the practicality NeuRA and find that it uses relatively little CPU power (less than 20% of a 800 MHz CPU core) to perform estimations. It also increases throughput by 15% on average when compared to Minstrel HT.

1.4 Thesis Organization

The rest of this thesis is organized as follows. Chapter 2 provides background information and describes related research. In Chapter 3, we describe the new algorithms that we have developed. We begin by describing the neural rate adaptation (NeuRA) algorithm which uses estimations from a neural network model to reduce the sampling overhead. We then present the offline statistically optimal algorithm which provides an upper bound on the throughput achievable by the best rate adaptation and frame aggregation algorithms. Chapter 4 describes the required components to perform trace-based evaluations. We first describe the process of collecting traces for training and evaluation and describe the different scenarios we use to collect traces. We then describe the technique we use to verify the correctness of the algorithms implemented in T-SIMn and describe the bugs we found and how they were fixed. Chapter 5 contains a comprehensive evaluation of the neural network model and a trace-based evaluation comparing NeuRA, the offline statistically optimal algorithm, and several widely-used algorithms. In Chapter 6, we describe a real-world prototype of NeuRA using the *ath9k* driver and evaluate its performance and the required processing power. Finally, we conclude the thesis and present ideas for future work in Chapter 7.

Chapter 2

Background and Related Work

2.1 Background

In this section, we provide an overview of concepts related to and required to better understand rate adaptation and frame aggregation. We then provide background information on trace collection and trace-based evaluation. Finally, we provide a bit of background information on neural network models.

2.1.1 Rate Adaptation

Modern WiFi standards contain several physical layer features that can be configured for transmitting packets [14]. These features are:

- **Modulation and Coding Scheme (MCS Index):** The MCS index specifies the modulation type and the coding rate to use for transmitting data frames. The modulation type specifies how digital data bits are encoded in an analog signal and vice versa. The coding rate specifies the ratio of useful data bits to the total number of bits transmitted (non-useful bits provide redundancy for error correction purposes). For example, MCS index 0 usually refers to the BPSK modulation type and a 1/2 coding rate.

- **Number of Spatial Streams (Antennas):** Beginning with 802.11n, WiFi networks support Multiple-Input/Multiple-Output (MIMO) communication. MIMO transmits several streams of data simultaneously using multiple antennas [15]. MIMO can increase the physical layer throughput of 802.11n devices by up to four times when using four antennas. The optimal number of spatial streams to use is variable and depends on channel conditions.

- **Guard Interval Length (GI):** The guard interval length specifies the amount of time that the transmitter is idle between sending different symbols. This is done to ensure that consecutive symbols do not interfere with each other. In 802.11n, 800 ns is referred to as a long guard interval (LGI) and 400 ns is referred to as a short guard interval (SGI).

- **Channel Width:** A single WiFi channel has a 20 MHz width. Starting with the 802.11n standard, WiFi devices are capable of aggregating channels (this is called channel bonding) to use 40 MHz or wider channel widths for transmission.

The combination of these physical layer features provides WiFi devices with a large number of choices when transmitting data frames. The physical rate refers to a specific combination of physical layer features. For example, MCS Index 5, 2 Spatial Streams, LGI, and 40 MHz refers to a single physical rate. In most indexing schemes, a different set of MCS indices are assigned for each set of spatial streams. So, in 802.11n networks, the previously mentioned physical rate is often referred to as MCS Index 13, LGI, 40 MHz.

The error rate of each physical rate is different for each environment. Error rates depend on the distance between the sender and the receiver, a wide variety of environmental parameters such as the location, shape and material of the objects in the environment, and WiFi and non-WiFi interference from other devices. As a result, the physical rate that will result in the highest achievable throughput is different in each environment.

WiFi devices must continually adapt to the constantly changing environment in order to transmit data using the best physical rate. An algorithm that performs this adaptation is called a rate adaptation algorithm. Rate adaptation algorithms are not specified in WiFi standards and are the responsibility of device manufacturers to implement. Performing rate adaptation has considerable overheads and costs associated with it. As a result, improving rate adaptation algorithms and reducing their overheads is an active area of research.

2.1.2 Frame Aggregation

Before the 802.11n standard, the WiFi MAC layer was highly inefficient because of the overheads imposed by acknowledgement frames and inter-frame gaps (data transmission throughput was a small fraction of the physical layer bitrate) [15]. The 802.11n standard added a feature called frame aggregation to increase the efficiency of WiFi MAC layer. Frame aggregation enables WiFi devices to aggregate up to 64 data frames called subframes or MAC Protocol Data Units (MPDUs) into a larger aggregated frame called an Aggregated MAC Data Protocol Unit (A-MPDU). The aggregated frame is then transmitted as a single unit. Furthermore, the receiver sends back a single block acknowledgement frame (Block ACK) to acknowledge the individual frames within an aggregated frame. Without frame aggregation a 450 Mbps physical rate (MCS Index 23, SGI, 40 MHz) cannot achieve more than 50 Mbps.

WiFi devices use frame aggregation algorithms to determine the best number of frames to aggregate. These algorithms usually aggregate as many frames as possible up to a maximum length (e.g., 64 frames) or up to the maximum transmission time permitted (e.g., 4 ms). Recent research has shown that aggregating as many frames as possible is not always the best choice and in some situations, reducing the aggregation length increases WiFi throughput [2, 7]. This is because for some receivers, frames towards the end of an aggregated frame experience much higher error rates than frames at the beginning. As a result, more complex frame aggregation algorithms have emerged in an attempt to continually try to determine the best aggregation length for each physical rate. Improving frame aggregation algorithms has become an active area of research.

2.1.3 Trace-Based Evaluation

T-SIMn [4] is an 802.11n trace-based simulator. It allows one to record traces from real-world WiFi experiments in a variety of settings using different client devices and then to simulate the running of different combinations of rate adaptation and frame aggregation algorithms using the recorded trace.

Conducting credible empirical evaluations is extremely difficult in WiFi networks because of highly variable channel conditions. Abedi et al. [3] show that even under controlled conditions with no WiFi or non-WiFi interference, it is extremely difficult to perform repeatable WiFi experiments or distinguish differences between competing alternatives. Trace-based evaluation, on the other hand, enables the fair comparison of different rate adaptation and frame aggregation algorithms because all algorithms are exposed to exactly the same channel conditions. At any point in time, the simulator can determine the expected throughput for the number of aggregated frames and the physical rate chosen by the algorithms of interest. Because real-world traces are used, each algorithm is subjected to identical channel conditions that replicate real-world conditions. We now explain how T-SIMn enables such comparisons.

During trace collection, the sending device transmits packets using all available physical rates in a round-robin fashion. Each packet is sent using the maximum possible number of aggregated frames for that rate and a variety of information about transmitted packets and the received block ACK frames is recorded in the trace. This information includes the physical rate used for transmitting each packet, the block ACK frame received for that packet (or a notion that no block ACK is received within the timeout period), the RSSI (received signal strength indication) of the block ACK frame, the total transmission time, and channel access information (the amount of time it took the device to acquire the channel). This information can be used to understand the environment and to replicate the same environmental conditions when performing trace-based evaluations.

During the trace collection process, a large number of frames are sent in a short period of time (as fast as possible) to gather statistics about the behaviour of all physical rates as quickly as possible. When running trace-based evaluations, T-SIMn uses the information from the trace in a window of time around the current time to determine the expected fate of each subframe for all rates.

The sender device used in trace collection needs to run a modified driver with custom rate adaptation and frame aggregation algorithms. As previously explained, the driver should use a round-robin scheme for selecting the physical rate and aggregate as many frames as possible when collecting the trace. It should also record the required information for every transmitted packet and dump it in a file to create the trace. T-SIMn provides a modified *ath9k* driver for devices supporting this driver. The receiver device, on the other hand, has no specific requirements and any WiFi device can be used.

In Chapter 5, we use T-SIMn to perform trace-based evaluations in order to fairly compare our proposed rate adaptation algorithm (NeuRA) and the offline statistically optimal algorithm (proposed in Section 3.2) to several other widely-used algorithms. The trace collection process is described in detail in Section 4.1.

2.1.4 Neural Networks

Neural networks are a powerful tool that are widely used as function estimators [13]. With enough layers and neurons and given enough data and processing power for training, they can approximate a wide variety of functions for regression and classification [28, 29]. Regression models approximate a function mapping the input variables to continuous real-valued output variables while classification models approximate a function mapping the input variables to discrete output variables.

In this thesis, we use a neural network model in the core of our rate adaptation algorithm that learns the relationships between physical rates during the training phase. The neural network then approximates a function mapping the throughput of a subset of rates that are sampled to the throughput of other non-sampled rates. As we explain in Chapter 3, this can be used to reduce the overhead of rate adaptation algorithms and improve their throughput.

Without proper supervision, neural network models may overfit to the training data and not learn generalizable relationships between the input and the output. Dropout [30] is a technique to overcome the problem of overfitting in neural networks. During the training phase, the Dropout technique randomly sets the value of a ratio of the neurons in each layer to zero and amplifies the value of the others. It forces the neural network to avoid converging to solutions that depend on specific connections between the neurons and/or specific input values. We use this technique in our neural network model to avoid overfitting the model on the training data and also force the neural network to learn multiple ways of predicting an output value. The latter is useful when we eliminate the least important features in the feature selection phase.

2.2 Related Work

In this section, we review previous research related to our work. In Section 2.2.1, we describe different approaches to rate adaptation and explain why sampling-based rate adaptation algorithms are the most popular choice for commercial devices. In Section 2.2.2, we describe several approaches to improving frame aggregation algorithms. Finally, in Section 2.2.3, we describe previous research on the existence of relationships between WiFi physical rates which is the basis of some of our work.

2.2.1 Rate Adaptation Algorithms

The problem of rate adaptation in WiFi networks (sometimes called “rate control” or “link adaptation”) is a long standing and widely researched problem. We review a subset of these methods (mostly for 802.11n and 802.11ac networks) that are related to our work. For a more comprehensive and detailed survey please see Yin et al. [31].

Sampling-Based Rate Adaptation

Many WiFi rate adaptation algorithms use sampling which was first introduced in the SampleRate algorithm [5]. Sampling-based algorithms periodically sample all or some of the physical rates and use those measurements to choose the rate with the highest expected throughput.

SampleRate [5] uses 10% of the frames for sampling. These frames are sent using random rates selected from all available rates to measure their throughput. Every 10 seconds, it calculates the throughput for all physical rates and chooses the one with the highest expected throughput. The rate with the highest throughput is used to transmit the other 90% of the frames that are not used for sampling. SampleRate [5] changes the rate only every 10 seconds which is considered very slow especially if the clients are mobile and the best physical rate changes quickly. Also, it was designed for 802.11g networks (with 8 physical rates) and its approach is not scalable to modern WiFi standards with many more physical rates.

Minstrel [11] is a sampling-based rate adaptation algorithm for 802.11g networks that works in a similar fashion to SampleRate. Minstrel adapts the rate more frequently than SampleRate and improves sampling by avoiding sampling the rates with very low or very high error rates. It also selects four different rates to be tried in order (called a retry chain) when one rate fails. During transmission, the WiFi device performs several physical layer retransmissions when an aggregated frame is not transmitted successfully. It starts by using the first rate in the retry chain and moves on to the next rate when a pre-specified number of failures are observed with the current rate. These features and optimizations make Minstrel a good choice for commercial devices. Minstrel HT [12] is a more recent variation of Minstrel that is designed to work with with 802.11n and 802.11ac networks. Minstrel and Minstrel HT are used in hundreds of millions of devices and are considered the most widely used rate adaptation algorithms for 802.11g and 802.11n networks respectively [31]. Minstrel HT is the default rate adaptation algorithm in the *mac80211* driver development framework of Linux kernel as well as the widely-used *ath9k* WiFi driver.

Minstrel HT uses a ratio of transmitted frames for sampling (called probe frames) and updates the four rates used in the retry chain several times per second. These four rates are: the rate with the highest measured throughput, the rate with the second highest throughput, the rate with the lowest error rate, and the lowest available rate. Minstrel HT also limits the amount of sampling by checking a variety of conditions such as: avoiding sampling the rates in the retry chain, avoiding sampling the rates that currently have an error rate of less than 5%, and heavily reducing the sampling frequency for rates that are slower than the first three rates in the retry chain. These optimizations considerably reduce the sampling overhead of Minstrel HT when compared to naïve sampling algorithms such as SampleRate. As a result of its practicality, it has become a widely-used algorithm. However, previous work suggests that Minstrel HT performs excessive sampling introducing a considerable overhead [1, 26, 10].

There are many other sampling-based rate adaptation algorithms that take a different approach to sampling than Minstrel HT. We next describe some examples of such algorithms. MiRA [27] is one of the earliest algorithms that considers MIMO (multiple spatial streams) physical rates. It classifies the physical rates into two MIMO modes ("single stream" and "double stream") and favours staying in the same MIMO mode unless it senses a significant change in the throughput. It performs little regular sampling and instead, upon sensing significant changes in the throughput of the currently used rate, it starts to sample more aggressively and samples the next lower and the next higher rate in the current MIMO mode as well as several rates in the other MIMO mode. MiRA may not react quickly if the current rate has a very low error rate and a faster rates becomes available while there is no significant change to current rate's throughput.

While MiRA only considers devices with two antennas, RAMAS [26] presents another rate adaptation algorithm that can be used with any number of antennas. RAMAS splits the physical rates into enhancement groups each representing a combination of the number of spatial streams, guard interval length, and channel width. These enhancement groups are ordered as shown in Table 2.1 by group index. RAMAS adapts the enhancement group and MCS index concurrently. To adapt each one, it uses a credit-based scheme that counts every successful frame as a positive credit and every failed frame as a negative credit. Whenever the credit exceeds a certain threshold, it increases the MCS index or enhancement group index by one and whenever the credit gets lower than a threshold, it decreases the MCS index or enhancement group index by one. Different credit thresholds are used for MCS index adaptation and enhancement group adaptation.

Unfortunately, many other modern devices implement rate adaptation in their proprietary firmware, making it almost impossible to evaluate those algorithms. However, Intel WiFi devices use a custom rate adaptation algorithm that is implemented in the open source *IwlWiFi* driver studied by Grünblatt et al. [16]. Intel iwl-mvm-rs is the most recent rate adaptation algorithm designed for Intel WiFi devices. It is the successor to Intel iwl-agn-rs and is now used with the most recent Intel chipsets. Since Intel WiFi devices are used in many modern laptops and personal computers, this algorithm is another example of a practical and widely-used algorithm. Intel iwl-mvm-rs splits rates into several groups (called columns) similar to RAMAS's enhancement groups. Rate groups (columns) for a two-antenna device using 802.11n rates are shown in Table 2.2 (A and B refer to the two antennas of the device). Every column has a list of successor columns that are

Table 2.1: Enhancement groups for RAMAS [26].

