A Study of 802.11 Bitrate Selection in Linux

Robert Copeland
Department of Computer Science
Johns Hopkins University
Baltimore, Maryland 21218
Email: bobc@cs.jhu.edu

***Abstract*—This paper investigates rate adaptation in 802.11 wireless networks, with a focus on algorithms currently available in the Linux operating system. The algorithms are compared with a simple rate adaptation algorithm from the literature, and modifications are presented that increase the performance of the existing routines in the studied scenarios. In order to compare simulated results with physical results, and to leverage the Linux software ecosystem, a new software simulator based on a virtual 802.11 device is presented.**

I. Introduction

In 802.11 wireless networks, data may be transmitted with any of a number of rates, from low-speed bitrates that are resilient under poor channel conditions, to high-speed rates that require a high signal level to function. The process of automatically selecting the rate that maximizes throughput, given the current channel conditions, is known as *rate adaptation* [1] and has been studied extensively.

The first published rate adaptation algorithm, Auto-Rate Fallback (ARF), is an extremely simple state machine that predicts the rate based on the most recent successful rate. SampleRate [2] is a popular algorithm that builds a statistical model of rates based on frame success rate and computed throughput. These two algorithms form the basis for others examined later in the paper.

Other algorithms attempt to predict the rate based on direct measurements of the signal level at either the transmitter or receiver [3], [4]. Unfortunately, these solutions often require changes to the MAC layer, or expensive low-bitrate broadcast packets.

Recently, loss differentiation has emerged as a promising improvement to frame-loss based algorithms, particularly in congested networks. As these also usually require changes to the 802.11 specifications [5], [6], or modifications to physical hardware [7], uptake of these algorithms in deployed systems has been slow. Consequently, it is instructive to study the algorithms currently in wide use in 802.11 LANs.

Simulation of rate algorithms has typically been performed using network simulators, such as ns2, originally developed for wired networks. While these systems work well for comparing different algorithms under controlled circumstances, by their nature it is difficult to compare experimental results with real-world trials. Moreover, simulations from the literature often fail to account for cross-layer effects that would impact practical implementations, such as routing delays, and TCP timeouts.

One observation is that a simulator may account for cross-layer effects implicitly, by directly using the networking stack of the operating system. One prior attempt to bridge the gap between research simulators and deployed systems is given in [8]. The authors present the library *libmac*, which allows experimenters to capture and inject frames using modified 802.11 device drivers. This system utilizes physical radios for packet collection and transport. For simulation purposes, it would be advantageous to instead use virtual radios and model the medium.

Thus, this paper introduces a simulator based on a virtualized 802.11 device driver, using the Linux `Mac80211` [9] wireless stack. In addition to capturing cross-layer effects, the proposed simulator provides the ability to directly compare experiments utilizing virtual devices with those from physical devices. This simulator is then used in an investigation of rate adaptation algorithms used in the Linux operating system.

Section III describes the assumptions made about the network and typical hardware devices. In section IV, the rate algorithms are briefly described. Section V formulates the channel models used in simulation. In section VI, a new 802.11 simulator that utilizes the Linux wireless stack is presented. In section VII, the rate algorithms are compared both in the simulator and in real world experiments. Finally, in section VIII, modifications to the Minstrel algorithm are proposed.

II. Differences from Previous Work

In this paper, three rate adaptation algorithms are examined: Minstrel, PID, and AARF [10]. AARF has been presented and reviewed in the literature, as has SampleRate [11], the predecessor of Minstrel. Yet, the author is unaware of published comparisons of Minstrel and PID, the two rate adaptation algorithms presently available in the Linux kernel 2.6.32.

Simulations of rate adaptation algorithms have previously been carried out in network simulators with the same or similar channel models as those used in this work. The cross-layer accuracy of such simulations relies in some part on the accuracy of models of other network layers. A new simulator is introduced that models only the 802.11 device and wireless medium while using the existing infrastructure for the remaining layers.

The virtual wireless device driver `mac80211_hwsim` existed prior to this project for testing `Mac80211`. In its more limited role as an API testing tool, the driver performed only

TABLE I: 802.11a Rate Set

Rate (Mbps)	Modulation	Coding Rate	Bits per OFDM symbol
6	BPSK	1/2	48
9	BPSK	3/4	48
12	QPSK	1/2	96
18	QPSK	3/4	96
24	16-QAM	1/2	192
36	16-QAM	3/4	192
48	64-QAM	2/3	288
54	64-QAM	3/4	288

basic operations and did not attempt to simulate the wireless medium. This kernel driver was rewritten to pass frames to user programs to ease development of the channel simulator.

III. Network Model

For this paper, the 802.11a PHY is used as the basis for experimentation. The newest standard, 802.11n, has recently been approved and provides 32 additional rates; however, the Linux rate adaptation API for 802.11n rates is still evolving at this time. The 802.11a rate set (Table I) is still in use as part of 802.11g, and provides a variety of speeds.

