A Model for Comparing Rate Adaptation Algorithms

Candy Yiu
Department of Computer Science
Portland State University
Portland, OR 97207
candy@cs.pdx.edu

Suresh Singh
Department of Computer Science
Portland State University
Portland, OR 97207
singh@cs.pdx.edu

ABSTRACT

Rate adaptation algorithms are critical to improving the throughput performance of WLANs. While previous studies have examined the performance of different algorithms in some detail using numerous measurements and simulations, there is a lack of a theory that would allow comparison across different channel conditions. This paper is a first step towards developing an abstract model that can allow us to (a) estimate or predict the performance of different algorithms and (b) allow us to make general statements about which algorithms would perform better, under what channel conditions. Our work is empirical in nature and uses three rate algorithms available in the Madwifi driver to examine the problem. We identify two key metrics, the speed of adaptation and the quality of adaptation, that taken together nicely encapsulate an algorithm's performance. We then show how these metrics predict the throughput behavior of the three rate algorithms considered with an accuracy of over 70%. Furthermore, we show that these metrics can be used to make very general statements comparing the behavior of any pair of algorithms over a wide range of channels.

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Algorithms

Keywords

802.11, Madwifi, Rate adaptation, Wireless protocol

1. INTRODUCTION

The problem of rate adaptation in 802.11 is fundamentally important to maximizing the throughput of WLANs. The main challenge is to estimate the channel dynamically and then adapt the transmission rate appropriately. Unfortunately, the problem of channel estimation is hard because the sender does not know the instantaneous channel condition at the receiver when it transmits a packet. Since no specific support is provided in 802.11 for providing instantaneous channel feedback, indirect approaches for channel estimation have been studied. One common technique uses MAC layer packet loss/retry counts at the transmitter. Since MAC layer retransmissions occur within tens of microseconds, the channel state is assumed to be unchanged. Then, depending on the pattern of loss, retries, and successes, different algorithms adapt rates differently.

Over the past decade, several rate adaptation algorithms have been proposed and evaluated either via simulations or in real implementations. Generally, the comparisons with other algorithms are a combination of qualitative analysis and measurements targeted at illustrating some behaviors. While one can generally infer the relative merits of one algorithm over another using these techniques, there does not exist any quantitative metric that can summarize the essential quality of these algorithms. In this paper we identify a pair of metrics that may be used to summarize the performance of any rate adaptation algorithm. We show how these metrics may be used to predict or estimate the throughput of an algorithm for any given channel. Finally we show how these metrics, considered together, may be used to state with confidence if one algorithm is better than another, and under what conditions.

Our work starts out with developing an answer to the simple question: "what does it mean for an algorithm to adapt?" As our analysis will show, there are two metrics that jointly answer this question: *the speed* at which an algorithm changes its rate for given channel conditions, and the *quality* of the adaptation as measured by the closeness to an optimal rate for those channel conditions. We measure these two metrics for three rate adaptation algorithms available in the Madwifi driver – SampleRate, Onoe, and AMRR. We then rigorously evaluate the efficacy of these metrics in terms of estimating how these algorithms perform for different channel conditions. Finally, we discuss how these metrics may be used to compare different algorithms to a first order approximation.

The remainder of the paper is organized as follows. In the next section we summarize related work on rate adaptation algorithms. Section 3 describes the two metrics we develop to quantify the performance of rate adaptation algorithms. These metrics are applied to various traces in order to evaluate their quality in section 4. The correctness

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.
WiNTECH'09, September 21, 2009, Beijing, China.
Copyright 2009 ACM 978-1-60558-740-0/09/09 ...$10.00.

of our approach is discussed in section 5. Section 4 develops a methodology to compare rate algorithms using the developed metrics. We summarize our results and directions for future work in section 7.

