Fingerprinting 802.11 Rate Adaptation Algorithms

Mariyam Mirza*, Paul Barford*†, Xiaojin Zhu*, Suman Banerjee*, Michael Blodgett*
*Computer Science Department, University of Wisconsin-Madison
†Nemean Networks

Abstract—**The effectiveness of rate adaptation algorithms is an important determinant of 802.11 wireless network performance. The diversity of algorithms that has resulted from efforts to improve rate adaptation has introduced a new dimension of variability into 802.11 wireless networks, further complicating the already difficult task of understanding and debugging 802.11 performance. To assist with this task, in this paper we present and evaluate a methodology for accurately *fingerprinting* 802.11 rate adaptation algorithms. Our approach uses a Support Vector Machine (SVM)-based classifier that requires only simple passive measurements of 802.11 traffic. We demonstrate that careful conversion of raw packet traces into input features for SVM is necessary for achieving high classification accuracy. We tested our classifier on the four rate adaptation algorithms available in MadWifi, the most popular open source driver for commodity wireless cards. The classifier performs with an accuracy of 95%-100%. We also show that the classifier is robust over a variety of network conditions if the training data includes a sufficient sampling of the range of an algorithm's behavior.**

I. Introduction

802.11 supports multiple data transmission rates at the physical layer to allow senders to maximize throughput based on channel conditions. The modulation schemes used to encode data at lower rates are more robust to channel noise that those used for higher rates. If the channel quality is good, *i.e.*, the signal-to-noise ratio (SNR) is high, then higher data rates will maximize throughput because the bit-error rate (BER) will be low. If the channel is noisy, lower data rates will maximize throughput because the high BER at higher data rates will lead to increased loss and MAC-layer backoffs, resulting in poor throughput.

Designing algorithms that allow wireless senders to converge to the optimal rate for prevailing channel conditions in a timely fashion is challenging due to the difficulty of determining the cause of packet loss [1], limitations of the PHY/MAC interface in commodity wireless cards [2], and the assumption that a higher transmission rate always results in higher loss for a given RF environment not always being true [3]. Many attempts have been made to address this challenge, resulting in a large number of rate adaptation algorithms, with different algorithms performing best under different network conditions.

This work was supported in part by NSF grants CNS-0716460, CNS-0831427, CNS-0905186, CNS-0747177, CNS-0855201, CNS-1040648 and CNS-0916955. Any opinions, findings, conclusions or other recommendations expressed in this material are those of the authors and do not necessarily reflect the view of the NSF.

In this paper, we describe a method for identifying or *fingerprinting* the rate adaptation algorithms used in an 802.11 environment. We envision this capability as being part of a toolkit for automated performance analysis and debugging of production networks. The need for automated analysis and debugging has become increasingly urgent as 802.11 networks have grown to support large user populations. Client devices set their own configurations and connect and disconnect at will. Wireless network administrators have little control over, and knowledge of, network configurations, and cannot rely on cooperation from clients for performance analysis and debugging. Thus, practical performance analysis and debugging efforts for large-scale wireless networks such as [4], [5] are typically based entirely on passive monitoring, which requires no support or participation from clients. The presence of multiple rate adaptation algorithms introduces a new dimension of variability into 802.11 wireless networks. However, to the best of our knowledge, none of the passive monitoring-based performance analysis and debugging efforts in the literature consider the impact of 802.11 rate adaptation algorithms, despite the fact that the choice of rate adaptation algorithms can have a major impact on network throughput. The rate adaptation algorithm fingerprinting capability will provide additional information to passive monitoring systems to facilitate wireless network performance analysis.

We begin by investigating the details of the four open source rate adaptation algorithms from the popular MadWifi driver that constitute the test cases for our study. Manual examination of implementations shows that the algorithms can result in many possible rate change permutations depending on the timing and pattern of packet transmissions and losses. The large space of permutations precludes a fingerprinting approach based on explicitly enumerating all possible cases of an algorithm's behavior and suggests the need for a learning-based approach for algorithm classification.

We develop a rate adaptation classifier using Support Vector Machines, a state-of-the-art machine learning technique, using carefully selected input features from passive packet traces. We then conduct extensive experiments in a laboratory environment to identify combinations of features that result in the most accurate classification capability. Our results show that a classifier trained with a robust set of features can exhibit classification accuracy as high as 95%-100%. We show that careful selection of input features is necessary for achieving high classification accuracy. We also demonstrate that a classifier generated in one set of network conditions can identify algorithms in a different set of network conditions as long as the training data includes a sufficiently broad sampling of an algorithm's behavior.

II. Related Work

Given the potentially large impact rate adaptation algorithms can have on 802.11 network throughput, it is not surprising that many research efforts over the past decade have focused on these algorithms. Rate adaptation algorithms fall into two categories, those that use physical layer information such as signal-to-noise ratio (SNR) [6]–[9], and those that use frame level information such as packet loss and throughput [3], [10], [11]. Specialized rate adaptation algorithms have also been developed for vehicular wireless networks, *e.g.,* [12], [13].

A number of projects such as [3], [8], [14] have focused on characterizing the performance of rate adaptation algorithms, analyzing whether particular algorithm design choices results in optimal throughput in a particular type of RF environment. Our work is complementary to these projects because it can be used to identify the algorithms deployed, and the knowledge of algorithm performance characteristics garnered from these other projects can be used to determine whether the rate adaptation algorithm is the cause of performance problems.

Several machine learning techniques, including SVMs, have been used to select optimal modulation and coding schemes based on physical layer parameters for MIMO systems (*e.g.*, [15]). Such efforts are complementary to our work because they use SVMs to optimize throughput while our work uses SVMs for algorithm identification.

III. Support Vector Machines

This section introduces Support Vector Machines: for more details see [16]. We are interested in predicting the identity of rate adaptation algorithms based on their observed characteristics. In statistical machine learning, this can be cast into a classification problem.