In addition, this paper is primarily interested in applications to small infrastructure networks. In ad-hoc and mesh systems, both the range of the network and number of nodes is often large. As a result, rates that work over long distances may be preferred to high throughput, short range rates. Also, in large networks, hidden terminals are common, leading to the frequent use of low-bitrate RTS/CTS protection.

A trend in consumer-oriented 802.11 hardware is the increasing use of so-called soft-MAC designs: devices consisting primarily of radios and small embedded CPUs where most of the 802.11 MAC Layer Management Entity (MLME) features are performed off-chip by the host computer. These designs are low cost and have the advantage of being software-updateable. Such designs often omit explicit rate control features, relying on the host to provide a rate or a set of candidate rates for a frame. A typical design is the Atheros 5212, in which each frame is accompanied by a multi-rate retry (MRR) descriptor. The descriptor consists of four candidate rates, r_1, r_2, r_3, r_4, along with a set of retry counts, c_1, c_2, c_3, c_4. The device will attempt to transmit a frame c_1 times at rate r_1, then c_2 times at r_2, etc., until the retry counts are exhausted or until an ACK is received. The simulator assumes a similar design.

IV. Rate Algorithms

ARF [12] is among the earliest developed automatic rate selection algorithms. In ARF, if packets are transmitted successfully a fixed number of times, then the rate is raised. If there is a frame loss immediately after a rate change, or if there are two consecutive failures, the rate is lowered. Adaptive Auto-Rate Fallback (AARF) [10] utilizes the basic results of ARF, but adds the notion of an exponentially increasing threshold for raising the rate. This is intended to correct the observed problem that the periodic failed transmission attempts at higher rates led to decreased overall throughput.

Minstrel, based on [2], takes a probabilistic approach. Ten percent of sent frames include a random probe rate as the first rate in the MRR chain. Success at each rate is recorded as packets are sent. Every 100 ms, the probabilities of success and computed throughput are updated for all packets, and these are combined with previous results using an exponentially weighted moving average. The MRR descriptor includes the two best throughputs followed by the best probability rate, then followed by the lowest available rate. The MRR retry counts are selected such that transmissions at a given rate for all attempts should take no more than 6 ms, and the entire transmission takes less than 24 ms.

PID is based on the concept of the proportional-integral-derivative feedback controller [13]. The algorithm adjusts the transmission rate to achieve a maximum of 14% transmission failures. Every 125 ms, the controller recomputes the average number of failed transmissions with an exponentially weighted moving average. If a large amount of frame loss is detected, the controller can enter a sharpening mode, in which large adjustments to the rate can be made to more quickly approach the targeted success percentage.

V. Channel Models

To characterize performance of rate scaling algorithms, this paper examines adaptability in both a slowly changing channel and a channel undergoing small-scale fading. For the former, channels are assumed to be additive white Gaussian noise (AWGN), for the latter a Rayleigh distribution is used. In all experiments, the following assumptions are made:

- Bit errors are independent
- Errors between symbols are independent
- Error correction utilizes hard decision boundaries
- Gray-coding is used for QAM points

In reality, bit errors have a bursty nature which cannot be fully corrected by techniques such as bit interleaving. However, error independence greatly simplifies bit error probability computations and is a common assumption from the literature.

Symbol error independence is provided by OFDM: due to separation in the frequency spectrum, errors on one subcarrier typically do not impact symbols on another. 802.11 devices employ soft detectors for error correction, which have been shown to perform better than hard decision boundaries [14]. Hard detectors, however, represent a worst-case scenario and are consequently useful for upper error bounds. Finally, the 802.11a QAM constellations are in fact gray-coded for increased resiliency in the decoder. This yields a useful approximation for the bit error rate given the symbol error rate (Eq. 3).

As none of the rate control algorithms examined in this paper directly use channel characteristics for prediction, inaccuracies in the channel model do not have a large effect on results.

The theoretical symbol error rates for BPSK and M-QAM (including QPSK) can be computed as follows [15]:

$$P_{BPSK} = \frac{1}{2}\ \mathrm{erfc}\left(\sqrt{\frac{E_s}{N_0}}\right) \tag{1}$$

$$P_{M-QAM} = 2\left[1 - \frac{1}{\sqrt{M}}\ \mathrm{erfc}\left(\sqrt{\frac{3}{2(M-1)}\frac{E_s}{N_0}}\right) - \left(1 - \frac{2}{\sqrt{M}} + \frac{1}{M}\right)\ \mathrm{erfc}^2\left(\frac{3}{2(M-1)}\frac{E_s}{N_0}\right)\right] \tag{2}$$