2. RELATED WORK

Several rate adaptation algorithms have been developed over the past decade or so. Most of them rely on channel feedback inferred from MAC layer retransmission/loss counts to adjust transmit rates. Thus, ARF [5] starts a connection at the highest bit rate and then drops down a rate if a packet is never ACK'ed. If ten successful transmissions occur (with no retransmissions), it goes up a rate. As noted in [7] ARF suffers from too many retransmissions in slowly changing channels. A more adaptive algorithm called AARF is proposed in [7] where the adaptation intervals are adjusted dynamically. The Madwifi [1] driver implements an algorithm called Onoe which is run every second. The rate adaptation is again based on success and retry counts though the adjustment is somewhat slower than AARF. Another algorithm implemented in Madwifi is called AMRR. This algorithm changes the estimation period using a simple binary exponential backoff algorithm. SampleRate [2] uses a more sophisticated method to determine the rates to use. It computes the average bit rate for each transmission rate (i.e., for a given transmission rate packets may need to be retransmitted therefore, the average bit rate for this transmission rate is calculated by normalizing over the retry time). SampleRate periodically probes the channel by sending a packet at a rate that has a higher average bit rate. Rates are changed based on packet loss/success counts. The RRAA algorithm [8] also uses loss data to infer short-term channel conditions. In addition, however, the algorithm uses a RTS/CTS filter to prevent collision losses from triggering a rate change.

A more reliable measure of channel conditions is rssi (Received Signal Strength Indicator) that is available at the receiver. If this value can be returned to the sender, then the sender can make precise rate adjustments to met a target packet error rate (for maximizing throughput). Unfortunately, there is no way within the standards to return this value to the sender. One approach explored in [9] is for the sender to retrieve the rssi value of the MAC ACK returned by the receiver. The assumption is that the channel is reciprocal and thus the sender rssi value reflects the receiver's rssi value. [4] also uses signal strength information to aid the transmitter in making its decision. As in [9], the information is obtained by assuming the reciprocity and stationarity of indoor channels. The benefit of both of these approaches is they do not need to use either probe packets or RTS/CTS packets. A hybrid approach is explored in [6] where the receiver returns the value of Clear Channel Assessment (CCA) to the transmitter via RTS/CTS packets. Furthermore, the RTS packets are used as probes to determine the state of the channel.

2.1 Discussion

In studying the various algorithms developed previously, one thing becomes clear – to a certain extent the comparisons reported tend to be a mix of measurement in specific channels and qualitative discussions of what makes one algorithm better than another. In this paper we seek to develop metrics that will allow a straightforward comparison of algorithms without resorting to a detailed study of specific behaviors of the algorithms under different channel conditions. An added benefit of such metrics is the ability to *predict* the performance of an algorithm under various channel conditions.

3. CHARACTERIZING THE ADAPTIVITY OF RATE ALGORITHMS

Figure 1 shows an example of how SampleRate adapts to a mobile channel. As the rssi of the channel varies, the transmission rate used also changes, with low rates used for poor channel conditions and high rates used for a good channel. The rapidity with which the algorithm changes its rate to match channel conditions directly impacts the achievable throughput. Therefore, throughput has been the metric of choice for characterizing algorithm performance under a variety of channel conditions. We however believe that throughput is a second order metric that hides important information about algorithm behavior. We assert that understanding *how well* an algorithm adapts requires an examination of the *speed* of adaptation as well as the *quality* of adaptation. With these two canonical metrics in hand, it is then possible to infer the throughput behavior of an algorithm and indeed to make general statements about relative performance of different algorithms under a range of channel conditions.

In order to develop an understanding of rate adaptation, we first make the key observation that rate algorithms switch between a small set of discrete rates even though the rssi may vary over a larger set of values. Define a rate to be *optimal* for a rssi value if the *expected throughput* using that rate is maximum over all rates. In Madwifi we can compute the expected throughput for a given rate by setting the transmit rate to a fixed value using iwconfig and running a program like iperf. If the channel remains stable, the same experiment can be run for different rates and we can easily find the optimal rate. Any rssi graph can now be converted to a optimal rate graph that makes it easier to understand how a rate adaptation algorithm adapts. The important thing to note here is that the optimal rate remains constant over relatively long periods even though the rssi changes over a much shorter time scale. It is because of this stability of the optimal rate that rate algorithms can adapt. This observation is also made in [2] where they note that rate adaptation on a time scale of 10 seconds is reasonable in many environments. Of course, for a mobile channel, rate adaptation should happen on a much shorter time scale. Indeed, this is where the behavior of several rate adaptation algorithms tends to be sub-optimal and, in some cases, even random [3].