Each run of a particular algorithm produces a series of feature vectors. We call each such feature vector $\mathbf{x}$ an *instance*. Each instance is represented by a d-dimensional real-valued vector $\mathbf{x} \in \mathbb{R}^d$. The label $y \in \{1,\ldots,C\}$ of an instance $\mathbf{x}$ is the identity of the algorithm underlying this run, where C is the number of distinct algorithms. For example, if a run of algorithm 2 produces 1000 feature vectors, we would have the following instance-label pairs: $(\mathbf{x}_1, y_1 = 2), (\mathbf{x}_2, y_2 = 2), \ldots, (\mathbf{x}_{1000}, y_{1000} = 2)$.

We have a training set which consists of multiple runs of all the algorithms under different conditions. The complete training set can be represented as a collection of n instance-label pairs $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$. A "test run" of an algorithm with unknown identity (but known to be one of those C algorithms) can be represented by its m instances $\{\mathbf{x}_i\}_{i=n+1}^{n+m}$. Our goal is to infer the class labels $y_{n+1} \ldots y_{n+m}$ in the test run. Note by definition $y_{n+1} = \ldots = y_{n+m}$. However, for computational convenience we adopt a two-stage procedure: In the first stage, we use a Support Vector Machine (SVM, discussed below) to predict the instance labels $y_{n+1} \ldots y_{n+m}$ which may be inconsistent (not all the same). In the second stage, we compute the single consensus label of the test run by a majority vote of the predicted instance labels. That is, the consensus label is the one which appear most frequently in $y_{n+1} \ldots y_{n+m}$.

To predict the instance labels, we train an SVM which can be understood as a function $\mathbb{R}^d \to \{1,\ldots,C\}$ that attempts to map any instance $\mathbf{x}$ to its class label y. For simplicity, we describe the linear, binary-label case $C = 2$, and refer the reader to the literature for the multi-label case. In this case, it is customary to encode the labels equivalently as $\{-1,1\}$ instead of $\{1,2\}$. A linear binary-label SVM estimates a real-valued function $f : \mathbb{R}^d \to \mathbb{R}$ with parameters $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$:

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b,$$

and predicts the class label according to $\text{sign}(f(\mathbf{x}))$. Training amounts to selecting an f that performs the best on the training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$. Performance here is measured by the so-called hinge loss function

$$L(f(\mathbf{x}), y) = \max(1 - yf(\mathbf{x}), 0)$$

which is to be minimized. The hinge loss is a surrogate (and in fact, a convex upper bound) of the 0-1 loss, which is one if the prediction $\text{sign}(f(\mathbf{x}))$ differs from the true label y, and zero otherwise. The hinge loss is preferred over the 0-1 loss because the former is easier to optimize due to its convexity.

Given the training set, training an SVM involves finding the function $\hat{f}$ that minimizes the hinge loss on the training set, plus a regularization term:

$$\hat{f} = \arg\min_f \sum_{i=1}^n L((f(\mathbf{x}_i), y_i) + \lambda \|f\|^2. \qquad (1)$$

The first term is the total loss on the training set. The function that minimizes this total loss "does best" on the training set. Such a function, however, may not be the one that produces the most accurate predictions on future test instances. This is because minimizing training set loss has the danger of *overfitting* the training data. One way to prevent overfitting is to regularize f by its norm $\|f\|^2$, with the intuition that we prefer a smoother function. The scalar λ balances training set loss and smoothness of f. The solution to the optimization problem (1) can be found efficiently using a quadratic program. The basic idea can be readily extended to multi-label cases. We use the multi-label linear SVM software SVM^{multiclass} [17].

We have used SVM-based methods for our earlier work on TCP throughput prediction for wireline and wireless environments in [18], [19]. The use of SVM in this case is considerably different because in the earlier work, the input feature set was small (less than 5 features), fairly obvious (*e.g.*, path properties such as loss rate and queuing delay), and the time scale of interest was on the order of several seconds. However, in this case, the feature set is large and non-obvious as described in Section V-C, and the time scale of interest is on the order of tens of milliseconds.

IV. Rate Adaptation Algorithms

The development of our classifier is based on the four 802.11 rate adaptation algorithms implemented by the latest version (0.9.4) of the popular open-source MadWifi driver [20]. We chose an open source driver so we could gain insight into algorithm behavior and validate the results of the classifier based on manual algorithm inspection. In this section we summarize the MadWifi rate adaptation algorithms. Our objective is to highlight the complexities of an algorithm's behavior, in particular the fact that an algorithm's behavior can change dramatically depending on the prevailing network conditions. The need for a learning-based classifier arises because the large number of packet rate and retransmission patterns that can occur with a given algorithm would be very difficult to enumerate explicitly.

The MadWifi driver is designed for wireless cards using Atheros chips, which implement multi-rate retries [21]. The Hardware Abstraction Layer (HAL) exports a *retry chain*, consisting of 4 ordered pairs of *rate/count* values, *r0/c0* through *r3/c3*. The hardware makes *c0* attempts to transmit a given packet at rate *r0*, *c1* attempts to transmit a packet and rate *r1*, and so on. Once the packet is successfully transmitted, the remainder of the retry chain is discarded.

The rate adaptation algorithms have three tasks, *(a)* to select rate *r* and count *c* values for the retry chain, *(b)* to determine the conditions under which the retry chain values are updated, and *(c)* to determine how often to check for the update condition. In the remainder of this section, we outline how the four algorithms perform these tasks. We present the algorithms in increasing order of complexity.

Onoe [22] tries to maximize throughput by selecting the highest transmission rate that results in loss rate below a certain threshold. It uses a system of *credits* to decide whether to change the current rate *r0*. The credit associated with *r0* is increased by one if less than 10% of packets in the last interval need retries, and *r0* is increased to the next highest rate when the credit exceeds 10. The credit associated with *r0* is decreased if more than 10% of packets need retries. *r0* is decreased to the next lowest rate if the average number of retries per packet exceeds one. The interval for evaluating loss rate and updating credits is 0.5–1.0 seconds. *r1* and *r2* are set to the two rates consecutively below the current *r0*, and *r3* is set to the lowest possible rate (6 Mbps in 802.11a/g). *c0* is set to 4, and *c1, c2* and *c3* are set to 2. Since *r0* is updated indirectly based on credits, and the retry chain is 10 packets long, *Onoe* is rather slow to adapt to changing network conditions.