When gray-coding is used, the bit error probability for M-QAM can be approximated by

$$BER \approx \frac{1}{\lg M} P_{M-QAM} \tag{3}$$

A computation of an upper bound for packet error probability when using binary convolutional codes for forward error correction with hard decision boundaries is computed as [16]:

$$P_{PER} \leq 1 - (1 - P_u(m))^L \tag{4}$$

Here, m is the modulation used and L is the number of bits in a frame. P_u is the union bound of the first event error probability, given by:

$$P_u(m) = \sum_{d=d_{free}}^{\infty} a_d P_d(m) \tag{5}$$

In this equation, d_{free} is the free distance of the convolutional code, and a_d is the total number of codewords with Hamming distance d from the correct codeword. Both d_{free} and the first 20 values of a_d have been precomputed for different convolutional codes in [17]. P_d is the probability of any incorrect path with Hamming distance d from the correct path, and is computed as [1]:

$$P_d(m) = \begin{cases} \sum_{k=(d+1)/2}^{d} \binom{d}{k} \rho^k (1-\rho)^{d-k} & \text{if } d \text{ is odd,} \\ \frac{1}{2}\binom{d}{d/2}\rho^{d/2}(1-\rho)^{d-d/2} + \sum_{k=d/2+1}^{d} \binom{d}{k}\rho^k(1-\rho)^{d-k} & \text{if } d \text{ is even.} \end{cases} \tag{6}$$

Here, ρ is the bit error rate computed for the appropriate modulation using eq. (1)-(3).

Together, eqs. (1-6) were implemented in Matlab to produce curves of throughput versus signal-to-noise ratio. The results (Fig. 1) are in agreement with those produced in [18]. These computed curves are used by the simulator to probabilistically drop packets according to a given SNR. Curves for packet lengths of 1500 and 100 octets were generated, allowing higher success probabilities for short packets such as 802.11 management frames.

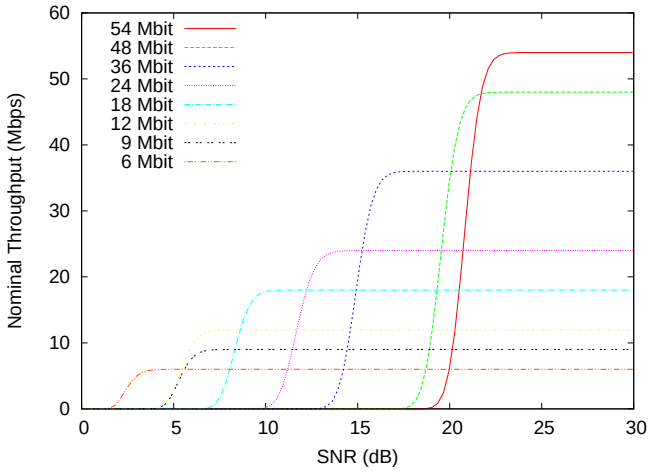

Fig. 1: Theoretical performance of 802.11a rates, 1500-byte frames

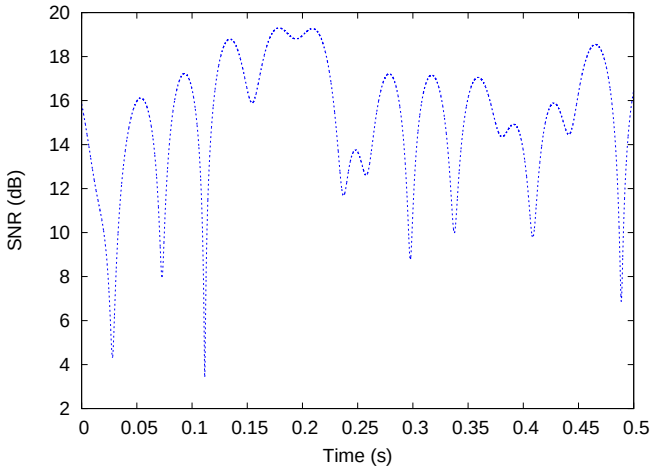

Fig. 2: Rayleigh fading with f_c = 5.0 GHz, v=1 m/s

To assess performance in small-scale fading, a modified Jakes model is used to simulate Rayleigh fading [19]. In this model, the small-scale loss due to fading is computed as a sum of sinusoids, each representing N rays arriving from different angles to a moving receiver.

$$\omega_n = \frac{2\pi f_c v}{N} cos(\alpha_n) \tag{7}$$

$$T(t) = \sqrt{\left(\frac{2}{N_0}\right)} \sum_{n=1}^{N_0} [cos(\beta_n) + j sin(\beta_n)] cos(\omega_n t + \theta_n) \tag{8}$$

N_0 is $N/4$, and the angles are $\alpha_n = 2\pi(n - 0.5)/n$ and $\beta_n = \pi n/N_0$, with normally-distributed phase θ_n. This model was implemented in Matlab, producing the waveform in Fig. 2.

VI. MAC80211 Simulator

A significant part of this exercise is the creation of a platform for experimentation inside the Linux operating system.

