NeuRA: Using Neural Networks to Improve WiFi Rate Adaptation

Shervin Khastoo
University of Waterloo
skhastoo@uwaterloo.ca

Tim Brecht
University of Waterloo
brecht@uwaterloo.ca

Ali Abedi
University of Waterloo
ali.abedi@uwaterloo.ca

ABSTRACT

Although a variety of rate adaptation algorithms have been proposed for 802.11 networks, 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 may lead to excessive frame loss or the inefficient operation of frame aggregation algorithms. In this paper, we design a novel Neural network-based Rate Adaptation algorithm, called NeuRA. NeuRA, significantly improves the efficiency of probing in sampling-based algorithms by using neural network models to predict the expected throughput of many rates, rather than sampling their throughput.

Despite decades of research on rate adaptation in 802.11 networks, there are no definitive results which determine which algorithm is the best nor 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, NeuRA reduces the gap in throughput between existing algorithms and the offline optimal algorithm by half. Finally, we implement NeuRA using the ath9k driver to show that the neural network processing requirements are sufficiently low and that NeuRA can be used to obtain statistically significant improvements in throughput when compared with the Minstrel HT.

CCS CONCEPTS

• **Networks** → **Wireless local area networks**; **Wireless access points, base stations and infrastructure**; **Network performance evaluation**; • **Computing methodologies** → *Machine learning*.

KEYWORDS

WiFi; 802.11; rate adaptation; link adaptation; rate control; neural networks; optimal algorithms; frame aggregration;

ACM Reference Format:
Shervin Khastoo, Tim Brecht, and Ali Abedi. 2020. NeuRA: Using Neural Networks to Improve WiFi Rate Adaptation . In *23rd International ACM Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM '20), November 16–20, 2020, Alicante, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3416010.3423217

1 INTRODUCTION

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 obtain such throughputs. As a result, modern WiFi devices must choose from a large number of different physical rates (up to 768).

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 like Minstrel HT [12] are highly popular in commercial devices and have been shown to work in a variety of conditions because they make decisions based on real-time measurements. 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, drivers such as *ath9k* disable frame aggregation during sampling. This also leads to a significant loss in channel efficiency. As a result, there is an inherent trade-off between the sampling overhead and the effectiveness of the rate adaptation algorithm. In this work, we focus on improving practical widely-used sampling-based algorithms and understanding how they perform when compared to the statistically optimal solution.

With the growing number of modulation and coding schemes in newer WiFi standards, naÃŕve 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 [2]. To our knowledge, no algorithms exist that utilize relationships between rates to infer information about one or more rates using feedback obtained from sampling other rates.

In this paper, we propose a neural network model for estimating the throughput of all physical rates based on the measured 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) to maximum the prediction power for estimating other rates. We implement a rate adaptation algorithm called NeuRA (Neural network-based Rate Adaptation) that utilizes estimations from the neural network model to reduce sampling overheads required for rate adaptation.

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 [27] 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).

In 802.11n and later WiFi standards, the optimal physical transmission rate depends on the optimal number of frames to aggregate for each rate. In this paper, we derive and compare against a statistically optimal offline algorithm that uses an oracle to choose the optimal physical rate and the number of frames to aggregate. This algorithm acts as an upper bound on the throughput that could be obtained using practical online algorithms (including NeuRA).

Conducting credible empirical evaluations is extremely difficult in WiFi networks because of constantly changing channel conditions [4]. Therefore, we perform trace-based evaluations 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. To study the practicality of using NeuRA, we evaluate a real-world prototype using the *ath9k* WiFi driver. We find that NeuRA improves the throughput over Minstrel HT while requiring relatively small amounts of processing power. The contributions of this paper are:

- 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 recursive feature elimination (RFE) to recursively reduce the number of sampling rates while ensuring that remaining rates incur low overheads and have good predictive power.
- We develop and describe an offline algorithm to calculate the statistically optimal solution to WiFi rate adaptation and frame aggregation. This provides an upper bound on the throughput that can be obtained using online algorithms.
- We find that 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. NeuRA provide throughputs that are relatively close to that of the offline statistically optimal. This is despite the offline algorithm's use of information about the future that is not available to online algorithms like NeuRA.
- 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 (< 20% of a 800 MHz CPU core) to perform estimations. It also increases throughput by 15% on average when compared to Minstrel HT.