Let us now return to the question of determining how well a rate algorithm adapts. In order to study this question, we need to identify the event(s) where the channel changes sufficiently to force a rate change. Indeed, the best possible scenario is one where the rate algorithm has adapted to a stable channel (stable in the sense that the optimal rate is constant) and then the channel changes so that a different rate is optimal. If the channel stays in this new state for a long enough length of time, we can measure the time taken for the rate to change. The shorter this time, the more adaptive an algorithm. A second metric that is evident in our studies is that even if the channel remains relatively stable for extended time periods, an algorithm may never

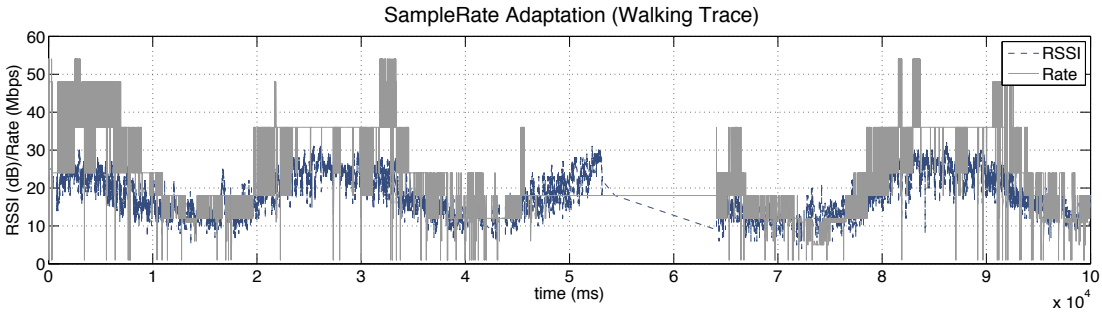

Figure 1: Rate adaptation for a mobile channel.

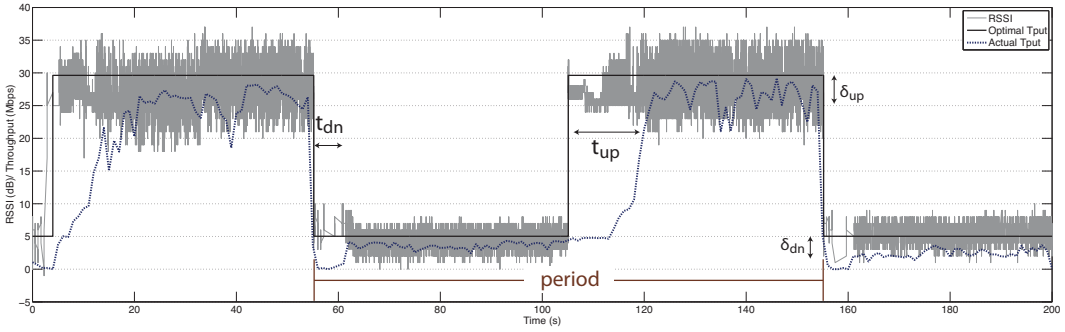

Figure 2: Illustration of t and ϵ.

achieve the optimal rate. For instance, even if rate 48Mbps is optimal, the rate algorithm may fluctuate between 36-54Mbps due to statistical variations in channel feedback. We define the *quality* of adaptation as the difference between the optimal throughput and the throughput achieved using the rates provided by the rate algorithm.

Figure 2 illustrates the two metrics described above. We see the optimal rate changing from a high value to a low value and back to a high value again. The time taken by the rate algorithm to adapt (dotted lines) t_{up}, t_{dn} (time to change from a low to high value and from a high to low value) and the quality of the adaptation δ_{up}, δ_{dn} are also shown. The δ values are the difference between the optimal throughout and the mean achieved throughput after the algorithm has adapted. We normalize the quality of the adaptation by dividing δ by the optimal throughput and denote that value as ϵ. In order to compute the t values, we calculate the time between when the channel changes to the time at which the mean throughput of the rate algorithm stabilizes. We assert that these two metrics can be used to describe any rate algorithm.

Before we present the results of our measurements, however, a discussion of the limitations of our approach are in order. Rate algorithms take some time (of the order of several seconds) to adapt to new channel conditions. Therefore, if the channel changes faster than this adaptation time, the performance of the rate algorithms can be unpredictable. In other words, our metrics will not be as accurate at estimating algorithm behavior for rapidly changing channels as compared to the case when the channel changes on the same time scale as that at which the rate algorithm can adapt. Another limitation of our metrics is that they are deterministic and are based on viewing the algorithms like black boxes. We only look at averaged algorithm performance and ignore completely the interplay between statistical feedback to the algorithm and its performance. Thus, there is some loss of accuracy but, as we show, our metrics perform surprisingly well despite these shortcomings.

