Evaluation of Efficient Link Reliability Estimators for Low-Power Wireless Networks

Alec Woo and David Culler
{awoo,culler}@cs.berkeley.edu

Abstract Real-time wireless link reliability estimation is a fundamental building block for self-organization of multihop sensor networks. Observed connectivity at low-power is more chaotic and unpredictable than in wireless LANs, and available resources are severely constrained. We seek estimators that react quickly to large changes, yet are stable, have a small memory footprint and are simple to compute. We create a simple model that generates link loss characteristics similar to empirical traces collected under different contexts. With this model, we simulate a variety of estimators, and uses the simple exponentially weighted moving average (EWMA) estimator, as a basis for comparison. We find that recently propose flip-flop estimators are not superior. However, our cascaded EWMA on windowed averaging is very effective.

1 Introduction

Vast networks of low-power, wireless devices, e.g., sensor networks, raise a family of challenges that are variants of classical problems in a qualitatively new setting. One of these is the question of link loss rate estimation based on observed packet arrivals. Traditionally, this problem has been addressed such contexts as determining the degree of error coding redundancy or the expected number of retransmissions on a link. In sensor networks, it arises as part of network self-organization. The network comprises a large number of resource constrained nodes communicating via low-power radios, such that each transmission from a node is potentially heard by a small subset of the overall network. Based on observing packets from other nodes and performing a set of local rules, which may generate additional packets, the network must form and maintain a multihop routing topology to support some higher level communication pattern, such as a data aggregation into a specific node. For example, the sink node could announce its desire to receive data and its depth from itself, namely zero. Nodes that receive this packet can determine their depth (one) and start generating packets, which carry the depth and the node number along with data. A periphery of depth-two nodes learn about nodes of depth one and can start sourcing data for the sink. However, these packets will have to be routed through one of the nearer nodes. Each node hears packets from several neighbors and choose one with smaller depth as its "parent" to route its traffic over the next hop. Progressively, more distant nodes learn of parents and a spanning tree is formed and continually maintained as data flows toward the sink. It will route around obstacles and find alternative paths when nodes fail or join.

The problem with this elegant, simple algorithm, and with many of its variants, is that low-power radio communication is highly variable due to external sources of interference in the spectrum, contention with other nodes, multipath effects, obstructions, and other changes in the environment, as well as node mobility. As a result, the who-can-hear-whom relation, i.e., the radio cell, is not binary nor is it static. From the perspective of a node, the set of other nodes it can hear and the loss probability to or from those nodes varies abruptly over time and with large magnitude. Even with perfect knowledge of the environment, these variations are extremely hard to predict. In general, we think of wireless connectivity as a statistical relationship, $P_{ij}(t)$, representing the probability of successful packet transmission from node i to node j and time t. It generally falls off with distance, but not uniformly so. Sometimes, geographically nearby nodes may have poor connectivity, whereas with a large field of nodes is it likely that some node far away will hear clearly due to positive interference. [6] Thus, simply hearing a packet is not a good enough basis for determining that two nodes are 'connected'. The shortest path tree formed by the rule above will typically include long hops with very low link reliability. It is likely to be much better to take more hops using better (typically shorter) links.

Instead, each node keeps track of a set of nodes that it hears, either through packets addressed to it or packets it snoops off the channel, and builds a statistical model of the connectivity to/from each. Thus, we would like to gain a good estimate of $P_{ij}(t)$ at each node j from packets it hears (and does not hear) so we can use this to make the set of *neighbors* in the connectivity graph well defined. For example, we may consider a node a neighbor only if its probability of communication exceeds some threshold, say 75%. Connectivity is not necessarily symmetric, but nodes can broadcast estimates to neighbors or assume that most good links are roughly symmetric and verify the reverse direction for links that are actually used. With this information, the shortest path algorithm can be greatly improved. Each node picks as its parent the neighbor with lowest depth and link success rate over 75%. This implies that the entire shortest-path tree will be comprised of good links. Moreover, if we have a good estimate of the link success rates, we can use more sophisticated path selection rules. For example, we may assume inductively that all nearer nodes maintain as estimate of the path success rate from them to the sink. Once we have an estimate of the local link success rate, we can locally combine the two estimates, assuming that success rate along the path from the neighbor is independent of how we arrive at the neighbor, to select the parent that will give the maximum success rate to the sink and we can record this as the path estimate for use by children.

Thus, maintaining many local link success (or loss) rate estimators is essential for self-organization into multihop routing routing topologies in sensor networks. However, there are several challenges. The storage capacity of the nodes is very constrained and its processor is not very powerful. Thus, the estimators must use very little space and be simple to compute. Furthermore, it is not sufficient that the estimator eventually converge, since the link status changes fairly quickly. We want the estimator to be agile and to have a small settling time, so route selection can adapt reasonably quickly to changes in the underlying connectivity. However, there are also many transient variations in the link, so we want an estimator that is stable, so the routing topology does not change too chaotically. Moreover, fluctuations and errors in the estimate may introduce (temporary) cycles in the routing graph due to inconsistent partial information used in the route selection. These desires are clearly in conflict, stable estimators tend to be less agile and agile estimators tend to be less stable, especially ones that are inexpensive to compute.

This situation motivates us to investigate the behavior of a wide range of link estimators in the context of low-power wireless connectivity for the purpose of multihop route selection. The basis for estimation is the sequence of packets that a node observes. Thus, we can view this as a series of binary events over time. We only get to observe the 'ones' directly - arrival of a packet. However, when we receive a packet we can infer the intervening zeros from the sequence number. Of course, if we stop hearing from a node, the zeros are silent and we cannot, in general, know the expected packet rate from the node, even if we know its sample rate, since it may be performing local data compression, as well as routing traffic for other nodes. So, additional measures are required to estimate silent losses.

We begin, in Section 2, by reviewing recent investigations of estimation techniques for related networking problems. Section 3 provides an empirical analysis of the behavior of the low-power wireless network we are using in a variety of contexts. Section 4 develops our link loss model, mathematical framework, and simulation methodology. Section 5 defines the metrics and tuning objectives, and studies the behavior of a variety of candidate estimators in this setting against the exponentially weighted moving average which serves as the baseline of comparison. Section 6 looks at the behavior of the best estimators and discuss implications on multihop routing protocols, and we conclude in Section 7.

2 Related Work

Passive probing to estimate link reliability for wired and power-rich wireless networks is well established. It is widely deployed over the Internet in protocols such as Internal Gateway Routing Protocol (IGRP) [4] and Enhanced IGRP (EIGRP) [3]. Reliability is measured as the percent of packets that arrived undamaged on a link. It is reported by the network interface hardware or firmware, and is calculated as a moving average. In IGRP, link reliability of a route equals the minimum link reliability along the path. Through experience and extensive empirical study, effective estimators have been developed and tuned to the characteristics of particular link technologies and usage models. We build on that work in designing estimators for use in multihop routing on a low-power, mobile wireless network. In IGRP, packet loss is not silent since each packet comes in on a particular link, with both sender and receiver known *a priori*. Incoming packets are always detected and damaged packets are losses. For the wireless scenario, the channel is a broadcast medium and packets can be damaged or totally missed by the receiver. Furthermore, the link error dynamics are expected to be very different at low-power. We adapt the moving average calculation of IGRP to handle silent losses to obtain two candidate estimators in our study.

