Mobile Network Estimation

Minkyong Kim and Brian Noble
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, MI 48109
{minkyong, bnoble}@umich.edu

Abstract

Mobile systems must adapt their behavior to changing network conditions. To do this, they must accurately estimate available network capacity. Producing quality estimates is challenging because network observations are noisy, particularly in mobile, ad hoc networks. Current systems depend on simple, exponentially-weighted moving average (EWMA) filters. These filters are either able to detect true changes quickly or to mask observed noise and transients, but cannot do both. In this paper, we present four filters designed to react quickly to persistent changes while tolerating transient noise. Such filters are *agile* when possible, but *stable* when necessary, adapting their behavior to prevailing conditions. These filters are evaluated in a variety of networking situations, including persistent and transient change, congestion, and topology changes. We find that one filter, based on techniques from *statistical process control*, provides performance superior to the other three. Compared to two EWMA filters, one agile and the other stable, it is able to offer the agility of the former in four of five scenarios and the stability of the latter in three of four scenarios.

Keywords

mobile network estimation, filtering, adaptive systems

1 Introduction

It is widely recognized that adaptation to changing networking conditions is critical to mobility [8, 13]. However, to adapt to network conditions one must first know what they are and how they change over time. Specifically, a mobile node must know the *available network capacity* that it could potentially utilize.

This is a difficult problem, since the effective latency and bandwidth between a mobile host and other nodes is constantly changing. This may be due to ad hoc topology changes, vertical handoff across connection alternatives, or wireless fading and shadowing [14, 23]. These causes are in addition to the more prosaic routing changes and congestion endemic to wired networks of even modest scale [16, 22]. Together, these factors conspire to produce frequent changes in available latency and bandwidth, and induce substantial noise in individual network observations.

Converting noisy observations into an estimate of available latency and bandwidth is an instance of the *filtering* problem. Observations are fed into a filter, which smoothes them in some way to produce estimates. Typically, systems employ an *exponentially-weighted moving average* (EWMA) filter to this problem. Given a new observation, an EWMA filter produces a new estimate as a linear combination of the old estimate plus the new observation, each given some weight.

Unfortunately, the *gain*, which determines the proportional weight assigned to the new observation and the old estimate, is fixed in traditional EWMA filters. When old estimates are given more weight, the filter provides good *stability*; it resists noise in individual observations. When new observations are given more weight, the filter provides good *agility*; it is able to detect performance changes quickly. Neither property is desirable at all times. Ideally, one would like to have a filter that is agile when possible but stable when necessary, depending on current circumstances. In other words, filters should be adaptive, just as other components of the system must be.

This paper describes our experiences designing and evaluating filters that trade stability for agility based on the prevailing situation. We have designed several candidate filters in an attempt to meet this goal. One, called *Flip-flop*, is a composition of an agile EWMA filter and a stable one. Flip-flop selects between them using a technique borrowed from statistical process control [20, 24, 26]. Two others, *Stability* and *Error-based*, use heuristics to vary the gain of an EWMA filter continuously, in order to select for agility or stability based on the behavior of observations and estimates, respectively. Finally, we have applied a variant of the well-known *Kalman* filter [9].

We subjected each candidate filter to a variety of network-

To appear in the Proceedings of the ACM Conference on Mobile Computing and Networking, Rome, Italy, June 2001.

ing scenarios, comparing them to two EWMA filters: one agile and the other stable. These scenarios include transient and persistent performance changes, the introduction of congestion, topology changes, and nodes moving in an ad hoc network. All of the filters produce similar estimates over the long-term; they provide similar *accuracy*. They differ only in the time required to arrive at accurate estimates, and their resistance to noise in steady state. In other words, they differ in agility and stability. We find that the Flip-flop filter provides agility comparable to the agile EWMA filter in four of the five settings we examined. Likewise, it provided stability comparably to the stable EWMA filter in three out of four settings.

2 Related Work

There is a large body of work concerning the estimation of network performance. However, prior work differs from ours in two important ways. First, most systems focus on finding the physical bandwidth of the *bottleneck link* rather than the available bandwidth between two end hosts over time. The bottleneck link bandwidth is that available along the path in the *absence* of congestion traffic. While there are applications that can make use of bottleneck link information, adaptive mobile systems are concerned with the bandwidth that is actually available to them over time.