2 RELATED WORK

The problem of rate adaptation in WiFi networks (sometimes called "Rate Control" or "link adaptation") is a long standing and widely researched problem. Here, we review a subset of these methods 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. [27].

Sampling-Based Rate Adaptation: Many WiFi rate adaptation algorithms use sampling which was first introduced in SampleRate [6]. Minstrel HT [12] is the most widely used rate adaptation algorithm that periodically probes recently unused physical rates and uses those measurements to choose the rate with the highest expected throughput [27]. It is the default rate adaptation algorithm in the *mac80211* driver development framework of Linux kernel as well as the widely-used *ath9k* WiFi driver.

MiRA [24] is one of the earliest sampling-based rate adaptation algorithms for MIMO WiFi networks. It zig zags between singles stream and two stream modes favouring MCS index changes over the MIMO mode changes. Even though MiRA uses a novel approach to deal with hidden terminals, later evaluations shows that Minstrel HT performs as well as MiRA on average [23]. RAMAS [23] splits the rates into enhancement groups each representing a combination of the number of spatial streams, guard interval length, and channel width. It then and adapts the enhancement group and MCS index concurrently. Intel iwl-mvm-rs is the rate adaptation algorithm used in the *IwlWiFi* driver designed for use with Intel WiFi devices used in many laptops and personal computers [13]. It splits rates into several groups in a similar fashion to RAMAS but performs group adaptation and MCS adaptation phases sequentially (rather than concurrently) [13].

RSSI/SNR-based Rate Adaptation: Several algorithms propose methods for determining and using some information about the signal quality at the sender or the receiver to make informed decisions about the best transmission rate to use for the current channel conditions [6, 7, 11, 14]. Sender-side RSSI is known to be an unreliable metric for choosing the best transmission rate [14]. Key issues with the receiver-side approaches are that they sometimes rely on information not widely available or accessible on all devices (e.g., channel state information) and the information collected at the receiver needs to be transmitted to the sender. Unfortunately, this requires modifying control frames [27] (which requires changing protocol standards) or relies on optional features of 802.11 protocols that to our knowledge are not implemented by chipset manufacturers. As a result, these approaches are not used in practice.

Other Rate Adaptation Methods: HiWiLA [18] adapts the transmission rate using a predesigned state transition graph. SmartLA [19] 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 in the algorithm's description and because it would require access to their training data. We find the claims made in the

SmartLA paper about substantial improvements puzzling because SmartLA uses all available 802.11ac rates and frame aggregation while the algorithms they compare with seem to be restricted to slower 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 20 difference). Across the variety of devices and scenarios we study, the gap between Minstrel HT and the *statistically optimal solution* is much lower (30% on average always less than 60%).

Frame Aggregation Algorithms: Frame aggregation appears in the 802.11n standard to enable aggregating up to 64 subframes (MPDUs) into an aggregated frame (A-MPDU) to increase the MAC layer efficiency. Researchers have observed that in some cases subframes towards the end of an aggregated frame incur higher error rates [8]. As a result, MoFA [8] and STRALE [9] limit the number of frames being aggregated in such cases. STRALE also additionally adapts the physical rate. PNOFA [3] uses a formula similar to Equation 1 (presented later) to calculate the expected throughput for all possible aggregation lengths and then chooses the best length for the current rate. They demonstrate better performance than MoFA and STRALE. In our offline optimal algorithm we calculate the statistically optimal aggregation length for each rate based on a formula adapted from the PNOFA paper [3]. We also include STRALE and PNOFA in our trace-based evaluations.