Group Index	Num. Spatial Streams	Guard Interval	Channel Width
0	1	800 ns	20 MHz
1	1	400 ns	20 MHz
2	1	800 ns	40 MHz
3	1	400 ns	40 MHz
4	2	800 ns	20 MHz
5	2	400 ns	20 MHz
6	2	800 ns	40 MHz
7	2	400 ns	40 MHz
8	3	800 ns	20 MHz
9	3	400 ns	20 MHz
10	3	800 ns	40 MHz
11	3	400 ns	40 MHz
12	4	800 ns	20 MHz
13	4	400 ns	20 MHz
14	4	800 ns	40 MHz
15	4	400 ns	40 MHz

sampled when the current best rate resides in that column. Intel iwl-mvm-rs performs sequential phases of MCS index adaptation and column adaptation in a loop as shown in Figure 2.1 [16]. MCS index adaptation samples one lower and one higher rate in the current column and switches to the best one. In the column scaling phase, every successor column of the current column is sampled to check if the expected throughput of any rate in those columns will be higher than the current rate's throughput.

As described, different approaches are proposed for sampling-based rate adaptation in previous work. To the best of our knowledge, none of these algorithms utilize relationships between physical rates to infer information about the reliability of non-sampled rates. In this thesis, we propose NeuRA (Neural Rate Adaptation) to improve the throughput of sampling-based rate adaptation algorithms by reducing their sampling overhead. NeuRA uses a neural network model to predict the throughput of non-sampled rates using the measured throughput of other rates. We compare our proposed algorithm to Minstrel HT and Intel iwl-mvm-rs because they are two practical and widely-used algorithms.

Table 2.2: Rate columns used in Intel iwl-mvm-rs [16].

Column	Antennas	Guard Interval	Next Columns
0	{A}	LGI	{1, 2, 4}
1	{B}	LGI	{0, 3, 4}
2	{A}	SGI	{0, 3, 5}
3	{B}	SGI	{1, 2, 5}
4	{A, B}	LGI	{0, 1, 5}
5	{A, B}	SGI	{1, 3, 4}

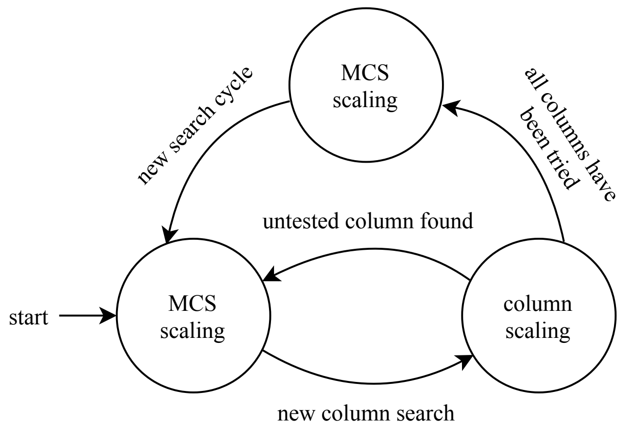


Figure 2.1: Different states of the Intel iwl-mvm-rs algorithm [16].

RSSI/SNR-based Rate Adaptation

Several algorithms propose methods for determining and using some information about the signal strength at the sender or the receiver to make informed decisions about the best transmission rate to use for the current channel conditions.

SampleLite [24] uses the sender-side RSSI of incoming frames (from acknowledgements) to select a baseline rate. To do so, it uses lookup tables to map different RSSI ranges to different values of each physical layer feature (e.g., if the RSSI is less than -67, use a 20 MHz channel width, otherwise, use a 40 MHz channel width). It then performs very limited sampling by trying one MCS index lower and one MCS index higher than the chosen rate and selecting the best one. The sender-side RSSI is known to be an unreliable metric for choosing the best transmission rate [17] and RSSI ranges proposed in SampleLite need to be recalibrated for different devices and channel conditions.

The best source of information about the signal strength would be obtained from the receiver side. There are several algorithms that use the signal strength at the receiver side to improve rate adaptation. Unfortunately, transmitting this information to the sender requires modifying control frames [31] (which requires changing protocol standards) or relies on optional features of 802.11 protocols that to our knowledge are not typically implemented by chipset manufacturers. Furthermore, such transmission incurs significant overhead by consuming bandwidth between the sender and the receiver. Finally, by the time the information is transmitted from the receiver to the sender, it may have become invalid due to changes in the channel [26].

The ESNR [17] algorithm uses the channel state information (CSI) matrix reported by the receiver device to predict which physical rates can transmit data reliably (have an error rate of less than or equal to 10%). Based on our observations, the best rate for transmission often has an error rate of greater than 10%. So this fixed threshold scheme may not be able to choose the best rate. Also, many WiFi devices do not report CSI [24, 6].

ARAMIS [10] provides an alternative solution to ESNR for devices that do not report CSI information. It uses the average SNR and separate SNR values from different antennas on the receiver side to lookup the best transmission rate from a lookup table. This table requires recalibration for different devices and environments.

Both ESNR and ARAMIS need a way to transmit the metrics from the receiver side to the sender and as previously explained, there are several issues with this approach. As a result, these algorithms are not used in practice.

Other Rate Adaptation Methods

HiWiLA [21] continually adapts the transmission rate using a series of state transitions in a predesigned state transition graph. SmartLA [22] builds a reinforcement learning model on top of HiWiLA to perform state transitions based on a reward (the current bit error rate). SmartLA is related to our work since it employs machine learning in rate adaptation. We had hoped to include SmartLA in our evaluation but we were not able to obtain code from the authors and were not able to replicate their implementation because of missing details regarding critical parameters used in their system and because it would require access to their training data.

HiWiLA and SmartLA report improvements of factors of 6 and 20 respectively when compared with Minstrel HT. We believe these results are misleading because SmartLA uses all available 802.11ac rates and frame aggregation while the algorithms they compare with are restricted to using only 802.11g or 802.11n rates and do not appear to use frame aggregation. As a result, in many evaluations Minstrel HT throughput does not exceed 25 Mbps while SmartLA achieves 500 Mbps (a factor of 20x difference). We believe their large improvements are mainly due to differences in configurations (which rates can be used and frame aggregation) rather than differences in algorithms. Across the variety of devices and scenarios we studied, the gap between Minstrel HT and the statistically optimal solution is much lower (about 30% on average and never more than 60%).

2.2.2 Frame Aggregation Algorithms

Researchers have observed that in some cases, subframes towards the end of an aggregated frame incur higher error rates than subframes in the beginning [7]. As a result, MoFA [7] and STRALE [8] attempt to calculate a good length to use for an aggregated frame (A-MPDU). STRALE also additionally adjusts the physical rate to one rate lower or one rate higher in an attempt to improve the throughput in certain situations.

PNOFA [2] is another algorithm that attempts to approximate the optimal frame aggregation length. It uses Equation 2.1 to calculate the expected throughput for all possible aggregation lengths and then chooses the best length for the current rate.

$$T_R(N) = \frac{\sum_{i=1}^{N}(1 - SFER_R(i))}{\tau_R(N)} \tag{2.1}$$

In Equation 2.1, $T_R(N)$ represents the expected throughput from aggregating N packets when using rate R. $SFER_R(i)$ is the average subframe error rate of the i-th subframe in an aggregated frame sent using rate R which is calculated using statistics from previously received block ACK frames. $\tau_R(N)$ is the time required to transmit an aggregated frame of length N using rate R.

PNOFA then adds a few frames to the optimal length to sample the error rate of later subframes in order to determine when to increase the aggregation limit when the best length increases. PNOFA is shown to provide better performance than MoFA and STRALE [2].

PNOFA also introduces an offline statistically optimal frame aggregation algorithm (which we call OSOFA). It uses Equation 2.1 to choose the best aggregation length, but an oracle is used to determine $SFER_R(i)$ values for future aggregated frames instead of calculating it based on previously received block ACK frames. In a trace-based evaluation environment, this can be calculated from the trace by averaging subframe error rates in a window of time centered around the current time (the same way the trace-based evaluator decides the fate of subframes in the next A-MPDU). OSOFA then uses the exact calculated length without adding any extra subframes. This algorithm provides an upper bound on how much frame aggregation algorithms can improve throughput.

In our offline statistically optimal rate and frame adaptation algorithm, we calculate the statistically optimal aggregation length for each rate in a similar fashion to OSOFA. We also include STRALE and PNOFA in our trace-based evaluations to compare them with other algorithms.

2.2.3 Relationships Between Physical Rates

Abedi et al. [1] show that there are relationships between the frame error rates of different physical rates used by WiFi devices. That study has focused on providing evidence that such relationships exist and their prevalence across different scenarios. They also implement a proof of concept prototype that assumes the frame error rates of long guard interval (LGI) rates are identical to their short guard interval (SGI) counterparts and reduces the sampling frequency to show that relationships between rates can potentially be used to reduce the sampling overhead in rate adaptation algorithms. We call this algorithm "Minstrel HT without LGI Sampling" and include it in our trace-based evaluations because it is designed to reduce the sampling overhead. Comparing this algorithm to our algorithm (NeuRA) shows the benefits of using a neural network model for relationship estimations.

While Abedi et al. [1] focus on discovering and showing that relationships exist, they have not determined precisely what those relationship are, how to find them, or which rates are the best predictors of other rates. We build on this previous work by creating NeuRA, a rate adaptation algorithm which uses a neural network model to estimate the throughput of a set of non-sampled rates given the measured throughput of a set of sampled rates. Furthermore, we propose a technique for determining the best set of rates to sample and perform a comprehensive evaluation of NeuRA using a fairly large set of traces representing different real-world conditions. We also demonstrate that it is plausible to implement NeuRA in real-world WiFi devices by creating a prototype and showing it can increase throughput when compared to Minstrel HT.

2.3 Chapter Summary

In this chapter, we provide background information on rate adaptation and frame aggregation algorithms and why they are an active area of research. Then we describe the trace-based evaluation methodology enabled by T-SIMn. We then describe the use of neural networks in function estimation which is the reason we use them in our proposed rate adaptation algorithm (NeuRA).

We review some relevant studies that solve the problem of rate adaptation, examine the different approaches used, and explain why sampling-based rate adaptation is the most practical approach and why it is widely used in commercial devices. We also review several frame aggregation algorithms and describe the work demonstrating the existence of relationships between WiFi physical rates.

Chapter 3

Proposed Algorithms

In this chapter, we first introduce the neural rate adaptation (NeuRA) algorithm in Section 3.1. NeuRA uses a neural network model to predict the expected throughput of some of the physical rates based on the measured throughput of others in an attempt to reduce the sampling overhead and increase the throughput of sampling-based rate adaptation algorithms. We also propose a method for selecting the best subset of rates for sampling. Next, in Section 3.2, we present the offline statistically optimal joint rate adaptation and frame aggregation algorithm. This algorithm uses an oracle to examine the fate of future frames to make statistically optimal choices of the physical rate and the number of frames to aggregate at each point in time. This provides an upper bound on the throughput achievable by online rate adaptation and frame aggregation algorithms.

3.1 Neural Rate Adaptation (NeuRA)

As described in Section 2.2.3, relationships exist between WiFi physical rates. It is expected that these relationships can be used to estimate the reliability of one or more rates based on the reliability of others. If such estimations are possible, they can be used to improve the efficiency of sampling-based rate adaptation algorithms by reducing the number of rates being sampled.

In this section, we use a neural network model to learn the relationships between the rates and use them to estimate the reliability of the rates not being sampled. We also propose a technique for finding the best set of rates to sample which we combine with models obtained from training our neural networks to implement our algorithm called NeuRA. We evaluate the performance of NeuRA in Chapters 5 and 6.

We first present the architecture of our neural network model in Section 3.1.1. Then, we describe the procedure used to process raw WiFi traces to generate data sets suitable for training and evaluating the model in Section 3.1.2. Finally, we explain the approach used for feature selection in order to find the best subset of rates to sample in Section 5.1.

3.1.1 Proposed Model

We propose the use of a feed forward neural network model that takes the effective throughput of a subset of physical rates (called the sampling set) as input and estimates the expected throughput of the rates not in the sampling set. The architecture of this model is shown in Figure 3.1. Based on our experiments with different architectures (not included here), this architecture provided the best accuracy over the training set for the two configurations used for training and evaluation (one with 32 physical rates and the other with 64).

The inputs to the model are the effective throughputs (i.e., measured throughputs) of the rates in the sampling set which can be of any size from 1 up to the total number of supported physical rates. The 3 hidden layers and the output layer of the model use the Rectified Linear Unit (ReLU) activation function. Each hidden layer contains 64 neurons and the output layer contains a neuron for each supported rate. Note that the neural network estimates the throughput of all available rates, however, we are only interested in the expected throughput of the rates that are not sampled. Also, we use a 10% dropout [30] after each hidden layer to avoid converging to solutions that depend on specific connections between the neurons and to avoid over fitting the model to the training data.

We implement and train the model using the Keras library [9] and use it for feature selection and prediction. For training parameters, we use a batch size of 50, the mean squared error (MSE) loss function, and the Adam optimizer [23]. We train the model for 1,000 epochs after which the loss function stabilizes.

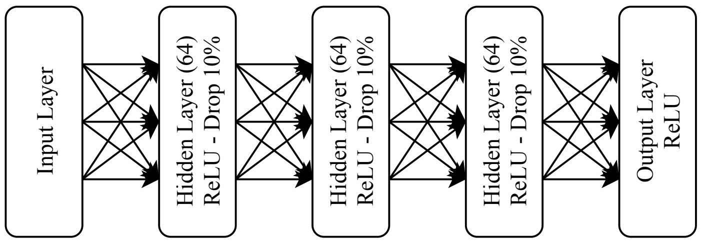


Figure 3.1: Structure of the proposed neural network model.

3.1.2 Data Set Preparation

We use recorded WiFi traces to train and evaluate the neural network model. During trace collection, an access point constantly transmits data packets to a client device using all available physical rates in a round-robin fashion. For each rate, the maximum frame aggregation length permitted for that rate is used. The block acknowledgment frames received from the client are recorded in the trace. As a result, traces include the required information to calculate the error rates for each subframe position within an aggregated frame within a time window. Section 4.1 describes the different scenarios used for trace collection. We use two disjoint sets for training and evaluation (a training set and a testing set) in order to perform a valid evaluation. We now explain the method used to process raw WiFi traces to generate data sets suitable for training and testing.

During our experiments (results are not shown here), we found that models that use effective throughput values obtain better results than those that use reliability (i.e., frame error rates). We believe that throughput values yield better results than frame error rates because neural networks try to minimize the average prediction error across all predicted values. However, an equal amount of error in frame error rates affects the expected throughput of different rates differently (e.g., a 3% difference in error rates results in larger differences in expected throughput for higher rates).

To create a data set, raw traces are processed so that the effective throughput of each rate is calculated using 1-second time windows. A 1-second window is used to include enough samples to obtain useful indications of error rates and the effective throughput for all physical rates while also capturing the channel state variability.

