Robust Rate Adaptation for 802.11 Wireless Networks

Starsky H.Y. Wong[1], Hao Yang[2], Songwu Lu[1] and Vaduvur Bharghavan[3]
Dept. of Computer Science, UCLA, 4732 Boelter Hall, Los Angeles, CA 90025 [1]
IBM T.J. Watson Research, 19 Skyline Drive, Hawthorne, NY 10532 [2]
Meru Networks, 1309 South Mary Avenue, Sunnyvale, CA 94087 [3]
{hywong1,slu}@cs.ucla.edu[1], haoyang@us.ibm.com[2], bharghav@merunetworks.com[3]

ABSTRACT

Rate adaptation is a mechanism unspecified by the 802.11 standards, yet critical to the system performance by exploiting the multi-rate capability at the physical layer. In this paper, we conduct a systematic and experimental study on rate adaptation over 802.11 wireless networks. Our main contributions are two-fold. First, we critique five design guidelines adopted by most existing algorithms. Our study reveals that these seemingly correct guidelines can be misleading in practice, thus incur significant performance penalty in certain scenarios. The fundamental challenge is that rate adaptation must accurately estimate the channel condition despite the presence of various dynamics caused by fading, mobility and hidden terminals. Second, we design and implement a new Robust Rate Adaptation Algorithm (RRAA) that addresses the above challenge. RRAA uses short-term loss ratio to opportunistically guide its rate change decisions, and an adaptive RTS filter to prevent collision losses from triggering rate decrease. Our extensive experiments have shown that RRAA outperforms three well-known rate adaptation solutions (ARF, AARF, and SampleRate) in all tested scenarios, with throughput improvement up to 143%.

Categories and Subject Descriptors: C.2.1 [Computer-Communication Networks]:Network Architecture and Design[Wireless communication]

General Terms: Design, Experimentation, Performance

Keywords: Rate Adaptation, 802.11

1. INTRODUCTION

Rate adaptation is a link-layer mechanism critical to the system performance in IEEE 802.11-based wireless networks, yet left unspecified by the 802.11 standards. The current 802.11 specifications mandate multiple transmission rates at the physical layer (PHY) that use different modulation and coding schemes. For example, the 802.11b PHY supports four transmission rates (1∼11 Mbps), the 802.11a PHY offers eight rates (6∼54Mbps), and the 802.11g PHY supports twelve rates (1∼54Mbps). To exploit such multi-rate capability, a sender must select the best transmission rate and dynamically adapt its decision to the time-varying and location-dependent channel quality, without explicit information feedback from the receiver. Such an operation is known as *rate adaptation*. Given the large numerical span among the available rate options, rate adaptation plays a critical role on the overall system performance in 802.11-based wireless networks, such as the widely deployed WLANs and the emerging mesh networks.

In recent years, a number of algorithms for rate adaptation [1, 2, 12, 8, 3, 4, 10, 5, 6, 7, 9] have been proposed in the literature, and some [1, 12, 8] have been used in real products. Their basic idea is to estimate the channel quality and adjust the transmission rate accordingly. This is typically achieved by using a few metrics collected at the sender and the associated design rules. The widely used metrics include probe packets [1, 2, 8], consecutive successes/losses [1, 2, 6], PHY metrics such as SNR [4, 3, 6], and long-term statistics [12]. Examples of the commonly used rules include increasing rate upon consecutive successes, using probe packets to assess new rates, etc. While all such metrics and rules seem intuitively correct and each design has its own merits, little is known about how effectively they perform in a practical setting. The fundamental problem is that real-world wireless networks exhibit rich channel dynamics including random channel errors, mobility-induced channel variation, and contention from hidden stations. Each of the above metrics and associated design rules has limited applicable scenarios. Consequently, each design has its own Achille's heel.

In this paper, we conduct a systematic and experimental study to expose the challenges for rate adaptation and explore new design space. To this end, we first use experiments and simple analysis to critically examine five design guidelines followed by most existing algorithms. These guidelines include: (1) *decrease transmission rate upon severe packet loss*, (2) *use probe packets to assess the new rate*, (3) *use consecutive transmission successes/losses to decide rate increase/decrease*, (4) *use PHY metrics to infer new transmission rate*, and (5) *long-term smoothened operation produces best average performance*. For experimental comparison, we implement three popular algorithms (ARF [1], AARF [2], SampleRate [8]) on a programmable AP platform, together with the ONOE algorithm [12] available in MADWiFi [17]. We not only identify the issues with these algorithms using experiments, but also take a microscopic view of their runtime behavior and gain insights on the root causes of the issues. Our experiments surprisingly show that the above

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
MobiCom'06, September 23–26, 2006, Los Angeles, California, USA.
Copyright 2006 ACM 1-59593-286-0/06/0009 ...$5.00.

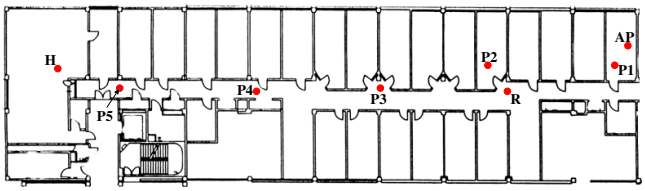


Figure 1: Experimental floor plan.

five seemingly valid guidelines can be quite misleading in practice, and may incur significant performance penalty of up to 70% throughput drop. In fact, we even discovered that with mild link-layer contention, these rate adaptation designs not only fail to facilitate throughput improvement, but also reduce the throughput and aggravate channel contention because rate decrease is falsely triggered.

To address these challenges, we design and implement a Robust Rate Adaptation Algorithm (RRAA) based on two novel ideas. First, we use *short-term loss ratio* in a window of tens of frames to opportunistically guide the rate selection. Such a loss ratio provides not only fresh but also dependable information to estimate the channel quality. Second, we leverage the per-frame RTS option in the 802.11 standards, and use an adaptive RTS filter to suppress collision losses with minimal overhead. We implement RRAA on a programmable AP platform and evaluate its performance using thorough experiments as well as field trials. Our results show that RRAA consistently outperforms three well-known algorithms of ARF, AARF and SampleRate in all scenarios with 802.11a/b channels, static/mobile stations, TCP/UDP flows, with/without hidden stations, and in controlled/uncontrolled environments. The throughput improvement of RRAA over these algorithms can be as high as 143% in realistic field trials.

The two key contributions of this paper are as follows. First, we provide a systematic critique on five design guidelines in the state-of-art rate adaptation algorithms. Second, we design, implement and evaluate a robust rate adaptation algorithm, which addresses all these identified issues and is fully compliant with the 802.11 standards.

The rest of the paper is organized as follows. Section 2 introduces the background and Section 3 describes our experimental system and methodology. Section 4 examines five design guidelines in existing rate adaptation algorithms. Section 5 presents the design of our robust rate adaptation algorithm, and Section 6 describes the implementation and evaluates its performance. Section 7 discusses the related work, and Section 8 concludes the paper.

2. BACKGROUND

We consider a practical 802.11-based wireless LAN or mesh network scenario. Both clients and access points (APs)/mesh routers use 802.11 a/b/g devices. The clients may roam but the APs/mesh routers are typically static. The physical layer operates at 2.4GHz for 802.11b/g or at 5Ghz for 802.11a, which only has a limited number (e.g., 3 for 802.11 b/g) of independent channels. Thus, multiple APs within a geographic locality may have to share one of these channels. This readily leads to *hidden stations* among APs and clients. In our campus building environment, we can easily sniff about 4∼10 APs and 50∼100 clients on any given 802.11 b/g Channel 1, 6, or 11 during regular office hours, and they may act as hidden stations among one another.

The default operation mode for wireless LAN/mesh network is the 802.11 DCF, in which an DATA-ACK exchange is performed between the client and the AP/mesh router after carrier sensing. To avoid collisions from hidden stations, the 802.11 standards recommend to use RTS/CTS handshake. However, in practice RTS/CTS is turned off in most deployed wireless LANs to reduce the signaling overhead.

Rate adaptation allows for each device to adapt the runtime transmission rate based on the dynamic channel condition. It has been used by all 802.11a/b/g devices in reality. An 802.11b device can use four rate options of 1, 2, 5.5, 11Mbps. An 802.11a device can use eight rate options of 6, 9, 12, 18, 24, 36, 48, 54Mbps. An 802.11g device can use all twelve rate options. The goal of rate adaptation is to maximize the transmission goodput at the receiver[1]. It exploits the PHY multi-rate capability and enables each device to select the best rate out of the mandated options. Rate adaptation is typically implemented at both the AP and the client, and the exact algorithm is left to the vendors.