3 NEURA

Abedi et al. [2] show that relationships exist between the frame error rates of many physical rates. It is expected that such 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. While Abedi et al. [2] focus on discovering 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.

In this paper, 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 use neural networks because with a sufficient number of layers and neurons that are trained on enough data, they can be used to approximate any function [25]. 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 Sections 6 and 7.

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

3.1 Proposed Model

We propose the use of a feed forward neural network model that uses the effective throughput of a subset of physical rates (called the sampling set) to estimate the expected throughput of the rates not in the sampling set. The architecture of this model is shown in Figure 1. Based on our experiments with different regression and classification model 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 [26] 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 [10] 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 [21]. We train the model for 1,000 epochs after which the loss function stabilizes.

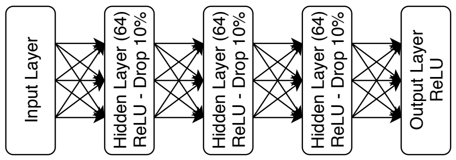

Figure 1: Structure of the proposed neural network model.

3.2 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 5 describes the scenarios used for trace collection. To perform a valid evaluation, we use two disjoint sets for training and evaluation (a training set and a testing set). We now describe the method used to convert our raw WiFi traces into data 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 [3] 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.

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

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 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 1: Format of the data set extracted from the traces. Each column represents a physical rate.

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

3.3 Feature Selection

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:

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

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 sample rate R. The intuition behind the equation is that we are interested in rates with the highest $\frac{\text{Estimation Power}}{\text{Sampling Time}}$ density 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. During this process, feature selection and model training are performed simultaneously.

4 OFFLINE 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 [4]) and the environments in which they are evaluated. An important and unresolved question is how well do these algorithms compare with one that makes optimal decisions. To that end, we now describe an algorithm for making statistically optimal decisions for both rate adaptation and frame aggregation. This requires knowledge about the fate of all subframes for current and future frames for all rates that could be 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 throughput can be used as an upper bound for what could be achieved by a perfect combination of rate adaptation and frame aggregation algorithms.

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 1 which is inspired by PNOFA [3] 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 6.3.

5 TRACE COLLECTION

T-SIMn [5] is an 802.11n trace-based simulator. It allows one to record traces from real-world WiFi experiments in a variety of settings and then simulate running different combinations of rate adaptation and frame aggregation algorithms on the trace. The technique used to collect traces is briefly described in Section 3.2.

We use two disjoint sets of traces for training and evaluation (the training and testing sets). 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 those raw traces are later used to evaluate different algorithms using T-SIMn. Trace-based evaluation enables a fair comparison of different rate adaptation/frame aggregation algorithms because all algorithms are exposed to exactly the same channel conditions.

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. We use a variety of receiving devices for trace collection because they do not require a modified driver.

Most commonly used WiFi devices (including most recent phones and laptops) have two antennas (and support two streams). Also, even though the highest available 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 [16]. As a result, we use the two configurations shown in Table 2 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. We train two separate models (Model A and Model B) because the two configurations have different available rates and different frequencies may have different relationships between rates.

Table 2: Two configurations used for trace collection.

Config	Carrier	# Antennas	Channel width	# Rates
A	2.4 GHz	2	20 MHz	32
B	5 GHz	2	40 MHz	64

We collect traces using several devices and several environments in which devices are expected to be used. Training scenarios are shown in Table 3. 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. Therefore, a total of 4 hours of data is used to train Model A (14,299 points) and 6 hours of data is used to train Model B (21,650 points).

Table 3: 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

Testing scenarios are shown in Table 4. Most of these traces are 20 minutes long while a few are under 20 minutes. These 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.

6 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.

Table 4: Scenarios for testing traces.

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

6.1 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 2 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 over fit 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.

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.