AMRR [11], like *Onoe*, tries to maximize throughput by selecting the highest transmission rate that results in loss rate below a certain threshold. If less than 10% of packets are lost in the last interval, the current *r0* is increased to the next highest rate, and if greater than 30% of packets are lost, it is decreased to the next lowest rate, otherwise it remains unchanged. Loss rate is evaluated every 10 packets. If a rate increase is attempted and it results in a loss, the interval for attempting the next increase is enlarged exponentially, up to a maximum of 50 packets. This is done to prevent unnecessary losses if the current transmission rate is the highest possible for the target loss rate. *r1* and *r2* are set to the two rates consecutively below the current *r0*, and *r3* is set to the lowest possible rate. *c0, c1, c2* and *c3* are all set to 1 to make the retry chain shorter and the algorithm more responsive compared to *Onoe*.

Sample Rate [3] selects *r0* by explicitly computing the rate most likely to maximize throughput in the prevailing network conditions, unlike *Onoe* and *AMRR*, which use the combination of loss minimization and rate maximization to estimate the best rate. *Onoe* and *AMRR* assume that a higher transmission rate will always result in a higher loss rate in a given environment. However, [3] shows this assumption to be incorrect, and shows that loss rate at a higher transmission rate may be lower depending on the modulation and encoding of the rates and the amount of noise in the RF environment. Motivated by these observations, *Sample Rate* explicitly computes throughput for a given rate based on the number of successful and failed transmissions and 802.11 parameters such as inter-frame spacing and ACK transmission time. *Sample Rate* changes *r0* when another rate begins to yield better throughput. Since a rate other than the next highest or next lowest from the current *r0* may yield the best throughput, the algorithm has to periodically sample all other rates. 10% of transmission time is used for sampling alternate rates. Rates for sampling are selected intelligently, with rates more likely to improve throughput selected more frequently. *r0* is changed if sampling indicates that another rate will result in higher throughput. *r1* and *r2* are no longer set to the two next lowest rates after *r0*, rather they are set to rates with the next lowest throughputs. Throughput is reevaluated for the current *r0* and other candidate rates periodically and is smoothed using EWMA, with 5% of the weight coming from the last evaluation interval. *r0* is changed if another rate's EWMA throughput is greater.

Minstrel [23] is the most advanced rate adaptation algorithm implemented by the MadWifi driver. It improves on two aspects of *Sample Rate*. First, it sets *c0, c1, c2* and *c3* based on *r0, r1, r2* and *r3* such that the retry chain

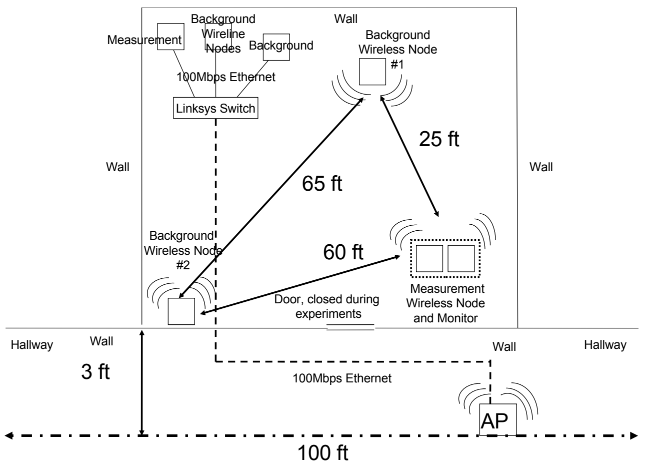

Background Wireline Nodes
Measurement
Background
Wall
100Mbps Ethernet
Linksys Switch
Background Wireless Node #1
25 ft
Wall
Wall
65 ft
Background Wireless Node #2
60 ft
Measurement Wireless Node and Monitor
Door, closed during experiments
Hallway
Wall
Wall
Hallway
3 ft
100Mbps Ethernet
AP
100 ft

Fig. 1: Laboratory testbed used to generate wireless network traces for rate adaptation algorithm classification.

completes within 26 ms, a time limit selected to minimize TCP performance deterioration in case of losses. Second, it was noted that even with *Sample Rate's* intelligent selection, sampling alternate rates resulted in use of low rates and low throughput. *Minstrel* tries to avoid this problem by more sophisticated sampling rate selection, the complete details of which can be found in [23]. The throughput is calculated in a manner similar to *Sample Rate*. It is reevaluated for the current *r0* and other candidate rates every 100ms. The value is smoothed using EWMA, with 25% of the weight coming from the latest 100ms interval, and *r0* is changed if another rate's EWMA throughput is greater.

V. Experimental Setup

In this section we describe our experimental testbed, how we used it to generate packet traces, and how we processed the traces for SVM-based algorithm classification.

A. *Experimental Environment*

Figure 1 illustrates the experimental testbed that we used to collect traces for classification. There are four primary components to the setup: wireline nodes, wireless nodes, one commodity access point (AP), and a monitor node. The wireline and wireless nodes are connected in a dumbbell topology via the APs and a switch. The AP is a Cisco AP1200, running IOS version 12.3(8), with single Rubber Duck antenna, and integrated 802.11a module/antenna. The switch is a commodity LinkSys 10/100 16-port Workgroup Hub. The wireline nodes and switch are connected to the AP via 100Mbps Ethernet connections. The maximum data rate for 802.11a is 54Mbps. Having 100Mbps wireline links insures that the wireless, rather than the wireline, part of the network is the throughput bottleneck.

The wireline and wireless nodes are identically configured Sun 4200 AMD Opteron 275 (dual Core) nodes, with 4 GB RAM, Intel 82546EB (e1000) chips, running CentOS 5.2. The wireless nodes are installed with R52-350 mini-PCI cards (Atheros 5414 chip). We used default wireless interface configurations for our experiments. These were: *(a)* no RTS/CTS, and *(b)* no MAC layer fragmentation.

Communication between wireless and wireline nodes is pairwise, *i.e.*, during an experiment, each wireless node sends data to a single, pre-assigned, wireline node and vice versa. One wireline-wireless pair is designated the measurement pair. Algorithm classification is done for packet traces from the measurement node pair. The other two node pairs generate background traffic. We refer to them as background pair one and background pair two. The monitor, an *AirPcapNx* adapter [24] is located next to the measurement node for all experiments. We use 802.11a channel 36 for all our experiments.