In the current 802.11 standard, a receiver does not provide explicit feedback information on the best rate or perceived SNR to the sender. Therefore, most practical rate adaptation algorithms [1, 2, 12, 8, 6, 10] make decisions solely based on the ACK, which is sent upon successful delivery of a DATA packet[2]. The sender assumes a transmission failure if it receives no ACK before a timeout.

3. EXPERIMENTAL METHODOLOGY

The evaluation results presented in this paper are all obtained from real experiments. In this section, we describe our experimental platform, setup and methodology.

Programmable AP Platform Our experiments are conducted over a programmable AP platform. The AP uses Agere 802.11a/b/g chipset and supports all three types of clients. The 802.11 MAC is implemented in the FPGA firmware, to which we have access. The platform has several appealing features that facilitate our research on rate adaptation. First, we can program our own rate adaptation algorithm, import other existing algorithms, and run them at the AP. Second, it provides per-frame control functionalities. We can perform per-frame tracing of various metrics of interests, such as frame retry count and SNR value. The maximum retry count and RTS option can be configured in real time on a per-frame basis. We may also control the transmission rate for each frame retry, similar to the MADWiFi device driver for Atheros chipset. Third, it supports all 802.11a/b/g channels and devices, and is compliant to 802.11 standards. Fourth, the feedback delay from the hardware layer is small; this implies that timely link-layer information is available to rate adaptation.

Experimental Setup We conduct all our experiments in a campus setting. Figure 1 shows the floor-plan of the building where we run experiments. Spot AP is the location of the programmable AP, and spot H is another AP spot inside a conference room. The AP at H periodically broadcasts packets for its clients, but acts as a hidden station for

[1]Goodput and throughput carry the same meaning in this paper.

[2]In this paper, we use packets and frames interchangeably, with a slight abuse of notation.

the programmable one at AP. Spots $P1$, $P2$, $P3, P4, P5, R$ represent six different locations where the receiving clients are placed during the experiments.

In most cases, the AP serves as the sender for the wireless traffic since we only implement the algorithms on the AP. All client devices run the Linux 2.6 kernel with CISCO Aironet 802.11a/b/g Adapters. The wireless device driver on the client side is MADWiFi.

Experimental Methodology We run experiments using various settings with static/mobile clients, on 802.11 a/b channels, with/without hidden stations. All these scenarios occur in realistic 802.11 networks. The static settings evaluate the stability and robustness of an algorithm, i.e., whether it can stabilize around the optimal rate and how sensitive it is to random losses. The mobility settings evaluate how responsive an algorithm is in adapting to significant channel variations perceived by mobile clients. Finally, the hidden-station settings assess how an algorithm performs under collision losses. This setup represents the real-life scenario of ad-hoc deployment of 802.11b/g APs sharing the 2.4GHz channels in residential, campus, or city environments.

To conduct repeatable experiments and provide fair comparison among different algorithms, we perform most experiments in a controlled manner to minimize the impact of external factors, such as people walking around, microwaves, and traffic from other AP or client devices. Specifically, these experiments are done during midnight when the offices are empty, and we use an additional sniffer to ensure no background traffic exists over the channel in use. We use Channel 14 for 802.11b experiments because our sniffer can detect at least 4 APs and 30∼50 clients over each of Channels 1, 6, 11 at all time. For 802.11a experiments, we use Channel 60, and no AP or client is using this channel during our experiments. Finally, we run a set of uncontrolled field trials, using 802.11b Channel 6, during the office hours to evaluate how different algorithms perform in realistic scenarios.

We conduct each experiment for multiple runs, and the result presented is the average over all runs. Each run lasts for 60 seconds in the static setting and the hidden station cases. Each mobility test lasts about 200 seconds. We experiment with both UDP and TCP traffic. For UDP, We use *iperf* [14] as the traffic generator, and vary the sending rate from 5Mbps to 30Mbps at an increment of 5Mbps. The results reported in the paper are based on the sending rate that produces the highest goodput. We also vary the packet size in our experiments. To stay focused, we only present the results with 1300-byte packets in this paper.

We evaluate and compare five rate adaptation algorithms in our experiments. Four of them, i.e., ARF [1], AARF [2], SampleRate [8], and the proposed RRAA, are implemented by us and run on our programmable AP. The fifth one, ONOE [12], is implemented by MADWiFi [17], and runs on the Linux client devices.

These selected algorithms provide a good sample of representative designs in the literature. ARF is the first rate adaptation design for 802.11 networks and has inspired many followup proposals [2, 5, 6]. It sends a probe packet upon either 10 consecutive transmission successes or timeout of 15 transmissions. A probe packet is sent at a rate higher than the current one in use. If the probe packet succeeds, ARF increases the transmission rate. Meanwhile, ARF decreases the rate upon two consecutive transmission failures. While the probing threshold is fixed as 10 consecutive successes in ARF, AARF improves the stability by doubling the probing threshold (with a maximum bound of 50) when a probe packet fails. The probing threshold is reset to its initial value of 10 whenever the rate is decreased. SampleRate is arguably the best reported algorithm for static settings. It maintains the expected transmission time for each rate, and updates it after each transmission. A frame is transmitted at the rate that currently has the smallest expected transmission time. In addition, for every 10 frames, SampleRate also sends one probe packet at another randomly selected rate. While in principle receiver-based designs such as RBAR [3] and OAR [4] may yield good performance, they cannot be implemented on current 802.11 devices without modifying the standard. Thus, we are unable to implement and evaluate such algorithms.

4. ON STATE-OF-THE-ART RATE ADAPTATION ALGORITHMS

Most current rate adaptation designs follow a few guidelines that are conceptually intuitive and seemingly effective. However, our study shows that they can be misleading in practice and incur significant performance penalty. In this section, we first revisit the solution space and categorize the existing designs in Section 4.1. We then critically examine five commonly adopted design guidelines using experiments and simple analysis in Section 4.2.

4.1 Solution Space for Rate Adaptation

At its core, each rate adaptation algorithm should possess at least two basic mechanisms:

Estimation: It either directly estimates the best transmission rate based on the current channel and network conditions, or indirectly infers the best rate by gauging how well the currently chosen rate performs.

Action: Given the latest estimation result, it decides when and how the transmission rate is updated.

Based on how these two mechanisms are implemented, we can categorize various rate adaptation designs into several general approaches.

4.1.1 Estimation

To design the estimation mechanism, one must address the following two questions. First, what information can be used in the estimation? Specifically, which layer is the information collected from, and what types of messages are taken into account? Second, given the collected information, how should the best transmission rate be estimated?

Which layer to use: There are three approaches that collect information from different layers of the protocol stack. The first one is the *Physical-layer* approach that uses SNR or other PHY metrics to directly estimate the channel quality (e.g., RBAR [3], or OAR [4]). The second one is the *Link-layer* approach that uses frame transmission results to indirectly infer the channel quality (e.g., ARF [1], AARF [2], SampleRate [8], ONOE [12]). The last one is the *Hybrid* approach that combines both PHY and link-layer information (e.g., HRC [6]).

Which message to use: The link-layer information can be collected based on either data or signaling frames, or both. For the *Data-frame* approach, rate adaptation uses

only the data frames to assess the channel quality. It can be further classified into two subcategories. The first one is a *probing* approach in which a few data frames are occasionally transmitted at a rate different from the current one to "probe" the channel (e.g., ARF [1], AARF [2], SampleRate [8]). The second one is the *non-probing* approach that never sends out probe frames (e.g., our proposed RRAA). For the *Signaling-frame* approach, rate adapataion takes into account the RTS/CTS handshake to better infer the causes of frame losses (e.g., RBAR [3], OAR [4], CARA [10]).

How to estimate: The physical-layer approach typically translates the measured SNR into a best transmission rate based on pre-defined mappings. On the other hand, the link-layer (or hybrid) approach needs to estimate the channel quality based on the outcome of previously transmitted frames. The estimation can be done via: 1) *Deterministic pattern* that treats consecutive frame successes/failures as the indication of good/bad channel condition (e.g., ARF [1], AARF [2]), and 2) *Statistical metrics* that use long-term or short-term frame statistics to statistically estimate the best possible rate (e.g., SampleRate [8] or RRAA).