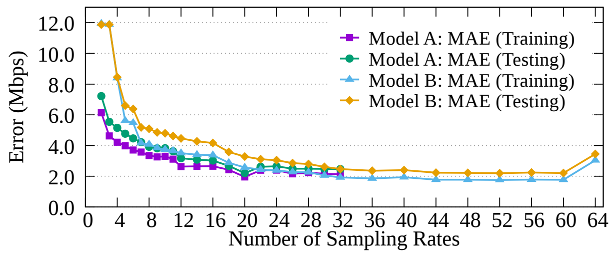

Figure 2: MAE for models with different numbers of rates. Similar MAE on the training and testing sets shows the model is general enough to hand unseen scenarios. The MAE is fairly low when at least 1/4 of the rates are sampled.

Relative RA (Rate Adaptation) Error: For each row of data in the data set (1-second time window), we only provide the model with the throughput of rates in the sampling set. The model then predicts the throughput of other rates and chooses the rate with the highest expected throughput.

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). Relative RA Error is then 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 3 shows Relative RA Error, and Figure 4 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.

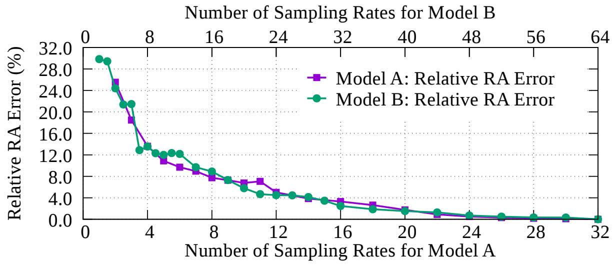

Figure 3: Relative RA Error measures the average throughput loss (as a result of inaccuracies in the predictions) when simulating rate adaptation on the testing data set. The error gets high when less than 1/2 of rates are sampled.

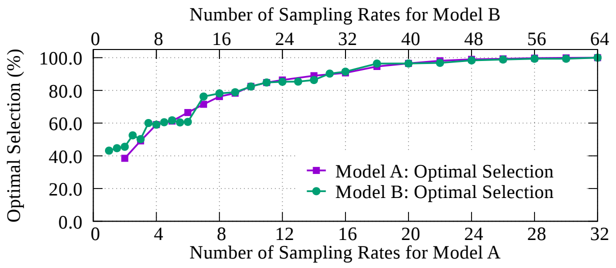

Figure 4: Optimal Selection measures the fraction of time that model predictions result in choosing a rate that performs at most 5% worse than the actual optimal rate. A reasonable value (>90%) requires sampling at least 1/2 of rates.

6.2 Feature Selection Evaluation

In this section, we evaluate the recursive feature elimination (RFE) approach to selecting 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 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. [2]. (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 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 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 (our importance calculation method considers sampling time) and results in significantly lower errors compared to other methods for both Model A and Model B.

Table 5: Models using different feature selection techniques. Rates chosen by RFE are both faster to sample and provide more accurate estimations.

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%

6.3 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 [5]. 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 [17] 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 neural network model.
- **Single rate sampling probability (F):** the probability of sampling a rate. So the probability of sampling is $N \times F$.
- **Frame aggregation algorithm (FAA):** options are "Default", "MoFA", and "PNOFA". "Default" is the 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 most consistently perform better than the others. The chosen parameters are listed in Table 6 for the two models and Table 7 shows the set of sampling rates used with these parameters. These parameters are used for all NeuRA results.

Table 6: Best parameters for NeuRA. Found empirically by trying all combinations of parameters on the training traces.

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

Table 7: 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

By examining Figure 3 and Figure 4, we see that the best sampling set size (best value of N) corresponds to the smallest sampling set with a relative rate adaptation error lower than 2% that also provides estimates that for at least 95% of the time lead to choosing a rate that is within 5% of the optimal rate.