Second, most of these systems rely on *active* probing of the network, injecting measurement traffic in addition to or instead of passive observation of traffic already present. This is likely to be unacceptable during times of sharp decreases in network performance. To the best of our knowledge, our work is unique in fully examining the problem of estimating available network performance using only passive observations.

Several systems attempt to discover available network performance using static-gain EWMA filters. For example, the round-trip time estimator in TCP uses a stable EWMA filter [11]. Odyssey, a platform for application-aware adaptation, employs a network estimator using an agile filter [21]. Unfortunately, both of these static filters suffer from their biases. The RTT estimator in TCP cannot track varying performance quickly, resulting in retransmission timeouts (RTOs) that are too aggressive. To compensate, RTOs are increased by a factor that accounts for observed variance. Odyssey suffers from the opposite problem. Occasionally, it is fooled into tracking transient changes in bandwidth and adapting too aggressively as a result. Odyssey applications are responsible for filtering out transient changes.

Keshav's work on flow control [15] provides much of the groundwork in the area of active probing. He proposed the *packet pair* technique: the use of two closely spaced packets to elicit bottleneck bandwidth. Keshav discussed the use of Kalman filters for network estimation, but rejected them because too little is known about the network state space to provide an optimal application. Instead, he employed a fuzzy logic estimator based on the same heuristic used by our Error-based filter with an added mechanism to resist occasional transient spikes. This mechanism would not improve performance in the presence of congestion-induced noise, which is endemic to cross-traffic. Keshav's estimator was designed for *rate-allocation servers* that exhibit much less noise than FCFS routers. Unfortunately, the current networking infrastructure consists primarily of FCFS routers, a domain Keshav's work explicitly excludes.

Several variations on packet pair improve its ability to generate estimates of bottleneck link bandwidth. Paxson [22] presents receiver-based packet pair (RBPP), which takes observations at the receiver that incorporate timing information from the sender for more accurate measurements. Lai [17] developed a further refinement, called receiver-only packet pair (ROPP). It depends only on timing information taken at the receiver, but approaches the effectiveness of RBPP. Lai also incorporates a mechanism called *packet windows* that increases the agility of packet-pair schemes, but leaves them susceptible to noise and transients. Varying the size of packet windows can bias ROPP for agility or stability, but this size is chosen statically.

There have been several approaches to estimating network performance through active probing. For example, Bolot [2] uses pairs of UDP packets to explore network state, but requires substantial amounts of bandwidth to do so. Downey's application of `pathchar` [7] uses ICMP packets with varying time-to-live fields, but also suffers from heavy bandwidth consumption. Carter and Crovella present tools to measure bottleneck and available bandwidth [6]. These tools rely on bursts of ICMP packets, sent in several phases, requiring substantial overhead. They assume that network conditions do not change during this process, limiting the granularity of changes that they can detect.

Lai's subsequent work [18] develops a more sophisticated network model that, combined with a technique called *packet tailgating*, estimates bottleneck link bandwidth of each link in a path. It generates much less active network traffic and makes fewer assumptions about router behavior than prior schemes. He estimates link bandwidth by determining the minimum delay experienced at each link. We believe that our Flip-flop filter could be used with this model to determine available bandwidth instead.

Balakrishnan's congestion manager [1] allows multiple conversations between two hosts—including those of protocols that normally do not provide congestion control—to effectively share bandwidth. It is based on an underlying layer that discovers network characteristics using a traditional, static-gain EWMA filter. We believe that the Flip-flop filter can be used in this system to improve the agility of applications without unduly sacrificing stability or accuracy.

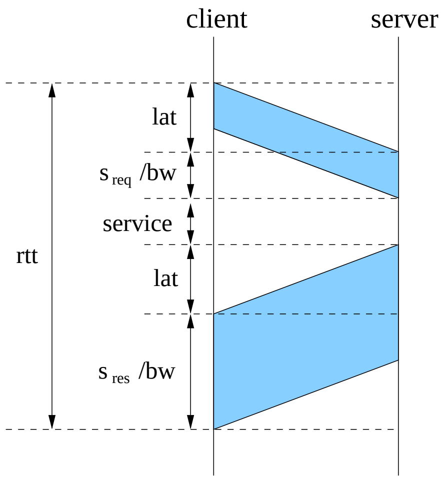
3 Making Observations

We derive estimates of network performance along a path by observing the behavior of packets on that path. These observations are made in light of a network model. This section describes the model underlying all of our estimators, and how we use it to obtain individual observations of network performance.