There is extensive prior work on modeling loss characteristics in various wireless networks. Nguyen*et al.* [7] use a trace-based approach for modeling wireless errors. Balakrishnan and Katz [8] collected error traces on WaveLAN and developed a Gilbert model. Konrad*et al.* [1] collected GSM traces and created a Markov-Based channel model. However, these wireless devices transmit at power levels that are two to three orders of magnitude greater than our radios. Thus, they potentially have very different reactions to background interference, environmental effects, and mobility. We draw upon the methodology developed in these studies to build an empirical characterization of our regime and study how well the established techniques carry over.

There is also a rich literature on network performance estimation, especially in the context of multicast and overlay topology management. Most of these efforts focus on active probing by injecting measurement traffic into the system, since direct measurement is not built-in as it is for link interfaces. For example, special multicast probes are used to estimate the internal multicast network packet loss rate and infer the overall topology. [12]. To minimize the power consumed by link estimation, we focus on passive techniques and avoid sending probe packets. Such additional active measures could be incorporated, yielding an interesting accuracy vs. energy trade-off.

Most prior work using passive estimation seeks to estimate a value from a large set where each observation is itself a meaningful approximation. For example, the round trip time estimator in TCP [13] can adjust its estimation based on each round trip time measurement. We must estimate the probability of reception from discrete events - the arrival of a packet or the silent failure to receive a packet. A single observation has little value; a history of observations must be used. Thus, estimators that have proved effective in other regimes may not be effective here. For example, a highly relevant study by Kim and Noble [11] studies the behavior of a collection of estimators in a context where both responsiveness and stability are desired in the face of sudden changes. By measuring round trip delay, they calculate the latest available bandwidth and filter it with an estimator. They assert that flip-flopping based on statistical process control between two EWMAs, with a agile and stable gain settings, provides the best estimator. Since each measurement reveals the latest estimate of available bandwidth, it can determine if the latest bandwidth falls within a certain prediction. If the agile estimation deviates so much that the process is likely to go out of control, the flip-flop drops the agile estimation and relies on the stable prediction. We do not have the same ability on a sample by sample basis. Instead, we develop estimators based on the flip-flop concept to gain a mix of agility and stability.

3 Empirical Link Characteristics

Empirical studies on the Berkeley mote platform [5] indicate that the connectivity of low-power wireless nodes is non-uniform and highly variable. [6] That study concludes that a probabilistic model of connectivity should be used. Although much more sophisticated statistical models have been proposed to characterize the inter-error intervals of wireless channels[2], we adopt this simple strategy and characterize the connectivity between nodes at a point in time by the probability of successful packet transmission. In this section, we seek to develop this characterization empirically.

3.1 Experimental platform

The widely available “Berkeley Mote” is used as the platform for this study[5]. It has a 4MHz Atmel microprocessor, with only 128kB of programmable memory and 1kB of data memory. The network uses a 916MHz, amplitude shift keying (ASK) radio, capable of delivering up to 50kbps. [9] The transmission power output is less than a milliwatt. TinyOS [10] provides a programming environment and a complete network stack on this platform. Its active message layer provides a connectionless packet abstraction, with a normal packet size being on the order of 30 bytes. A DC-balanced SECDED scheme is used for encoding each byte. A 16-bit CRC is computed over the entire packet and determines successful packet reception.

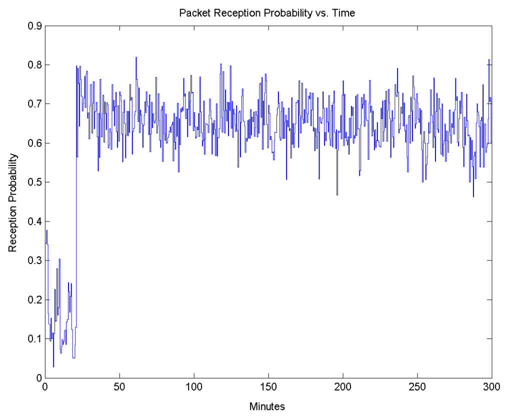
3.2 Experimental Setup

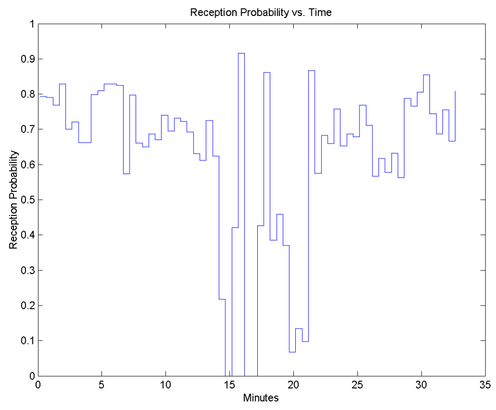
We collect packet loss traces over time with a single sender generating packets at a uniform rate and recording the successful arrivals at one receiver. The placement and environment of the nodes is varied. To capture ’real life’ behavior, all experiments are done in real settings rather than in an RF isolated chamber. Nodes are placed 3 inches above the ground. Any node movement is done by a person picking up and moving the node. The sender rate is 8 packets/s, with each packet containing a sequence number. The receiver redirects each successful packet over the serial port to a computer where the trace is stored. With sequence numbers in all traces, we can infer losses and generate a sequence of success/loss events, S, over time that constitute the trace. We calculate packet success rate over time, $W(t)$, as the fraction of packets received in each 30 second interval.

3.3 Empirical Traces

We present three traces to illustrate typical observed behavior. Figure 1(a) shows how packet reception rate reacts to a change in distance when the transmitter at 15 feet from the receiver is moved 7 feet closer. No further changes are made for more than four hours. The distance change results in a discrete change in packet success rate from approximately 15% to 65%. Furthermore, at either distance there is substantial variation, presumably due to interference on the RF channel, however, the distribution appears stationary.

Figure 1(b) shows how packet reception rate changes when a person stands in the vicinity receiver. The reception probability is very sensitive to the person’s position with discrete changes of substantial magnitude. At some times it blocks communication, at other times it has no effect, and at others it actually improves matters. These traces are from an indoor environment, however, our outdoor traces show similar behavior.





(a) Distance effects on $W(t)$. Nodes are placed 15 feet apart. After 15 minutes, the transmitter is moved to a distance of 8 feet, where it remains for four hours.

(b) Obstruction effects on $W(t)$. A person deliberately stands beside the receiver in the interval 15-20 minutes.