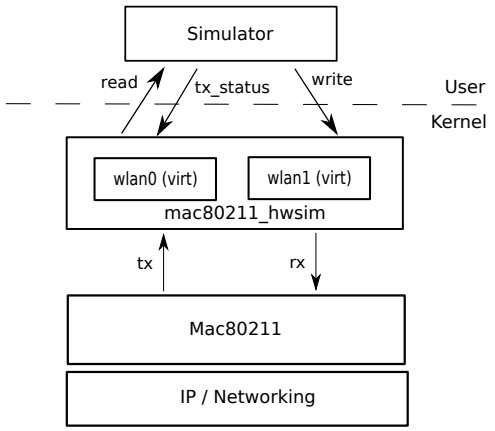

Fig. 3: Mac80211 Simulator Architecture

The 802.11 implementation in Linux is known as `Mac80211`, a software MAC layer used by soft-MAC wireless devices.

A diagram of the simulator architecture is given in figure 3.

The simulator consists of a user-space program and the `mac80211_hwsim` kernel device driver. The driver maintains a queue of packets that are to be sent on a given virtual device. When the simulation program opens a special control device file, two virtual network devices are created, and a file descriptor is allocated for each device. Reading from one file descriptor returns any frame that the virtual device has to send, and writing a frame to a file descriptor results in a successful reception of the frame by that virtual device. The simulator reports success or transmission failure of a given frame by way of an ioctl system call; these results are fed back into the rate selection algorithms by mac80211. Any number of wireless stations can be simulated by chaining devices, though the present exercise only creates a pair of stations.

The user-space program simulates the wireless medium by reading frames from both descriptors and probabilistically dropping frames as they are written to the other descriptor. For a given frame, the simulator will:

- Compute the probability of error-free transmission at the given bitrate, packet length, and signal level.
- Randomly discard the frame according to the probability from step 1.
- Repeat steps 1-2 for each available bitrate and retry count, until the frame has been successfully sent, or all retries have been exhausted.
- Delay according to the total frame transmission time.
- Write the frame to the other virtual device in the case of successful transmission.
- Output the effective instantaneous rate.

The total transmission time of a data frame in the 802.11 distributed coordination function (DCF) is composed of the DCF interframe space (DIFS), a contention window backoff cw, the frame transmission time t_f, and the ACK transmission time, t_a. Values for the 802.11a OFDM PHY are given in table II [20].

TABLE II: 802.11 OFDM PHY Parameters

Parameter	Value
Slot time (t_{slot})	9 μs
SIFS	16 μs
DIFS	2 $\times t_{slot} + SIFS = 18 + 16 = 34$ μs
CW_{min}	$15 \times t_{slot} = 135$ μs
CW_{max}	$1023 \times t_{slot} = 9207$ μs

The number of OFDM symbols for a packet of L bits at bitrate b is computed as follows:

$$N_{DBPS}(b) = 4b \tag{9}$$

$$N_{SYM}(L,b) = \left\lceil \frac{L_{service} + L + L_{pad}}{N_{DBPS}} \right\rceil \tag{10}$$

$$= \left\lceil \frac{16 + L + 6}{N_{DBPS}} \right\rceil \tag{11}$$

N_{DBPS} is the number of data bits per OFDM symbol, $L_{service}$ is the length of the PLCP service header (16 bits), and L_{pad} is the length of padding (6 bits). The total frame transmission time is the symbol transmission time ($t_{SYM} = 4\mu s$) times the number of symbols, plus the time required to send the PLCP preamble ($16\mu s$) and the PLCP signal header ($4\mu s$):

$$t_f(L,b) = 16 + 4 + t_{SYM} \times N_{SYM}(L,b) \tag{12}$$

The ACK transmission time includes the short interframe space plus the time to send a 14 octet packet:

$$t_a(b) = SIFS + t_f((14)(8), b) \tag{13}$$

Thus, the full time to transmit a packet is the sum of transmission times for each attempt i, and the ACK at the final bitrate, if applicable:

$$t = t_a(b_n) + \sum_i cw_i + t_{f_i}(L, b_i) + DIFS \tag{14}$$

The backoff parameter cw_i is a uniform random variable on the interval $[0, CW]$, where CW begins at CW_{min} and exponentially increases for each retransmission until reaching CW_{max}. For repeatable results, the simulator instead uses the expectation $E[cw_i] = CW/2$.

When simulating real-time behavior (for example, to test cross-layer TCP throughput), the simulator will delay for a time of t μs for each frame transmission. To validate this model, average TCP throughput was measured using the `iperf` utility: each rate was tested for 60 seconds, first with the simulator and then with a physical device. The results are given in table III.

As exponential backoff has a large impact on the total transmission time, it is useful to consider not just the successful rate, but the time of failed retries. For that purpose, this paper measures an *effective instantaneous rate*, which is simply the length in bits of a frame divided by the complete transmission time.