4.1.2 Action

In general, there are two approaches to adjusting the rate based on the aforementioned estimation. The first approach is *Sequential rate adjustment*, which means to increase (or decrease) the current rate by one level when the channel is good (or bad). In other words, the rate is adjusted by at most one level at a time. The second one is *Best rate adjustment*, which means to immediately change the rate to the one that may yield the best performance. In such cases, the rate may jump or drop by multiple levels at a time.

4.2 Critiques on Current Design Guidelines

Given the above solution space for rate adaptation, the state-of-the-art algorithms have been using several design guidelines streamlined through the extensive practice of various algorithms over the past fifteen years. In this section, we use real experiments to provide a critique on them. We show that while such guidelines are useful in certain presumed scenarios, they can be misleading in other cases. In the worst case, they yield unexpected, erroneous results.

4.2.1 Guideline #1: Decrease transmission rate upon severe packet loss

The first guideline states that, whenever severe packet loss occurs, rate adaptation should decrease its current transmission rate. This has been widely used in almost all existing algorithms [1, 2, 12, 8, 6]. Severe packet loss is typically detected via excessive retry counts, beyond-threshold frame losses, or successive transmission failures. The original motivation for this rule is that, whenever the link condition between the sender and the receiver deteriorates and thus incurs significant losses at the current rate, the sender switches to lower rates to adapt to the worsening channel condition.

The above rule is easily broken in practice when hidden stations exist. In the presence of hidden stations, a receiver may experience significant packet losses. This subsequently triggers rate adaptation at the sender to decrease its rate according to the stated guideline. However, the sender should *not* decrease its transmission rate upon hidden-station induced losses, because reducing the rate cannot solve the contention problem. In fact, reducing the rate will make channel contention even worse because it prolongs the transmission time for each packet, which aggravates channel collisions and further reduces the rate.

	ARF	AARF	SampleRate	FixedRate
Goodput (Mbps)	0.65	0.56	0.58	1.46
Loss Ratio	61%	60%	59%	60%

Table 1: Performance of different rate adaptation algorithms in the presence of hidden stations.

Our experiment also confirms the above findings as shown in Table 1. We place a hidden AP at spot H, a receiver at spot R, and a sender at spot AP. The sending AP and the hidden AP are not aware of each other. We compare ARF, AARF, and SampleRate with the operation of turning off rate adaptation and using fixed rate (called FixedRate). In the absence of hidden AP, all algorithms are sending more than 95% of packets at the highest rate 11Mbps, and the frame loss ratio is also relatively low at about 5.5%. However, when the hidden AP starts to broadcast packets to its clients at a mild rate of 0.379Mbps, receiver R experiences about 60% losses for all algorithms. The heavy loss triggers the sender's rate adaptation algorithm to decrease its sending rate to 1Mbps. Overall, this leads to the throughput around 0.56∼0.65Mbps for ARF, AARF, and SampleRate. In contrast, if we turn off rate adaptation and fix the rate at 11Mbps, the throughput is 1.46Mbps, about 124.6% to 160.7% higher than that of the three algorithms. These experimental results show that improper rate adaptation not only fails to improve the system performance, but also reduces the achievable throughput. More results in the hidden station environment will be presented in Section 6.

The fundamental problem is that rate adaptation may experience much richer set of packet loss scenarios in practice, which are well beyond the simplistic one of only fading/path loss envisioned by the original designs. The guideline of decreasing rate upon severe packet loss does not apply in other lossy scenarios. The rate adaptation solution has to differentiate various losses and react accordingly.

4.2.2 Guideline #2: Use probe packets to assess possible new rates

The second guideline uses one or multiple *probe* packets to learn the channel status at transmission rates other than the one currently in use. The probe packets are data frames sending at a *different* transmission rate. The goal is to determine whether other rates will yield better performance. If the results from such probe packets indeed lead to higher throughput, rate adaptation will switch to the new, typically higher rate. In the existing algorithms, probe packets will be sent out at the next higher rate after multiple successful transmissions at the current rate [1, 2, 6], or at a randomly selected rate once every tens of packets [8].

The above design guideline also has two downsides, as illustrated by our simple analysis. First, a successful probe can be misleading and trigger incorrect rate increase. In the literature, algorithms such as ARF, AARF, and HRC use a single probe packet to assess the channel at a higher rate. Whenever such a single probe packet is successfully transmitted at the higher rate, such algorithms decide to increase the rate accordingly. Now consider the following realistic case. The sender and the receiver have near-perfect trans-

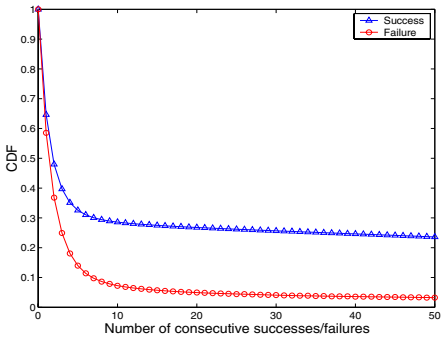


Figure 2: CDF of an additional success/failure transmission after n consecutive success/failure transmissions.

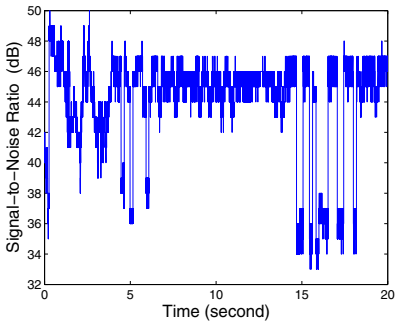


Figure 3: Evolution of SNR over time.

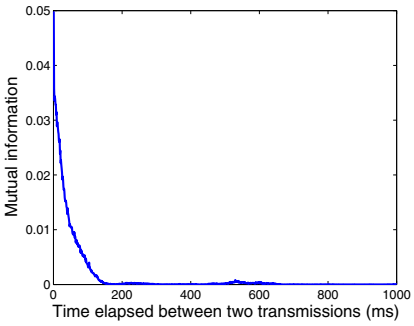


Figure 4: Mutual Information of two packets separated by x ms in time.

missions at 12Mbps, but suffer from 40% loss at a higher rate of 18Mbps. That means, a single probe sent at 18Mbps has 60% chance to get through. However, the expected throughput for sending packets at 18Mbps with 40% loss is smaller than the one at 12Mbps with near zero loss. From our observations, most immediate higher rates in reality have less than 50% loss percentage. Therefore, the chance of success for a single probe is usually higher than 50%, and such a success can be misleading.

Second, an unsuccessful probe can incur severe penalty on future rate adaptation. We use the SampleRate algorithm [8] as an example to illustrate the problem. In SampleRate, probe packets are regularly sent at every ten-packet interval and transmitted at a randomly chosen rate. The implementation of SampleRate in MadWiFi uses exponential weighted moving average (with a weighting factor of 0.05 for the latest sample) to statistically update the per-packet expected transmission time at a given rate. The real issue is that such a statistical update based on probe is too sensitive to (possibly rare) failure of probe packets. This happens when the expected transmission times for two rates are very close. For example, in 802.11a/g devices, the lossless transmission times at 54Mbps and 48Mbps are 534ms and 560ms[3], respectively, for 1400B packets. Consider the case where SampleRate currently operates at 48Mbps but probes at 54Mbps. A single probe failure (say, the total retry count is 4) at 54Mbps will update the expected transmission time at 54Mbps as 625ms. Therefore, this single probe failure prevents the rate adaptation from switching to 54Mbps for an extended period of time. A detailed calculation shows that it takes 25 lossless probes for SampleRate to change 625ms into a value smaller than 560ms, the lossless time at 48Mbps. Considering that each probe packet is sent once every ten frames, it takes 250 frame transmissions for SampleRate to eventually switch to 54Mbps. The consequence is that this suboptimal rate reduces throughput even with rare probe failure. Our real experiments also confirm this discovery in reality, as we will document in Section 6.

The fundamental problem is that statistically small number of probe samples may yield inaccurate rate adaptation. It can be overly optimistic upon a probe success, or too pessimistic upon a probe failure.

[3]We obtain these values by using the equation given in [8].

4.2.3 *Guideline #3: Use consecutive transmission successes/losses to increase/decrease rate*

The third guideline states that, upon multiple consecutive transmission successes (say, 10 in ARF [1], AARF [2], and HRC [6]), the current rate should be increased to the next higher rate; upon back-to-back transmission failures (say, 2 in ARF, AARF and HRC, or 4 in SampleRate), the rate should be decreased to a lower one.