3.1 Measuring RTT

We represent the end-to-end path between two hosts with a simple fluid-flow model [19]. In this model, the sequence of hops from source to sink can be represented as a single service queue with latency lat and bandwidth bw. A packet's delay is simply: $delay = lat + size/bw$. The terms lat and bw are considered to be the *effective* latency and bandwidth along a path rather than those of some physical link. As traffic along the path increases, effective latency increases and effective bandwidth decreases.

An individual client observes network performance to a server by sending a request to that host, receiving a response, and measuring the total elapsed time. We assume that such observations happen only as a side effect of normal traffic exchanged between one machine and another. They are not made through active probing to avoid overhead during times of decreasing network performance. Requiring more traffic to detect such situations will only degrade conditions.



This figure illustrates the delays experienced by a request/response pair in our fluid-flow model.

Figure 1: Observing Network Performance

The total elapsed time consists of the time to transmit the request to the server, the time required to service the request, and the time required to transmit the response. This process is depicted in Figure 1. The round-trip time, RTT, with this model is:

$$RTT = (lat + \frac{s_{req}}{bw}) + service + (lat + \frac{s_{res}}{bw}) \quad (1)$$

where s_{req} and s_{res} are the sizes of request and response, respectively.

There are several things to note about this model. First, not all measured costs are imposed by the network. We assume that the service time is small relative to network costs. If this assumption does not hold, the server must report service time in its response packet, so that the client can account for it. Second, this scheme does not require synchronized clocks; round-trip times are only determined at the client, and service times are only measured at the server. This is convenient, since synchronizing clocks at fine grain requires GPS [25] or some other external time source. Unfortunately, in exchange for this convenience, we must assume that network performance is symmetric. In other words, we assume that it is the same from server to client as from client to server. While this is not always true, it is an unavoidable consequence of the lack of synchronized clocks.

3.2 Discounting Self-Interference

Our goal is to determine the effective latency and bandwidth available to this host. Therefore, estimates should be independent of its behavior, but dependent on the other traffic present along the path. When a host transmits two requests very close to one another in time, the second will be queued behind the first, inflating the second's RTT. We must discount such *self-interference* [22]. To do so, we use the bandwidth estimate to determine self-imposed queuing delay.

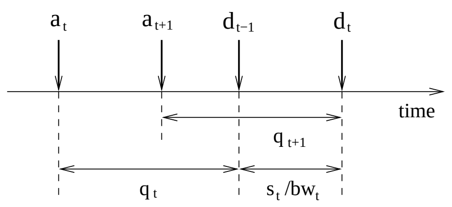


Figure 2: Self-Interference Queueing Delay

Figure 2 illustrates how we compute the self-interference queueing delay. When message m_{t+1} arrives at the modeled queue at a_{t+1}, it must wait until the previous message m_t departs at d_t. So the queueing delay, q_{t+1}, is defined as: $q_{t+1} = d_t - a_{t+1}$. Since we do not explicitly measure the departure time, we compute q_{t+1} using the following:

$$q_{t+1} = max\left\{q_t + \frac{s_t}{bw_t} - (a_{t+1} - a_t), 0\right\} \quad (2)$$

where s_t is the size of message m_t, and bw_t is the bandwidth estimate generated at a_t.

3.3 Computing Spot Values

After the self-interference queueing delay is discounted, the round trip time consists of the contributions made by effective latency and bandwidth. In order to separate these two, we must make two consecutive observations with different total sizes. The latency and bandwidth obtained using two observations are called *spot values*; they represent a single snapshot of network performance.

There are two subtleties that must be considered when computing spot values. First, we assume that the observations producing a spot value experience the same networking conditions. This is not always true. When conditions are sufficiently different, one of the spot values is negative. We ignore such observations. Second, if consecutive request-response pairs have identical total size, we cannot solve for latency and bandwidth. In such cases, we determine which parameter, latency or bandwidth, has been more stable over recent observations. We assume the stable parameter has not changed, and solve for the remaining one with the new observation.