4. QUANTIFYING THE PERFORMANCE OF RATE ALGORITHMS

In order to perform a systematic study of rate adaptation behaviors, we need to have repeatable experiments. Specifically, we need to create channel conditions with the following properties:

1. The channel needs to exhibit long periods of stable behavior interspersed with sudden changes. In other words, we need to create square wave patterns such as in Figure 2. These patterns best expose the adaptive behavior of algorithms.

2. Similar channel conditions need to be used to measure t, ϵ values for different algorithms.

3. The period of the square wave as well as its amplitude need to be varied to study how or if t, ϵ values are affected.

To this end, we exploit the ability to change transmit powers in Madwifi on a per-packet basis to create repeatable channel models. The sender and receiver are placed in the same locations for all experiments. We use iperf() to generate UDP traffic continuously and we use 1450 byte packets. At the iperf server we dump the rssi values for each packet received along with a time stamp and the transmit rate used.

4.1 Finding t and ϵ

Based on several initial measurements, we decided to use a long period of 100s for the square wave channel. Thus, the channel switches between two rssi values every 50s. This gives the algorithms sufficient time to reach steady state at the high and low rssi values before we force a change. The second parameter we study is the amplitude of the change between large and small rssi values. The expectation is that a large drop in rssi will result in high packet losses causing faster adaptation whereas a smaller change may take longer time to adapt. We consider four different values of changes in amplitude (ratio of high to low) – 17.5dB, 15.5dB, 13dB, 9dB. We run the experiments for different channels at least five times for each of three algorithms – Onoe, SampleRate, and AMRR. Within each run we identify the transitions from high to low and low to high values. The values of t and ϵ are then computed for each of these transitions. The channel used for the experiments is quiet and therefore the effect of backoff is minimized.

For each trace, we first convert the rssi values to the optimal rate value. Next, we use 1s buckets and compute the optimal throughput and throughput achieved by the rate algorithm in each bucket. Figure 3 shows an example of these transformations for one trace of the AMRR algorithm. Using the throughput traces, we now find the t and ϵ values for the algorithm. Table 1 gives us the mean values for the metrics for the three algorithms.

There are several things of note in Table 1. First, observe that t_{dn} (in seconds) is generally smaller than t_{up} in almost all cases. This is because when rssi falls significantly, the high rate being used causes significant packet loss that forces the rate algorithm to adapt quickly. However, when we go from a low rssi to a high rssi, the old slow rate continues performing well (though sub-optimally). This results in slower adaptation. The quality ϵ of the adaptation is generally good with values above 0.7. However, there are some instances where the adaptation is very poor (about 0.4). This is an interesting phenomenon that needs to be explored within the context of the specific algorithms but is outside the scope of the current paper.

5. EVALUATING THE CORRECTNESS OF THE METRICS

As we discussed previously, our goal in computing t and ϵ values is to develop metrics that allow us to compare algorithms as well as to estimate algorithm performance under different channel conditions. Indeed, before we can make general statements about relative algorithm performance, it is important to determine the "goodness" of the metrics. Since one of the primary methods used to compare rate algorithms is to study their throughput performance under a range of channels, we compare the measured throughput with the model-derived throughput for the three algorithms.

5.1 Estimating Throughput Performance

Let us assume that we are given the rssi values for a channel at the receiver. Our goal is to estimate the throughput obtained when using a particular rate algorithm. The algorithm we use for estimating the throughput is as follows:

Throughput Estimation:

- Given the rssi plot, create a plot of optimal throughput using one second buckets.
- Given a high-to-low or low-to-high transition, we need to find the point at which the rate algorithm will have adapted to the new channel conditions. One way to do this is to simply use the t values from Table 1. However, that method only works if the channel is stable for long enough to enable the rate algorithm to stabilize. Furthermore, using the t values does not tell us the throughput for the time that the algorithm is adapting the rate.