Figure 1: Empirical traces of link characteristics.

Figure 2 shows a more complex scenario that we will use throughout our analysis. The experiment begins with the sender placed 14 feet from receiver. After 9.5 minutes, the sender is placed 8 feet from the receiver. At 17.5 minutes, it is placed 4 feet from the receiver. At 21 minutes, it is moved back to 12 feet from the receiver. Finally, at 26.5 minutes, it was placed 4 feet from the receiver again. This results in a series of steps between stationary regions.

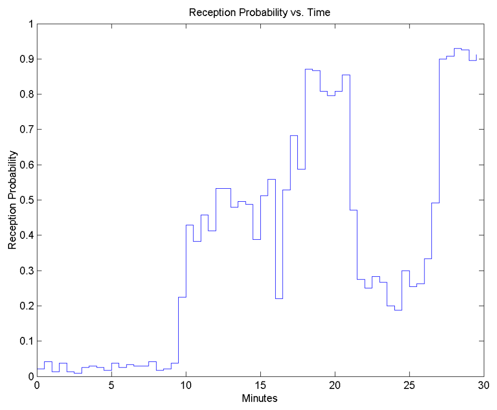


Figure 2: Movement effects on $W(t)$. Transmitter is deliberately moved to different distances at various times.

Based on these traces, a reasonable model of the communication behavior is given by the probability of packet reception over time, which changes in discrete steps from close to 0% to close to 100%. We would like to develop estimators that can track these discrete changes rapidly and yet also lock on to the stationary regions, despite statistical variation.

4 Synthetic Link Connectivity Model

In order to explore the estimator design space, we construct a simple synthetic trace generator based on these empirical traces. The input to the generator is the target packet success rate over time. It follows discrete steps as in Figure 2. Estimators observe the successful packets and estimate the underlying rate.

4.1 Trace Generation

Since the outcome of packet reception is either loss or success, a simple model is to treat each packet reception as a Bernoulli Trial, with 1 denoting success and 0 loss, where p equals probability of success or $1 - p$ equals probability of loss. The sequence of packet receptions is a Bernoulli Process, and the number of packet successes over n trials follows the Binomial distribution. If this process is described by binomial random variable, M, $E(M) = n * p$. Packet success rate is itself a random variable, $\hat{W}$, with expected value equals to $\frac{E(M)}{n}$, which equals p.

The assumption of this model is that each trial is independently and identically distributed. In reality, this may not be the case, but we can compare the resulting behavior to observed traces. To investigate whether W, follows the Binomial distribution when there is no observable physical influence, we plot quantiles extracted from the stationary portion of our data in Figure 1(a) against quantiles derived from the theoretical Binomial distribution. The expected value of W from the data set is 65%, so we set the expected value in the Binomial distribution to this value. By a quantile, we mean the fraction of points below the given value. The quantile-quantile graph is shown Figure 3. If the data in Figure 1(a) follows the Binomial distribution, the data set should be linear along the 45 degree line.

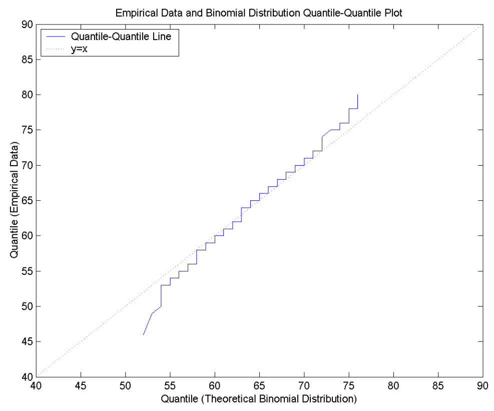


Figure 3: Quantile of empirical data against quantile of binomial distribution.

Figure 3 shows a good match when the quantile is near the mean, but deviates slightly at both extremes. This suggests that the empirical data has a larger degree of variance than the Binomial distribution model.

Nonetheless, Figure 3 suggests that the Binomial distribution is a fairly good model to approximate reality.

4.2 Simulation Model

We use this simple mechanism to generate packet loss traces for use in evaluating different estimators in simulation. Let $U \in \{0, 1\}$ be a uniform random variable. At each packet arrival at time t, if $U <= p(t)$ then the packet is received successfully ($M(t) = 1$); otherwise, it is a loss ($M(t) = 0$). Estimators observe the sequence M and produce an estimate $\hat{P}(t)$, which can be compared to the underlying target, $p(t)$.

To model changes of link quality resulting from mobility or obstacles at the receivers, we make $p(t)$ a piecewise function of time. To model Figure 2, $p(t)$ is a sequence of steps shown in Table 1.

t (minutes)	$p(t)$
0 - 9.5	2.63%
9.5 - 17.5	46.57%
17.5 - 21	83.40%
21 - 26.5	28.22%
26.5 - 30	91.18%

Table 1: Definition of $p(t)$ to model Figure 2

Figure 4 shows the time series comparison between the empirical trace and the simulated trace using $p(t)$. As with the empirical trace, the simulated $\hat{W}(t)$ is obtained by taking the faction of packets received in each 30 second interval. The simulated trace captures the essence of the empirical trace, except with a slightly smaller degree of variance overall.

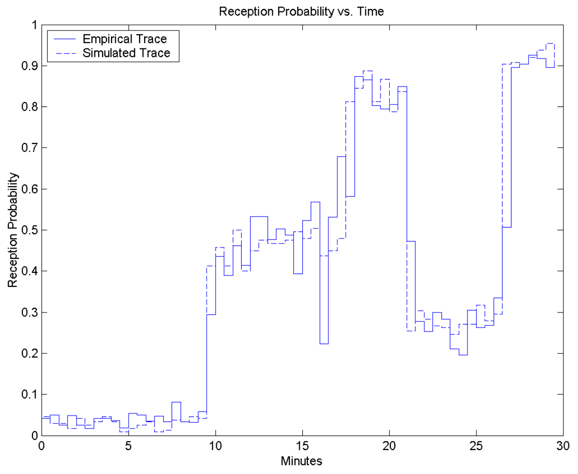


Figure 4: Time series comparison of empirical traces with simulated traces.

5 Estimators

In this section, we layout the general form of our candidate estimators. We describe a set of metrics for evaluation, and discuss the relationship between agility, stability, and amount of history required for estimation. We then define objectives in tuning the estimators, and present each estimator along with the corresponding parameters for meeting the tuning objectives.

5.1 General Framework

Our goal is to design link reliability estimator that is responsive, yet stable, reasonably accurate, simple with little computational requirement, and memory efficient. Inputs to the estimators include packet arrivals, M, and periodic timer events, T. Each packet contains a transmitter ID and a link sequence number. Since a lost packet does not generate any message arrival events, every M is equivalent to zero or more packet loss events followed by a packet success event.