B. *Experimental Protocol*

The measurement node transferred 8MB files with 5 seconds between transfers. There were three levels of background traffic: no background traffic, one node pair generating background traffic, and two node pairs generating background traffic. The first background node pair transfers 4MB files with an interval of 2 seconds between transfers, and the second pair transfers 512KB files with 1 second between transfers. Both background nodes use *Sample Rate*, the default MadWifi rate adaptation algorithm.

For each background traffic level, 100 files per rate algorithm were transferred for each of the four algorithms. The file transfers for the different algorithms were interleaved in sets of 25 to compensate for possible external interference that could effect the integrity of classification.

The training set consists of 10% of samples selected uniformly at random from the first half of the transfers at each background traffic level. The test set consists of all samples from the second half of transfers at each background traffic level. We constructed 5 different training sets via random sampling, and tested all of them using the second half of the transfers. The classification accuracy values in Section VI are the averages of the five runs.

C. *Feature Selection for Algorithm Classification*

We process packet traces for rate adaptation algorithm classification in the following manner. We generate a training/test feature vector for every instance where the transmission rate of the *kth* and *k+1th* 802.11 data packet transmitted by the measurement node is different. The rate transition is the center of the packet window over which we compute features. The feature values were normalized by subtracting the mean of each feature from the respective feature values and then dividing by the standard deviation before training and testing.

We use two different feature sets, detailed below. The difference in the feature sets is that they represent information at varying levels of granularity for training and testing. *Feature Set 1* aggregates the information in the feature vector window, while *Feature Set 2* exposes detailed packet trace information. The results in Section VI will show that having packet trace information at varying levels of granularity is essential for accurate classification.

Feature Set 1: The following features are computed for a window of size m_1 around a rate transition event. Features *(1)–(4)* are computed once for the entire window, and *(5)–(12)* are computed individually for each of the eight 802.11a data rates. Each feature in the list below corresponds to two distinct features, one computed for a window of size m_1 before the rate transition event, and the second for a window of size m_1 after the rate transition event.

(1) The number of packets in the window.

(2) The packet reception probability, defined as the number of non-retry packets divided by the total number of packets in the window.

(3) The fraction of packets in the window that are unique, defined as the number of distinct 802.11 sequence numbers observed divided by the total number of packets.

(4) The trace accuracy, which is defined as $\frac{1 - \#\ of\ missing\ 802.11\ sequence\ numbers}{total\ \#\ of\ packets}$.

(5) The number of packets with each rate in the window.

(6) The number of packets with each rate that are retries in the window.

(7,8,9) The minimum, median, and maximum distance in packets for packets with each rate from the center of the window. Distance in packets is calculated in terms of the number of packets in the trace rather than packet sequence numbers. When a packet of a particular rate does not occur in a window, distance is set to a constant high value.

(10, 11, 12) The minimum, median, and maximum distance in packets for a retry packet with each rate from the center of the window.

Hence, for any value of m_1, there are a total of 136 features, $2*4$ for the four features (*1–4*) before and after the center of the window, and $8*8*2$ for the eight features (*5–12*) for each of the eight rates before and after the center of the window.

We consider the following different values of m_1: 100ms, 200ms, 300ms, 400ms, 500ms, 10 packets, 20 packets, 30 packets, 40 packets, 50 packets, 1-5 802.11 retries around a packet transition event.

Feature Set 2: For this set, the following three features corresponding to each packet in a window of size m_2 are included in the feature vector:

(1) The transmission rate of the packet.

(2) A binary value indicating whether the packet is a retry.

(3) The difference in sequence numbers between the packet and the center of the window.

We use the following values of m_2: 5, 10, 15, 20, 25, 30, 35, 40, 45 and 50 packets. In this case, since there are 3 features per packet and the window size is m_2 packets before and after a rate transition, the total number of features per feature vector is $3*((2*m_2)+1)$[1].

Finally, we constructed an *all features* vector, which is the concatenation of feature vectors for all window sizes for both feature sets, and contains 3720 features. In Section VI, due to space considerations we present classification accuracy results for *all features*, all window sizes for *Set 2*, and $10, 20, 30, 40$ and 50 packet window sizes for *Set 1*, because these feature sets yielded the most interesting results.

VI. Results

We conducted two sets of experiments one week apart, which we refer to as *Experiment 1* and *Experiment 2*, both using the protocol described in Section V-B. For results presented in Section VI-A, training and test sets, constructed as described in Section V-B, were drawn from the same experiment. For results presented in Section VI-B, the classifier trained on data from *Experiment 1* was tested on data from *Experiment 2* and vice versa to investigate the portability and robustness of the classifier.

We considered two classification metrics, *transfer classification accuracy* and *sample classification accuracy*. The first metric indicates whether the rate adaptation algorithm for the whole transfer is classified correctly. The classification for the whole transfer is determined by the majority of the classifications of the individual samples (feature vectors) in the transfer. We report the fraction of transfers classified correctly and incorrectly for each rate adaptation algorithm. The second metric is defined as the percentage of samples classified as a particular algorithm for each transfer. This is a measure of the confidence we can have in the classification of a transfer. We used this metric to guard against accuracy inflation, because a transfer classification is considered correct whether 100% of the samples from the transfer are classified correctly or whether 25.1% of the samples are classified correctly and the other 74.9% of the samples are split evenly between the remaining three classes. We found that *sample classification accuracy* closely followed *transfer classification accuracy*. Due to space considerations, we only present *transfer classification accuracy* in this paper.