We obtain the effective throughput of a physical rate R in a time window $w(t)$ using the following equation inspired by PNOFA [2] with the maximum aggregation length (N). We add channel access time to the formula from PNOFA to obtain more accurate throughputs for scenarios with variable channel access times.

$$T_R(N, w(t)) = \frac{\rho \times \sum_{i=1}^{N}(1 - SFER_R(i, w(t)))}{\bar{C}(w(t)) + \tau_R(N)} \tag{3.1}$$

for $w(t)$ we use a 1-second time window containing time t. $T_R(N, w(t))$ represents the expected throughput (in Mbps) from aggregating N packets when using rate R in time window $w(t)$. The numerator is the expected number of successful bits and the denominator is the expected time to transmit the aggregated frame. ρ is the number of data bits in each subframe (MPDU) for which we use a typical value of 12,320 (1,540 bytes). $SFER_R(i, w(t))$ is the average subframe error rate of the i-th subframe in an aggregated frame sent using rate R in time window $w(t)$. This is calculated from the Block ACK frames in the trace. $\bar{C}(w(t))$ is the average channel access time in time window $w(t)$ and is calculated from the channel access information in the trace. $\tau_R(N)$ is the time required to transmit an aggregated frame of length N using rate R which is calculated based on protocol standards using the T-SIMn library.

Finally, we divide all throughput values by the maximum physical rate in our traces (300 Mbps) to scale all values to the normalized $[0, 1]$ range to make them suitable for training the neural network model. An example of the resulting data set which contains the effective throughputs calculated for all rates in 1-second time windows is shown in Table 3.1. We construct the training data set by concatenating the data sets extracted from multiple training traces and construct the testing data set by concatenating the data sets extracted from the testing traces.

Table 3.1: Format of the data set extracted from the traces.

Time (s)	$TPut_1$	$TPut_2$	...	$TPut_{64}$
0	0.015	0.039	...	0.0
1	0.016	0.035	...	0.0
...	...	...	...	...
2399	0.009	0.027	...	0.0

3.1.3 Feature Selection Technique

An important component of NeuRA is the technique used for feature selection (or feature elimination). Feature selection is necessary since we have to choose a subset of physical rates to be sampled which can be given to our model as input. Only the training data is used for feature selection. To explain our approach to feature selection, suppose we have N input features (rates) and we want to eliminate K features. The goal is that the remaining $N - K$ features should provide the most accurate throughputs estimations for all other rates. Additionally, the K eliminated rates should ideally reduce the sampling overhead as much as possible.

To accomplish this, first, we train the neural network using N input rates. Then we calculate the importance $\delta(R)$ of each input rate R as follows:

$$\delta(R) = \frac{\sum_{d \in T} |\frac{\partial L}{\partial R}(d)|}{\tau_R}$$

where T is the training data set, d is a row (1-second time window) in the training data, L is the loss function of the neural network, and τ_R is the time required to sample rate R using a single frame.

This equation sums the absolute value of the derivative of the neural network's loss function with respect to the effective throughput of input rate R (i.e., the impact of rate R on the neural network's loss function) over the entire training set and divides it by the time required to probe rate R. The intuition behind the equation is that we are interested in rates with the highest $\frac{\text{Estimation Power}}{\text{Probing Time}}$ value. This equation was empirically found to choose rates that result in better training set accuracy when compared to other scoring schemes we studied (not included here).

After assigning the importance values, we eliminate the K input rates with the lowest importance scores. Since we use the dropout technique, we expect the neural network to learn different dependencies that exist between the rates and not rely on a single dependency for each rate.

Using the described elimination scheme, we start with the initial sampling set containing all supported rates. Then we perform Recursive Feature Elimination (RFE) to reduce the size of the sampling set. After each step, we drop K input features (rates), retrain the network using the remaining input rates, and recalculate the feature importance scores. This is done repeatedly until all desired set sizes have been determined. Since using $K = 1$ for the whole process makes feature selection very slow, we set $K = 4$ for set sizes greater than 32, decrease it to $K = 2$ when we reach 32 input rates, and to $K = 1$ when we reach 12 input rates.

3.2 Offline Statistically Optimal Algorithm

For decades, new rate adaptation and frame aggregation algorithms have been invented and compared with each other. These evaluations depend on the techniques being used for comparison (which are prone to problems and errors [3]) and the environments in which they are evaluated. An important and unresolved question is how well these algorithms compare with one that makes optimal decisions. This is an important question because if they do well relative to an offline optimal algorithm under a wide variety of conditions, future research may not be required.

To that end, we now describe an algorithm for making statistically optimal decisions for both rate adaptation and frame aggregation aspects. This requires knowledge about the fate of all subframes for current and future aggregated frames and for all rates that could have been chosen.

Since this method uses information from the future and does not incur sampling overheads, it is superior to all online (practical) algorithms. The value in this algorithm is that its resulting throughput can be used as an upper bound for what could be achieved by a perfect combination of rate adaptation and frame aggregation algorithms. That is, it provides a basis for determining how close to or far from our algorithm and other widely-used algorithms perform when comparing the throughput.

Our algorithm first determines the best aggregation length for each physical rate and the effective throughput associated with that aggregation length. To do so, we use Equation 3.1 which is inspired by PNOFA [2] to calculate the expected throughput values $T_R(N, w(t))$ for a time window centered at time t and for all possible values of N (up to the maximum aggregation length for rate R). Defining $w(t)$ as a symmetric time window centered around time t allows us to estimate the expected throughput of each rate R with any aggregation length N at time t.

After calculating $T_R(N, w(t))$ for all values of N, we determine the value N that maximizes the expected throughput and store it along with its expected throughput as the best possible choice for rate R. The algorithm then chooses the rate with the maximum expected throughput along with the computed optimal aggregation length for time t.

While this algorithm may appear only to be of theoretical interest, we are able to implement this algorithm in the T-SIMn simulator. Because T-SIMn uses a trace-driven approach to evaluating frame aggregation and rate adaptation algorithms, this algorithm can be implemented by allowing the simulator to look ahead into the future to compute the statistically optimal decisions for these choices. This is possible because by design, the traces contain information about the fate of each subframe for all available rates and the statistically optimal solution can be computed for a given window size. The results of running this offline optimal solution on the traces and its comparison with practical algorithms are presented in Section 5.3.

3.3 Chapter Summary

In this chapter, we first present the required components to build a rate adaptation algorithm that uses predictions from a neural network model in order to reduce the sampling overhead. We present the architecture of the proposed neural network model, training parameters, the process used to extract training and testing datasets from raw WiFi traces, and a feature selection technique to select the best subset of WiFi physical rates to sample and use as the neural network model's input.

Next, we present the offline statistically optimal algorithm which makes optimal choices for the number of frames to aggregate and the physical rate to use at each point in time. We explain that the design of T-SIMn and its use of real-world traces makes it possible for us to implement this algorithm in T-SIMn and determine an upper bound for the throughput of a trace. Using T-SIMn, both of these algorithms are evaluated on a variety of real-world WiFi traces and compared to widely-used algorithms in Chapter 5.

Chapter 4

Evaluation Methodology

In this chapter, we describe two components that are necessary for performing a comprehensive and credible trace-based evaluation. In Section 4.1, we describe the process of collecting two diverse sets of real-world WiFi traces. One set is used for training the model and another set is used for evaluation purposes. In Section 4.2, we explain the technique of using synthetic traces to verify the expected behaviour of the rate adaptation and frame aggregation algorithms implemented in T-SIMn. We also describe three bugs that we have found and fixed using this verification technique.

4.1 Trace Collection

In this thesis, we use two disjoint sets of traces for training and evaluation (the training and testing sets) in order to perform a valid and credible evaluation of our model. The data set extracted from the training traces is used to train the models, select the best set of rates to sample and to determine the best set of parameters to use for NeuRA. The data set extracted from the raw testing traces is used to evaluate the accuracy of the models and the raw testing traces are later used to evaluate different algorithms using T-SIMn.

It is important to collect a diverse set of traces for both training and evaluation. Traces should cover stationary and mobile clients, congested and unoccupied WiFi channels, different environments (e.g., locations within office spaces), and different devices in case relationships are different for different scenarios. A diverse set of training traces helps the model to learn more generalizable relationships between rates while a diverse set of testing traces helps us to evaluate the models under a wide variety of conditions. Additionally, the testing set should also include traces that are different from all training scenarios (e.g., different client devices or different conditions) as well as some traces similar to the training scenarios. This way, we can evaluate the accuracy of the model on both previously seen and unseen scenarios.

A TP-Link TL-WDN4800 802.11n PCI-E wireless card which runs a modified version of the *ath9k* driver (included in T-SIMn) is used as the sending device. The receiving device does not require a modified driver so a variety of receiving devices are used for trace collection.

Most commonly used WiFi devices (including most recent phones and laptops) have two antennas. Also, even though the highest channel width in the 802.11n standard is 40 MHz, most devices will not use a 40 MHz channel width when using the 2.4 GHz carrier frequency due to channel congestion [19]. As a result, we use the two configurations shown in Table 4.1 for trace collection. For Configuration A a WiFi channel is shared with other active devices (which is typical in 2.4 GHz networks) and for Configuration B an unoccupied 5 GHz channel is used. Because the two configurations have different available rates and because different frequency spectrums may have different relationships between physical rates, we train two separate models (Model A and Model B).

Table 4.1: Two configurations used for trace collection.

Config	Spectrum	# Antennas	Channel Width	# Rates	Channel Condition
A	2.4 GHz	2	20 MHz	32	Congested
B	5 GHz	2	40 MHz	64	Unoccupied

We collect traces using several devices and several environments in which devices are expected to be used. Training scenarios are shown in Table 4.2. Six traces are collected for Model A (using Scenarios T1 to T6) and nine traces are collected for Model B (using Scenario T1 to T9). The length of each training trace is 40 minutes. Testing scenarios are shown in Table 4.3. These traces are 3 - 20 minutes long. Testing scenarios are chosen to cover most devices, client states, and environments.

Scenarios A1 to A7 are used to evaluate Model A and scenarios B1 to B7 are used for Model B. Scenarios A1 to A4 and B1 to B3 are similar to a training scenario while scenarios A5 to A7 and B4 to B7 are different from all training scenarios. We collect traces using four different receiving devices: a Samsung Galaxy Note 5 phone (SM-N920C), a TP-Link TL-WDN4200 USB adapter, a Huawei P20 phone (EML-L09C), and an Intel 8265 laptop WiFi card.

Two experiments labelled “extra movement” refer to moving and shaking the device while walking with the device in hand. These are included to evaluate rate adaptation algorithms in cases of high mobility and on previously unseen scenarios. In “toy train” scenarios, the device is mounted on a toy train which simulates movement with a constant speed. “Close AP” refers to a scenario with about 1 meter between the client (receiver) and the access point (sender) and “Distant AP” refers to a distance of about 10 meters.

Table 4.2: Scenarios for training traces.

Scenario	Device	State	Description
T1	SM-N920C	Stationary	Close AP
T2	SM-N920C	Stationary	Distant AP
T3	SM-N920C	Walking	Environment 1
T4	SM-N920C	Walking	Environment 2
T5	SM-N920C	Toy train	Slow Speed
T6	SM-N920C	Toy train	Fast Speed
T7	TL-WDN4200	Stationary	Close AP
T8	TL-WDN4200	Stationary	Distant AP
T9	TL-WDN4200	Walking	Environment 1

4.2 Using Synthetic Traces to Verify Algorithms

In this thesis, we use trace-based evaluation to compare NeuRA and the offline statistically optimal algorithm to several widely-used algorithms. In this section, we perform tests to ensure that algorithms implemented in T-SIMn behave as expected. Anomalies in the expected behaviour of the algorithms can be used to find potential bugs.

To do so, we create synthetic traces for which we can roughly predetermine the expected behaviour of different algorithms. In such traces, the fate of all subframes for all rates is known for a specified time interval and does not change during that interval. Additionally, the trace should work with all existing algorithms and be consistent with the assumptions made by different algorithms (e.g., a lower MCS index should have an error rate less than or equal to a higher MCS index, and subframes at the beginning of an aggregated frame should have lower error rates than subframes towards the end).

Table 4.3: Scenarios for testing traces.

Scenario	Device	State	Description	Length
A1	SM-N920C	Stationary	Distant AP	20 Min
A2	SM-N920C	Walking	Environment 1	20 Min
A3	SM-N920C	Walking	Environment 2	20 Min
A4	SM-N920C	Toy train	Fast Speed	20 Min
A5	TL-WDN4200	Walking	Extra Movement	5 Min
A6	EML-L09C	Stationary	Distant AP	3 Min
A7	Intel 8265	Walking	Environment 1 + 2	5 Min
B1	SM-N920C	Stationary	Close AP	20 Min
B2	SM-N920C	Walking	Environment 2	20 Min
B3	SM-N920C	Toy train	Slow Speed	20 Min
B4	SM-N920C	Walking	Extra Movement	10 Min
B5	TL-WDN4200	Walking	Extra Movement	5 Min
B6	EML-L09C	Walking	Environment 1 + 2	3 Min
B7	Intel 8265	Walking	Environment 1 + 2	5 Min

We have run all rate adaptation and frame aggregation algorithms in T-SIMn using this trace to validate if they perform as expected in different time intervals. As a result, we have found and patched bugs in three of these algorithms. After patching these bugs, all algorithms seem to behave as expected on our synthetic traces. Note that this method helps us to find bugs by spotting unexpected behaviours but it does not prove the correctness of algorithms. Section 4.2.1 describes the process of creating synthetic traces and Section 4.2.2 describes the bugs we have found and patched in the algorithms.

4.2.1 Trace Creation

We generate the synthetic trace so that during a time interval all rates below a threshold always succeed and all higher rates always fail. Similarly, we choose a maximum aggregation length so that all subframe with an index less than or equal to that length succeed and all subframes past that length fail during the time interval. By doing so, we can understand and predict the rate and the aggregation length to which a good algorithm will eventually converge.

We use a Python script to generate the trace information based on the desired best rate and frame length. Similar to the traces collected from real-world experiments, this trace contains information about all physical rates at all times. To simplify the trace and because it is not relevant to examining the correctness of the algorithm implementations, we assume that the channel access time is zero (i.e., there is no interference from other WiFi and non-WiFi devices).

In order to create traces that match the expectations of different algorithms, we use the following relationships between rates:

- Increasing the MCS index, increases the error rate because a denser modulation or a less-redundant coding scheme is used [15].
- Increasing the number of spatial streams, increases the error rate [24].
- Increasing the channel width, increases the error rate [24].
- Short guard interval (SGI) rates have a higher error rate than their long guard interval (LGI) counterparts as they use a less conservative inter-frame gap [15].

The trace is created using different patterns for multiple 1-minute intervals. During that interval, the behaviour of each rate is unchanged. For each 1-minute interval, a random physical rate is chosen as the maximum rate that successfully transmits the data (R_{max}) during that interval. Based on the relationships described above, all rates that are supposed to have a higher error rate than R_{max} fail during that interval and all rates that are supposed to have a lower error rate than R_{max} successfully transmit the data during that interval. To be more specific, each physical rate (R) that satisfies all of the following conditions is successful during the interval while all other rates fail.