The periodic timer event provides a synchronous input to the estimator that allows it better estimate losses when message events are infrequent. For example, if a node were to disappear no later message events would occur, yet the connectivity estimate should go to zero. One temporal assumption is that higher layer protocols can provide a minimum message rate, R, for neighboring nodes. If R is known, estimators can safely infer the minimum number of packet losses over the time period, T, and compensate accordingly.

For each estimator, there is a continuous tuning space. To make fair comparisons among different estimators, we pick two meaningful points in the tuning space as our tuning objectives. One point is to tune for best agility given a stability target. The other is to tune for best stability given a agility target. Individual estimators are well tuned to meet these objectives before being compared against each other. We use the simulated trace, $\hat{W}(t)$, shown in Figure 4, as input to the tuning process. The data rate was set to be 8 packets/s, which is the same as the empirical trace. We also set R to this value in order to drive the best out of these estimators.

This general framework is captured in Figure 5.

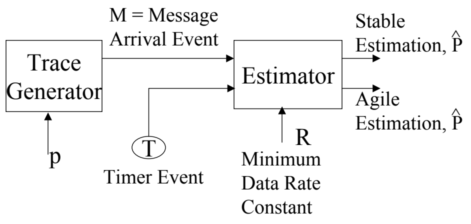


Figure 5: General framework of passive link estimators.

5.2 Metrics

The following metrics are used to evaluate estimators relative to our framework. *Settling Time* is the length of time after a step in $p(t)$ before $\hat{P}$ reaches within $\pm\epsilon\%$ of $p(t)$ and remains within that error bound. We use a threshold of $\epsilon = 10\%$. *Crossing Time* is the length of time after a step in $p(t)$ before $\hat{P}$ first crosses $\pm\epsilon\%$ of $p(t)$. Since $E(\hat{P}) = p$ is known to us, we can compute *mean square error* $(p - \hat{P})^2$ which captures not only the degree of error, but also places a higher penalty to large overshoot or undershoot. *Coefficient of variance* measures how stable an estimator is after reaching steady state. *Sum of errors* is used to capture if the estimator is biased which may lead to systematic errors. Finally, memory resources and computation complexity measures how efficient is each estimator.

5.3 Agility, Stability, and History

It is important to understand the tradeoff between estimation stability, agility, and the amount of history used to generate the estimation. In general, a larger history yields a more stable, but less agile estimation. In the case of the Binomial distribution, p also determines the variance of the number of outcomes that are 1's since $\sigma^2(X) = n * p * (1 - p)$. The variance of $\hat{P}$ is $\frac{\sigma^2(X)}{n^2} = \frac{p*(1-p)}{n}$, which should peak if $p = 0.5$ for a given n. To decrease the variance in $\hat{P}$, the number of samples n should be increased. The problem arises if we look at the standard deviation rather than variance. Since $\sigma(\hat{P}) \propto \frac{1}{\sqrt{n}}$, decreasing the standard deviation requires a quadratic increase in history size. This analysis suggests that tuning for stable estimation of $\hat{P}$ may lead to an increase in history size that poses a resource concern for some types of estimators and reduces agility. It is difficult to obtain a agile estimator with small error or variance, unless p is either close to 0 or 1.

5.4 Tuning Objectives

We tune each estimators being studied for two settings: stable and agile. For stable estimators, we aim to minimize the settling time while requiring the total error $\epsilon < 10\%$ and tune each estimator accordingly. For agile estimators, we aim to minimize the total error while requiring that the crossing time, (i.e. $\hat{P}$ reach $\pm 10\% of p$), within 40 packet opportunities. The crossing time is chosen arbitrarily since our concern is to reveal the different shortcomings among different estimators when they are tuned to the same objective. However, it is about half of what we expect for stable estimators. With such tuned estimators, we compare settling time, crossing time, mean square error, and coefficient of variance, as well as memory resources and computational requirement. Settling time is only meaningful for the stable estimators.

5.5 Terminology

We first establish the relevant terminology before presenting the different estimators. We let $\hat{P}$ be the current estimation, t be the time stamp of the last M event, m be the number of known missed packets based on sequence number difference, and k be a guess on the number of missed packets based on R over a time window between the current T event and the last M or T event. Let l be the number of packet losses we feed into the estimator. For every T event, $l = k$. For every M event, $l = max(m - k, 0)$.

5.6 EWMA(α): Basis of Comparison

The exponentially weighted moving average (EWMA) estimator is very simple and memory efficient, requiring constant storage of the old estimate for any kind of history tuning. Since EWMA is so widely used, it is the estimator that we use as the basis for comparison. EWMA uses a linear combination of infinite history, weighted exponentially. It has the property of being reactive to small shifts and is often used as an agile detector in many statistical process control applications.

The algorithm works as follows. Let $0 < \alpha < 1$ be the tuning parameter. At any M or T event, repeat $\hat{P} = \hat{P} * \alpha$ for l times. If it is a M event, compute $\hat{P} = \hat{P} * \alpha + (1 - \alpha)$. If the input is a T event, $l = k = \lfloor(\text{current time} - t) * R\rfloor$. That is, on successive T events, without intervening M events, additional losses are accumulated into k. If the input is a M event, a packet must have been received successfully. Therefore, we set t equals to current time stamp. To calculate m, we extract the sequence number from the successful packet in M and subtract it from the last sequence number heard plus one. With m calculated, $l = max(m - k, 0)$ and we set k=0. Note the process of maintaining k, t, last sequence number heard, and the calculation of l apply to all estimators and is orthogonal the actual estimator's algorithm. The implementation of EWMA will take 4 bytes (floating point) or 1 byte (fixed point) to store $\hat{P}$ and the amount of computation involved is 2 multiplications and 1 additions.

Figure 6(a) shows $\hat{P}(t)$ of the tuned, stable estimator, with $\alpha = 0.99$. It reveals that to keep within 10% error, EWMA is already set very close to its maximum gain of 1. With such a large gain, agility is not to be expected. The crossing time for EWMA is 167 packets while settling time is close to 180 packets. Figure 6(b) shows the agile version with $\alpha = 0.9125$. It is probably not a useful estimator since it has large overshoots and undershoots, which is expected since EWMA is sensitive to small shifts. Nevertheless, the agile version is good for detecting disappearance of a neighboring node over a relatively short time. Note that a small decrease of α from 0.99 to 0.9125 has a large effect on agility and error, something we do not normally see in other contexts. Furthermore, in practice, representing α using fixed point to avoid heavy weight floating point operations may create extra complexity since α needs to be quite precise.