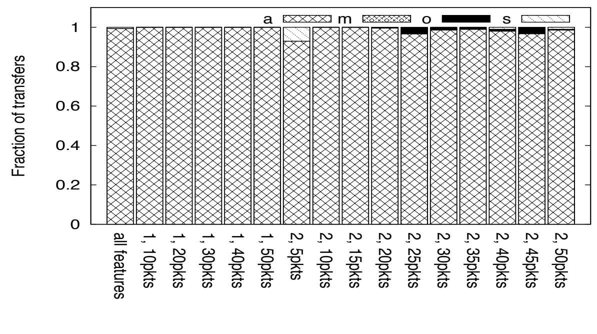
A. Classification Accuracy

Figure 2 illustrates the algorithm classification accuracy for the case where the training and test sets are drawn from the same experiments. We present results from two different runs of the same experiment, conducted in the same laboratory environment but one week apart, because wireless network conditions are difficult to replicate due to external interference effects. Table I presents the distribution of transmission rates for the two experiments, and it can be seen that there is a significant difference in the distributions in many cases.

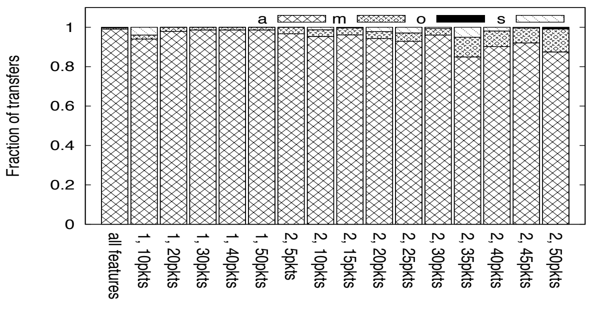
[1]+1 for the packet at the center of the window.

Fig. 2: Stacked histograms illustrating the fraction of transfers classified for each algorithm for *Experiment 1* and *Experiment 2* conducted one week apart in the same laboratory setting. In each case, the training and test set was drawn from the same experiment. The individual figure labels indicate the experiment and the correct algorithm. The x-axis indicates the feature set (1 or 2) and window size used. The y-axis indicates the fraction of transfers classified for a certain algorithm and feature set, e.g., *Set 1, 20 pkts* for *Experiment 2, Onoe* shows that approximately 70% of transfers were classified correctly as *Onoe*, 25% were classified incorrectly as *Minstrel* and 5% were classified incorrectly as *AMRR*. Algorithms are indicated in the key with the first letter in their name.