Our experiments show that the above design guideline is not valid in many practical scenarios. In our experiments, we turn off rate adaptation as well as the frame retry. We place the AP and the client at different spots, and manually fix the transmission rate which gives the highest throughput, for each experimental run. The success/failure event for each packet transmission is recorded inside the AP. The cumulative distribution probability for n consecutive transmission successes/failures is plotted in Figure 2.

We make two observations from Figure 2. First, the probability for a failed transmission following at least two consecutive failures is only 36.8%. This result means that with 63.2% chance, a successful tranmission will occur after two or more consecutive failures. Therefore, decreasing current rate upon consecutive transmission failures may not be the right choice statistically. Second, the probability for a successful transmission following at least 10 consecutive successes is only 28.5%. Thus, the chance for a failed transmission following 10 consecutive successes is 71.5%. This result indicates that consecutive transmission successes cannot be a reliable metric to predict the next transmission rate.

The fundamental problem is that most realistic scenarios exhibit randomly distributed loss behaviors. Any deterministic pattern of transmission successes/failures may not occur with large probability in all cases.

4.2.4 *Guideline #4: Use PHY metrics like SNR to infer new transmission rate*

The next guideline suggests to use the physical-layer metrics, such as the signal-to-noise ratio (SNR), for rate estimation. This has been used to estimate rates in the presence of client mobility in HRC [6]. It has also been applied in the receiver-based rate adaptation algorithms such as RBAR [3] and OAR [4]. In theory, such physical-layer metrics may indeed lead to an accurate rate estimation.

However, the above design rule encounters severe difficulties in practical 802.11 systems for two reasons. First,

Sampling intervals (ms)	5000	1000	500	100
UDP Goodput (Mbps)	14.9	15.3	16.5	17.1

Table 2: Performance of ONOE with different sampling intervals.

earlier experimental studies [13, 8] already show that there is no strong correlation between SNR and delivery probability at a rate in a general case. Second, our experiments show that the SNR variations also make the rate estimation highly inaccurate. In our experiment, we send back-to-back UDP packets from the AP to the client (located at P1) and sample the SNR value. Figure 3 plots the SNR value for each received packet. The figure indicates that it is common for the SNR value to have variations of 5dB between consecutive transmissions. In some cases, the variation can be as large as 10∼14dB. This large SNR variation can easily lead to more than one-rate-level deviation out of the multi-rate options when translating to the transmission rate, based on the goodput versus SNR mapping (Figure 7 of [5]).

In our experiments, we also tried other two physical-layer metrics, the background energy level (with the intention to differentiate fading loss and collision loss), and received signal strength indication (RSSI). However, neither metric can be directly used to estimate the rate accurately, nor can one be used to differentiate loss causes.

4.2.5 *Guideline #5: Long-term smoothened operation produces best average performance*

This guideline suggests to use long-term smoothened operation in the presence of random losses over the channel. The operation can be either rate estimation or rate change action, or both. In rate estimation, this rule recommends to use long-term statistical information to estimate the optimal transmission rate. For example, popular algorithms ONOE [12] and SampleRate [8] both collect packet-level statistics (in terms of loss and throughput) over a period of one to ten seconds. In rate change decision, this rule suggests to only change rates infrequently, say, once every 1 or 10 seconds. In both cases, the underlying hypothesis is that long-term estimation/action will smoothen out the impact of random errors and lead to best average performance. Our experiment and analysis based on information theory invalidate both claims.

We first show through experiments that long-term rate estimation and rate change action over large sampling periods will not yield best average performance. The experiment is conducted using the ONOE algorithm implemented in MADWiFi. ONOE uses one second as the default sampling interval. It changes its rate based on the packet-level loss statistics collected over each sampling period. In the experiment, the sender located at P2 of Figure 1 uses ONOE to send packets to the AP. We vary the sampling period and the results are given in Table 2. The table clearly shows that small sampling period of 100ms actually produces the best average performance in the long term. Using large sampling period may lead to 12.9% throughput reduction. In fact, similar results have also been reported in early studies (Figure 3.5 of [8]). One reason for this performance drop is that the algorithm is unable to exploit the short-term *opportunistic gain* over the wireless channel, which typically occurs at the time scale of hundreds of milliseconds.

We next use the concept of *mutual information* [16] to show that long-term rate estimate over large sampling periods does not help even in the presence of random loss. Mutual information indicates the mutual dependency of two random variables, i.e., how much information one random variable can tell about the other. We treat the transmission success/failure event at a given time as a random variable and calculate the mutual information for two events at different time instants. In an experimental setting similar to section 4.2.3, we disable rate adaptation and the frame retry, and record the time for each success/failure transmission. We then calculate the mutual information for each pair of packets separated by an interval of x ms. Figure 4 plots the mutual information evolution with respect to different x. The figure shows that their mutual information becomes negligible when two packets are separated by more than 150ms over time. This implies that the success/failure event occurred 150ms earlier can barely provide any useful information for the current rate estimation. We also conduct similar experiments at different locations. All results show that mutual information diminishes when the sampling period becomes larger than 150∼250ms. This experimental result shows that large sampling periods, ranging from a few seconds to tens of seconds, do not lead to more accurate rate estimation. In this case, how to assign different weighting factors for samples over time becomes a challenging issue.

The next experiment shows that long-term, infrequent rate change decision may also lead to performance penalty. The experiment is set up for the client mobility case, in which a person carries the receiver and walks at approximately constant pedestrian speed of 1m/s. The experiment is in 802.11b and the route is $P1 \rightarrow P2 \rightarrow P3 \rightarrow P4 \rightarrow P5 \rightarrow P4 \rightarrow P3 \rightarrow P2 \rightarrow P1$ (see Figure 1), and each trip takes about 200 seconds. In the test, we compare the performance of ARF and SampleRate[4]. Both ARF and SampleRate use relatively short-term rate estimation. ARF sends a probe packet no later than 15 transmissions. SampleRate implementation uses EWMA with a factor of 0.05, which implies that roughly only the recent 50 samples carry major weights in the estimation. However, the rate change actions in both algorithms are quite different. ARF allows for rate change every 10 or 15 packets, while SampleRate takes 2 seconds to switch to a new rate (unless four consecutive losses trigger rate decrease). Our experimental results show that, the average UDP goodput for ARF and SampleRate are 3.85Mbps and 3.50Mbps, respectively. ARF performs 10% better than SampleRate in the mobile client case, which shows that the delayed rate-change decisions hurt the responsiveness of SampleRate.

5. DESIGN

In this section, we present the design of RRAA, a Robust Rate Adaptation Algorithm for 802.11-based wireless networks. Overall, RRAA tries to maximize the aggregate throughput in the presence of various channel dynamics. Specifically, it seeks to achieve the following goals:

[4]We implemented SampleRate based on their source codes in MADWiFi, which is different from the algorithm described in [8]. First, while [8] averages the transmission time over a 10-second window, the implementation uses EWMA without any window. Second, while [8] suggests per-packet rate decision, the rate is only changed every 2 seconds or upon four consecutive losses in the implementation.

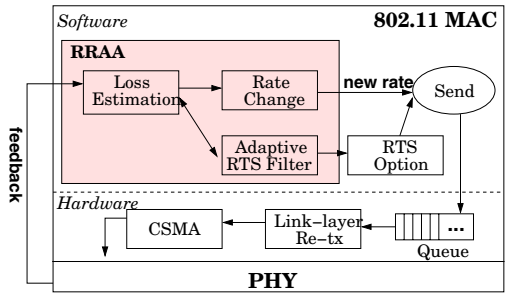


Figure 5: Modules and interactions in RRAA.

- Robust against random loss: The design should maintain stable rate behavior and throughput performance in the presence of mild, random channel variations.
- Responsive to drastic channel changes: The algorithm should respond quickly to significant channel changes. Specifically, we want the design to be highly responsive in the following two scenarios: (a) The algorithm is able to quickly track the rate decrease/increase associated with the channel change, when the channel quality deteriorates/improves as a mobile user walks away/towards the AP. (b) The algorithm is able to respond properly in the presence of severe channel degradation induced by interfering sources, e.g., hidden terminals due to other 802.11 devices, and sources such as microwaves and cordless phones operating in the same frequency band.