5.7 Flip-Flop EWMA($\alpha_{stable}, \alpha_{agile}$)

A Flip-Flop between two EWMA estimators, with a stable and a agile setting, is suggested to be the best estimator to provide both agility and stability [11]. The use of statistical control theory provides the basis for switching between the two estimators. To explore the possibility of using such an estimator, we set the agile and stable EWMA tunings to be as in the EWMA study. Since the α_{stable} has a 10% noise margin, a simpler approach to switch between agile and stable estimator when their difference is greater than 10%. Note that flip-flop can switch in two directions. One is to be agile by default. When the agile estimation is out of control and deviates more than 10% difference from the stable estimation, we fall back on to the stable estimation. The resulting $\hat{P}(t)$ is shown in Figure 6(c). The other approach is to be stable by default, but switch to the agile estimation since it can detect sudden change such as mobility much earlier. The switching occurs when the agile estimation deviates from the stable one by more than 10%. The resulting $\hat{P}(t)$ is shown in Figure 6(d).

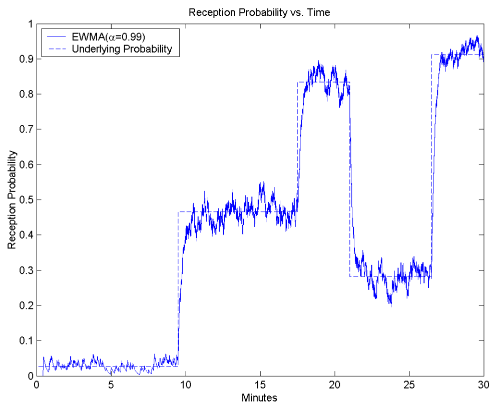
The two graphs suggest that the flip-flop idea does not provide an advantage over the simple EWMA in our setting. This is because fluctuations of the agile estimator are so bad that it only introduces instability and error. The study in Kim*et. al* [11] does not show the dynamics of either estimator separately over time, so it is difficult to isolate why it makes so much better in that setting.

5.8 Moving Average(n)

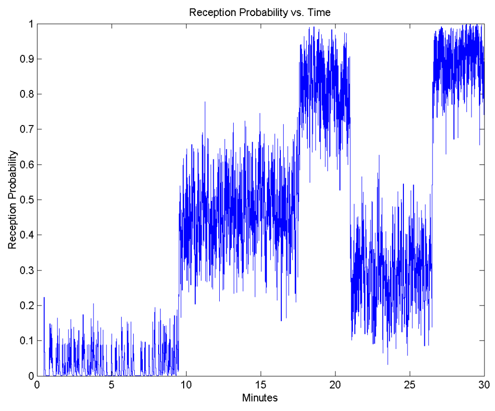
The moving average estimator is another simple estimator that is widely used, including in IGRP routers. The algorithm works as follows. Let n be the tuning parameter for the maximum size of a history window of bits, $h \in 0, 1$. At any given event, append l zeros to the end of h. If it is a M event, append 1 to the end of h, and take the last n elements in h. Then, $\hat{P} = \frac{\sum_{i=1}^{s} h(i)}{s}$.

To avoid overestimating $\hat{P}$ in the case when there are only a few samples, the estimator gives no estimation, $\hat{P} = 0$, if the number of samples is below some threshold, ϕ. The implementation of such algorithm will take $\lceil \frac{n}{8} \rceil$ bytes to store h and the amount of computation involved in computing $\hat{P}$ is n bit shifts, 1 addition, and $log(n)$ shifts rather than a full division. Note that for ease of implementation, the tuning process takes n in multiples of 8.

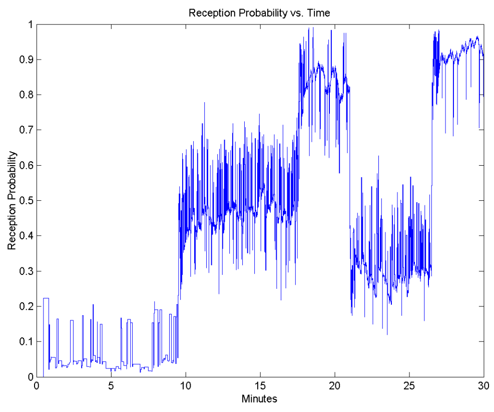
Figure 6(e) illustrates $\hat{P}(t)$ of the tuned, stable moving average estimator with stable settings of $n = 144$. It achieves a settling and crossing time of about 120 packets, so it is much more agile than EWMA. However, with $n = 144$ or 18 bytes of storage per estimator, it is expensive to use to keep track of a reasonable number of neighboring nodes. Figure 6(f) shows the agile case with $n = 24$. Moving average appears to have less error and variance in its estimation compared to EWMA. In Figure 6(b), EWMA is more sensitive to small changes.



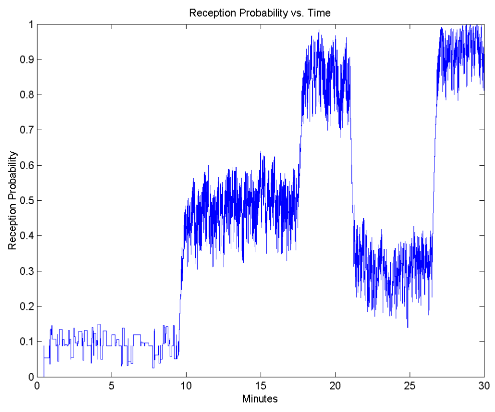
(a) Stable EWMA($\alpha = 0.99$) and $p(t)$.



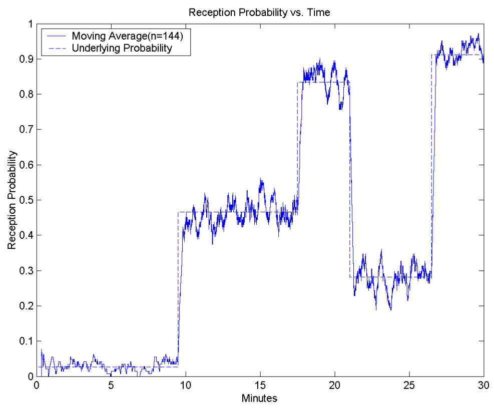
(b) Agile EWMA($\alpha = 0.9125$) estimation.



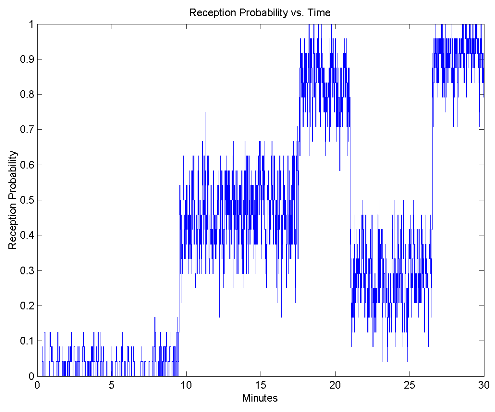
(c) Flip-Flop EWMA($\alpha_{stable} = 0.99, \alpha_{agile} = 0.9125$). It uses the stable estimation if the agile estimation goes beyond 10% from the stable estimation.



(d) Flip-Flop EWMA($\alpha_{stable} = 0.99, \alpha_{agile} = 0.9125$). It uses the agile estimation if that goes beyond 10% from the stable estimation.