TABLE III: TCP Throughput for 802.11 bitrates

Rate (Mbps)	Simulated TCP (Mbps)	Measured TCP (Mbps)
54	21.7	20.6
48	18.8	19.6
36	16.4	16.9
24	13.9	13.0
18	11.4	10.4
12	8.4	7.99
9	6.68	6.39
6	4.77	4.07

VII. Analysis of Rate Adaptation Algorithms

In order to evaluate rate adaptation algorithms, two virtual devices are created with the simulator and separated into distinct network namespaces. The `hostapd` access point software is attached to one virtual device, while the other device associates with it. The `iperf` utility is used to generate TCP traffic from the STA to the AP.

As an initial experiment, each rate adaptation algorithm is evaluated for basic fitness over a slowly degenerating AWGN channel. The channel initially has an SNR of 30 dB for a warm-up period of five seconds, then decreases by .05 dB every second. The effective instantaneous rates of all frames are computed, including backoff penalties for retransmissions. For comparison, an idealized rate controller which always picks the rate with highest throughput and zero retransmissions is also simulated (Fig. 4).

Minstrel proves most adept in this simple scenario, performing close to the ideal and outperforming PID and AARF by a large margin. Loss rate with PID was very high, causing `iperf` to stall, and leading to eventual disassociation from the AP by the `Mac80211` connection watchdog. Investigation revealed several problems with this implementation:

- PID lacks MRR, so any attempt at sending at an inappropriate rate usually results in a packet drop after the initial retries are exhausted
- Frame success is improperly accounted, leading to an under-estimated error rate
- Due to a sign error in fixed-point math, attempts to correct the error term are sometimes made in the wrong direction, destabilizing the algorithm

As a result of these issues, the algorithm selects a much higher rate than it should, resulting in many drops and very poor or no throughput. These issues have been corrected in the algorithm *Modified-PID*. The modified algorithm still exhibits a high degree of variation in the rates chosen.

AARF under-performs in the presence of packet loss. There were two reasons for this: first, any two consecutive packet losses results in a decrease in the rate. If these losses are coincidental but the overall probability of success on the rate is still good, AARF will lower the rate prematurely. In a physical system, such bursty errors are to be expected. Secondly, in a SoftMAC design, there is large latency between determination of a rate and processing of the first frame that used that rate. If a determination is made to raise the rate, then packets already queued will utilize the old rate, and some number of packets will be queued at the new rate before the first frame at the new rate has been transferred. The result is rate oscillation of the sort described in [10]. Between 16 and 20 dB, for example, AARF oscillated between the 36 Mbps and 48 Mbps rate, with the first attempt at 48 Mbps usually failing.

TABLE IV: Sample MRR descriptors

Entry	AARF		Minstrel	
	rate	count	rate	count
1	24	1	24	8
2	18	1	18	3
3	12	1	24	8
4	6	11	6	3

In Fig. 5, TCP throughput for the same experiment is given. As time progresses and SNR decreases, both Minstrel and AARF maintain good overall throughput. Modified-PID can only manage about 5 Mbps below 20 dB. An interesting artifact is the presence of huge loss spikes around rate transitions with Minstrel. Conversely, AARF is much more consistent. As will be seen in the next experiment, differences in the MRR descriptor are a likely cause.

To measure performance in small-scale fading situations, each algorithm was tested for one minute in a simulated Rayleigh channel with a maximum Doppler shift of 16.6 Hz at 5.0 GHz (Fig. 6). Here, AARF outperforms both Minstrel and PID, managing 11.5 Mbps compared to Minstrel's 7.05 Mbps and PID's 7.28 Mbps. Fig. 7 illustrates the cause for the discrepancy. At time $t = 6.83$, both algorithms have selected the 24 Mbit rate, and both experience a deep fade. However, AARF has many fewer retries in the MRR descriptor compared to Minstrel (Table IV).

AARF succeeds on the second attempt at 18 Mbits, while Minstrel attempts 24 Mbits eight times in a row, and finally succeeds at 18 Mbits. Including backoff, the retries take nearly 20 ms, time during which AARF is able to send 20 additional packets. At time $t = 6.86$, Minstrel unsuccessfully sends a probe for the 54 Mbps rate, and this is followed by excessive retries for two more packets. While a single retry per rate may be too low for some realistic scenarios, this experiment demonstrates the importance of backoff in determining overall throughput.

In order to compare these results to performance with actual radios, each algorithm was tested with `iperf` at distances of 0.5, 5, and 10 meters for one minute, for three trials at each distance. The transmitter is an Atheros AR2417 running Linux 2.6.32.3 with the ath5k driver, and the AP is an RaLink RT2561 running `hostapd` on Linux 2.6.30.5 with the rt61 driver. A single trial is representative of the others and is plotted in Fig. 8. In Fig. 9, mean and standard deviation of each rate algorithm is given for different distances.