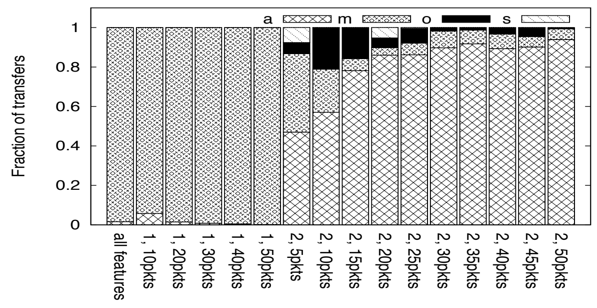
(a) Experiment 1, *AMRR*



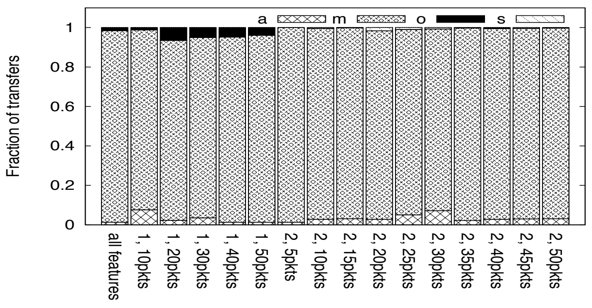
(b) Experiment 2, *AMRR*



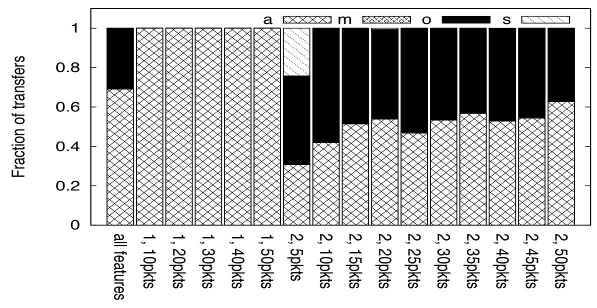
(c) Experiment 1, *Minstrel*



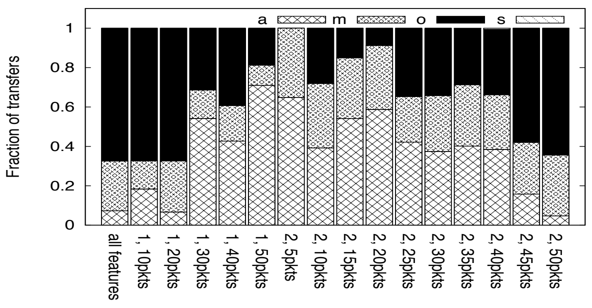
(d) Experiment 2, *Minstrel*



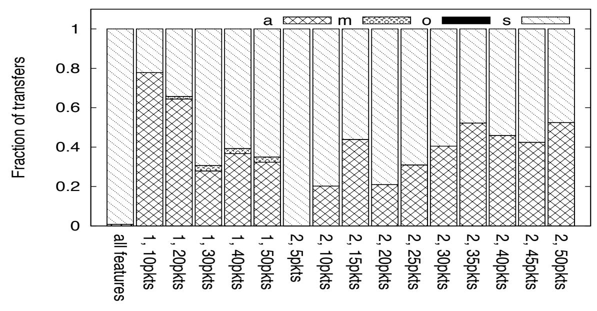
(e) Experiment 1, *Onoe*



(f) Experiment 2, *Onoe*



(g) Experiment 1, *Sample Rate*



(h) Experiment 2, *Sample Rate*

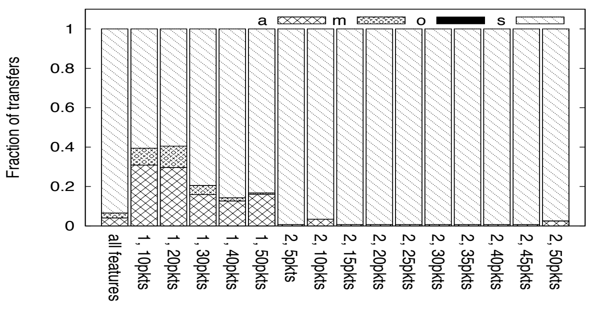


TABLE I: Distribution of packets across transmission rates for four target algorithms. The first column indicates the algorithm (algorithms are identified by the first letter in their names), the background traffic level (nobg for no background traffic, bg1 for one background node pair generating traffic, and bg2 for two background node pairs generating traffic), and the experiment number (1 or 2). The columns labeled with rates indicate the percentage of packets in each experiment transmitted at that rate. *Rate Change* indicates the percentage of times two consecutive packets were transmitted at different rates. *Retry* indicates the percentage of packets that were retries. *Consecutive Retry* indicates the percentage of times the *k+1th* packet was a retry given that the *kth* packet was a retry. This is an estimate of how far down the retry chain the algorithm had to go in case of a loss. The last three values are measures of the information content of a packet trace. Algorithms that have higher values are likely to be classified with higher accuracy because they provide SVM with a larger amount of information over a given time window, enabling SVM to generate a better distinguishing signature. All values are aggregates for all transfers in a given experiment.

Experiment	**6Mbps**	**9Mbps**	**12Mbps**	**18Mbps**	**24Mbps**	**36Mbps**	**48Mbps**	**54Mbps**	**Rate Change**	**Retry**	**Consec. Retry**
a, nobg, 1	0.3	0.0	0.1	0.7	9.6	78.1	11.1	0.0	31.8	16.5	6.2
a, nobg, 2	0.4	0.1	0.6	1.3	15.3	77.2	5.0	0.0	30.3	15.7	6.2
a, bg1, 1	0.3	0.1	1.3	6.7	37.7	51.2	2.8	0.0	28.6	15.4	9.8
a, bg1, 2	0.6	0.6	2.8	8.7	35.7	50.0	1.6	0.0	29.1	15.7	10.9
a, bg2, 1	0.3	0.7	1.4	5.4	31.3	55.4	5.5	0.0	30.7	16.4	9.2
a, bg2, 2	0.6	0.5	2.4	8.1	48.7	39.4	0.3	0.0	31.6	17.0	11.0
m, nobg, 1	0.0	0.0	0.1	0.2	0.5	92.4	4.0	2.8	7.4	14.2	42.9
m, nobg, 2	0.3	0.0	0.0	0.6	2.8	89.7	3.9	2.5	10.3	23.0	46.8
m, bg1, 1	0.2	0.1	0.1	0.3	2.8	88.9	5.1	2.6	13.1	24.6	44.9
m, bg1, 2	0.5	0.1	0.2	1.5	15.5	75.9	3.9	2.4	13.4	27.8	47.8
m, bg2, 1	0.4	0.0	0.1	0.5	1.9	89.5	4.9	2.6	10.5	22.7	41.3
m, bg2, 2	0.6	0.1	0.3	1.2	12.6	78.5	4.3	2.4	14.9	30.3	50.8
o, nobg, 1	0.0	0.0	0.0	0.0	4.9	94.8	0.2	0.0	4.1	10.9	32.8
o, nobg, 2	0.2	0.0	0.0	0.5	46.7	52.6	0.0	0.0	5.8	15.1	38.7
o, bg1, 1	0.2	0.0	0.0	0.7	33.0	66.2	0.0	0.0	7.7	18.8	30.4
o, bg1, 2	0.3	0.0	0.1	1.8	81.8	16.0	0.0	0.0	5.4	17.3	26.8
o, bg2, 1	0.3	0.0	0.0	1.4	56.1	42.1	0.0	0.0	5.8	17.2	26.6
o, bg2, 2	0.3	0.0	0.1	1.9	82.4	15.2	0.0	0.0	5.1	17.2	26.7
s, nobg, 1	0.2	0.0	0.2	2.9	41.5	54.5	0.7	0.1	9.7	8.2	23.9
s, nobg, 2	0.4	0.0	0.5	11.9	61.7	25.2	0.2	0.0	10.7	9.0	25.9
s, bg1, 1	1.1	0.0	2.9	18.9	49.2	27.4	0.4	0.1	11.6	16.8	25.5
s, bg1, 2	0.8	0.0	4.5	24.7	57.2	12.7	0.1	0.0	11.3	16.3	25.3
s, bg2, 1	1.7	0.0	4.6	17.1	40.3	36.0	0.2	0.1	13.2	17.9	29.7
s, bg2, 2	0.9	0.0	4.7	22.5	54.6	17.0	0.1	0.0	11.8	16.6	25.8

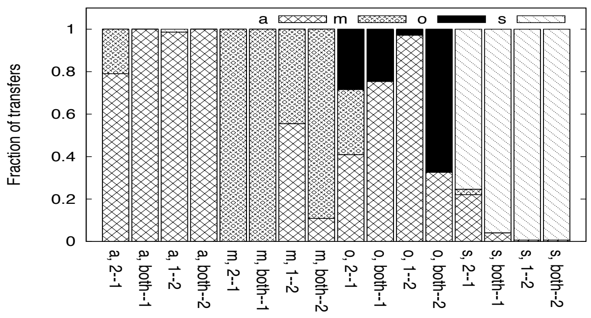


Fig. 3: Stacked histogram showing the fraction of transfers classified correctly and incorrectly for each algorithm, training set and test set combination. This figure illustrates the potential for the portability of the classifier. All results presented are for the *all features* set. The x-axis indicates the correct algorithm and the training and test sets, *e.g.*, *a, 2–1* indicates that AMRR is the correct algorithm, the training set consisted of data from *Experiment 2* and the test set consisted of data from *Experiment 1*, while *m, both–2* indicates that Minstrel is the correct algorithm, the training set consisted of data from both *Experiments 1 and 2*, and the test set consisted of data from *Experiment 2*.

Figure 2 shows that for all algorithms except *Onoe* (2e,2f), the classification accuracy for *all features* is high, ranging from 95% to 100%. Also, for all cases except *Experiment 1, Onoe* 2e, the accuracy for *all features* is very close to that of the highest accuracy individual feature set for that experiment. Both observations support our learning-based approach. Their combination means two things. First, classification accuracy is high. Second, feature set and window size selection are simplified because the concatenation of all feature sets and window sizes in *all features* results in accuracy equal to that of the most accurate individual feature set. This occurs even though the most accurate feature set varies by algorithm and experimental run.