(e) Stable moving average ($n = 144$) and $p(t)$.



(f) Agile moving average with ($n = 24$).

Figure 6: $\hat{P}(t)$ for different estimators at both stable and agile configuration.

5.9 Time Weighted Moving Average (TWMA)(n,w)

The moving average estimator applies the same weight on all packets within the sliding window. A common improvement is to apply a weighting function which places heavier weight on more recent samples so the estimation can be more adaptive to temporal changes. The basic algorithm works the same as the moving average except with an addition of a time weighted function, w. The tuning parameters for this estimator are n and w. In our study, we stick to one weighting function, w, and only tune n. While w is not the perfect function, it serves a purpose for observing the effect on weighting.

The w that we choose works as follows. Let h be a sequence $\in 0, 1$ and s be the current size of h. Let w be a sequence of length s. w applies weight=1 for the most recent $\lceil s/2 \rceil$ packets in h and decreases the weight for the least recent $\lfloor s/2 \rfloor$ samples linearly from 1 to $\frac{1}{\lceil s/2 \rceil}$. Therefore, $\hat{P} = \frac{\sum_{i=1}^{s} w(i)*h(i)}{\sum_{i=1}^{s} w(i)}$.

The implementation of this estimator also takes $\lceil \frac{n}{8} \rceil$ bytes to store h bits. As for the amount of computation, since $h \in 0, 1$, the multiplications can be turned into additions. As a result, there are n additions and 1 division. This carries more complexity compared to moving average. Since w is different for different s, a fixed size lookup table can be used to store all w given n is fixed. Note that for ease of implementation, the tuning process also takes n in multiples of 8.

Figure 7(a) illustrates $\hat{P}(t)$ of the tuned, stable TWMA estimator with $n = 168$ and Figure 7(b) shows the agile version with $n = 32$. Visual comparison of the same figures for moving average show the two are very similar and both have better settling time and less high frequency fluctuation that EWMA. However, the effect of the weighting function is twofold. First, it increases the history required to achieve our stability objective from 144 to 168, making it less memory efficient as compared to moving average. Second, it is likely that w requires floating point operations which we try to avoid. However, w improves over moving average by decreasing the average settling time from 122 to 113 packet time, while maintaining the same amount of error and variation. As for the agile case, $n = 32$ is also greater than 24 for the moving average case. Nonetheless, its mean square error and coefficient variance are also smaller than EWMA and moving average.

5.10 Flip-Flop Packet Loss and Success Interval with EWMA (FFPLSI) ($\alpha_{success}, \alpha_{loss}, ff$)

The packet loss interval is the number of consecutive successful packets in between two successive packet loss events. The greater the interval, the better is the reception probability. An estimation of the packet loss interval adapts slowly to bursts of packet successes, but reacts quickly to bursts of packet losses.

The estimator works as follows. The tuning paramter is α_{loss}. Let I be the current average of loss interval and i be the most updated number of consecutive success when a packet loss is detected. The average I is computed using a EWMA as follows. For each i, $I = I * \alpha_{loss} + (1 - \alpha_{loss}) * i$. At any instance, $\hat{P}(t) = \frac{I}{I+1}$.

The packet success interval is the reverse of the packet loss interval. It measures the number of consecutive packet losses in between two successive packet success instances. The estimation of this average corresponds to the average bursts of errors. The greater the interval is, the worse is the quality of the link. Packet success interval adapts slowly to bursts of packet losses but it reacts quickly to bursts of packet successes.

The algorithm of packet success interval is similar to packet loss interval, with I being the average packet success interval and i being the most updated number of consecutive loss when a packet success is detected. The tuning parameter is $\alpha_{success}$. For each i, $I = I * \alpha_{success} + (1 - \alpha_{success}) * i$. At any instance, $\hat{P}(t) = 1 - \frac{I}{I+1}$.

The flip-flop mechanism can be used to attempt to capture the best of both worlds. For stability, packet loss interval should be used when successes are frequent ($\hat{P} >= 50\%$) while packet success interval should be used when losses are frequent ($\hat{P} < 50\%$). We call this configuration as ff=STABLE. For agile estimations, it should be the reverse, and we call this ff=AGILE.

Since EWMA is used for averaging, the implementation of this estimator is very efficient. For each entry, it only takes 2 bytes to store the intervals and 2 bytes (fixed point) or 8 bytes(floating point) to store $\hat{P}$. Like EWMA, parameters tuning do not affect the storage requirement.

Figure 7(c) shows $\hat{P}(t)$ for the tuned, stable (ff=STABLE) estimator, with $\alpha_{success} = 0.98$ and $\alpha_{loss} = 0.98$. This estimator is very stable and smooth around both extremes at 0 and 100%. However, the slow rising edges show that its settling and crossing time are much larger compared to other estimators, even with EWMA. The agile case is shown in Figure 7(d), with $\alpha_{success} = 0.85$ and $\alpha_{loss} = 0.85$. This estimator is not a good agile candidate, since its fluctuations are larger than EWMA.

5.11 Window Mean with EWMA (WMEWMA) (t, α)

So far, all estimators that we discussed update the estimation for every M event. It is possible to perform low-pass filtering by taking an average of a time window and adjust the estimation using the latest average. The average is actually an observation of $\hat{P}$ and EWMA can be used to more filtering. The tuning parameters are t and α. Let t be the time window represented in number of message opportunities between two T events, and $0 < \alpha < 1$.

The algorithm works as follows. $\hat{P}$ is only updated at each T event. In the time window t between two T events, let r be the number of received messages (i.e. number of 1's in M events), and f be the sum of all losses, l, from all M events and the current T event. The mean $\mu = r/(r + f)$, and $\hat{P} = \hat{P} * \alpha + (1 - alpha) * \mu$.

For each entry in this estimator, it will take 2 bytes for storing r and f, and 1 byte (fixed point) or 4 byte (floating point) for storing $\hat{P}$. The amount of computation for $\hat{P}$ involves 2 additions, 1 division, and two multiplications. The computation is done per T event rather than per M event. Similar to EWMA, storage requirement of this estimator is independent of tuning.

Figure 7(e) shows $\hat{P}(t)$ of the tuned, stable estimator with $t = 30$ message time and $\alpha = 0.60$. The observed settling time and crossing time is relatively small, and the high frequency components in the estimation are clearly removed as compared to Figure 6(a) in the EWMA case. In fact, its settling time is comparable to fastest time weighted moving average. Figure 7(f) shows the agile version with $t = 10$ message time and $\alpha = 0.3$. Although the windowing effect has low pass filtering effect, using a small t actually creates shifts to which EWMA is sensitive. As a result, the performance in the agile scenario does not show significant improvement over EWMA.