- $Num\_Spatial\_Stream(R) <= Num\_Spatial\_Stream(R_{max})$
- $MCS\_Index(R) <= MCS\_Index(R_{max})$
- $Channel\_Width(R) <= Channel\_Width(R_{max})$
- $Guard\_Interval(R) >= Guard\_Interval(R_{max})$

Additionally, a aggregation length limit (LEN_{limit}) is chosen randomly from a set of possible values ($\{1, 4, 16, 32\}$) for each time interval. For all rates, all subframes in an aggregated frame that have an index greater than LEN_{limit} fail all the time. Every rate also has a maximum aggregation length that is determined by the number of frames that can be sent in a specified time limit (4 ms in T-SIMn) which we refer to as LEN_{max}. As a result, during each 1-minute interval, the optimal physical rates is R_{max} and the optimal frame aggregation length for that rate is $LEN_{best}(R_{max}) = min(LEN_{max}(R_{max}), LEN_{limit})$.

Note that these synthetic traces represent extreme conditions that are never seen in the real world. For example, using a wider channel width or more spatial streams usually increases the throughput because their higher bitrate usually outweighs their higher error rates. As a result, many algorithms perform poorly on some of the time intervals. However, even though these traces do not represent real-world conditions, they help us to test several properties of rate adaptation and frame aggregation algorithms, because we know the behaviour of the trace at different time intervals.

We have generated several different synthetic traces and used them to validate the behaviour of different algorithms in T-SIMn. Here, we present one of these traces that triggers all three bugs it helped us to find and fix. Table 4.4 shows the maximum physical rate that succeeds (R_{max}), the maximum possible aggregation length for R_{max} which we call $LEN_{max}(R_{max})$, the aggregation length limit (LEN_{limit}) for each 1-minute interval in this trace, and the best aggregation length for R_{max} which is $LEN_{best}(R_{max}) = min(LEN_{max}(R_{max}), LEN_{limit})$. The optimal throughput for each time interval is achieved by aggregating $LEN_{best}(R_{max})$ frames using rate R_{max}. The throughput of this optimal configuration for each interval is also shown in the table.

Time intervals that have a lower aggregation length limit (LEN_{limit}) than $LEN_{max}(R_{max})$ are expected to maximize the throughput improvements caused by non-trivial frame aggregation algorithms such as STRALE and PNOFA and algorithms that aggregate as many frames as possible are expected to perform poorly during those intervals. We now describe how we use this trace to find and fix bugs in some of the T-SIMn algorithms.

4.2.2 Finding and Fixing Bugs

Using the synthetic traces, we found a bug in the STRALE implementation, a bug in Minstrel HT w/o LGI Sampling presented by Abedi et al. [1], and a bug in our porting of the Intel iwl-mvm-rs algorithm from the work of Grünblatt et al. [16] to T-SIMn. We now describe how we found and fixed these bugs.

Table 4.4: Maximum rate and aggregation length for each time interval in the synthetic trace.

Interval (s)	Max Physical Rate (R_{max})	Max Len For R_{max}	Aggr Limit (LEN_{limit})	Best Len For R_{max}	Max Tput (Mbps)
[0, 60)	2 SS, MCS 5, 20 MHz, LGI	32	4	4	64.1
[60, 120)	2 SS, MCS 1, 20 MHz, LGI	8	1	1	17.0
[120, 180)	2 SS, MCS 6, 40 MHz, LGI	32	32	32	201.1
[180, 240)	1 SS, MCS 4, 40 MHz, SGI	29	4	4	58.3
[240, 300)	2 SS, MCS 2, 20 MHz, SGI	14	1	1	23.4
[300, 360)	2 SS, MCS 6, 20 MHz, LGI	32	4	4	69.4
[360, 420)	2 SS, MCS 6, 20 MHz, SGI	32	32	32	114.4
[420, 480)	2 SS, MCS 0, 20 MHz, SGI	4	16	4	12.4
[480, 540)	1 SS, MCS 6, 20 MHz, LGI	19	32	19	52.3
[540, 600)	2 SS, MCS 7, 40 MHz, LGI	32	1	1	44.9
[600, 660)	2 SS, MCS 7, 20 MHz, SGI	32	16	16	116.3
[660, 720)	1 SS, MCS 5, 40 MHz, SGI	32	4	4	71.0
[720, 780)	1 SS, MCS 6, 20 MHz, SGI	21	1	1	29.4
[780, 840)	1 SS, MCS 5, 20 MHz, SGI	19	32	19	51.7
[840, 900)	2 SS, MCS 4, 40 MHz, SGI	32	4	4	89.4

STRALE

The bug in STRALE was found by comparing the throughput of STRALE with the throughput of Minstrel HT. Minstrel HT aggregates as many frames as possible while STRALE reduces the aggregation length when shorter aggregated frames result in higher throughput. In this synthetic trace, the best aggregation length is considerably lower than the aggregation length chosen by Minstrel HT during many time intervals. As a result, STRALE is expected to perform better than Minstrel HT during those time intervals. Note that in real-world experiments, STRALE does not always perform better than Minstrel HT in scenarios with a relatively high channel access time (high interference). However, as previously explained, there are no channel access delays in the synthetic trace for validation.

Counter to our expectation, we observed (as shown in Figure 4.1) that STRALE was providing throughput lower than Minstrel HT during the time intervals [360, 420) and [780, 840). In these two specific intervals, STRALE was supposed to aggregate as many frames as possible but it was choosing an aggregation length much lower than expected.

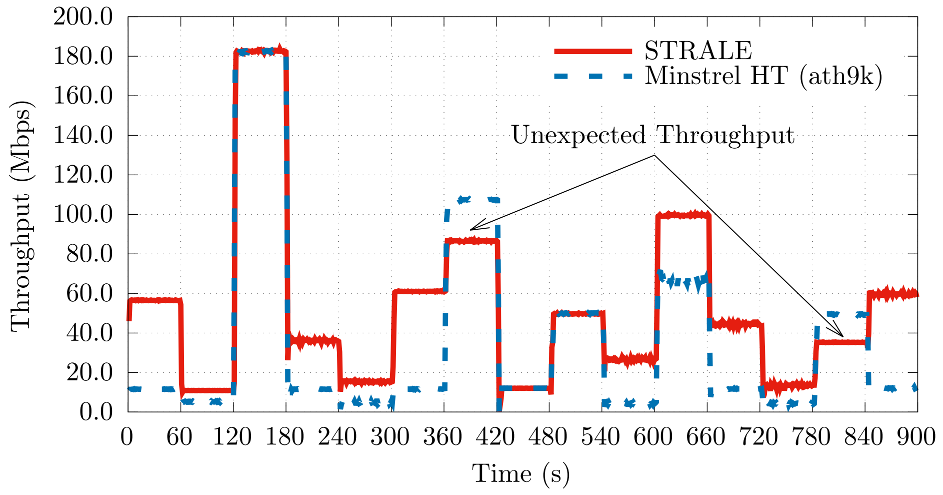


Figure 4.1: STRALE versus Minstrel HT before STRALE bug fix.

After investigating the problem, we found that the block ACK feedback generated by T-SIMn is written to indices 0 to $len - 1$ of an array in one function, while the STRALE code was reading from indices 1 to len of the same array in another function. This was causing STRALE to obtain incorrect information about which subframes are acknowledged and resulted in incorrect decisions regarding the number of frames to aggregate.

Listing 4.1 shows the code snippets containing this bug and Listing 4.2 shows the same code snippets after fixing the bug. Figure 4.2 shows STRALE throughput after fixing this bug. The lower throughput of STRALE compared to Minstrel HT is not visible any more after the fix.

Minstrel HT without LGI Sampling

The Minstrel HT without LGI Sampling algorithm is used by Abedi et al. [1] as a proof of concept to show the benefits of reducing the sampling overhead of Minstrel HT. As described in Section 2.2.3, this algorithm reduces the sampling frequency by only sampling SGI (Short Guard Interval) rates and approximating the error rate of LGI (Long Guard Interval) rates to be equal to their SGI counterparts. It is one of the rate adaptation algorithms that we compare against in Chapter 5.

Listing 4.1: STRALE code snippets containing the bug.

```
// STRALE reading the subframe error rate array (line with bug)
for (i = 1; i < aggr_num + 1; i++) {
    thpt += (1 − an−>sfer[tidno][i]) ∗ thpt_div;
}
...
// Another function filling the subframe error rate array
for (int i = 0; i < feedback.fa.num_mpdus(); i++) {
    an_.sfer[0][i] = 1 − feedback.acks[i];
}
```

Listing 4.2: STRALE code snippets after fixing the bug.

```
// STRALE reading the subframe error rate array (bug fixed)
for (i = 0; i < aggr_num; i++) {
    thpt += (1 − an−>sfer[tidno][i]) ∗ thpt_div;
}
...
// Another function filling the subframe error rate array
for (int i = 0; i < feedback.fa.num_mpdus(); i++) {
    an_.sfer[0][i] = 1 − feedback.acks[i];
}
```

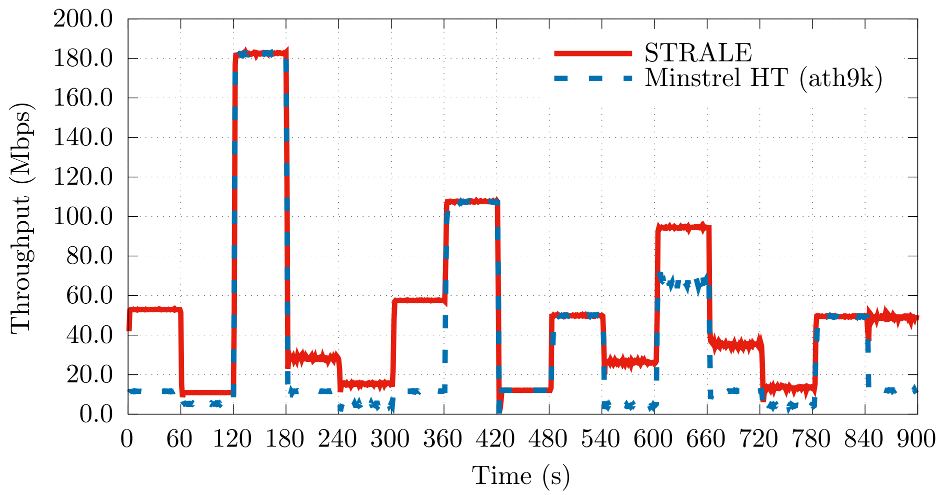


Figure 4.2: STRALE versus Minstrel HT after STRALE bug fix.

Although an SGI rate and its LGI counterpart may have similar error rates in some real-world scenarios, this is not the case in our synthetic trace. In this trace, by design, all SGI rates fail during the time intervals during which the maximum physical rate (R_{max}) is an LGI rate. Because this algorithm only samples SGI rates, it is expected to provide near-zero throughput during these intervals. On the other hand, during the time intervals during which R_{max} is an SGI rate, the LGI rates have an error rate similar to their SGI counterparts. As a result, this algorithm is expected to provide throughput similar to or higher than Minstrel HT during these intervals.

When running this algorithm on the synthetic trace, we observed that it provides unusually low throughput during the time intervals [360, 420) and [600, 660) during which the maximum rate is an SGI rate. This is shown in Figure 4.3. After investigation, we found that sometimes sampling comes to a complete halt for more than a minute when using this algorithm. We found that the problem was with a part of Minstrel HT code that was modified to reduce the amount of sampling. Minstrel HT uses a variable called *sample_wait* to pause sampling until a specified number of frames are sent. In this algorithm, *sample_wait* was unconditionally being multiplied by a factor of two when receiving any feedback from the simulator. In reality, *sample_wait* should be multiplied by that factor only when *sample_wait* is reset by Minstrel HT.

We modified the code to fix this bug. Listing 4.3 shows a code snippet from Minstrel HT w/o LGI Sampling that contains this bug and Listing 4.4 shows the same code snippet after fixing the bug.

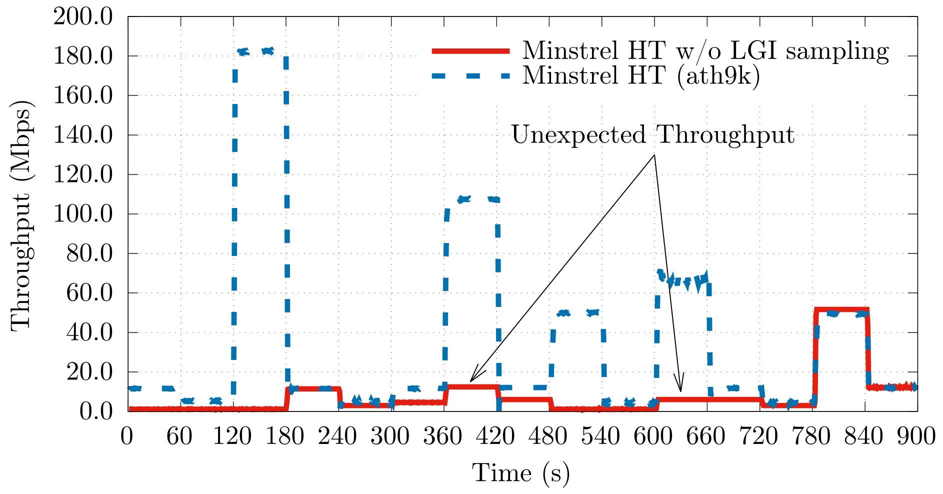


Figure 4.3: Minstrel HT w/o LGI sampling versus Minstrel HT before the bug fix.

Listing 4.3: The code snippet containing the bug found in Minstrel HT w/o LGI Sampling.

```
// Minstrel HT resetting sample_wait
if (!mi->sample_wait && !mi->sample_tries && mi->sample_count > 0) {
    mi->sample_wait = 16 + 2 * MINSTREL_TRUNC(mi->avg_ampdu_len);
    mi->sample_tries = 1;
    mi->sample_count--;
}
// Minstrel HT w/o LGI Sampling multiplying sample_wait
// Bug: The following block should be inside the previous if block
if (uw_minstrel_stats_mode != MINSTREL_DEFAULT) {
    mi->sample_wait *= sampleReduceFactor;
}
```

Listing 4.4: The code snippet in Minstrel HT w/o LGI Sampling after fixing the bug.

```
// Minstrel HT resetting sample_wait
if (!mi->sample_wait && !mi->sample_tries && mi->sample_count > 0) {
    mi->sample_wait = 16 + 2 * MINSTREL_TRUNC(mi->avg_ampdu_len);
    mi->sample_tries = 1;
    mi->sample_count--;
    // Minstrel HT w/o LGI Sampling multiplying sample_wait
    // Bug fixed by moving the following block
    if (uw_minstrel_stats_mode != MINSTREL_DEFAULT){
        mi->sample_wait *= sampleReduceFactor;
    }
}
```

Figure 4.4 shows the throughput of this algorithm after fixing the bug. As can be seen, the throughput of this algorithm increases significantly in the previously mentioned time intervals after the fix.