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 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. LGI rates are assumed to be equal to their SGI counterparts. This algorithm was included in Abedi et al. [2].
(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. [13] 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 6 with predictions and rate changes being made every 1 ms.
(8) **Offline Statistically Optimal:** As described in Section 4, 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 combination (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.

We have generated several synthetic traces with known behaviours to verify the expected behaviour of all algorithms (not shown here). Interestingly, these enabled us to find a minor bug in the original STRALE implementation and a bug in our porting of the Intel iwl-mvm-rs algorithm. These bugs have been fixed and the simulation results obtained using the synthetic traces now match the expected results. A thorough description of this process and more details of the experimental results can be found in Shervin Khastoo's thesis [20].

The results of our trace-based evaluations are shown in Figure 5. All throughputs are shown relative to that of Minstrel HT (1). Table 8 presents the average throughput of 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, all differences are statistically significant and confidence intervals are not applicable.

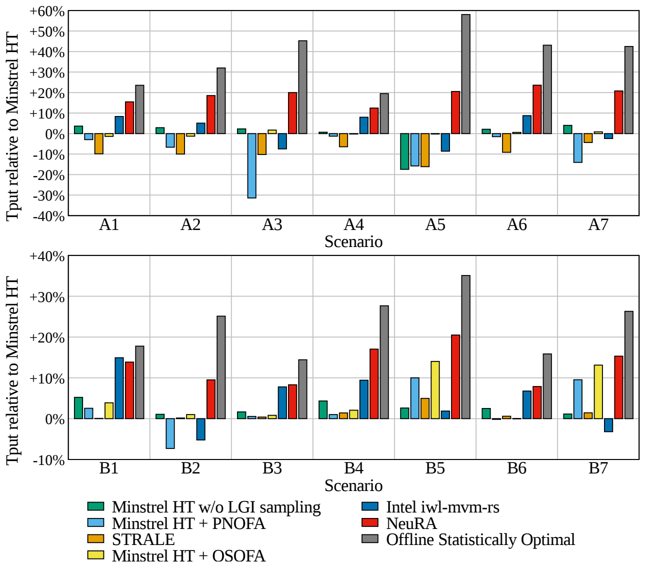

Figure 5: Average throughput relative to Minstrel HT. NeuRA improves the throughput on previously seen and unseen scenarios. The offline statistically optimal throughput is an upper bound and is not achievable in practice.

Table 8: Average throughput of Minstrel HT (1) 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

6.4 Observations

Here, we list several observations made from examining the results of our trace-based evaluation (Figure 5.) 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 case. 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 6 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%).

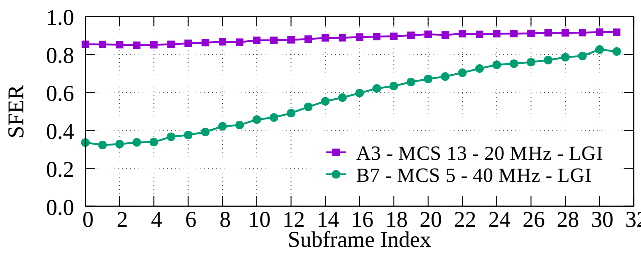

Figure 6: Examining the rates with highest variability in subframe error rates (SFER) suggests that only some scenarios benefit from limiting the frame aggregation length.

7 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 [4] to neutralize the effect of changes in the environment on the comparison.

The architecture of this prototype is shown in Figure 7. It consists of a modified *ath9k* Linux kernel module and a user space process for performing predictions. During NeuRA's setup phase the sampling set and sampling frequency are sent to the driver.

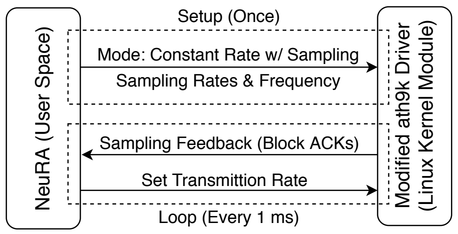
NeuRA (User Space)
Modified ath9k Driver (Linux Kernel Module)
Setup (Once)
Mode: Constant Rate w/ Sampling
Sampling Rates & Frequency
Sampling Feedback (Block ACKs)
Set Transmittion Rate
Loop (Every 1 ms)

Figure 7: NeuRA implementation using *ath9k* driver.