5.12 Alternative Estimation Techniques

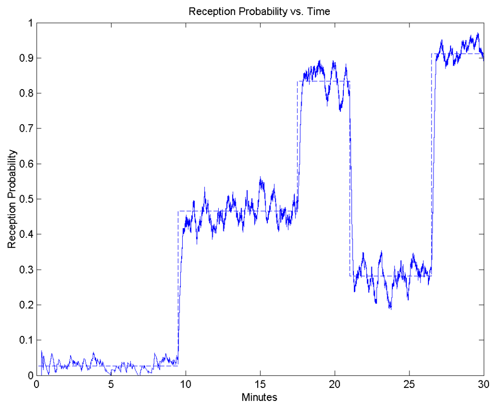
The resource constraints on our platform significantly limit the amount of processing and storage one can do, which narrows the choice of estimator significantly. Computing a statistically meaningful median upon previous estimates already raises a concern on storage resources, given there are potentially many neighboring nodes one need to estimate to perform routing. Similarly, the rich literature on estimation techniques, such as linear regression or Kalman filter, are not practical on our platform.

6 Discussion

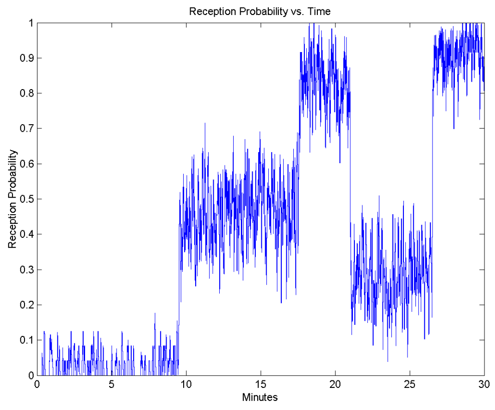
With our controlled study of stable and agile estimators in hand, and we return to the question of what is best estimator relative to the metrics we are considering. We discuss how these results may affect multihop routing and give guidelines on what to expect when designing such protocols.

6.1 Stable Estimators

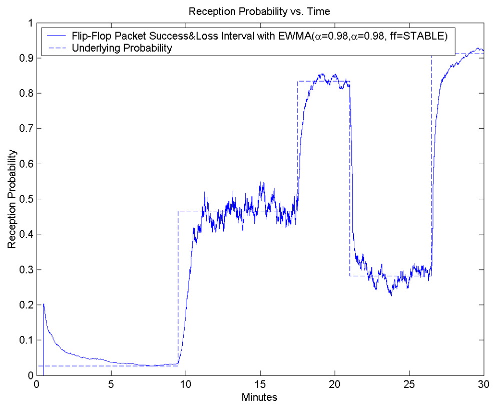
Table 2 summarizes the result of the stable estimators. We first look at sum of errors. The value should approach 0 if the estimator is unbiased. For all estimators, the sum of errors is small, showing that they are not biased. The mean square error penalizes estimators that have large overshoots or undershoots. FFPLSI is very stable at the extremes. As a result, it achieves the smallest mean square error as expected, but other estimators come close to it. The coefficient of variance measures the effectiveness of the estimator in staying within the true value. Flip-Flop Packet Loss and Success Interval with EWMA has the largest coefficient of variance while EWMA has the best. Again, values for other estimators are relatively close. The major determining factor is the settling time. Moving Average, TWMA, and WMEWMA all have much smaller settling times than the rest of the estimators. It is desirable to have the most agile estimator which can still stay within 10% of the true value, even if it does not have the best mean square error and coefficient of variance. One would hope that the crossing time for EWMA and FFPLSI will be much smaller than the actual setting time. However, from the Figures of $\hat{P}(t)$ and Table 2, it is clear that the crossing times are only slightly smaller than the settling times. Another important constraint is storage space. Since Moving Average, and TWMA does not have constant storage space,



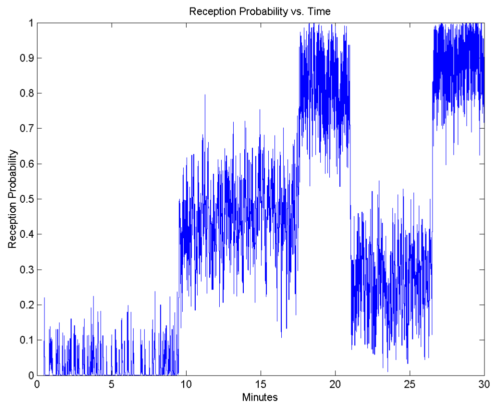
(a) Stable time weighted moving average estimation ($n = 168$) and $p(t)$.



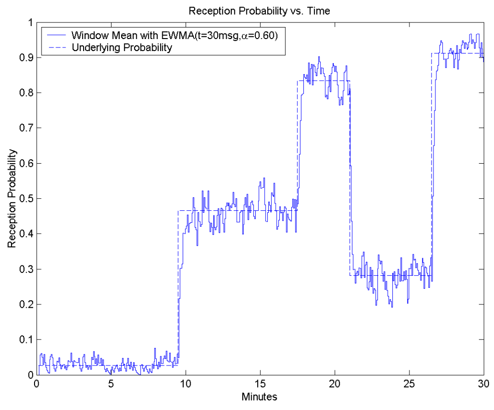
(b) Agile time weighted moving average estimation ($n = 32$).



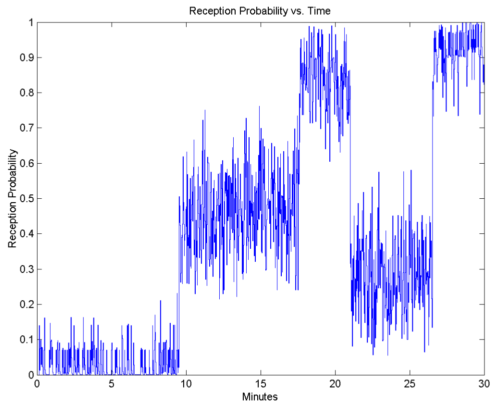
(c) Stable flip-flop packet loss and success interval with EWMA estimation ($\alpha_{success} = 0.98, \alpha_{loss} = 0.98, ff =$ STABLE) and $p(t)$.



(d) Agile flip-flop packet loss and success interval with EWMA estimation ($\alpha_{success} = 0.85, \alpha_{loss} = 0.85, ff =$ AGILE).



(e) Stable window mean with EWMA($t = 30, \alpha = 0.6$) and $p(t)$.



(f) Agile window mean with EWMA($t = 10, \alpha = 0.3$).

Figure 7: $\hat{P}(t)$ for different estimators at both stable and agile configuration.