The classification accuracy for *Onoe* is lower than all other algorithms across all feature sets. This is explained by considering the *Rate Change* column of Table I. This column indicates the rate change frequency, *i.e.*, the percentage of times two consecutive packets in a trace have different transmission rates. *Onoe* has the lowest rate change frequency, 4.1% to 7.7%. *Sample Rate* has the next lowest rate change frequency, 9.7% to 13.2%, roughly twice that of *Onoe*. This difference means that over a given packet window, a *Sample Rate* trace will provide SVM with twice as much information to generate a distinguishing signature compared to *Onoe*, resulting in higher classification accuracy for *Sample Rate*. The fact that *Onoe* has by far the lowest rate change frequency is consistent with what we know about the four algorithms. *Onoe* is the slowest to change rates because of its use of credits to calculate rate change. Also, it has the longest and slowest-to-change retry chain with a *c0* value of 4, which further reduces the frequency of rate change and hence the information content of packet traces. All other algorithms change rates at a frequency of roughly 11%-30% due to shorter retry chains and properties such as sampling alternate rates or trying a higher rate after every 10 successful packet transmissions at a given rate, yielding traces with a higher information content and therefore better classification accuracy.

Another observation from Figure 2 is that different window sizes and feature representations (*Set 1* versus *Set 2*) have different classification accuracies for the different algorithms and even for different experiments with the same algorithm. For example, *Set 1* has high classification accuracy for both experiments with *Minstrel*, while *Set 2* has high accuracy only for *Experiment 2* (Figures 2c,2d). Also, for *Sample Rate*, *[Set 2, 5 pkts]* is the only feature set and window combination that has high accuracy for *Experiment 1* (other than *all features*), while a number of feature set and window combinations have high accuracy for *Experiment 2*.

To explain why different window sizes and feature representations result in different classification accuracy, we have to consider how transmission rate changes between two successive packets. One reason for rate change is retry chain traversal due to packet loss. The second is sampling of other candidate rates, as in *Sample Rate* and *Minstrel*. The third is due to the algorithm deciding that the current *r0* is suboptimal and picking a new *r0*. Rate change due to the traversal of the retry chain occurs at the finest time granularity, from one packet to the next, for the length of the retry chain. Rate change due to sampling occurs at the second finest granularity (10 packets). Rate change due to computation of a new optimal *r0* occurs at the coarsest granularity (tens of packets, 1 or more seconds).

Consider a hypothetical algorithm that changes rates only due to retry chain traversal. Such an algorithm's distinguishing signature would be captured most effectively by highly detailed information over a small window, such as *[Set 2, 5 pkts]*, because all rate changes would occur at the granularity of individual packets. If *Set 1* and a larger window are used, the fine-grained information that is the algorithm's signature will be lost. Consider another hypothetical algorithm that only changes rate based on long-term loss and throughput feedback, *i.e.,* when it re-evaluates the optimal rate, it sets all rates in the retry chain to that one optimal value. Such an algorithm's distinguishing signature would be captured most effectively by features that *(a)* are computed over longer windows, and *(b)* can filter out short term variations and capture longer term throughput, loss and rate change behavior, such as *[Set 1, 50 pkts]*.

Practical algorithms change rates at all three time granularities, so features computed at varying levels of detail at a range of time granularities are necessary to comprehensively capture an algorithm's signature. This is why *all features*, a concatenation of all feature set and window size combinations, always has the highest classification accuracy. However, in a given run for an algorithm, rate might change for one particular reason more often than for others. If the dominant cause of change is retry chain traversal, we expect *[Set 2, 5 pkts]* to be the most accurate classifier, as for *Sample Rate, Experiment 1*. If, however, the dominant cause of change is reevaluation of the optimal rate, we expect *Set 1* classifiers to be more accurate than *Set 2* classifiers, as for *Minstrel, Experiment 1*. Based only on passive packet traces, it is difficult to identify with certainty the cause of rate change from one packet to the next.

B. Classifier Portability

We investigate the portability of classifiers by testing the classifier trained on *Experiment 1* on transfers from *Experiment 2* and vice versa. We conclude that classifier portability depends on the relative information content in the training and test data. If the training data's information content is greater than or equal to that of the test data, the classifier is as accurate for the test data from the other experimental run as it is for test data from its own experimental run, otherwise it is less accurate.

Figure 3 illustrates the classification accuracy when a classifier trained on data from one experiment is tested on data from the second experiment. The classification accuracy for *Onoe* is low across all training and test sets due to low information content of traces, as discussed above. The classification accuracy for *AMRR* and *Sample Rate* is high (70% or higher) in all cases. Table I shows that this is the case even though the distribution of packets by transmission rate is different for the two experimental runs for each algorithm. What is similar about the two runs for each algorithm is the information content of the traces measured in terms of the last three metrics in Table I.

The interesting case in Figure 3 is *Minstrel*, where the classifier trained on data from *Experiment 1* performs poorly on test data from *Experiment 2* (case *m, 1–2*) with an accuracy of only 40%, but the classifier trained on *Experiment 2* performs very well on *Experiment 1* (case *m, 2–1*) with an accuracy of 100%. Table I shows that for all the three measures reported, *Set 2* has higher information content than *Set 1*. This means that the classifier trained on *Experiment 1* captures a limited set of the algorithm behavior compared to that present in *Experiment 2* test data, resulting in poor classifier accuracy. However, when the classifier is constructed with data from both *Experiment 1* and *Experiment 2*, its accuracy on *Experiment 2* tests increases dramatically, suggesting that enhancing a ported classifier with current data is beneficial.

These results have important implications for the practical use of our methods. They show that if the training set has high enough information content, the resulting classifier is portable, regardless of the difference in the distribution of packet transmission rates. Classifier portability is a desirable property in general because it reduces training time in new environments. Classifier portability is essential for wireless networks because the RF environment is constantly changing, and retraining every time is expensive. The results in this section also show that the accuracy of a poorly-performing classifier can be improved by updating it with training data from the current environment.