  To deal with these twin problems, we convert the t data to a *slope* value γ that describes the rate of change in Mbps/sec and use this linear model to determine the throughput for the transition period and to determine when the algorithm stabilizes.
- The above step will tell us the time periods when the algorithm is adapting as well as the periods when the algorithm has stabilized. For the stable periods, we estimate throughput by multiplying the optimal throughput with ϵ from Table 1.

Figure 4 illustrates the rssi, actual throughput, and estimated throughput curves for AMRR. As we can see, the agreement between the estimated and actual throughput curves is quite good. Indeed, in this case, the average actual throughput is within 90% of the estimated.

5.2 Experiments to Validate the Metrics

To study how well the metrics can be used to estimate or predict algorithm performance, we need to measure actual algorithm performance under various channel conditions and compare the obtained throughput with the estimated throughput. However, an underlying issue here is what channels should be used for this purpose? Clearly, in order for the answer to be general, we need to cover all possible channel models, but this is unrealistic. Therefore, the approach we follow is to identify those parts of real traces that force rate changes and use those as the basis to create artificial traces for our study.

We observe that there are three basic types of changes in the rssi values that occur most often – the square wave, the sawtooth, and the ramp shown in Figure 5. The sawtooth and the ramp models represent channel changes in the mobile case. The sudden drop in the ramp model corresponds to environment changes, such as turning a corner or entering a stairwell. The square pulse model is present in most traces and represents fluctuating rssi values due to changes in the multipath. In each of the models, the *period* defines how quickly the channel changes and hence it effects the adaptivity of the rate algorithms.

5.3 Results of the Validation

For all the three channel models, we studied the estimation versus actual performance of the three rate algorithms for the following periods: 625ms, 1.25s, 2.5s, 5s, 10s, 20s, 30s, 40s, and 50s. The ratio of the maximum to minimum rssi was generally around 18dB though it tended to vary somewhat. As previously, we computed actual throughput in one second buckets and also computed the average

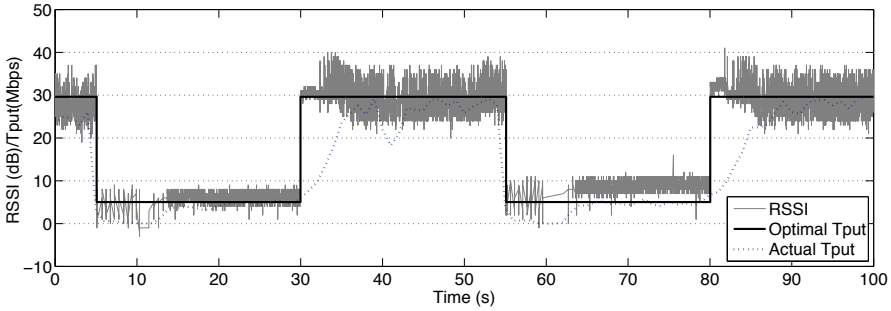

Figure 3: Illustration of the process for estimating t and ϵ.

	t_{up}				t_{dn}			
step size	17.5dB	15.5dB	13dB	9dB	17.5dB	15.5dB	13dB	9dB
Sample	15.14	7.33	6.67	3	6.14	2.75	2	1
Onoe	13	25.6	17.67	1	3.86	5.33	6.5	2.5
Amrr	6.6	2.67	3.25	1	8.33	3.2	4	2.25
	ϵ_{up}				ϵ_{dn}			
step size	17.5dB	15.5dB	13dB	9dB	17.5dB	15.5dB	13dB	9dB
Sample	0.82	0.87	0.89	0.9	0.82	0.65	0.44	0.71
Onoe	0.25	0.45	0.79	0.77	0.78	0.42	0.44	0.68
Amrr	0.9	1	0.99	0.95	0.89	1	0.45	0.73

Table 1: Measured values of adaptation time t and quality ϵ.

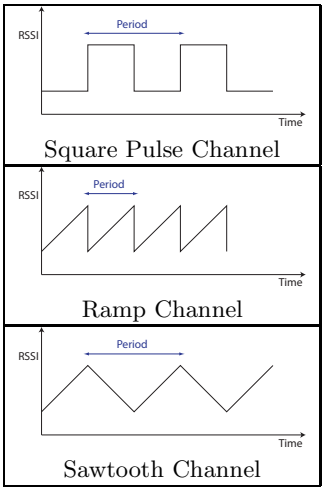

Figure 5: Channel models.