Estimator	Settling Time (Packet Time)	Crossing Time (Packet Time)	Mean Square Error ($\%^2$)	Coefficient of Variance	Sum of Errors	Storage per Entry (bytes)	Computation Overhead
EWMA	178	167	$9.32x10^{-4}$	0.19%	0.10%	1	(2 mul.,1 add.)
Moving Average	122	122	$11x10^{-4}$	0.22%	0.26%	$\lceil \frac{n}{8} \rceil$	(1 add., $n + log(n)$ shifts)
TWMA	113	113	$11x10^{-4}$	0.22%	0.23%	$\lceil \frac{n}{8} \rceil$	(1 div., n add.)
FFPLSI	292	271	$8.98x10^{-4}$	0.33%	-0.18%	2	(4 mul., 2 add.)
WMEWMA	118	113	$13x10^{-4}$	0.27%	0.16%	3	(2 mul.,1 div, 2 add.)

Table 2: Simulation results of all estimators in stable settings.

Estimator	Settling Time (Packet Time)	Mean Square Error ($\%^2$)	Coefficient of Variance	Sum of Errors	Storage per Entry (bytes)	Computation Overhead
EWMA	21	$65x10^{-4}$	2.41%	0.28%	1	(2 mul.,1 add.)
Moving Average	23	$55x10^{-4}$	2.1%	0.24%	$\lceil \frac{n}{8} \rceil$	(1 add., $n + log(n)$ shifts)
TWMA	25	$48x10^{-4}$	1.87%	0.22%	$\lceil \frac{n}{8} \rceil$	(1 div., n add.)
FFPLSI	23	$80x10^{-4}$	3.14%	-0.98%	2	(4 mul., 2 add.)
WMEWMA	21	$70x10^{-4}$	2.7%	0.18%	3	(2 mul., 1 div, 2 add.)

Table 3: Simulation results of all estimators in agile settings.

WWEWMA seems to be the best choice given it is well balanced in all dimensions.

6.2 Agile Estimators

Table 3 summarizes the performance of the agile estimators. For sum of errors, FFPLSI is five times larger than its stable counterpart. EWMA also has 3 times increase. This suggests that these two estimators can be biased in agile configuration. Our agile settings decrease the settling time by 5 to 10 times relative to the stable settings. However, mean square error and coefficient of variance increase by roughly the same factor. It appears that agile estimator is only useful to discover a significant change in link reliability quickly, such as disappearance of a node.

6.3 Multihop Routing

The motivation of the study was to understand how the performance of link estimations affect higher level algorithms in particular multihop routing. Our results suggest that one can either build a agile estimator with large errors or a stable estimator with a settling time of about 100 packets. For routing purposes, stable estimator is a clear choice. Even so, the estimate will only be accurate to about 10%. In choosing a parent for routing, fluctuations of up to about 10% should be tolerated before switching for better alternative. If reliability is used as a cost metric, care must be exercised to avoid cycles due to variations in the reliability estimate.

6.4 Estimation with Empirical Trace

Figure 8 shows how WMEWMA estimator with the stable setting, performs on the empirical trace that shaped the trace generator. Our final choice estimator tracks the empirical trace well. The degree of overshoot and undershoot is higher than in simulation. This is expected since real traces $W(t)$ have larger variances that $\hat{W}(t)$. As a result, estimators tuned using $\hat{W}(t)$ should be tuned for more stability when applied in real situations.

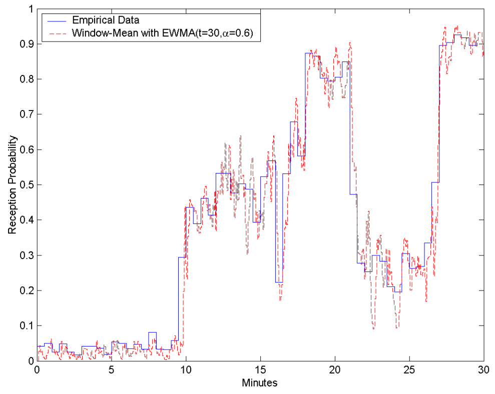


Figure 8: WMEWMA estimations using empirical data.

7 Conclusion

Real time estimation of link reliability is vital in any self-organizing network, wired or wireless, since routing paths should never be constructed over poor links. In the case of low-power wireless networks, our empirical traces show that link reliability varies dramatically with changes in position or with the environment. This observation justifies the need to estimate link reliability in the case of routing, since a path over two 80% reliability links is clearly better than a one hop path over a 50% reliability link.

Our traces also indicate that link reliability fluctuates over time like a random process, even when no observable physical changes are present. We model such fluctuations using a binomial distribution, with step responses to model mobility and obstacles. We found that such a model is a good approximation of the empirical data and it provides a simple tool for evaluation of the estimator design space.

During the search process of different estimators, the resource constraints of our platform filters out many complex or inefficient estimation techniques and helps us focus on a few efficient estimators. Using the different metrics that we defined, we found that EWMA over an average in a time window (WMEWMA) performs best overall. It provides stable estimates within 10% that react to large steps in about 100 packet times. Furthermore, the storage requirement is constant for all tunings, making it an attractive estimator for our resource constrained platform. Our study has narrowed down the estimator design space into this one estimator and suggests a reasonable parameter setting. What remains is a final process of tuning it in real settings.

References

[1] A. Konrad, B.Y. Zhao, A.D. Joseph, Reiner Ludwig. Explicit loss notification and wireless web performance. In *ACM MSWIM*, 2001.

[2] B. Mandelbrot. Self-similar error clusters in communication systems and the concept of conditional stationarity. In *IEEE Transactions on Communication Technology COM-13*, pages 71–90, 1965.

[3] Bob Albrightson and J.J. Garcia-Luna-Aceves and Joanne Boyle. EIGRP - a fast routing protocol based on distance vectors.

[4] C. Hedrick. An introduction to IGRP. In *Rutgers - The State University of New Jersey Technical Publication, Laboratory for Computer Science*, August 1991.

[5] Crossbow, Inc. *www.xbow.com.*

[6] Deepak Ganesan, Bhaskar Krishnamachari, Alec Woo, David Culler, Deborah Estrin and Stephen Wicker. Complex behavior at scale: An experimental study of low-power wireless sensor networks. In *Technical Report UCLA/CSD-TR 02-0013*, 2002.

[7] G.T. Nguyen, R. Katz. A trace-based approach for modeling wireless channel behavior. In *Proceedings of the Winter Simulation Conference*, pages 597–604, 1996.

[8] H. Balakrishnan, R. Katz. Explicit loss notification and wireless web performance. In *IEEE Globecom Internet Mini-Conference*, 1988.

[9] Jason Hill, David Culler. A wireless-embedded architecture for system level optimization. In *UC Berkeley Technical Report*, 2002.

[10] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, Kristofer Pister. System architecture directions for networked sensors. In *ASPLOS*, 2000.

[11] M. Kim, B. Noble. Mobile network estimation. In *ACM Mobicom*, 2001.

[12] R. Cceres, N. G. Duffield, J. Horowitz, F. Lo Presti, and D. Towsley. Loss-based inference of multicast network topology. In *IEEE Conference on Decision and Control*, 1999.

[13] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM*, pages 314–329, 1988.