VII. Summary

In this paper, we describe a new methodology for accurately identifying the 802.11 rate adaptation algorithms deployed in a target wireless network. Our learning-based approach obviates the need to explicitly characterize the full spectrum of each rate adaptation algorithm's behavior, a difficult if not impossible task given the large number of possible rate change permutations. Our approach is practical because it is based entirely on passive monitoring of target networks. We test our methodology under varying network conditions and show that by using a comprehensive feature set, a classifier can be highly accurate, correctly identifying algorithms 95%-100% of the time for three of the four algorithms implemented by MadWifi, the most popular open source wireless interface driver. We show that careful selection of inputs for training the classifier is essential for high accuracy. We demonstrate the potential for classifier portability across a range of operating conditions. We envision rate adaptation algorithm fingerprinting becoming part of passive monitoring-based suites for wireless network performance analysis. In future work, we plan to construct classifiers for a wider variety of wireless interface drivers.

References

[1] S. Rayanchu, A. Mishra, D. Agrawal, S. Saha, and S. Banerjee, “Diagnosing Wireless Packet Losses in 802.11: Separating Collision from Weak Signal,” in *The 27th IEEE Joint Conference of the IEEE Computer and Communications Societies, INFOCOM*, Phoenix, AZ, USA, April 2008.

[2] M. Vutukuru, H. Balakrishnan, and K. Jamieson, “Cross-layer Wireless Bit Rate Adaptation,” in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Barcelona, Spain, August 2009.

[3] J. Bicket, “Bit-rate Selection in Wireless Networks,” Master's thesis, Massachusetts Institute of Technology, February 2005.

[4] Y.-C. Cheng, J. Bellardo, P. Benkö, A. C. Snoeren, G. M. Voelker, and S. Savage, “Jigsaw: Solving the Puzzle of Enterprise 802.11 Analysis,” in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pisa, Italy, September 2006.

[5] R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, “Analyzing the MAC-level Behavior of Wireless Networks in the Wild,” in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Pisa, Italy, September 2006.

[6] G. Holland, N. H. Vaidya, and P. Bahl, “A Rate-Adaptive MAC Protocol for Multi-Hop Wireless Networks,” in *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking, MOBICOM*, Rome, Italy, July 2001.

[7] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. W. Knightly, “Opportunistic Media Access for Multirate Ad Hoc Networks,” in *Proceedings of the Eighth Annual International Conference on Mobile Computing and Networking, MOBICOM*, Atlanta, Georgia, USA, September 2002.

[8] S. H. Y. Wong, S. Lu, H. Yang, and V. Bharghavan, “Robust Rate Adaptation for 802.11 Wireless Networks,” in *Proceedings of the Seventh Annual International Conference on Mobile Computing and Networking, MOBICOM*, Los Angeles, CA, USA, September 2006.

[9] J. Kim, S. Kim, S. Choi, and D. Qiao, “CARA: Collision-Aware Rate Adaptation for IEEE 802.11 WLANs,” in *25th IEEE International Conference on Computer Communications, Joint Conference of the IEEE Computer and Communications Societies, INFOCOM*, Barcelona, Spain, April 2006.

[10] A. Kamerman and L. Monteban, “WaveLAN-II: A High-Performance Wireless LAN for the Unlicensed Band,” *Bell Labs Technical Journal*, Summer 1997.

[11] M. Lacage, M. H. Manshaei, and T. Turletti, “IEEE 802.11 Rate Adaptation: A Practical Approach,” in *MSWiM '04: Proceedings of the 7th ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, 2004.

[12] G. Judd, X. Wang, and P. Steenkiste, “Efficient Channel-aware Rate Adaptation in Dynamic Environments,” in *Proceedings of the 6th International Conference on Mobile Systems, Applications, and Services, MobiSys*, Breckenridge, CO, USA, June 2008.

[13] J. Camp and E. W. Knightly, “Modulation Rate Adaptation in Urban and Vehicular Environments: Cross-layer Implementation and Experimental Evaluation,” in *Proceedings of the 14th Annual International Conference on Mobile Computing and Networking, MOBICOM*, San Francisco, California, USA, September 2008.

[14] C. Yiu and S. Singh, “A Model for Comparing Rate Adaptation Algorithms,” in *Proceedings of the Fourth ACM Workshop on Wireless Network Testbeds, Experimental Evaluation and Characterization, WINTECH*, Beijing, China, September 2009.

[15] R. Daniels and R. Heath, “Online Adaptive Modulation and Coding with Support Vector Machines,” in *Proceedings of the IEEE European Wireless Conference*, Lucca, Italy, April 2010.

[16] B. Schölkopf and A. J. Smola, *Learning With Kernels*. Cambridge, MA: MIT Press, 2002.

[17] “Multi-Class Support Vector Machine.” [Online]. Available: http://svmlight.joachims.org/svm_multiclass.html

[18] M. Mirza, J. Sommers, P. Barford, and X. Zhu, “A Machine Learning Approach to TCP Throughput Prediction,” in *The International Conference on Measurement and Modeling of Computer Systems, ACM SIGMETRICS*, San Diego, CA, USA, June 2007.

[19] M. Mirza, K. Springborn, S. Banerjee, P. Barford, M. Blodgett, and X. Zhu, “The Challenges of TCP Throughput Prediction for Wireless Networks,” in *In Proceedings of the IEEE Communications Society Conference on Sensor, Mesh, and Ad Hoc Communications and Networks, SECON 2009*, Rome, Italy, June 2009.

[20] “The MadWifi Project.” [Online]. Available: http://madwifi-project.org

[21] “Atheros Wireless LAN 2.4/5-GHz 802.11a/b/g 108 Mbos Turbo Radio-on-a-Chip WLAN Networking Products and Technology Overview.” [Online]. Available: http://www.atheros.com/pt

[22] “Implementation of the Onoe Algorithm.” [Online]. Available: http://madwifi-project.org/browser/madwifi/branches/madwifi-0.9.4/ath_rate/onoe/onoe.c

[23] “Description of the Minstrel Algorithm.” [Online]. Available: http://madwifi-project.org/browser/madwifi/branches/madwifi-0.9.4/ath_rate/minstrel/minstrel.txt

[24] “AirPcap.” [Online]. Available: http://www.cacetech.com/products/airpcap.html