Mean	$\gamma_{up}(Mbps/sec)$	$\gamma_{dn}(Mbps/sec)$	ϵ_{dn}	ϵ_{dn}
Sample	1.89	4.149	0.87	0.65
Onoe	0.68	1.42	0.56	0.58
Amrr	5.68	3.63	0.97	0.77

Table 2: Mean values of γ, ϵ used for our validation study.

	Square Pulse	Sawtooth	Ramp
Sample	0.7490	0.8235	0.8940
Onoe	0.7928	0.7976	0.7464
Amrr	0.7315	0.9557	0.7306

Table 3: Summary of quality of the estimation.

throughput over a 200s run using iperf. To calculate the estimated throughput, we used the slope γ and ϵ values given in Table 2. The values for ϵ are the *mean* values from Table 1. The values for γ are also the mean values of slope and are obtained by computing the slope as an angle in radians for each transition that was used to generate the t data in Table 1. We then average the angles over all transitions and calculate *tan* of that angle. The reason we chose to use mean values for the parameters from Table 2 rather than the more accurate values from Table 1 is that for the sawtooth and ramp channel models, the changes in rssi occur in small steps and therefore none of the values from Table 1 can be used individually, the average is more meaningful here.

Figure 6 plots the estimated and actual measured throughput graphs for the three channel models for all the different period lengths. We first note that the estimation of throughput for the Square channel model is very accurate for all three algorithms. Indeed, the estimated throughput curves also have the same trend as the actual throughput curves. The estimation matches the actual throughput curves for AMRR and Onoe for the Sawtooth channel model as well. However, the estimation for SampleRate is poor with the estimation algorithm over estimating throughput by a significant amount. The fit for the Ramp model is also good for the three algorithms with the AMRR fit being the worst. Table 3 summarizes the fit quality averaged over all period lengths and all runs (measured as the ratio of estimated to actual throughput).

Finally, to further illustrate the applicability of our model to actual traces, we show how the estimation algorithm compares with actual performance of SampleRate for a mobile

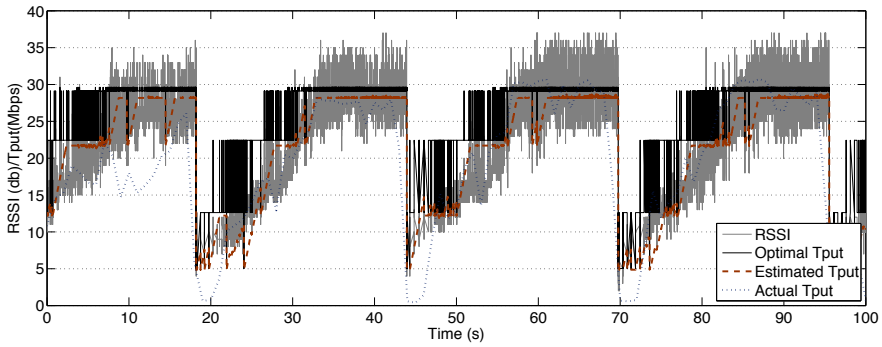

Figure 4: Illustration of how the estimation algorithm works.

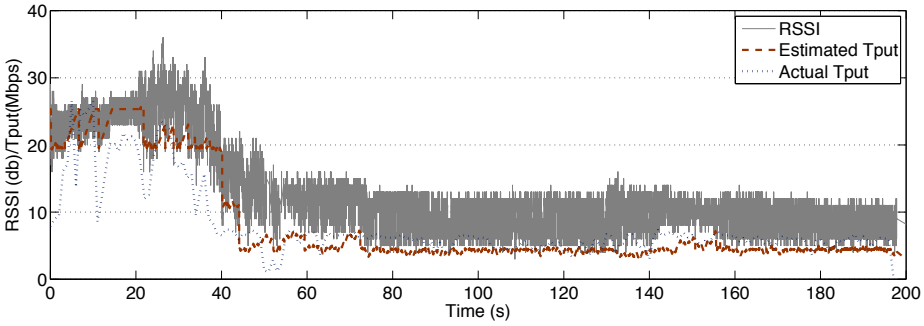

Figure 7: Illustration of the estimation algorithm with a real trace.

user in Figure 7. We see that the estimation algorithm tracks the behavior of the actual SampleRate very closely. Indeed, we observe similar behaviors in other real-life traces as well.