4 Producing Estimates

Typically, systems use EWMA filters to smooth noisy network observations. Such filters take the form:

$$E_t = \alpha E_{t-1} + (1-\alpha)O_t \tag{3}$$

where E_t is the newly generated, smoothed estimate, E_{t-1} is the prior estimate, and O_t is the current observation. The term α is called the *gain*, and determines the filter's reactivity. If the gain is large, old estimates will dominate and the filter will be slow to change, making it stable. For example TCP's RTT filter has a gain of 7/8. In contrast, filters with low gain will tend to be agile. Odyssey's network filters use gains as low as 1/8 to detect changes quickly. In the remainder of this section, we describe our four adaptive filters.

4.1 Flip-Flop Filter (FF)

The first filter, called Flip-flop, consists of two EWMA filters. One is agile, with a gain of 0.1, and the other is stable, with a gain of 0.9. A controller selects between the two. The underlying principle of this controller is to employ the agile filter when possible, but fall back to the stable filter when observations are unusually noisy. It employs a *control chart* [20] to make this decision.

Control charts are commonly used to provide statistical process control in manufacturing applications. They plot the sample mean, $\overline{x}$, of a controlled quantity against the desired population mean, μ, over time. The plot includes two *control limits*: the upper control limit (UCL), and the lower control limit (LCL). Usually, the control limits are defined to be $\mu \pm 3\sigma_{\overline{x}}$, where $\sigma_{\overline{x}}$ is the sample standard deviation. When a sample exceeds the control limits, the process is judged to be out of control. This is called the *3-sigma rule* [26].

We apply this technique to filter selection, but must make allowances for our domain. First, we do not know the true latency and bandwidth at any point; this is needed to generate a population mean. Second, those quantities are expected to change over time. Control charts are primarily used only to detect such shifts in mean, but we also want to recalibrate our control to the new mean value. Finally, we do not know the population standard deviation in advance, but need it to establish the control limits.

To address these shortcomings, we periodically change the center line and limits of the control chart, using the *moving range* to approximate the standard deviation. The moving range, $\overline{MR}$, is the average of the differences between adjacent points, $|x_i - x_{i-1}|$. The control limits use the moving range as a substitute for the sample standard deviation. Since the true value of $\overline{MR}$ may change over time, we smooth it. The center line, which represents the population mean, is set to an exponentially-weighted moving average of the estimated value, $\overline{x}$, to account for mean shifts. The control limits are then:

$$\overline{x} \pm 3\frac{\overline{MR}}{d_2} \tag{4}$$

where d_2 estimates the standard deviation of a sample given its range. When the range is from a sample of two, as it is for $\overline{MR}$, the value of d_2 is approximately 1.128 [20]. In the process control literature, this type of control chart is called the *individual-x chart* [24].

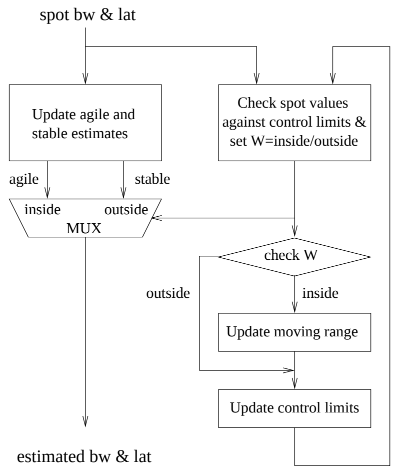
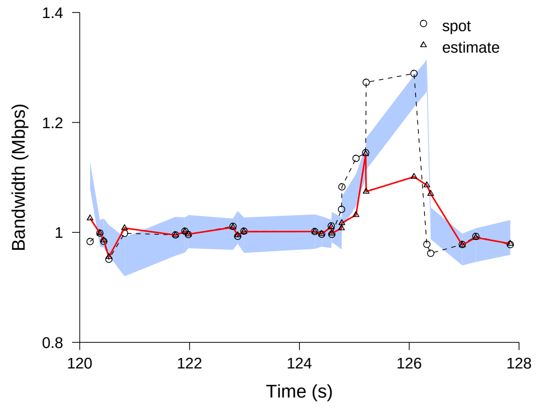


Figure 3: Flip-Flop Filter

Figure 3 shows how the Flip-flop filter chooses between its two EWMA filters and when it update its control values. As long as spot values fall within the 3-sigma limits, we use the agile filter. If the values fall outside the limits, we fall back to the stable one. In other words, when spot observations are unusually variable, the filter dampens its estimates.