The modified driver writes all received Block ACKs into kernel message ring buffer which is read by the NeuRA process. NeuRA keeps a 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, 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 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 [15] to perform the predictions in C++. We used the same parameters and sampling rates as used in the trace-based evaluations.

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 [22] 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 [4] (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 9 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 for 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. 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 9: NeuRA prototype scenarios (tput for Minstrel HT).

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

Figure 8 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 the graphs show that NeuRA produces higher average throughput (14% and 16%) than Minstrel HT. We also performed experiments with mobile clients but in those scenarios, confidence intervals are wide and overlap due to constant changes in the environment. 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%.

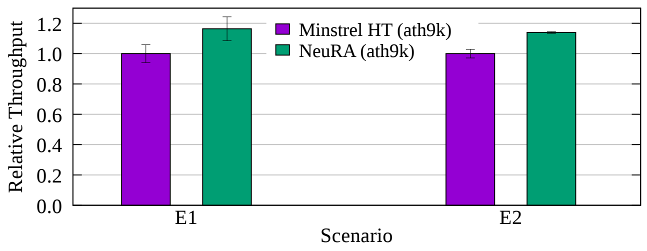

Figure 8: Real-world experiments with the NeuRA prototype show that it improves the throughput.

8 CONCLUSIONS

In this paper, we present NeuRA, a neural network based rate adaptation algorithm. NeuRA learns the relationships between the throughput of WiFi 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 reduce the number of rates that are sampled, thus decreasing sampling overhead, making good choices in rates to select and increasing WiFi throughput. Additionally, we derive an previously unknown offline algorithm to calculate the statistically optimal combination of the number of frames to aggregate and rate to choose. This provides an upper bound on the throughput that can be obtained by practical online algorithm.

Trace-based evaluations are used to compare the performance of NeuRA and the statistically optimal solution with widely-used rate adaptation and frame aggregation algorithms. 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 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 provide 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 and find that it uses a relatively low amount processing power to increase the average throughput by 15% when compared to the default Minstrel HT scheme. While our evaluations of NeuRA have focused on using 802.11n devices, we believe that even better results may be obtained when using 802.11ac and 802.11ax devices with more physical rates.

We plan to make the traces and source code used in this paper publicly available [1].

ACKNOWLEDGMENTS

We acknowledge the Natural Sciences and Engineering Research Council of Canada (NSERC) and Google for financial support and thank David Radke for his feedback on an earlier draft of this paper.