6. USING THE METRICS TO COMPARE ALGORITHMS

A second application of our abstract model is to be able to compare algorithms against one another. One way we can do this is to use the estimated model for each algorithm for given channels and then state that if the estimated model predicts that algorithm A is better than B, then indeed algorithm A is better than B if we were to run the actual experiments. If we perform this test on the data sets described in the previous section, we see that the estimation generally does correctly identify which of any pair of algorithms perform better. Table 4 summarizes the results of this study with each number denoting the fraction of time our estimation algorithm correctly identified the better algorithm of any pair.

A more general way of comparing rate algorithms is to compare the value of γ and ϵ values and infer relative performance. For instance, if we examine the throughput data in Figure 4, we see that AMRR almost always performs better than the other two algorithms. On studying the summary of slope and quality data from Table 2 the reason for this becomes apparent. Let us examine the ϵ values first. We see that these values are highest for AMRR implying that after AMRR stabilizes, it is closest to optimal. The second metric γ denotes how quickly the algorithm adapts and a larger value is better. We see that AMRR adapts much faster to an increase in rssi values as compared to the other algorithms. However, it adapts somewhat slowly compared to SampleRate when rssi falls. Interpreting this data is interesting because we have to take it in the context of the other γ and ϵ values. Note that the number of high-low and low-high transitions in a channel are almost equal. Therefore, we need to consider the γ_{up} and γ_{dn} values jointly. The difference in the γ_{up} values is 3.79 Mbps/sec in favor of AMRR while it is only 0.519 Mbps/sec in favor of SampleRate. Given that there are a similar number of up and down changes in rssi and that AMRR has a higher ϵ value, overall it is easy to see that AMRR will obtain a higher throughput on average.

Indeed, based on the above discussion, we can use the following empirical algorithm to compare which of two algorithms A or B performs better. Define the following quantities:

$$\alpha = \gamma^A_{up} + \gamma^A_{dn} - \gamma^B_{up} - \gamma^B_{dn}$$

and,

$$\beta = \epsilon^A_{up} + \epsilon^A_{dn} - \epsilon^B_{up} - \epsilon^B_{dn}$$

Using these quantities, we can now state the following:

1. If $\alpha > 0$ and $\beta > 0$ the A is the better algorithm in general.
2. If $\alpha > 0$ and $\beta < 0$ then A is better for those channels that show rapid changes in optimal rates over time. Examples would be mobile channels.
3. If $\alpha < 0$ and $\beta > 0$ then B is better for relatively stable channels where changes in optimal rates happen much more slowly than the adaptation time.

	Sample vs AMRR	Sample vs Onoe	Onoe vs AMRR
Square	1.0	0.9	1.0
Sawtooth	1.0	0.875	1.0
Ramp	0.75	1.0	0.875

Table 4: How well the model predicts relative algorithm performance.

We note that this empirical algorithm is based on steady-state assumptions of rate algorithm behavior. In examining traces, it is clear that on some occasions, due to large channel variations and/or collisions due to hidden terminals, the behavior of the rate algorithm is unstable. In these instances, we believe that the empirical algorithm will not work correctly.

7. CONCLUSIONS

The problem examined in this paper is understanding and quantifying what it means for an algorithm to adapt to changing channel conditions. We examine three rate algorithms implemented in Madwifi - Onoe, SampleRate, and AMRR. For each algorithm, we carefully study its adaptive behavior using manufactured channel conditions that expose the adaptive behavior best. We then measure the rate of adaptation as well as the quality of adaptation. Using these two metrics we develop an algorithm that can estimate the achieved throughput for any algorithm given a channel rssi trace. We show the estimation to be relatively accurate and, more importantly, we show that the metrics can be used to make general statements comparing different algorithms.

There are two main future directions for this work. First, we need to develop a better model for the transition periods where the channel has changed and the rate algorithm is trying to adapt. The linear model we have developed has error because in reality the adaptation of algorithms follows a more exponential process. Second, we need to validate the comparison algorithm presented in the previous section by applying it to more pairs of algorithms.

8. ACKNOWLEDGEMENTS

We would like to thank Ron Fairley and Tektronix for allowing us use of their Anechoic chamber. This work was funded by the NSF under award number CNS-0722008.