While the existing algorithms may achieve certain aspects of the above goals, they are unable to achieve all of them.

The design of RRAA is based on two novel ideas. First, it uses *short-term loss ratio* to assess the channel and opportunistically adapt the runtime transmission rate to dynamic channel variations. Second, it leverages the RTS option in an adaptive manner to filter out collision losses with small overhead. As shown in Figure 5, RRAA consists of three closely interacting modules:

- *Loss Estimation*: It assesses the channel condition by keeping track of the frame loss ratio within a short time window ($5 - 40$ frames in our implementation).
- *Rate Change*: It decides whether to increase, decrease, or retain the rate based on the estimated loss ratio.
- *Adaptive RTS Filter*: It selectively turns on RTS/CTS exchange to suppress collision losses, and adapts the number of RTS-protected frames to the collision level.

We next describe the loss estimation and rate change design in Section 5.1, then the adaptive RTS filter in Section 5.2, and finally their integration in Section 5.3.

5.1 Loss Estimation and Rate Change

To ensure standard compatability, RRAA uses only the link-layer information of frame successes or losses in deciding the transmission rate. However, unlike existing link-layer-based rate adapatation algorithms [1, 2, 8], RRAA never uses probe packets to assess possible new rates. Instead, RRAA always adjusts the rate based on the frame loss ratio over the previous *short-term* time window. Compared to a probe frame which either succeeds or fails, the loss ratio over many transmission samples provides more dependable information to estimate the rate. As a result, the rates selected by RRAA are highly robust to the random channel losses.

```
1  R=highest_rate;
2  counter=ewnd(R);
3  while true do
4     rcv_tx_status(last_frame);
5     P = update_loss_ratio();
6     if( counter == 0 )
7        if     (P > PMTL) then R = next_lower_rate();
9        elseif (P < PORI) then R = next_high_rate();
10       counter = ewnd(R);
11    send(next_frame,R);
12    counter--;
```

Figure 6: Loss estimation and rate change in RRAA-BASIC.

The loss estimation and rate change algorithm in RRAA is illustrated in Figure 6, which we refer to as RRAA-BASIC in this paper. In RRAA-BASIC, each rate is associated with three parameters: an estimation window size, a *Maximum Tolerable Loss* threshold (MTL), and an *Opportunistic Rate Increase* threshold (ORI). We will describe in Sections 5.1.1 and 5.1.2 how these parameters are chosen to select the best rate. For simplicity, we first assume identical lengths for all frames, then relax this assumption in Section 5.1.3.

The algorithm starts with the highest rate (i.e., 54 Mbps for 802.11a/g or 11 Mbps for 802.11b), and adapts the transmission rate in the following manner. Whenever a new rate is chosen, it is used to transmit the next *ewnd* frames, which we call an *estimation window*. The loss ratio is estimated based on how many frames over this window are lost. Specifically, the runtime loss ratio is calculated as

$$P = \frac{\#\_lost\_frames}{\#\_transmitted\_frames} \quad (1)$$

where both numbers of lost frames and transmitted frames are counted over the window and include all re-tries. When the window finishes, a new rate is chosen based on the estimated loss ratio. As shown in Lines 7-11 of Figure 6, the rate is decreased to the next lower one if the loss ratio is larger than P_{MTL}. It is increased to the next higher one if the loss ratio is smaller than P_{ORI}. In these two cases, a new estimation window starts for the newly selected rate. However, if the loss ratio is between P_{MTL} and P_{ORI}, the current rate is retained, and the estimation window keeps *sliding forward*, until the loss ratio of the most recent *ewnd* frames leads to a rate change. Once the rate is changed, a new estimation window is started accordingly.

The intuition behind the above algorithm is that a sufficiently low (high) loss ratio indicates good (bad) channel conditions, thus the rate should be increased (decreased) accordingly. However, it is non-trivial to design a rate adaptation algorithm based on this seemingly simple idea. In particular, the design must address two issues: (i) what are the loss ratio thresholds for rate increase and decrease (P_{MTL} and P_{ORI}) respectively? and (ii) how long is the estimation window (*ewnd*)? We next address these two questions.

5.1.1 Loss Ratio Thresholds

In RRAA, the loss ratio thresholds are chosen such that the new rate can maximize the expected throughput in the

Rate (Mbps)	Critical Loss Ratio (%)	P_{ORI}	P_{MTL}	$ewnd$
6	N/A	50.00	N/A	6
9	31.45	14.34	39.32	10
12	22.94	18.61	28.68	20
18	29.78	13.25	37.22	20
24	21.20	16.81	26.50	40
36	26.90	11.50	33.63	40
48	18.40	4.70	23.00	40
54	7.52	N/A	9.40	40

Table 3: RRAA implementation parameters for 802.11a.

next window. In short, when the loss ratio exceeds P_{MTL}, the expected throughput at the current rate becomes lower than that at the next lower rate, thus the rate should be decreased. On the other hand, when the loss ratio is below P_{ORI}, the channel is very likely ready for higher rates, and thus the rate should be opportunistically increased.

We first consider P_{MTL}, the threshold for rate decrease. Given a fixed frame size, we can assess the loss-free throughput at different rates, then define a *critical* loss ratio as follows. For any rate R other than the base rate out of the multiple rate options, let the next lower rate be R_-. The critical loss ratio P^* for R is defined as:

$$P^*(R) = 1 - \frac{Throughput(R_-)}{Throughput(R)} = 1 - \frac{tx\_time(R)}{tx\_time(R_-)} \quad (2)$$

That is, with a loss ratio of P^*, the goodput at R becomes the same as the loss-free goodput at R_-. In other words, one should not tolerate a loss ratio larger than $P^*(R)$ at rate R to maximize goodput, assuming the channel becomes lossless when the rate drops. However, given that the loss ratio at R_- is likely non-zero in practice, we choose $P_{MTL} = \alpha P^*(R)$, where $\alpha \geq 1$ is a tunable parameter, to anticipate certain level of losses at R_-.

Equation (2) also tells us that, given the transmission rate and the frame size, the throughput can be calculated based on the transmission time, which includes all PHY and MAC overheads such as PHY preamble, SIFS, DIFS, slot time, ACK, and random backoff. In our implementation, we use the minimum backoff in the calculation, which provides us the maximum P_{MTL} and optimistic rate decisions.

Next we explain how to set P_{ORI}, the threshold for rate increase. The challenge here is the lack of a good method to estimate how the loss ratio changes when the rate increases. As a result, the design may suffer either from premature rate increase (when the threshold is too high) or from missing the opportunity to switch to a higher rate (when the threshold is too low). In our design, we use a simple heuristic that sets $P_{ORI} = P_{MTL}(R^+)/\beta$, where $P_{MTL}(R^+)$ is the P_{MTL} of the next higher rate. The rationale is that, the loss ratio at the current rate R has to be small enough to make the consequent rate increase stabilize at R^+ and not quickly jump back to R. This way, the algorithm will not keep on oscillating between R and R^+. An illustrative example for these two thresholds in 802.11a is shown in Table 3 with $\alpha = 1.25$ and $\beta = 2$.

5.1.2 Estimation Window Size

The estimation window size *ewnd* is a critical parameter in RRAA that affects the accuracy of loss ratio estimation. It is well known in statistics that estimation based on a small number of samples may significantly deviate from the actual value, due to randomness in the samples. This seems to argue for the use of large estimation windows. However, as shown in Section 4, the channel condition may fluctuate a lot as time evolves. Thus the loss ratio in a large window is less meaningful in guiding rate adaptation, as it is skewed by obsolete samples. This problem can be further exacerbated with multiple active stations, because the packets of one station are spread out due to channel contention (for upstream traffic) or multiplexing at the AP (for downstream traffic).

Our design balances the two conflicting needs of obtaining meaningful statistics and avoiding obsolete information. Specifically, we count the window size by the number of transmitted frames. Because our algorithm compares the loss ratio against P_{ORI} and P_{MTL}, we first ensure $\frac{1}{ewnd} < P_{ORI}$, i.e., the estimated loss ratio is at least at a granularity finer than the thresholds. We then gradually increase the window size for higher rates because they can transmit more frames within the same time period as compared to lower rates. As an illustrative example, the final *ewnd* settings for all rates in 802.11a are shown in Table 3.