REFERENCES

[1] 2020. NeuRA Web Page. https://cs.uwaterloo.ca/~brecht/neura.
[2] Ali Abedi and Tim Brecht. 2016. Examining Relationships Between 802.11n Physical Layer Transmission Feature Combinations. In *MSWiM*. 229–238.
[3] Ali Abedi, Tim Brecht, and Omid Abari. 2020. PNOFA: Practical, Near-Optimal Frame Aggregation for Modern 802.11 Networks. In *MSWiM*.
[4] A. Abedi, A. Heard, and T. Brecht. 2015. Conducting Repeatable Experiments and Fair Comparisons using 802.11n MIMO Networks. *Operating Systems Review* 49, 1 (2015), 41–50.
[5] Ali Abedi, Andrew Heard, and Tim Brecht. 2016. T-SIMn: Towards the High Fidelity Trace-Based Simulation of 802.11n Networks. In *MSWiM*. 83–92.
[6] John C. Bicket. 2005. *Bit-rate Selection in Wireless Networks*. Master's thesis. Massachusetts Institute of Technology.
[7] S. Biswas, J.C. Bicket, E. Wong, R. Musaloiu-E, A. Bhartia, and D. Aguayo. 2015. Large-scale Measurements of Wireless Network Behavior. In *SIGCOMM*. 153–165.
[8] S. Byeon, K. Yoon, O. Lee, S. Choi, W. Cho, and S. Oh. 2014. MoFA: Mobility-aware Frame Aggregation in Wi-Fi. In *CoNEXT, Dec., 2014*. 41–52.
[9] S. Byeon, K. Yoon, C. Yang, and S. Choi. 2017. STRALE: Mobility-aware PHY rate and frame aggregation length adaptation in WLANs. In *INFOCOM, 2017*. 1–9.
[10] François Chollet et al. 2015. Keras. https://keras.io.
[11] L.B. Deek, E.G. Villegas, E.M. Belding, S.-J. Lee, and K.C. Almeroth. 2015. A practical framework for 802.11 MIMO rate adaptation. *Computer Networks* 83 (2015), 332–348.
[12] Felix Fietkau. 2010. Minstrel_HT: new rate control module for 802.11n. https://lwn.net/Articles/376765/.
[13] Rémy Grünblatt, Isabelle Guérin Lassous, and Olivier Simonin. 2019. Simulation and Performance Evaluation of the Intel Rate Adaptation Algorithm. In *MSWiM, November 25-29, 2019*. 27–34.
[14] D. Halperin, W. Hu, A. Sheth, and D. Wetherall. 2010. Predictable 802.11 packet delivery from wireless channel measurements. In *SIGCOMM, Aug. 2010*. 159–170.
[15] T. Hermann et al. 2016. Frugally Deep. https://github.com/Dobiasd/frugally-deep.
[16] Tim Higgins. 2012. Bye Bye 40 MHz Mode in 2.4 GHz. https://www.smallnet-builder.com/wireless/wireless-features/31743-bye-bye-40-mhz-mode-in-24-ghz-part-1. Accessed: 2020-04-01.
[17] Wenzel Jakob, Jason Rhinelander, and Dean Moldovan. 2016. pybind11 – Seamless operability between C++11 and Python. https://github.com/pybind/pybind11.
[18] R. Karmakar, S. Chattopadhyay, and S. Chakraborty. 2015. Dynamic link adaptation for High Throughput wireless access networks. In *IEEE International Conference on Advanced Networks and Telecommuncations Systems*. 1–6.
[19] R. Karmakar, S. Chattopadhyay, and S. Chakraborty. 2017. SmartLA: Reinforcement learning-based link adaptation for high throughput wireless access networks. *Computer Communications* 110 (2017), 1–25.
[20] Shervin Khastoo. 2020. *NeuRA: Using Neural Networks to Improve WiFi Rate Adaptation*. Master's thesis. University of Waterloo.
[21] D.P. Kingma and J. Ba. 2015. Adam: A Method for Stochastic Optimization. In *International Conference on Learning Representations*.
[22] ESnet / Lawrence Berkeley National Laboratory. 2014. iperf3. https://software.-es.net/iperf/.
[23] D. Nguyen and J.J. Garcia-Luna-Aceves. 2011. A practical approach to rate adaptation for multi-antenna systems. In *ICNP*. 331–340.
[24] I. Pefkianakis, Y. Hu, S.H.Y. Wong, H. Yang, and S. Lu. 2010. MIMO rate adaptation in 802.11n wireless networks. In *MobiCom*. ACM, 257–268.
[25] Tomaso Poggio and Federico Girosi. 1990. Networks for approximation and learning. *Proc. IEEE* 78, 9 (1990), 1481–1497.
[26] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. 2014. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* 15, 1 (2014), 1929–1958.
[27] W. Yin, P. Hu, J. Indulska, M. Portmann, and Y. Mao. 2020. MAC-layer rate control for 802.11 networks: a survey. *Wireless Networks* (2020), 1–38.