Intel iwl-mvm-rs

Intel iwl-mvm-rs always uses the highest available channel width which is 40 MHz in our synthetic trace. Even though using the highest available channel width usually provides the highest throughput in the real-world, this is not the case in our synthetic trace. During the time intervals during which the maximum physical rate (R_{max}) is a 20 MHz rate, all 40 MHz rates fail by design. So we expect Intel iwl-mvm-rs to get zero throughput during these intervals. On the other hand, during the time intervals during which R_{max} is a

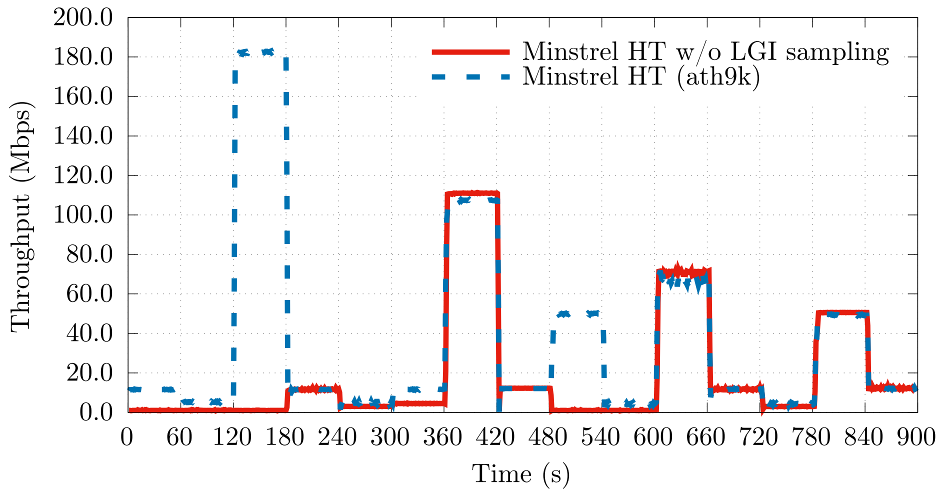


Figure 4.4: Minstrel HT w/o LGI sampling versus Minstrel HT after the bug fix.

40 MHz rate, Intel iwl-mvm-rs is expected to successfully converge to this rate.

When we ran this algorithm on the synthetic trace, we saw that it provides non-zero throughput during the first two time intervals [0, 60) and [60, 120), even though only 20 MHz rates work during those intervals. After investigating, we found that even though the channel width is set to the maximum available width after every column scaling operation, when the algorithm first starts, it uses a 20 MHz channel width before the first column scaling operation. This was clearly not the intended behaviour of the algorithm and the issue was introduced when we ported the algorithm to T-SIMn. We fixed the issue by setting the channel width to the maximum available channel width when the algorithm starts. Figure 4.5 shows the difference in throughput in the first two 1-minute time intervals before and after fixing the bug.

4.2.3 The Value of Synthetic Traces

By creating synthetic traces for which the expected behaviour of many algorithms can be predicted, we managed to find and fix three bugs in algorithms implemented in the T-SIMn trace-based evaluator. Finding and fixing these bugs allow us to perform credible evaluations to compare the performance of different rate adaptation and frame aggregation algorithms which we present in Chapter 5. We believe that performing these tests is necessary to find potential implementation bugs before performing evaluations.

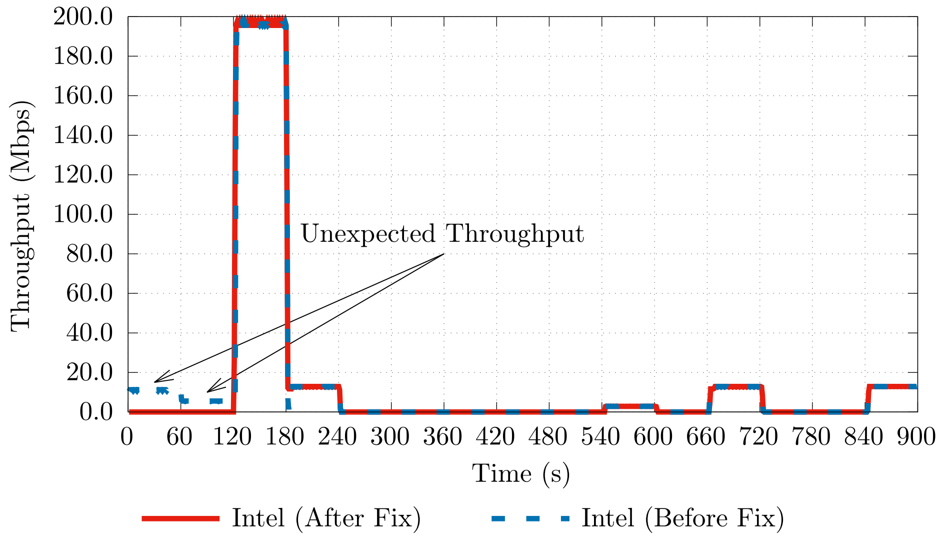


Figure 4.5: Behaviour of Intel iwl-mvm-rs port before and after the bug fix.

4.3 Chapter Summary

In this chapter, we describe the two widely-used WiFi configurations that we use for trace collection, the use of two separate sets of traces for each configuration (one for training and one for evaluation), and different real-world scenarios and different devices from which we have collected the traces.

Next, we describe the technique of creating synthetic traces and using them to verify the implementation of the algorithms we use in our trace-based evaluation. We explain the process of finding and fixing three bugs that were identified during this process and conclude that this kind of validation is necessary to perform credible evaluations.

Chapter 5

Evaluation

In this section, we evaluate the accuracy of the trained models and the approach we use for selecting the best sampling rates. Then we compare the performance of NeuRA and the statistically optimal algorithm to a variety of state-of-the-art sampling algorithms using trace-based evaluation.

5.1 Model Evaluation

After the feature selection phase, we train a neural network model for varying sizes of sampling sets between 2 rates and the total number of supported rates. To evaluate the effectiveness of the neural network's ability to provide accurate estimations, we first examine the Mean Absolute Error (MAE) for the neural network's predictions compared with the measured values from the traces.

Mean Absolute Error (MAE): MAE is computed as follows. For each row of data in the data set (1-second time window), the throughput of sampling rates is fed to the neural network model to predict the throughput of all rates. Then the absolute difference between the predicted throughputs and actual throughputs is calculated. MAE represents the average of these absolute differences over all rates and all time windows. It is also multiplied by 300 Mbps to scale the value from the $[0, 1]$ range to a Mbps throughput value.

Figure 5.1 shows the MAE over the training and the testing set for models with different numbers of rates. Note that Model A predicts the throughput of 32 rates while Model B predicts the throughput of 64 rates. Therefore, the performance of the two models on a specific size of sampling set cannot be compared directly. The small difference of MAE between the training and the testing sets shows that the model is not overfit to the training data and is general enough to predict cases it has not seen in the training set. We note that the MAE fluctuates between 2 Mbps and 4 Mbps for reasonable sizes of the sampling set (i.e., when more than a quarter of rates are used). This shows that if the size of the sampling set is reasonable, the neural network model can effectively predict the throughput of most of the non-sampling rates most of the time.

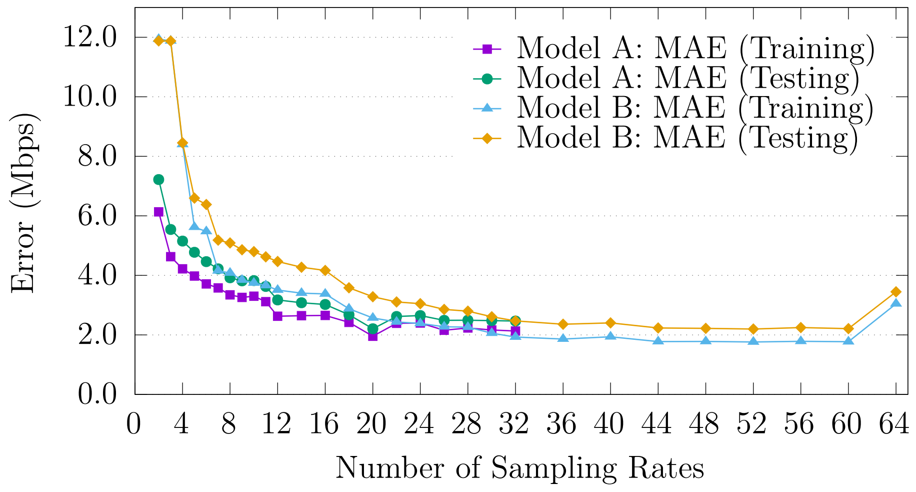


Figure 5.1: Mean Absolute Error (MAE) for models with different numbers of rates.

One of the parameters the feature selection method takes into account is the time required to sample the rates. Therefore, an interesting metric to observe is the sampling time.

Sampling Time: The total time required to probe every rate in the sampling set once. To obtain this value for a sampling set, the time required to sample an A-MPDU of size 1 is calculated for each rate. To do so, we use the T-SIMn library which includes highly accurate timing for packet transmission and receiving acknowledgements based on protocol standards. Then, we sum up the sampling time of all rates to calculate this metric.

Figure 5.2 shows the sampling time for sampling sets of different sizes. When the sampling set size is large and is decreased, sampling time decreases more rapidly than for smaller sets as there are more slower rates to eliminate in the sampling set. The rate at which the sampling time decreases slows down as the size of sampling set gets smaller. For example, the sampling time decreases more when reducing the size of the sampling set from 64 to 60 than when reducing it from 60 to 56. Additionally, there are small jumps at some points in the graph (e.g., when reducing the sampling set size from 9 to 8 in Model A) which suggests some slower rates are eliminated later than some faster rates as they were more important for the predictions.

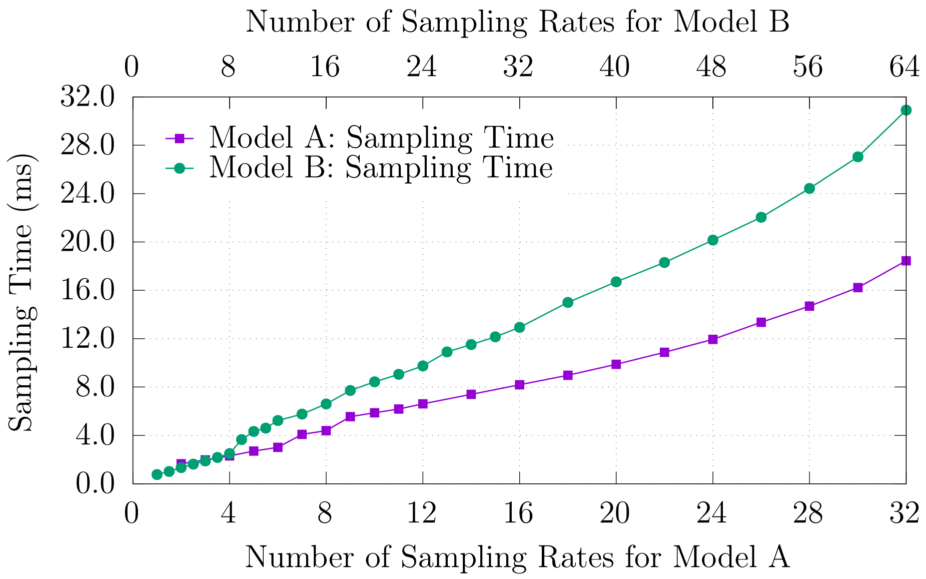


Figure 5.2: Comparison of the sampling time for different sampling set sizes.

A low MAE value for a rate estimation model does not necessarily translate to a small loss in the throughput when performing rate adaptation using that model. The average relative error when comparing two rates can be as high as $2 \times MAE$. Also, if a single important rate is predicted with a high error at some points, it may result in a poor choice of rates and the throughput may drop significantly. To evaluate the effectiveness of the neural network model for use in a rate adaptation algorithm, we define some rate adaptation metrics below.

RA (Rate Adaptation) Error: For each row of data in the data set (1-second time window), we choose a rate using the model. This is done by choosing the highest throughput from the measured sampling rates and the other rates estimated by the model. Then the resulting throughput of the chosen rate is calculated by looking at its actual throughput in the data set at that point of time (T_{model}) which is compared to the maximum throughput among all rates at that point of time (T_{max}) (which is available in the trace). The **absolute RA error** is then calculated by averaging the $T_{max} - T_{model}$ difference over all time windows and the **relative RA error** is calculated by averaging the $\frac{T_{max}-T_{model}}{T_{max}}$ relative differences over all rows (time windows).

Optimal Selection: After determining the rates selected by the model at each point in time, we then calculate the percentage of rows (time windows) that the model's choice results in a throughput within 5% of the optimal throughput. (i.e., $T_{model} >= 0.95 \times T_{max}$). We call this metric optimal selection.

Figure 5.3 shows absolute RA error, Figure 5.4 shows relative RA error, and Figure 5.5 shows optimal selection for models with different numbers of sampling rates. These metrics are calculated on the testing set to evaluate the model's performance on cases it has not seen before. Two different x-axes are shown for Model A and Model B as they have different numbers of supported rates. As can be seen, the model does a good job of choosing rates when the size of the sampling set is large enough (e.g., contains at least half of the supported rates). However, as we lower the number of sampling rates, the error increases and the optimal selection decreases considerably.

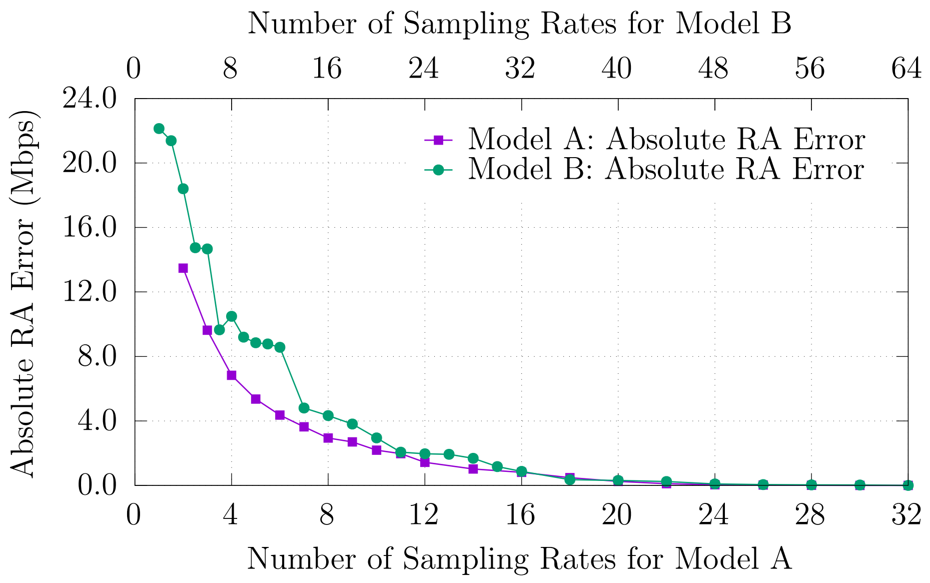


Figure 5.3: Comparison of absolute RA error for different sampling set sizes.