To improve the responsiveness to rapid channel fluctuation, e.g., due to mobility, we use an optimization technique that allows for the transmission rate to be changed in the middle of an estimation window. After each frame is sent, we calculate the best (worst) possible loss ratio, assuming the remaining frames in the window all succeed (fail). If the best possible loss ratio already exceeds P_{MTL}, the rate is immediately decreased and a new estimation window starts. Similarly, the rate is immediately increased when the worst possible loss ratio is smaller than P_{ORI}.

5.1.3 Miscellaneous Issues

Now we describe how RRAA addresses three practical issues of idle stations, multiple active stations, and variable packet sizes.

With an idle station, it may take a long duration to transmit *ewnd* frames. In such cases, the accumulated frame loss ratio becomes obsolete, thus cannot be used to accurately guide the rate adapatation. To handle idle stations, we include a timeout mechanism in the loss ratio estimation. An estimation window is flushed after certain period of time (1 seconds in our experiments) even if less than *ewnd* frames have been sent. After that, the current rate is retained while a new estimation window starts.

We also handle multiple active stations by using small estimation windows. It takes roughly 20ms to transmit *ewnd* frames, which is smaller than the duration (say, 150 ms) required to retain the correlation across transmissions. Thus, with a medium number (e.g., 8) of active stations, our design still works well despite that the transmissions of these stations are multiplexed. As the number of active stations increases, the performance of RRAA may gradually degrade. However, to our best knowledge, no existing designs can well address this issue, and the fundamental barrier is that rate adaptation cannot obtain sufficient samples to infer the channel quality before it deviates significantly.

To handle variable packet sizes, we categorize the packet sizes into multiple groups, and maintain a loss ratio for each of them. Accordingly, the algorithm makes separate rate decisions for packets with different sizes. We note that this technique is similar to the one in the SampleRate implemen-

```
1  RTSWnd = 0;
2  RTScounter = 0;
3  while ture do
4     rcv_tx_status(last_frame);
5     if(!RTSOn and !Success) then
6        RTSWnd++;
7        RTScounter = RTSWnd;
8     elseif(RTSOn xor Success) then
9        RTSWnd = RTSWnd/2;
10       RTScounter = RTSWnd;
11    if(RTScounter > 0) then
12       TurnOnRTS(next_frame);
13       RTScounter--;
```

Figure 7: Adaptive RTS (A-RTS) filter in RRAA.

tation of [8], while other algorithms such as ARF, AARF, and ONOE assume the same frame size for all packets.

5.2 Adaptive RTS Filter

To be robust against hidden terminals, RRAA uses an adaptive RTS filter to suppress collision losses when it estimates the loss ratio. The basic idea is to leverage the per-frame RTS option in 802.11 standards, and *selectively* turn on RTS/CTS exchange to suppress collision losses. While RTS is well known as an effective means to handle hidden terminals, the main challenge faced by our design is to decide *when* and *how long* RTS should be turned on or off.

Design Options There are two straightforward design choices to suppress hidden-station-induced losses. The first one is to turn on RTS for every frame. However, the downside is the excessive overhead of RTS/CTS exchange, which can be significant with high transmission rates. This is also why RTS is disabled in most real-life 802.11 wireless networks. Therefore, when RRAA decides to turn on RTS, the throughput gain from better rate adaptation must outweigh the overhead of RTS.

The second choice is to turn on RTS upon a frame loss and turn off RTS upon a frame success. This is the design used in CARA [10] to handle a different problem of multi-client contention in a single collision domain. However, when hidden terminals exist, it suffers from the drawback of *RTS oscillation*, which alternates on and off for RTS. In the worst case, one of every two frames is lost, resulting in more than 50% throughput reduction. From the rate adaptation perspective, the sender may still experience heavy collision losses and eventually drop the data rates.

Adaptive Scheme in RRAA As illustrated in Figure 7, RRAA uses an adaptive RTS (A-RTS) scheme to adapt to the dynamic collision level incurred by hidden stations. The key state maintained by A-RTS is $RTSwnd$, the RTS window size within which all frames are sent with RTS on (the actual number of frames being sent with RTS on is recorded by $RTScounter$). $RTSwnd$ is initially set as 0, which disables RTS. It is then adapted to the estimated collision level as follows. When the last frame was lost without RTS, $RTSwnd$ increments by one because the loss was potentially caused by collisions. However, when the last frame transmission was lost with RTS, or succeeded without RTS, $RTSwnd$ is halved because the last frame clearly did not experience collisions. When the last frame succeeded with RTS on, $RTSwnd$ is kept unchanged. Note that $RTSwnd$ is an integer. When it is halved from the value 1, it becomes 0 which disables RTS again. An example showing $RTSwnd$ evolution and RTS on/off schedule is illustrated in Figure 8.

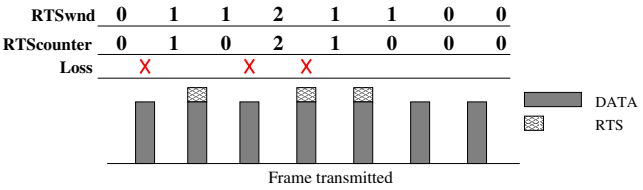


Figure 8: An example of $RTSwnd$ evolution.

```
1  while true do
2     rcv_tx_status(last_frame);
3     A-RTS();
4     if(!RTSFail) then
5        RRAA_BASIC();
6        if(RTSWnd > 3) then
7           fix_re_tx_rate();
```

Figure 9: The complete RRAA design: Integrating RRAA-BASIC and A-RTS.

With A-RTS, more frames are sent with RTS on when collision losses are severe, as $RTSwnd$ is gradually increased by such losses. This not only protects the frame transmissions from collisions but also avoids the loss ratio estimation being poisoned by the collision losses. On the other hand, when collision losses are mild or absent, $RTSwnd$ is small (say 1 or 2) due to multiplicative decrease, and the overhead of RTS/CTS exchange is minimized. We will evaluate the effectiveness of A-RTS via experiments in Section 6.

5.3 Integrating RRAA-BASIC and A-RTS

While RRAA-BASIC and A-RTS address channel fluctuations and hidden terminals respectively, one might think that a simple combination of both algorithms would suffice. However, their coherent integration is actually non-trivial because *RTS frames may also be lost due to collisions with the hidden terminals*.

Such RTS losses have dual impacts on rate adaptation. First, when RTS fails, it is considered as a frame loss even though the data frame is not transmitted at all, which tends to over-estimate the loss ratio. Second, the RTS losses will trigger retries executed in the firmware of the Atheros/Agere chipset, which use rates lower than the one set by the software rate adaptation algorithm. Because an RTS frame is much smaller than a typical data frame, subsequent re-tries of RTS may still collide with the ongoing data transmission of a hidden terminal. In the worst case, the data frame is discarded or transmitted at the base rate during the last retry executed by the chipset firmware, regardless of the decisions made by the software rate adaptation.

As shown in Figure 9, RRAA addresses these two issues by applying two checks in integrating RRAA-BASIC and A-RTS. First, a frame loss due to RTS failure is not counted toward the loss ratio estimation. Secondly, when $RTSwnd$ in the A-RTS algorithm exceeds a threshold (3 in our experiments), which indicates severe collisions, we disable the firmware rate adaptation. That is, the firmware is informed to fix the rate in the re-transmissions as specified by the software rate adaptation module.

6. IMPLEMENTATION AND EVALUATION

In this section, we describe our implementation effort and

evaluate the performance of RRAA, using both controlled experiments and uncontrolled field trials.

6.1 Implementation

We implement RRAA-BASIC and RRAA on a programmable AP platform. We also import the implementation of three other algorithms, namely ARF, AARF and SampleRate, into our platform for comparison purposes.

There are two non-trivial challenges that our implementation must address. First, our AP platform avoids floating-point calculation, thus the runtime short-term loss ratio and the associated two thresholds are not directly applicable. To address this issue, we count the number of lost frames, rather than to calculate the decimal loss ratio. Specifically, we maintain a counter to record the number of lost frames within the current estimation window, while the loss ratio thresholds are translated into the number of frame losses before we load them into the AP.

Second, to filter out collision losses, RRAA needs to know whether a frame loss is incurred by RTS failures or by data transmission errors after a RTS success. While the exact loss type may ultimately be available from the 802.11 chipsets, many existing systems do not use this information. In our implementation, we develop a RTS failure detection technique as follows. The key idea is that the transmission time of a 20-byte RTS frame is much shorter than that of a typical DATA frame. Thus by checking the time duration of a transmission, we can infer whether RTS has failed or not.