9. REFERENCES

[1] The madwifi project. http://madwifi-project.org, 2009.

[2] John C. Bicket. Bit-rate selection in wireless networks. Master's thesis, MIT, 2005.

[3] J. Camp and E. Knightly. Modulation rate adaptation in urban and vehicular environments: Cross-layer implementation and experimental evaluation. In *ACM Mobicom*, San Francisco, CA, September 14 - 19 2008.

[4] G. Judd, X. Wang, and P. Steenkiste. Efficient channel-aware rate adaptation in dynamic environments. In *ACM MobiSys*, Breckenridge, CO, June 17-20 2008.

[5] A. Kamerman and L. Monteban. Wavelan-ii: A high-performance wireless lan for for the unlicensed band. *Bell Labs Technical Journal*, pages 118–133, Summer 1997.

[6] J. Kim, S. Kim, S. Choi, and D. Qiao. Cara: collision-aware rate adaptation for ieee 802.11 wlans. In *IEEE INFOCOM*, 2006.

[7] M. Lacage, M. H. Manshaei, and T. Turletti. Ieee 802.11 rate adaptation: a practical approach. In *ACM MSWiM*, 2004.

[8] S. H. Y. Wong, H. Yang, S. Lu, and V. Bharghavan. Robust rate adaptation for 802.11 wireless networks. In *ACM MOBICOM*, pages 146–157, 2006.

[9] J. Zhang, K. Tan, J. Zhao, H. Wu, and Y. Zhang. A practical snr-guided rate adaptation. In *IEEE INFOCOM 2008 Minisymposium*, April 2008.

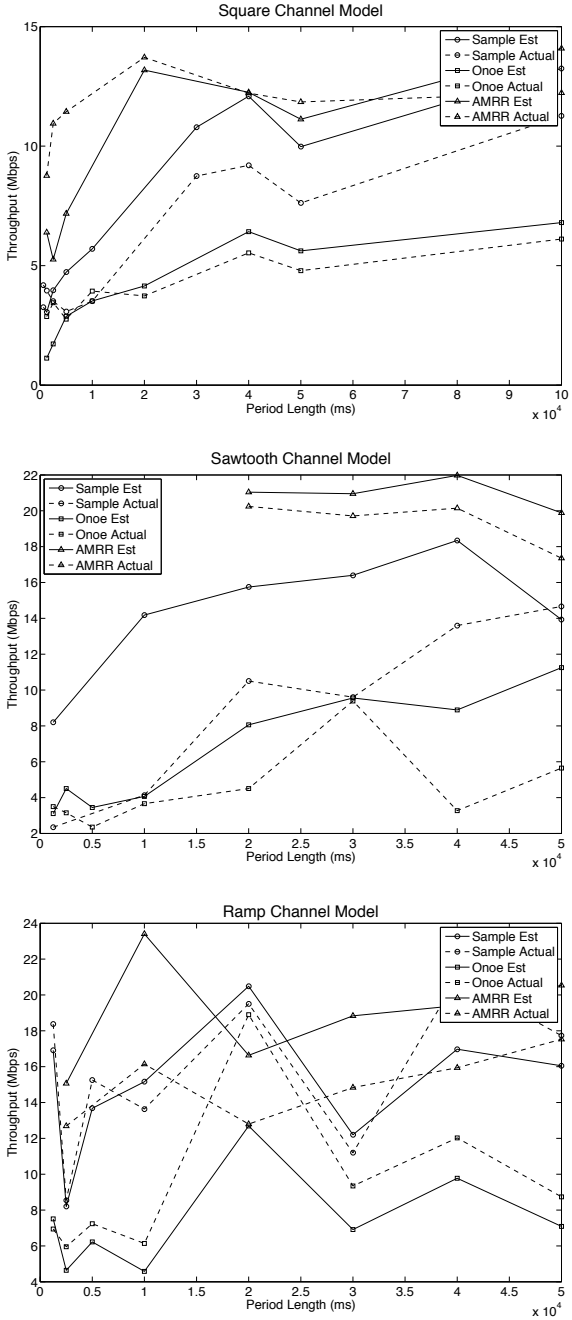

Figure 6: Comparison of actual versus estimated throughput.