5.2 Feature Selection Evaluation

In this section, we evaluate our recursive feature elimination (RFE) approach to choosing the best rates for sampling. To do so, we consider the sampling sets of equal size chosen by our approach and other approaches to selecting rates and train separate models for each

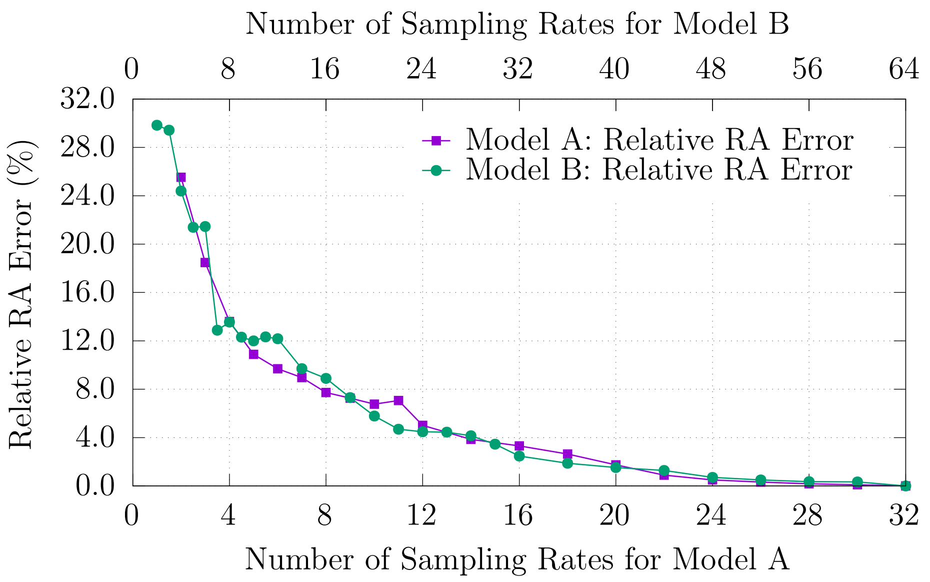


Figure 5.4: Comparison of relative RA error for different sampling set sizes.

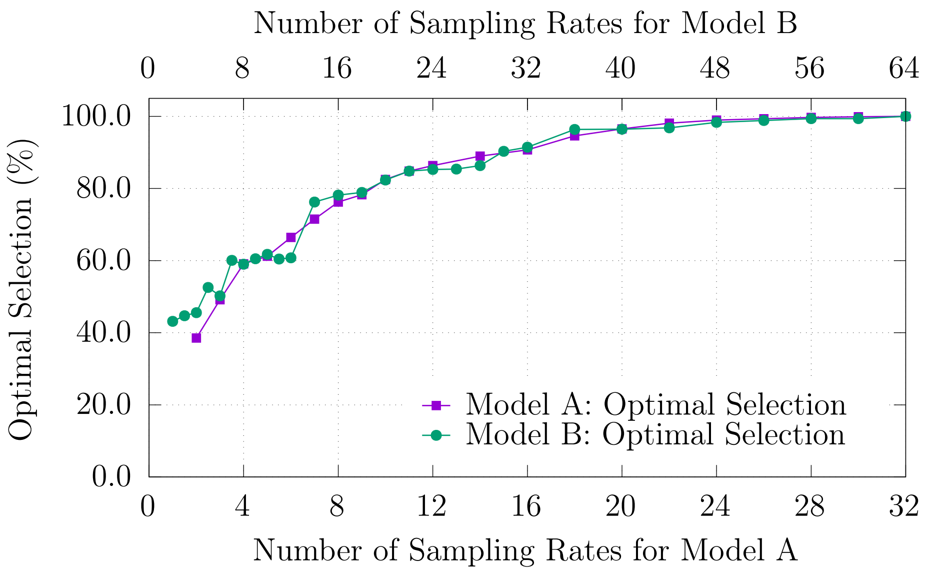


Figure 5.5: Comparison of optimal selection for different sampling set sizes.

of these sampling sets. Then we compare the performance of the trained models on the testing set to see which sampling set provides the most accurate estimations.

We compare the RFE approach to two other approaches. (1) uses only SGI (short guard interval) rates, chosen because this was the sampling set used by Abedi et al. [1]. (2) uses a random set of sampling rates picked from the supported rates. To compare with the SGI method, the size of the sampling set for our RFE and models is set to half of the supported rates (i.e, 16 rates for Model A and 32 rates for Model B).

Table 5.1 shows the previously defined metrics for the different models trained with these three approaches to rate selection on the testing set. Sampling time is the total time required to probe every rate in the sampling set once. As can be seen, rates selected by RFE are both faster to sample and result in significantly lower errors compared to other methods for both Model A and Model B.

Table 5.1: Comparison of models using different feature selection techniques.

Method	Sampling Time	MAE	Relative RA Error	Optimal Selection
SGI (A)	8.9 ms	4.1 Mbps	7.7%	70.7%
Random (A)	9.5 ms	4.0 Mbps	7.4%	66.6%
RFE (A)	8.2 ms	3.0 Mbps	3.3%	90.7%
SGI (B)	15.0 ms	3.9 Mbps	3.1%	84.7%
Random (B)	16.0 ms	3.9 Mbps	5.1%	76.8%
RFE (B)	12.9 ms	2.5 Mbps	2.5%	91.4%

5.3 Trace-Based Evaluation

As discussed previously, a trace-based evaluation is the most sound way to compare different rate adaptation and frame aggregation algorithms because different algorithms are all exposed to the same channel conditions. We use T-SIMn because it has previously been shown to be extremely realistic and highly accurate [4]. We have implemented NeuRA and other algorithms using the rate adaptation algorithm class in T-SIMn. T-SIMn is written in C++ and we have added a set of python bindings using pybind11 [20] to enable using the Keras models with T-SIMn. The NeuRA implementation uses 3 main parameters described below.

- **Number of sampling rates (N):** The size of the sampling set used in the the neural network model.

- **Single Rate Sampling Probability (F):** This is the probability of sampling an individual rate. Therefore, the probability of sampling is $N \times F$.

- **Frame aggregation algorithm (FAA):** options are: “Default”, “MoFA”, and “PNOFA”. “Default” is the frame aggregation algorithm from the *ath9k* driver which aggregates as many frames as possible.

When NeuRA needs to select a rate, with a probability of $N \times F$ it chooses the next sampling rate and sends a probe A-MPDU of size 1. Otherwise (with a probability of $1 - N \times F$), it chooses the rate predicted to result in the highest throughput. Every 1 ms (of simulated time), the best rate for transmission is updated based on the sampling results and estimations from the neural network model. This is done by calculating the effective throughput of the sampling rates based on their measured frame error rates. The throughput of the sampling rates is then fed to the neural network to estimate the throughput of the other rates.

The possible values of N are between 2 and the total number of supported rates. We tested F values between 0.001 and 0.01 using 0.001 increments. We have run simulations with all possible combinations of these three parameters for all training traces and have chosen the parameters that achieve the highest throughput on average. The chosen parameters are listed in Table 5.2 for the two models and Table 5.3 shows the set of sampling rates used with these parameters. These parameters are used for all NeuRA results. We see that the best sampling set size (best value of N) obtains a relative rate adaptation error less than 2% (in Figure 5.4) and and an optimal selection value greater than 95% (in Figure 5.5).

Table 5.2: Best parameters for NeuRA (found empirically).

Model	Best N	Best F	Best FA
A	20	0.004	Default
B	36	0.004	PNOFA

We now compare the performance of the following rate adaptation/frame aggregation algorithm combinations on the testing traces. These were chosen from the best algorithms

Table 5.3: Sampling rates used in best NeuRA configurations.

Model	Group	MCS Indices
A (2.4 GHz)	20 MHz - LGI	4, 5, 6, 7, 10, 11, 12, 13, 14, 15
	20 MHz - SGI	4, 5, 6, 7, 10, 11, 12, 13, 14, 15
B (5 GHz)	20 MHz - LGI	7, 12, 13, 14, 15
	20 MHz - SGI	6, 7, 11, 12, 13, 14, 15
	40 MHz - LGI	3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15
	40 MHz - SGI	2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15

that are either available in T-SIMn or that we could find an implementation for. While conducting our experiments, we observed that STRALE always performs better than MoFA and MoFA is therefore not shown in our result to reduce clutter.

1. **Minstrel HT (ath9k):** This is the default rate adaptation algorithm in the ath9k driver. It uses the default frame aggregation algorithm to aggregate as many frames as possible.

2. **Minstrel HT w/o LGI sampling:** The same as (1), except the sampling frequency is decreased by half by only sampling SGI rates. The error rate of LGI rates is assumed to be equal to their SGI counterparts. This algorithm was included in Abedi et al. [1].

3. **Minstrel HT + PNOFA:** The same as (1), but it uses the PNOFA frame aggregation algorithm.

4. **STRALE:** This is using the STRALE holistic approach to frame aggregation and rate adaptation.

5. **Minstrel HT + OSOFA:** The same as (1), but the Offline Statistically Optimal Frame Aggregation (OSOFA) length for the current rate is chosen by examining the past and future information in the trace (as is done in (8)). It provides an upper bound on the how much the throughput of Minstrel HT can be improved with better frame aggregation.

6. **Intel iwl-mvm-rs:** The rate adaptation algorithm used in recent Intel WiFi devices. Intel WiFi cards are a popular choice for laptops and desktop computers. We have

used the code from Grünblatt et al. [16] and ported it to T-SIMn. The default frame aggregation algorithm is used since the frame aggregation mechanism of Intel devices is not known.

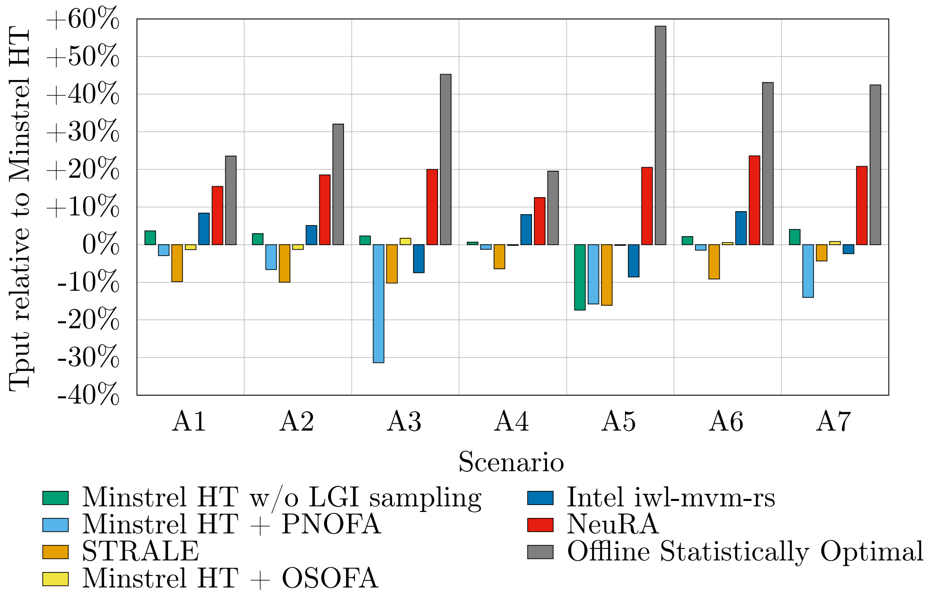
7. **NeuRA:** NeuRA with parameters from Table 5.2. The rate is changed every 1 ms based on the predictions of the neural network model. This short period ensures that predictions are frequent enough to evaluate the potential of our model.

8. **Offline Statistically Optimal:** As described in Section 3.2, the past and future information in the trace is used to make statistically optimal choices for both the rate and the aggregation length for each transmission.

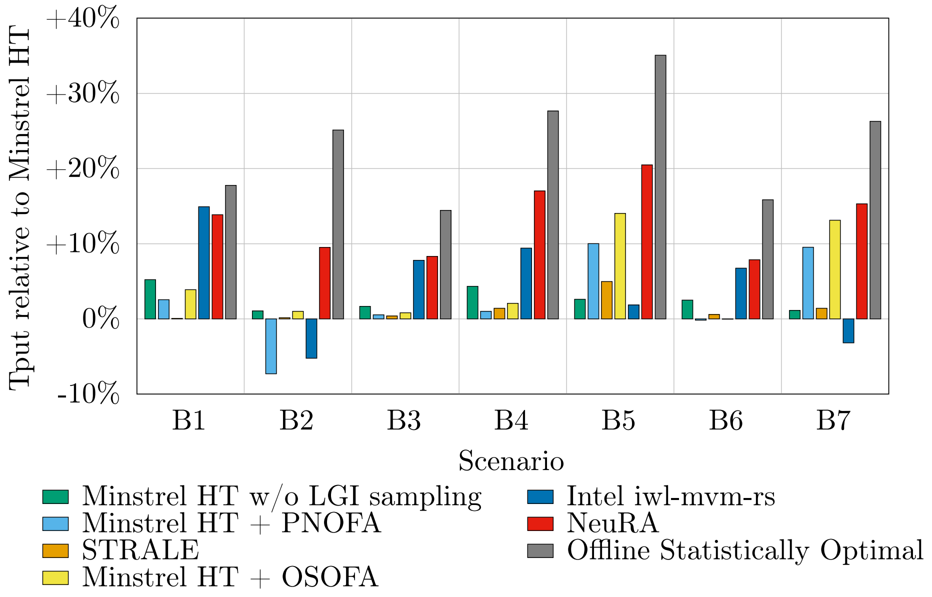
The algorithms (5) and (8) are not practical combinations as they require information about the future. However, they are included because they provide useful upper bounds on throughput that could be obtained by improving rate adaptation and frame aggregation algorithms.

The results of our trace-based evaluations are shown in Figure 5.6. All throughputs are shown relative to that of Minstrel HT (1). Table 5.4 presents the average throughput obtained by Minstrel HT (1) on each of the testing traces. These traces cover a wide spectrum of link capacities between the sender and receiver. As can be seen in these graphs, NeuRA consistently improves the throughput on scenarios that are similar (but not identical) to those in the training set and those that are different from those in the training set (unseen scenarios). Interestingly, some of the biggest improvements are obtained in the previously unseen scenarios (probably due to their higher mobility). Note that in these comparisons, for each scenario, all algorithms are subject to identical conditions from a single WiFi trace. As a result, the differences are statistically significant.

To provide better insights into differences between the behaviours of Minstrel HT, NeuRA, and the offline statistically optimal algorithm, we show the throughput of these algorithms over time for two different scenarios. Figure 5.7 shows the throughputs for Scenario A7 (a mobile scenario) and Figure 5.8 shows the throughputs for Scenario B1 (a stationary scenario). For clarity, in these graphs, the throughput is plotted using averages over 5-second time windows. As can be seen in Figure 5.7, throughput varies significantly over the course of the experiment. Throughput is higher when the device is closer to the access point and lower when it is farther away. In this experiment, you can see the ranking of each algorithm (in terms of throughput) is fairly consistent throughout the experiment. In Figure 5.8, the throughout of all algorithms is much higher overall than in Figure 5.7. It is also less variable because the client device is stationary. In this experiment, the difference



(a) Trace-based evaluation results for Configuration A traces.