Specifically, we use the hardware feedback timestamps (in granularity of μs) to approximate the time duration T_1 spent by the current transmission, which includes all frame retries. In our system, we can extract the number of retries and the backoff value used in each retry. This way, we can calculate T_2 as the time spent to complete all frame retries without any RTS failure. If the difference between T_1 and T_2 exceeds a threshold, we infer that a RTS failure has occurred. Note that this technique cannot detect the loss types with perfect accuracy due to unknown factors such as hardware processing latency and carrier sensing time. However, our experiments have confirmed that it can detect the vast majority of RTS failures in practice.

6.2 Performance Evaluation

In this section, we evaluate the performance of RRAA-BASIC and RRAA using extensive experiments and field trials. We compare them to three existing algorithms, i.e., ARF, AARF and SampleRate, in various settings of static clients, mobile clients, and hidden stations. The results show that RRAA constantly outperforms other algorithms in all scenarios. Note that the performance of ARF, AARF and SampleRate fluctuates significantly across different scenarios.

6.2.1 Static Clients

We first evaluate RRAA and RRAA-BASIC in static settings where the AP and the clients remain stationary throughout the experiment. The goal is to assess how well these algorithms, as well as ARF, AARF, and SampleRate, respond to random channel losses. We perform tests at five different locations, P1, P2, P3, P4, P5 of Figure 1, where the receiving client perceives quite different channel condition and transmission rate. We run both UDP and TCP, over 802.11a and 802.11b channels. For the 802.11a tests, since our system does not implement RTS over 802.11a channels, we only evaluate RRAA-BASIC. The result does not include P5, since P5 is out of the 802.11a communication range.

Figures 10(a) and 10(b) show the goodput results on 802.11a channels for RRAA-BASIC, ARF, AARF, and SampleRate at four locations of P1∼P4, using UDP and TCP, respectively. In all cases, RRAA-BASIC always outperforms the other three algorithms. For the UDP case, its throughput gain ranges from 4.5% to 67.4% at these four locations. In the TCP scenario, the throughput gain of RRAA-BASIC is between 10.3% and 45.3% compared with the worst of the other three algorithms.

To understand why RRAA-BASIC is better than others in the static setting, we record the rate decision made by each algorithm during the experiment. Figure 11 plots the rate distribution at the sender when the client is located at spot P3. The figure shows that RRAA-BASIC transmits 79% of its packets at the rate of 24Mbps, while the other three algorithms send only 59%∼66% of packets at this rate. Both ARF and AARF are too sensitive to random channel losses. Therefore, ARF and AARF sent around 28% and 37% of packets at 18Mbps. Compared with ARF and AARF, RRAA-BASIC will not decrease its transmission rate unless the loss ratio is greater than the threshold. This shows that RRAA-BASIC is more robust to random channel losses. Our experiments at other locations also reveal similar rate distributions.

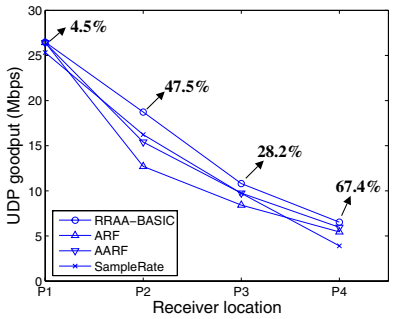
The UDP throughput results for different algorithms at five locations using 802.11b channels are plotted in Figure 10(c). In general, RRAA-BASIC performs better than RRAA because RRAA-BASIC never sends the probing RTS messages by default. The maximum performance difference between RRAA and RRAA-BASIC is 4.6%, which happens at spot P5. However, RRAA still achieves 0.3%∼48.2% throughput gain compared with the other three algorithms.

Comparing Figures 10(a) and 10(c), we also make another interesting observation. SampleRate works better than ARF and AARF in the 802.11b environments. This is because SampleRate tolerates some random channel errors and does not drop to lower transmission rates as frequently as ARF and AARF. However, SampleRate performs worse than both ARF and AARF at spots P1 and P4 in 802.11a. P1 is the closest location from the receiver to the AP sender. The poor performance of SampleRate at P1 is already explained in Section 4.2. Our experiments show that SampleRate transmits only 82% of its packets at 54Mbps, while all other algorithms transmit 99% of packets at this rate. The poor performance of SampleRate at P4 can be explained by one heuristic used by its implementation. Its implementation[5] explicitly skips 9Mbps in its rate decisions. Therefore, SampleRate only uses 12Mbps and 6Mbps at P4, while skipping the rate of 9Mbps for its packets. Unfortunately, most transmissions at 12Mbps have failed. In contrast, ARF and AARF send 50% of their packets at 9Mbps at P4, while RRAA transmits 67% of data packets at 9Mbps.

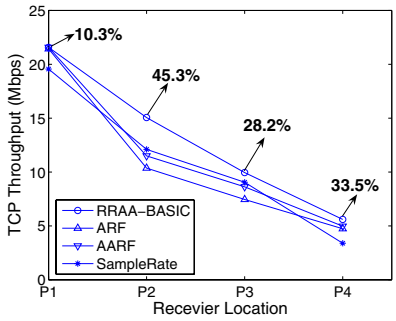
6.2.2 Mobile Clients

We now compare RRAA and RRAA-BASIC with other algorithms ARF, AARF, and SampleRate, in the case of client mobility. While the static setting mainly assesses stability and robustness of a rate adaptation algorithm, the

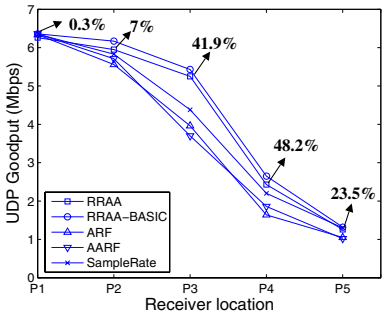
[5]We used the source code of MADWiFi dated on March 14, 2006.



(a) UDP goodput in 802.11a.



(b) TCP throughput in 802.11a.



(c) UDP goodput in 802.11b.

Figure 10: TCP/UDP performance of ARF, AARF, SampleRate, RRAA-BASIC and RRAA. The numbers pointed by arrows are performance gains of RRAA (in 802.11b) or RRAA-BASIC (in 802.11a) over the worst one.

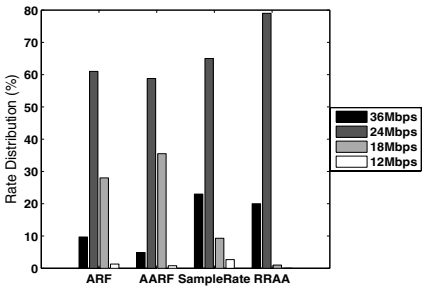


Figure 11:
Distribution of rates in each algorithm when the receiver is located at P3 (802.11a).

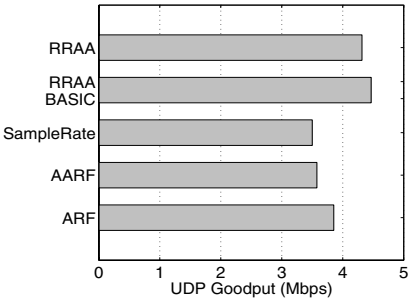


Figure 12:
UDP goodput in 802.11b with client mobility.

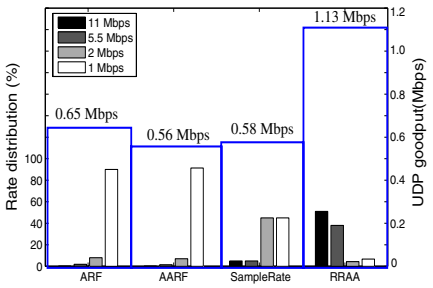


Figure 13:
UDP throughput and rate decision distribution in the hidden-terminal case (802.11b).

mobile setting gauges its responsiveness. The experimental setup is the same as the one we have described in Section 4.2.5.

Figure 12 shows the average throughput of different algorithms using UDP for the mobile client. We can see that both RRAA and RRAA-BASIC perform better than ARF, AARF, and SampleRate. The throughput improvements of RRAA over ARF and SampleRate are about 10.0% and 27.6%, respectively. This clearly demonstrates that RRAA is highly responsive to significant channel variations incurred by client mobility.