In this test, no single algorithm has a large performance advantage over the others. At 0.5 meters, the radio can transmit successfully at 54 Mbps, so it is expected that all will perform reasonably well. At five meters, some packets can be periodically sent at high bitrates, but loss is frequent, leading to large-scale oscillations seen by all rate algorithms. At ten

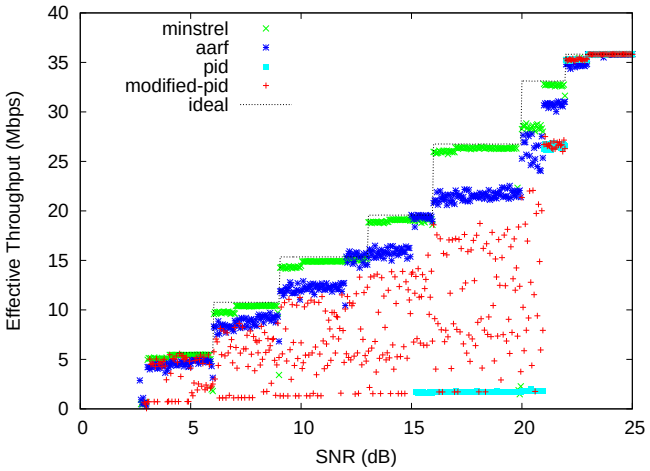

Fig. 4: Effective rate of rate control algorithms

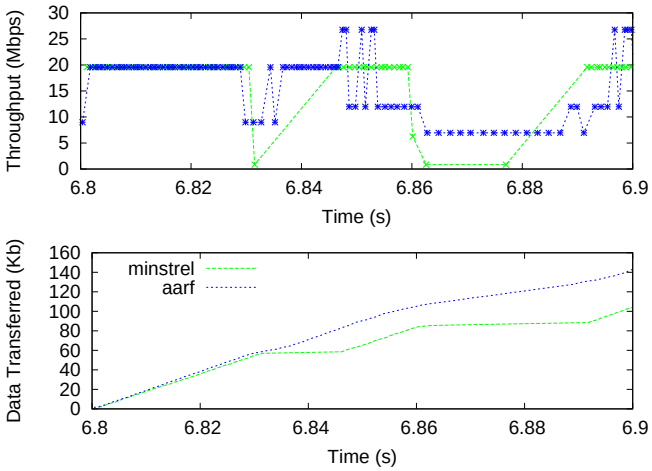

Fig. 7: Impact of excessive retries on throughput

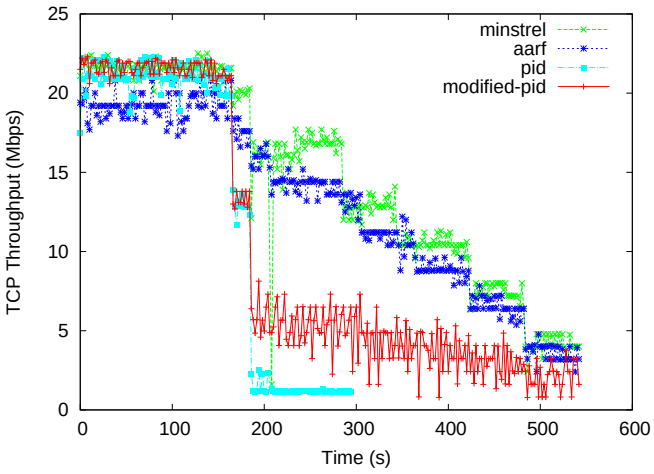

Fig. 5: TCP throughput of rate control algorithms

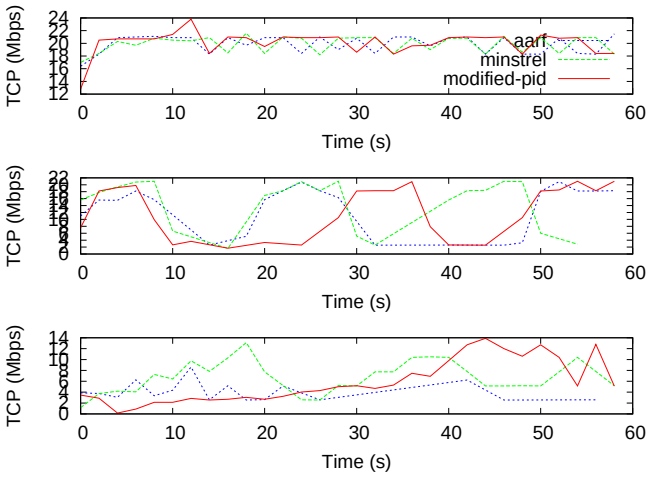

Fig. 8: TCP performance, physical radios $(d = .5, 5, 10m)$

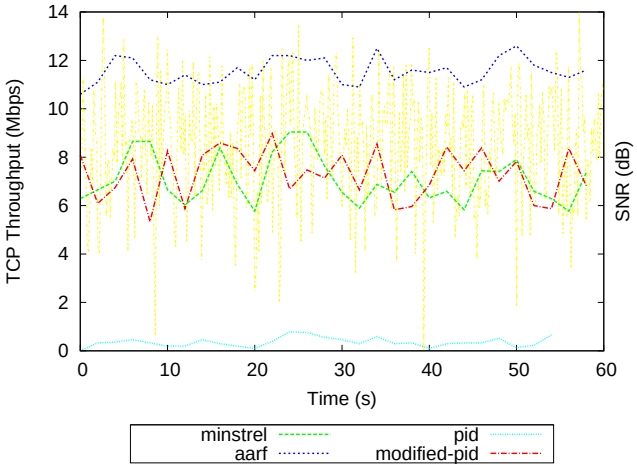

Fig. 6: TCP throughput in Rayleigh fading channel

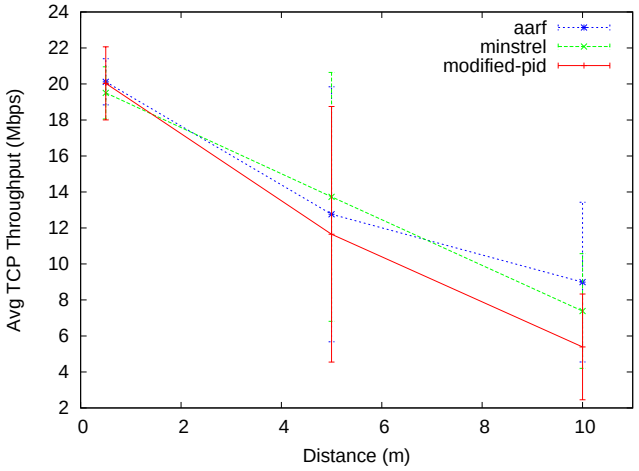

Fig. 9: Mean TCP performance with physical radios

meters, AARF slightly outperforms Minstrel, but variance is quite large in these measurements.

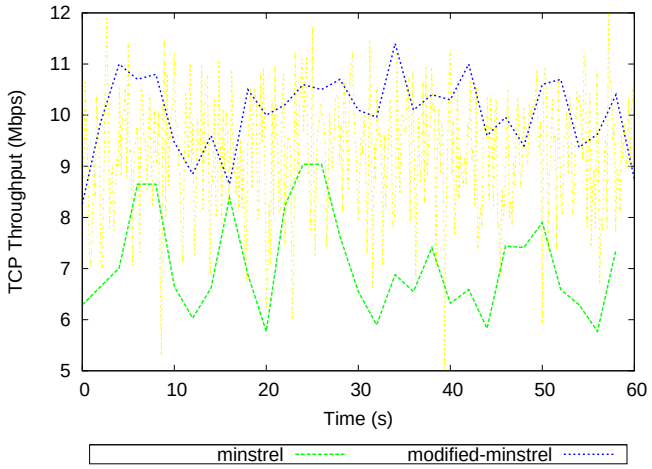

Fig. 10: Simulated performance of modified Minstrel

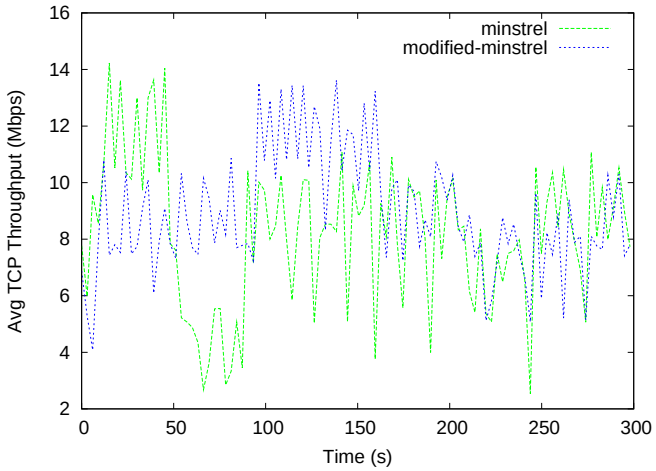

Fig. 11: Physical performance of modified Minstrel

TABLE V: Experimental TCP throughput with Minstrel (Mbps)

Test	Minstrel	Modified-Minstrel
Simulated	6.154	9.867
Physical	8.01	8.93

VIII. Proposed Modifications

The simulations show that Minstrel is a well-performing algorithm in general, producing results close to the ideal in good conditions. However, the large retransmission penalty in 802.11 reduces its potential throughput in instances of greater loss. Consequently, a reduction in the number of retries is likely to improve the overall throughput achieved by Minstrel. The existing transmission time estimate takes into account backoff, but this value is reset for every MRR slot. Thus, the following changes have been made to the algorithm:

- The transmission time estimate is computed as the MRR slots are created, ensuring backoff time for the second slot depends on that of the first.
- The best throughput and best probability rate are not repeated in the MRR slots if they are the same. Instead, the next best rate is used in this case.

These changes were implemented in Minstrel, resulting in Fig.10 (simulated) and Fig.11 (physical). In the physical experiment, throughput was tested with `iperf` for 5 minutes at a distance of $d = 7m$. In all other respects, the experimental setup from the previous section was used. The results are in table V. In simulation, a consistent improvement of 60.5% was seen, but there was little performance difference with actual hardware.