(b) Trace-based evaluation results for Configuration B traces.

Figure 5.6: Trace-based evaluation results relative to Minstrel HT.

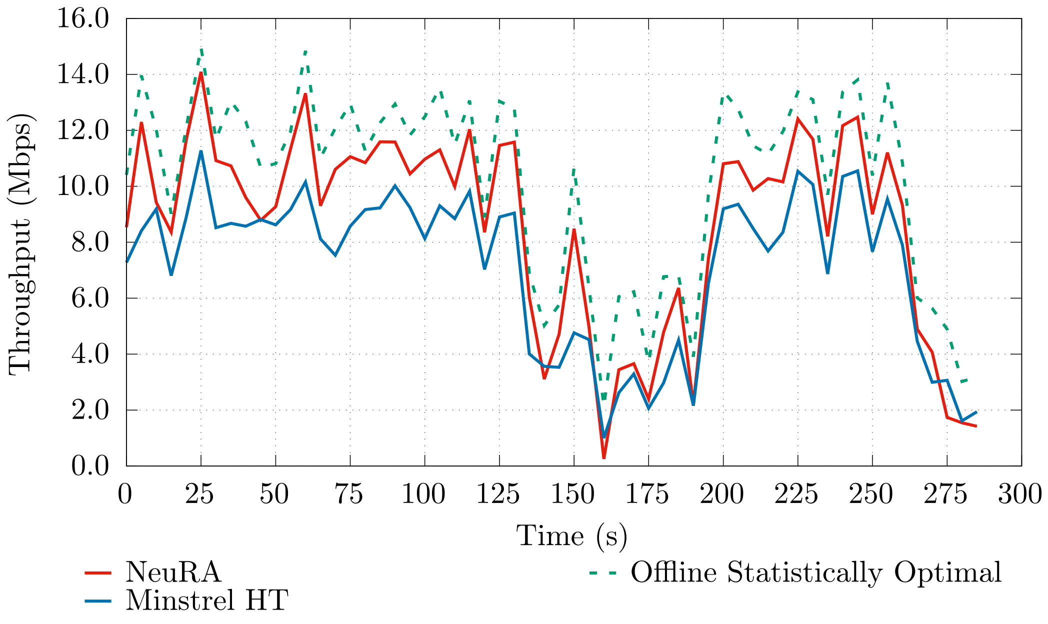


Figure 5.7: Comparison of the throughput over time for Scenario A7 (mobile).

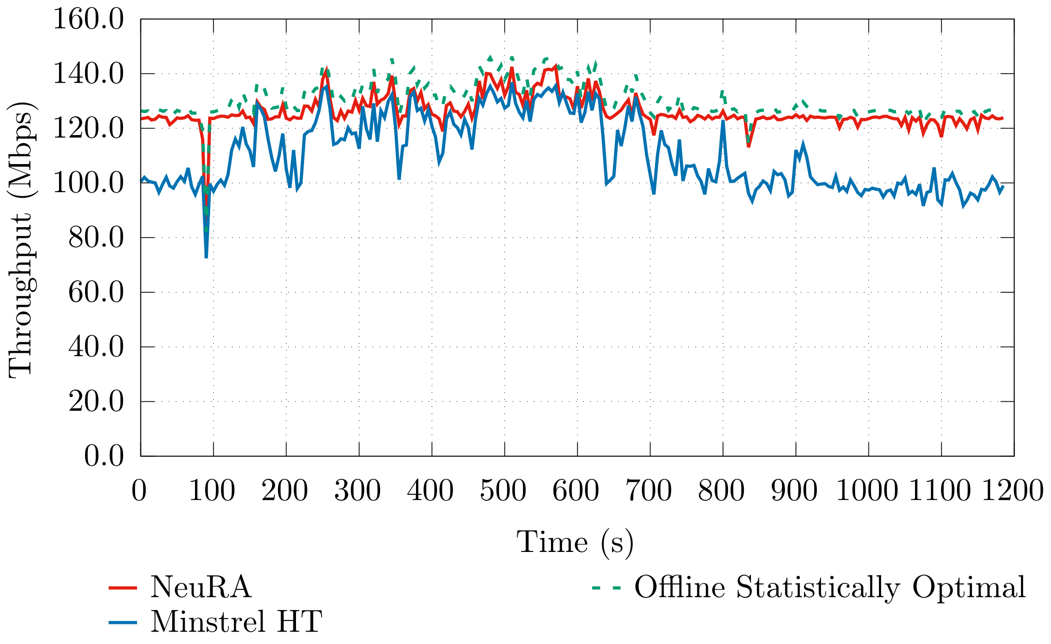


Figure 5.8: Comparison of the throughput over time for Scenario B1 (stationary).

between Minstrel HT and other algorithms is more pronounced at the beginning and end of the experiment. It also shows that NeuRA appears to be mostly effective at tracking the offline statistically optimal algorithm during those same periods of time.

Table 5.4: Average throughput of Minstrel HT (ath9k) on traces.

Scenario	A1	A2	A3	A4	A5	A6	A7
Tput (Mbps)	18.7	15.9	5.7	27.1	6.0	16.6	7.1
Scenario	B1	B2	B3	B4	B5	B6	B7
Tput (Mbps)	110.7	28.5	118.4	55.7	43.3	84.3	39.8

5.4 Observations

Here, we list several observations made from examining the results of our trace-based evaluation (Figure 5.6). Recall that these traces are mainly collected from commonly used environments representative of those in which devices would actually be used.

Key Observations 1: NeuRA achieves up to 24% (16% on average) higher throughput than Minstrel HT and up to 32% (13% on average) higher throughput than Intel iwl-mvm-rs. Furthermore, if we compare NeuRA with the maximum throughput across all practical combinations, NeuRA still provides throughput up to 20% (9% on average) higher. Also, NeuRA's throughput is almost never lower than any other practical algorithm (except Scenario B1 where Intel outperforms NeuRA by 0.8%).

Key Observations 2: The offline statistically optimal solution achieves throughput up to 58% (30% on average) higher than Minstrel HT, up to 74% (28% on average) higher than Intel iwl-mvm-rs, and up to 58% (22% on average) higher than the maximum throughput among practical combinations (except NeuRA). It shows that the widely-used algorithms can be improved but not the amounts reported in some previous research. When compared with NeuRA, the offline statistically optimal algorithm achieves only up to 31% (12% on average) higher throughput. Another way of looking at these results is that NeuRA reduces the gap between the practical algorithms and the offline optimal algorithm by roughly half.

Other Observations 1: Minstrel HT w/o LGI sampling (2) performs up to 5% (1% on average) better than vanilla Minstrel HT. It shows that if the relationships between rates are not used with a proper prediction model, the improvement is not significant and the throughput may even decrease in some cases. Furthermore, we note that Intel iwl-mvm-rs performs up to 15% (3% on average) better than Minstrel HT. However, in several scenarios it performs up to 8% worse.

Other Observations 2: We note that Minstrel HT + OSOFA (5) improves Minstrel HT by less than 2.5% on average. Even though there is significant improvement of up to 14% for Scenarios B5 and B7, by comparing OSOFA (5) and Offline Statistically Optimal (8) we see that frame aggregation has a less significant role than rate adaptation for the devices and scenarios used in this study. Limiting the aggregation length is only useful when there is high variability in the subframe error rates of the rates being used. Figure 5.9 shows the maximum SFER variability for A3 (no impact from subframe position) and for B7 (significant impact from subframe position). We observe that our cellphone devices (SM-N920C and EML-L09C) show much less SFER variability (mostly flat curves) when compared to other devices. Also, both STRALE and PNOFA perform worse than vanilla Minstrel HT in 2.4 GHz scenarios (A1-A7). In 5 GHz scenarios (B1-B7), STRALE obtains some minor improvements (up to 5%) while in some case PNOFA obtains slightly larger improvements (up to 10%).

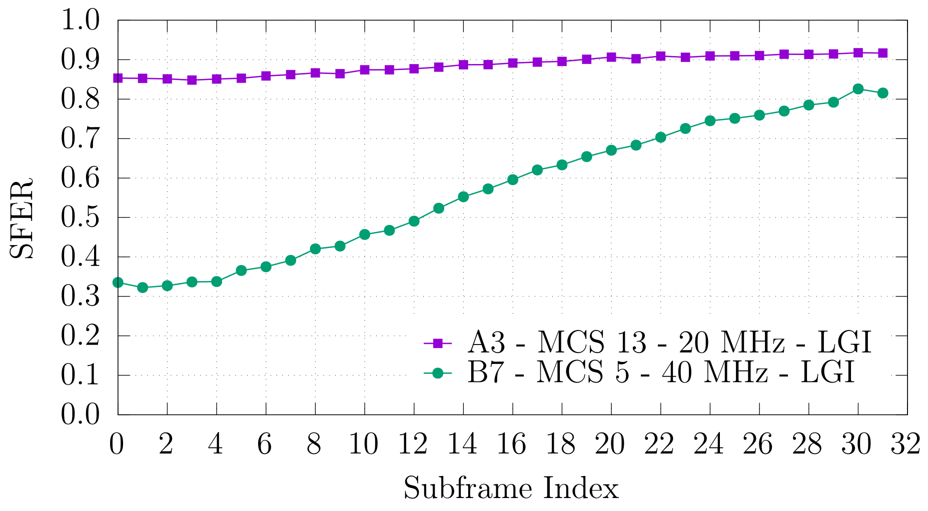


Figure 5.9: Comparison of the variability in subframe error rates (SFER) for A3 and B7.

5.5 Chapter Summary

In this chapter, we first evaluate the neural network model used in NeuRA. We study metrics of accuracy such as mean absolute error (MAE) and also define and measure several metrics related to the performance of the model in rate adaptation. We observe that our model does not overfit the training data. Using a reasonable number of sampling rates, the model provides accurate estimations for non-sampled rates to be used during rate adaptation. We then evaluate our approach to feature selection by comparing it with two other approaches (random rate selection and using only SGI rates for sampling). We show that our approach selects rates that are both faster to sample and that provide significantly more accurate estimations.

Next, we perform trace-based evaluation using T-SIMn to compare NeuRA and the offline statistically optimal algorithm to widely-used and state-of-the-art algorithms. We observe that NeuRA improves throughput for all scenarios (even scenarios with previously unseen client devices and movement behaviours). NeuRA performs up to 24% (16% on average) better than Minstrel HT and up to 32% (13% on average) better than Intel iwl-mvm-rs. Additionally, the upper bound provided by the offline statistically optimal algorithm is up to 58% (30% on average) higher than Minstrel HT and up to 31% (12% on average) higher than NeuRA. We conclude that NeuRA can effectively increase throughput and decrease the gap between practical sampling-based algorithms and the upper bound given by the offline statistically optimal algorithm by a half.

Chapter 6

Real-World Prototype

We have seen that NeuRA consistently improves the throughput of rate adaptation algorithms in trace-based evaluations. However, that evaluation does not consider the required processing power and other obstacles that may prevent NeuRA from working in real-time. In this section, we develop and describe a real-world NeuRA prototype developed for the *ath9k* WiFi device driver and compare its performance with the default rate adaptation algorithm (Minstrel HT). Empirical evaluations can be quite difficult to perform correctly when comparing the performance of different rate adaptation algorithms because of the high variability in WiFi channel state. To ensure that the evaluation is sound, we utilize the randomized multiple interleaved trials method [3] to neutralize the effect of changes in the environment on the comparison.

6.1 Prototype Design

The architecture of this prototype is shown in Figure 6.1. It consists of a modified *ath9k* Linux kernel module and a user space process for performing predictions. When NeuRA is started, a specified set of rates is sampled with a prescribed frequency.

The modified driver writes all received Block ACKs into kernel message ring buffer which is read by the NeuRA process. NeuRA keeps an exponentially weighted moving average (EWMA) of the frame error rate for each sampling rate and updates it whenever it reads a block ACK of that rate. Every 1 ms NeuRA calculates the effective throughput of the sampling rates and feeds them to the neural network model to predict the throughput of each rate. The three best rates are then sent to the *ath9k* driver via debugfs to be

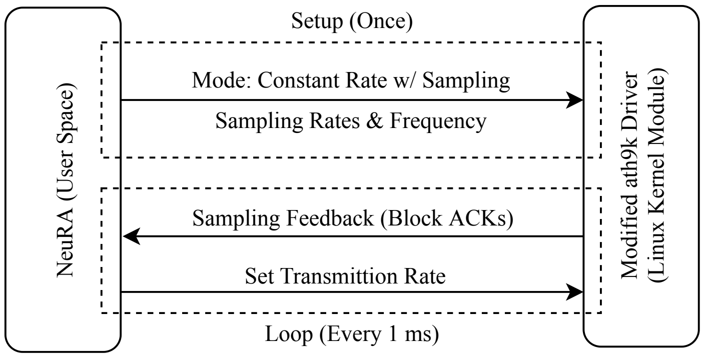


Figure 6.1: Architecture of NeuRA prototype implemented using *ath9k* driver.

used as the transmission and retry rates. The user space process is written in C++ for performance reasons. We use the same Keras models derived from our training set and use the frugally-deep library [18] to perform the predictions in C++. We used the same parameters and sampling rates as used in the trace-based evaluations.

The rate adaptation algorithm is not identical to the one used in trace-based simulations. Because setting the transmission rates is done asynchronously there may be a small delay (a few packets) between when new rates are set and when they are used for transmitted packets. On the other hand, setting the rates asynchronously prevents the driver from waiting for NeuRA to perform its calculations. Also, the default frame aggregation algorithm is used with NeuRA in the prototype. Furthermore, instead of choosing one best rate which is used by T-SIMn, three rates are selected to fill the retry chain in the *ath9k* driver similar to Minstrel HT (the fourth rate in the retry chain is usually not set when frame aggregation is enabled).

6.2 Evaluation

We perform several experiments to evaluate our prototype. In each experiment, we compare the maximum achievable throughput using Minstrel HT and NeuRA. We saturate the link between the sender and receiver using iperf3 [25] to send UDP packets. We exclude the first and last 2 seconds of each experiment (the warm up and cool down periods). We use 20 randomized multiple interleaved trials [3] (10 randomly interleaved trials for Minstrel HT and 10 for NeuRA). The length of each trial is 14 seconds (10 seconds when excluding warm up and cool down).

Table 6.1 describes the scenario for each experiment. E1 is performed with Configuration A (2.4 GHz, 20 MHz, 2 antennas) and E2 is performed with Configuration B (5 GHz, 40 MHz, 2 antennas). The TP-Link TL-WDN4800 802.11n PCI-E card and *ath9k* driver are used as the wireless access point (sender). This runs on an AMD Phenom™ II X4 955 Processor at 800 MHz. We observe that the average CPU utilization of NeuRA does not exceed 20% of a single core when with predictions done every 1 ms. Even though predicting every 1 ms may seem aggressive, we use it to be consistent with our implementation of NeuRA in T-SIMn. Moreover, it lets us measure the CPU utilization while heavily utilizing the neural network model. We observe that it takes 2-6 packets (depending on the rates used) from the time a new set of transmission rates are set (by the user-space process) until those rates are used in transmission.