Another interesting observation is that SampleRate performs the worst out of all algorithms. In its implementation, SampleRate tries to limit the minimum duration between successive rate changes to be at least 2 seconds. In contrast, both ARF and AARF perform better because they have a timeout mechanism that results in probing the channel at least once every 15 packets.

6.2.3 *Setting with Hidden Terminals*

In this experiment, we evaluate whether RRAA can quickly infer collision losses and adjust its rate accordingly in the presence of hidden terminals. The experimental setting is the same as the one described in Section 4.2.1.

Figure 13 plots the UDP goodput and the distribution of transmission rates for the four algorithms. The results show that RRAA always performs the best. Its throughput gain over SampleRate and AARF is about 101%, and its gain over ARF is about 74%. We also observe that RRAA sends 50% of its packets at 11Mbps, while ARF and AARF send more than 85% of packets at 1Mbps and SampleRate transmits about 42% of its packets at 1Mbps and 2Mbps each. It is clear that ARF, AARF and SampleRate have reduced their rates to 1∼2Mbps due to the collision losses incurred by the hidden AP. In RRAA, we can differentiate most of such losses from fading errors using the adaptive RTS filter mechanism described in Section 5.

6.2.4 *Field Trials*

After we have gained insights on the pros and cons of different algorithms using controlled experiments, we finally conduct a series of uncontrolled field trials over a two-day period. The purpose is to understand how these algorithms perform under realistic scenarios, in which various sources of dynamics co-exist in a complex manner.

Our first field trial involves static clients only. We run six sets of experiments and each lasts an hour. The time span is over 6 hours from 4∼10pm. Each experiment uses four static clients, two of them are located at spot P1, and the other two are placed at spot P2. A TCP connection is run from the AP sender to each receiver. We intentionally select Channel 6, which is also used by other clients and APs in the same building. During our experiments, the sniffer detects about 7∼11 APs and 77∼151 clients over Channel 6. People

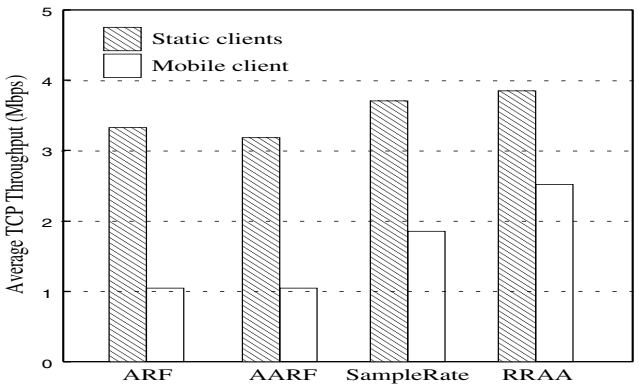


Figure 14: TCP performance in field Trials.

are walking in the corridor, stepping into and out of offices, and even turning on microwaves during the experiments.

Figure 14 shows the experimental results with static clients in the first field trial. The throughput gain of RRAA over SampleRate is 3.8% on average, and the gain increases to 15.3% over ARF. In all cases, RRAA outperforms the other three algorithms.

We also conduct another field trial using the mobile setting presented earlier in Section 4.2.5, except that we use Channel 6 during the 6-hour trial. As the client moves around, it apparently experiences hidden terminals once a while, due to other APs and active clients on Channel 6. Figure 14 gives the throughput of different algorithms. The results show that, RRAA achieves throughput improvement of 35.6% over SampleRate and 143.7% over ARF during each one-hour experiment.

7. RELATED WORK

Rate adaptation has been an active research topic in recent years and a number of algorithms [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12] have been proposed. It is one of the few algorithms that are left to the vendors by the IEEE 802.11 standard, yet its design is critical to the overall system performance. Most existing algorithms follow the design guidelines identified in Section 4. They do not address all the robustness issues against random channel loss, mobility, and hidden-terminal-induced contention loss. Therefore, they do not work well and suffer from performance penalties in realistic settings with mobile clients, static clients with lossy channels, or hidden clients/APs.

The two components in RRAA bear superficial similarity to some existing designs but have fundamental differences from them. The loss ratio threshold-based scheme seems similar to the one used in ONOE [12]. However, ONOE uses long-term estimation (i.e. 1 second), while RRAA uses short-term metrics to exploit the *opportunistic gain* associated with transient channel dynamics. In addition, the threshold values in ONOE are set in a rather ad hoc manner, while RRAA computes its thresholds based on the expected cost and gain associated with the rate changes. The RTS mechanism is also used in CARA [10]. However, CARA seeks to solve a different problem of multiple clients contending for the same AP over wireless LANs. Because CARA did not intend to address rate adaptation in hidden-terminal settings, its design is much simpler than the adaptive RTS filter in RRAA and suffers from RTS oscillation with hidden stations. Moreover, CARA is designed on top of ARF, thus inheriting the drawbacks of ARF.

8. CONCLUSION

Rate adaptation offers an effective means to facilitate system throughput improvement in 802.11-based wireless networks by exploiting the physical-layer multi-rate option upon dynamic channel conditions. In this paper, we have critiqued on five design guidelines for existing algorithms, and proposed a new Robust Rate adaptation Algorithm (RRAA). The key insight learned is that the rate adaptation algorithm has to infer different loss behaviors and take adaptive reactions accordingly. We have implemented RRAA on a programmable AP platform and compared it with three other popular algorithms of ARF, AARF, and SampleRate. Through extensive experiments, we demonstrate that RRAA consistently outperforms all these algorithms, and improves throughput by up to 35.6% over SampleRate and by up to 143.7% by ARF in field trials. We believe that our solution will benefit the widely-deployed 802.11-based WLANs as well as the emerging mesh networks.

9. ACKNOWLEDGMENTS

We appreciate the constructive comments by our shepherd, Dr. Edward Knightly and the anonymous reviewers. We also thank Dr. Songchun Zhu, Mr. Jerry Cheng and Mr. Xiaoqiao Meng for many helpful discussions.

10. REFERENCES

[1] A. Kamerman and L. Monteban. WaveLAN II: A high-performance wireless LAN for the unlicensed band. *Bell Labs Technical Journal*, Summer 1997.

[2] M. Lacage, M. H. Manshaei, and T. Turletti. IEEE 802.11 Rate Adaptation: A Practical Approach. *ACM MSWiM*, 2004.

[3] G. Holland, N. Vaidya, and V. Bahl. A Rate-Adaptive MAC Protocol for Multihop Wireless Networks. *ACM MOBICOM*, 2001.

[4] B. Sadeghi, V. Kanodia, A. Sabharwal, and E. Knighlty. Opportunistic Media Access for Multirate Ad Hoc Networks. *ACM MOBICOM*, 2002.

[5] D. Qiao, S. Choi, and K. Shin. Goodput Analysis and Link Adaptation for IEEE 802.11a Wireless LANs. *IEEE TMC*, 1(4), October 2002.

[6] I. Haratcherev, K. Langendoen, R. Lagendijk and H. Sips. Hybrid Rate Control for IEEE 802.11. *ACM MobiWac*, 2004.

[7] I. Haratcherev, K. Langendoen, R. Lagendijk and H. Sips. Fast 802.11 Link Adaptation for Real-time Video Streaming by Cross-Layer Signaling. *ISCAS*, 2005.

[8] J. Bicket. Bit-rate Selection in Wireless Networks. *MIT Master's Thesis*, 2005.

[9] D. Qiao and S. Choi. Fast-responsive Link Adaptation for IEEE 802.11 WLANs. *IEEE ICC*, 2005.

[10] J. Kim, S. Kim, S. Choi, and D. Qiao. CARA: Collision-aware Rate Adaptation for IEEE 802.11 WLANs. *IEEE INFOCOM*, 2006.

[11] S. Choi, K. Park, and C. Kim. On the Performance Characteristics of WLANs: Revisisted. *ACM SIGMETRICS*, 2005.

[12] Onoe Rate Control. *http://madwifi.org/browser/trunk/ath_rate/onoe*.

[13] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-Level Measurements from an 802.11b Mesh Networks. *ACM SIGCOMM*, 2004.

[14] Iperf v2.0.2. *http://dast.nlanr.net/Projects/Iperf/*.

[15] AiroPeek v2.0. *http://www.wildpackets.com/*.

[16] Thomas M. Cover, Joy A. Thomas. Elements of Information Theory. John Wiley & Sons, 1991.

[17] Multiband Atheros Driver For WIFI. http://madwifi.org/.