IX. Conclusions and Future Work

In this paper, a novel 802.11 simulator has been presented that provides experimenters with an easy way to leverage the existing operating system stack and user-space software present on the Linux platform. The simulator has been demonstrated by examining rate adaptation algorithms currently used in Linux under different channel models. In the process, deficiencies were found in the existing rate adaptation algorithms and modifications proposed. MAC layer collision avoidance backoff was experimentally shown to play a key role in the achievable network throughput.

Although simulated results proved difficult to replicate in real-world testing, it is felt that better testing conditions would help this problem. For example, running experiments in a radio-controlled environment such as the ORBIT [8] testbed will likely be instructive. The ability to run experiments on both virtualized and non-virtualized hardware in ORBIT nodes could prove useful in developing accurate wireless models.

Potential avenues for future work include expanding the simulator to support scriptable topologies from existing simulators; developing and adding more complete channel models; and making more aspects of the networking stack user-controllable. Rate adaptation continues to be heavily studied in 802.11, particularly as 802.11n brings many new rates. With 802.11n, sampling algorithms like Minstrel will need to manage a much larger probability matrix, while ensuring that probes and retransmissions do not overly reduce throughput. Cross-layer validation of 802.11n-aware rate adaptation algorithms will likely be important as existing algorithms are adapted.

References

[1] D. Qiao and S. Choi, "Goodput enhancement of IEEE 802.11 a wireless LAN via link adaptation," in *Proc. IEEE ICC01*, vol. 7. Citeseer, 2001, pp. 1995–2000.

[2] J. Bicket, "Bit-rate selection in wireless networks," Ph.D. dissertation, Massachusetts Institute of Technology, 2005.

[3] G. Holland, N. Vaidya, and P. Bahl, "A rate-adaptive MAC protocol for multi-hop wireless networks," in *Proceedings of the 7th annual international conference on Mobile computing and networking*. ACM New York, NY, USA, 2001, pp. 236–251.

[4] G. Judd, X. Wang, and P. Steenkiste, "Efficient channel-aware rate adaptation in dynamic environments," in *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*. New York, NY, USA: ACM, 2008, pp. 118–131.

[5] *CARA: Collision-Aware Rate Adaptation for IEEE 802.11 WLANs*, April 2007.

[6] S. Rayanchu, A. Mishra, D. Agrawal, S. Saha, and S. Banerjee, "Diagnosing wireless packet losses in 802.11: Separating collision from weak signal," in *IEEE INFOCOM 2008. The 27th Conference on Computer Communications*, 2008, pp. 735–743.

[7] M. Vutukuru, H. Balakrishnan, and K. Jamieson, "Cross-layer wireless bit rate adaptation," in *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication*. New York, NY, USA: ACM, 2009, pp. 3–14.

[8] K. Ramachandran, H. Kremo, M. Gruteser, P. Spasojevic, and I. Seskar, "Scalability analysis of rate adaptation techniques in congested ieee 802.11 networks: An orbit testbed comparative study," in *IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, 2007. WoWMoM 2007*, 2007, pp. 1–12.

[9] "Mac80211," http://linuxwireless.org/.

[10] M. Lacage, M. Manshaei, and T. Turletti, "IEEE 802.11 rate adaptation: a practical approach," in *Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*. ACM New York, NY, USA, 2004, pp. 126–134.

[11] Y. Yang, M. Marina, and R. Bagrodia, "Experimental evaluation of application performance with 802.11 PHY rate adaptation mechanisms in diverse environments," *Proc. WCNC06*, pp. 2273–2278.

[12] A. Kamerman and L. Monteban, "WaveLAN-II: A high-performance wireless LAN for the unlicensed band," *Bell Labs Technical Journal*, vol. 2, no. 3, pp. 118–133, August 1997.

[13] M. Willis, "Proportional-Integral-Derivative Control," *Department of Chemical and Process Engineering, University of Newcastle*, 1999.

[14] A. Assalini, M. Trivellato, and S. Pupolin, "Performance analysis of OFDM-OQAM systems."

[15] J. Proakis, *Digital Communications*, 4th ed. McGraw-Hill Science/Engineering/Math, August 2000.

[16] M. Pursley and D. Taipale, "Error probabilities for spread-spectrum packet radio with convolutional codes and Viterbi decoding," *Communications, IEEE Transactions on [legacy, pre-1988]*, vol. 35, no. 1, pp. 1–12, 1987.

[17] P. Frenger, P. Orten, T. Ottosson, and A. Svensson, "Multi-rate convolutional codes," 1998.

[18] M. Manshaei and T. Turletti, "Simulation-based performance analysis of 802.11 a wireless LAN," in *Proc. International Symposium on Telecommunications (IST03)*. Citeseer.

[19] P. Dent, G. Bottomley, and T. Croft, "Jakes fading model revisited," *Electronics Letters*, vol. 29, no. 13, pp. 1162–1163, 1993.

[20] "IEEE 802.11-2007, Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," June 2007.