Table 6.1: NeuRA prototype experimental scenarios (throughput for Minstrel HT).

Exp	Model Used	Client	Description	Avg. Tput
E1	Model A	TL-WDN4200	Stationary, Close AP	58.3 Mbps
E2	Model B	TL-WDN4200	Stationary, Close AP	159.7 Mbps

Figure 6.2 compares the relative throughput of Minstrel HT and NeuRA with 95% confidence intervals for each experiment. In these stationary experiments, confidence intervals do not overlap and show that NeuRA produces higher average throughput than Minstrel HT. Comparing the average throughputs shows 14% and 16% improvements with NeuRA when compared to Minstrel HT. We also performed experiments with mobile clients but in those scenarios, confidence intervals are wide and overlap due to high variability caused by constant changes in the environment. Therefore, we do not include those results because they only demonstrate the difficulty of conducting repeatable experiments in environments with highly variable channel conditions.

Note that this prototype is designed to study the practicality of NeuRA and measure its required computation power, not to maximize throughput. The low measured CPU utilization demonstrates that NeuRA processing power requirements are relatively small and should easily be supported by access point CPUs or an application-specific integrated circuit (ASIC). We also found reducing the prediction interval to 5 ms does not seem to affect throughput but lowers CPU utilization to 12%.

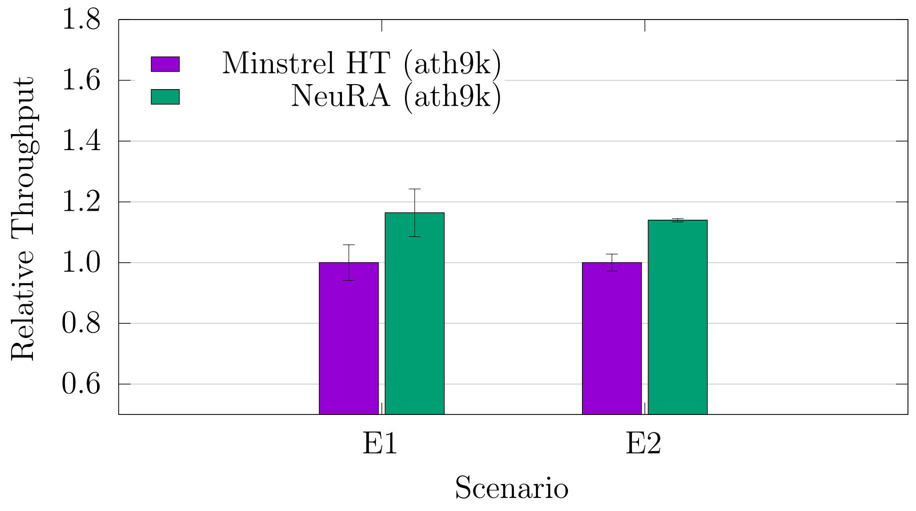


Figure 6.2: Comparison of throughput for Minstrel HT and the NeuRA prototype.

6.3 Chapter Summary

In this chapter, we describe the real-world prototype of NeuRA which we develop for the Linux *ath9k* WiFi device driver. We implement NeuRA as a user-space process that communicates with the *ath9k* driver through the debugfs interface. The prototype uses the same neural network models and sampling parameters as the T-SIMn implementation of NeuRA. However, several changes are made to integrate the algorithm with the *ath9k* driver. We use the randomized multiple interleaved trials (RMIT) technique to perform fair comparisons between our prototype and the default rate adaptation algorithm in *ath9k* (Minstrel HT). We observe that NeuRA increases the throughput by 16% on average when compared to Minstrel HT. Furthermore, it requires relatively little processing power to obtain those improvements.

Chapter 7

Conclusions and Future Work

7.1 Thesis Summary

WiFi devices use rate adaptation and frame aggregation algorithms to continually adapt to highly variable channel conditions and choose the best physical rate and number of frames to aggregate when transmitting. Although a variety of techniques have been proposed for rate adaptation, sampling-based rate adaptation algorithms have gained popularity in commercial devices because they make decisions based on real-time measurements and work in a variety of conditions.

Sampling-based algorithms measure the throughput of different physical rates by periodically using them for transmission (e.g., 10% of the frames) which is called sampling and then choosing the rate with the highest throughput for other transmissions. During sampling, data has to be sent using non-optimal rates. Also, frame aggregation is usually disabled when sampling because the reliability of the rates used for transmission is unknown. As a result, even though modern algorithms such as Minstrel HT implement several optimizations to avoid sampling every rate, sampling still imposes considerable overhead and reduces throughput.

Previous work has shown that relationships exist between the reliability of different physical rates that can potentially be used to reduce the sampling overhead [1]. In this thesis, we present NeuRA, a neural network based rate adaptation algorithm. NeuRA learns about the relationships between the throughput of different physical rates and uses those relationships to estimate the expected throughput of some rates based on the measured throughput of other rates. We use a novel application of recursive feature elimination

(RFE) to choose the best set of rates to sample. These rates and the derived models are then used to make good choices in rates to sample and to reduce the number of rates that are sampled, thus decreasing sampling overhead and increasing WiFi throughput.

Additionally, we derive a previously unknown offline algorithm to calculate the statistically optimal combination of the number of frames to aggregate and the physical rate to use at each point in time. We implement this algorithm in T-SIMn to calculate an upper bound on the throughput for a recorded WiFi trace. This provides an upper bound on the throughput that can be obtained by practical online rate adaptation and frame aggregation algorithms.

Trace-based evaluations are used to compare the performance of NeuRA and the offline statistically optimal algorithm with widely-used rate adaptation and frame aggregation algorithms. We find that the offline optimal solution provides an upper bound on throughput which is up to 74% (30% on average) higher than Minstrel HT and up to 74% (28% on average) higher than Intel iwl-mvm-rs. We conclude that several-fold improvements over Minstrel HT shown in some previous work [21, 22] are unlikely to be obtained in typical real-world scenarios.

Furthermore, we observe that the neural network model learns (from training data) fairly generalizable relationships between rates that work well on previously unseen devices, types of client motion and environments. NeuRA effectively reduces the gap between practical sampling-based algorithms and the upper bound given by the offline statistically optimal algorithm by a half and performs up to 24% (16% on average) better than Minstrel HT and up to 32% (13% on average) better than Intel iwl-mvm-rs. Interestingly, NeuRA provides throughputs that are surprisingly close to that of the offline statistically optimal algorithm (especially given that the offline algorithm uses information about the future that is not available to NeuRA).

Finally, we implement a prototype of NeuRA using the *ath9k* driver to evaluate its practicality. We find that it requires a relatively small amount of processing power (less than 20% of a single 800 MHz core) to increase the average throughput by 15% when compared to the default Minstrel HT scheme.

7.2 Future Work

In this section, we present some ideas for future work to extend the offline statistically optimal algorithm and NeuRA.

7.2.1 Near-Optimal Rate Adaptation Algorithm

The offline statistically optimal algorithm we propose in this thesis uses an oracle to obtain information about the fate of future subframes. As a result, the algorithm can choose the best aggregation length and best physical rate at each point in time. Another interesting algorithm to explore is an algorithm that works like our offline statistically optimal algorithm but only uses information from the past (i.e., it is not permitted to use information about the fate of future subframes). This algorithm would approximate the error rate of each subframe for each physical rate based only on their past behaviour. Because online practical algorithms do not have access to information about the future, this algorithm provides a bit of a more realistic upper bound for the throughput of practical rate adaptation and frame aggregation algorithms. In future work, we plan to implement this algorithm in T-SIMn and compare it to NeuRA and widely-used algorithms using trace-based evaluations.

7.2.2 More Rates, Newer Standards

Our evaluations of NeuRA have focused on using 802.11n devices with two antennas. This is because of the limitations of existing trace collection and trace-based evaluation tools and because 802.11ac devices implement frame aggregation and rate adaptation in proprietary closed-source firmware. We believe that NeuRA may obtain even better results when using devices with more physical rates and newer standards (e.g., 802.11ac and 802.11ax devices).

When dealing with a large number of physical rates, multiple neural network models can be used to reduce the size of each model (e.g., one for predicting the best physical rate group and another for predicting the best MCS index). Furthermore, a new trace collection methodology needs to be developed because the trace collection tool used with T-SIMn has a lower precision when used with a large number of physical rates. Additionally, because newer WiFi devices usually implement rate adaptation and frame aggregation algorithms in their proprietary firmware, integrating the model in device firmware requires cooperation from industrial WiFi chip makers. As a result, using NeuRA with newer WiFi standards is left as future work.

7.2.3 Adaptive Configuration for NeuRA

Our NeuRA implementation uses parameters such as the sampling frequency and the size of the sampling set. In this thesis, we choose a constant set of parameters that work best on average and achieve the highest throughput across the scenarios we study. However, the best sampling frequency and the best sampling set size varies slightly across different scenarios. To further reduce the sampling overhead, a predictive system (e.g., another neural network model) could be used to periodically predict the best set of sampling parameters. Information such as the average frame error rate, RSSI variability and the average throughput can be used to predict the best set of parameters.

7.3 Concluding Remarks

We believe that relationships between WiFi physical rates can be leveraged to reduce the sampling overhead of rate adaptation algorithms. Our implementation of NeuRA in T-SIMn and our prototype implementation show promising improvements over widely-used rate adaptation algorithms such as Minstrel HT and Intel iwl-mvm-rs. Furthermore, the results from the offline statistically optimal algorithm suggest that there is not much room for improvement over NeuRA. We look forward to implementing NeuRA on devices using newer WiFi standards and a larger number of physical rates by cooperating with device manufacturers to address the outstanding issues. NeuRA is expected to have an even more significant impact on the throughput of those devices.

References

[1] Ali Abedi and Tim Brecht. Examining relationships between 802.11n physical layer transmission feature combinations. In *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 229–238, 2016.

[2] Ali Abedi and Tim Brecht. PNOFA: Practical, near-optimal frame aggregation for modern 802.11 networks. submitted for publication, 2020.

[3] Ali Abedi, Andrew Heard, and Tim Brecht. Conducting repeatable experiments and fair comparisons using 802.11n MIMO networks. *ACM SIGOPS Operating Systems Review*, 49(1):41–50, 2015.

[4] Ali Abedi, Andrew Heard, and Tim Brecht. T-SIMn: Towards the high fidelity trace-based simulation of 802.11n networks. In *Proceedings of the 19th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 83–92, 2016.

[5] John Charles Bicket. Bit-rate selection in wireless networks. Master's thesis, Massachusetts Institute of Technology, 2005.

[6] Sanjit Biswas, John Bicket, Edmund Wong, Raluca Musaloiu-E, Apurv Bhartia, and Dan Aguayo. Large-scale measurements of wireless network behavior. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 153–165, 2015.

[7] Seongho Byeon, Kangjin Yoon, Okhwan Lee, Sunghyun Choi, Woonsun Cho, and Seungseok Oh. MoFA: Mobility-aware frame aggregation in Wi-Fi. In *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies*, pages 41–52, 2014.

[8] Seongho Byeon, Kangjin Yoon, Changmok Yang, and Sunghyun Choi. STRALE: Mobility-aware PHY rate and frame aggregation length adaptation in WLANs. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[9] François Chollet et al. Keras. https://keras.io, 2015.

[10] Lara Deek, Eduard Garcia-Villegas, Elizabeth Belding, Sung-Ju Lee, and Kevin Almeroth. A practical framework for 802.11 MIMO rate adaptation. *Computer Networks*, 83:332–348, 2015.

[11] Felix Fietkau. Minstrel rate control algorithm for mac80211. https://wireless.wiki.kernel.org/en/developers/documentation/mac80211/ratecontrol/minstrel.

[12] Felix Fietkau. Minstrel_HT: New rate control module for 802.11n. https://lwn.net/Articles/376765/, Mar 2010.

[13] Kunihiko Fukushima and Sei Miyake. Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position. *Pattern recognition*, 15(6):455–469, 1982.

[14] Matthew Gast. *802.11 Wireless Networks: The Definitive Guide*. O'Reilly, 2005.

[15] Matthew Gast. *802.11n: A Survival Guide*. O'Reilly, 2012.

[16] Rémy Grünblatt, Isabelle Guérin-Lassous, and Olivier Simonin. Simulation and performance evaluation of the Intel rate adaptation algorithm. In *Proceedings of the 22nd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 27–34, 2019.

[17] Daniel Halperin, Wenjun Hu, Anmol Sheth, and David Wetherall. Predictable 802.11 packet delivery from wireless channel measurements. *ACM SIGCOMM Computer Communication Review*, 40(4):159–170, 2010.

[18] T. Hermann et al. Frugally deep. https://github.com/Dobiasd/frugally-deep, 2016.

[19] Tim Higgins. Bye Bye 40 MHz Mode in 2.4 GHz. https://www.smallnetbuilder.com/wireless/wireless-features/31743-bye-bye-40-mhz-mode-in-24-ghz-part-1, 2012. Accessed: 2020-04-01.

[20] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. pybind11 – Seamless operability between C++11 and Python. https://github.com/pybind/pybind11, 2016.

[21] Raja Karmakar, Samiran Chattopadhyay, and Sandip Chakraborty. Dynamic link adaptation for high throughput wireless access networks. In *2015 IEEE International Conference on Advanced Networks and Telecommuncations Systems (ANTS)*, pages 1–6. IEEE, 2015.

[22] Raja Karmakar, Samiran Chattopadhyay, and Sandip Chakraborty. SmartLA: Reinforcement learning-based link adaptation for high throughput wireless access networks. *Computer Communications*, 110:1–25, 2017.

[23] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[24] Lito Kriara and Mahesh K Marina. SampleLite: A hybrid approach to 802.11n link adaptation. *ACM SIGCOMM Computer Communication Review*, 45(2):4–13, 2015.

[25] ESnet / Lawrence Berkeley National Laboratory. iperf3. https://software.es.net/iperf/, 2014.

[26] Duy Nguyen and JJ Garcia-Luna-Aceves. A practical approach to rate adaptation for multi-antenna systems. In *2011 19th IEEE International Conference on Network Protocols*, pages 331–340. IEEE, 2011.

[27] Ioannis Pefkianakis, Yun Hu, Starsky HY Wong, Hao Yang, and Songwu Lu. MIMO rate adaptation in 802.11n wireless networks. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, pages 257–268, 2010.

[28] Tomaso Poggio and Federico Girosi. Networks for approximation and learning. *Proceedings of the IEEE*, 78(9):1481–1497, 1990.

[29] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[31] Wei Yin, Peizhao Hu, Jadwiga Indulska, Marius Portmann, and Ying Mao. MAC-layer rate control for 802.11 networks: a survey. *Wireless Networks*, pages 1–38, 2020.