The moving range is updated only when the spot values fall within the 3-sigma limits. This prevents the moving range from becoming too wide. The center line and control limits are adjusted for each packet received.



The shadowed area represents the values between the upper and lower control limits. Around 125 seconds, the Flip-flop filter switches from the agile filter to the stable one. Each filter is run independently of the other. Therefore, when switching to the stable EWMA filter, the global estimate drops without a corresponding low observation.

Figure 4: Flip-Flop Filter Example

Figure 4 shows an example of the Flip-flop filter. The shadowed area represents the values between the upper and the lower control limits, the dotted lines show the spot values, and the solid line plots the estimate over time. When the spot values are outside of the control limits, the controller selects the stable filter; at other times, it chooses the agile filter. For example, around 125 seconds, the Flip-flop filter switches from the agile filter to the stable one because the spot value is above the upper control limit, causing a drop in estimate. This drop is a result of maintaining the two static-gain filters independently; the stable filter is consistently more conservative in following changes.

4.2 Stability Filter (SF)

The second filter is called the Stability filter. Like the Flip-flop filter, the Stability filter dampens estimates in proportion to the variance of spot observations. However, rather than using variance to select between static-gain filters, we use a measure of the variance to dynamically change the gain of a single filter.

The goal of the Stability filter is to dampen estimates when the network exhibits unstable behavior. As consecutive observations diverge, the instability increases. This, in turn, increases the gain, making the filter stable. Our measure of instability is similar to the moving range used in the Flip-flop filter. To compute instability, U, we use a second EWMA filter:

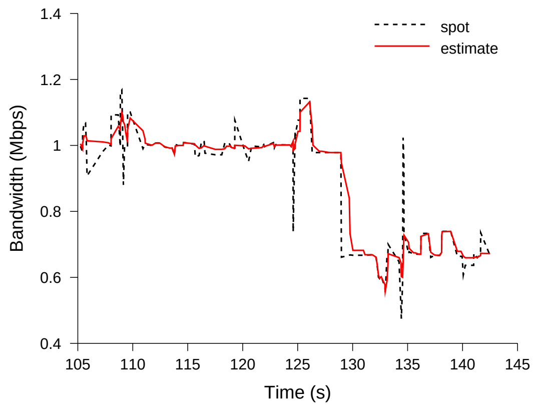


Figure 5: Stability Filter Example

$$U_t = \beta U_{t-1} + (1-\beta)|x_t - x_{t-1}| \qquad (5)$$

where x_t is the spot value measured at time t, and β is 0.6; this value was chosen empirically to minimize estimation error under the network scenarios presented in Section 5. We set the gain to be:

$$\alpha_t = \frac{U_t}{U_{max}} \qquad (6)$$

where U_{max} is the largest instability seen in the ten most recent observations.

An example of the Stability filter is shown in Figure 5. The dotted line shows the changing spot values, and the solid line tracks estimated values. The filter is relatively robust against large changes in performance, but tracks small changes well.

4.3 Error-Based Filter (EF)

The Error-based filter follows the general form of the Stability filter, but takes a different approach in adapting its gain. Rather than vary gain based on the variance in network observations, it bases gain on the predictive power of its estimates. When the Error-based filter produces estimates that match well with reality, these estimates are given more weight through higher gain. When the filter does not accurately match observed values, we decrease its gain so that it can converge more quickly.

The error at an individual observation is the difference between the past estimate and the current observation: $|E_{t-1} - O_t|$. Rather than use raw error values at each step, we filter these errors through a secondary EWMA filter; it then plays a role similar to that of U_t in the Stability filter. Estimator error, Δ_t, is:

$$\Delta_t = \gamma\Delta_{t-1} + (1-\gamma)|E_{t-1} - O_t| \qquad (7)$$

where γ is 0.6. This value was chosen empirically in the same manner as β. We then set the gain of the Error-based

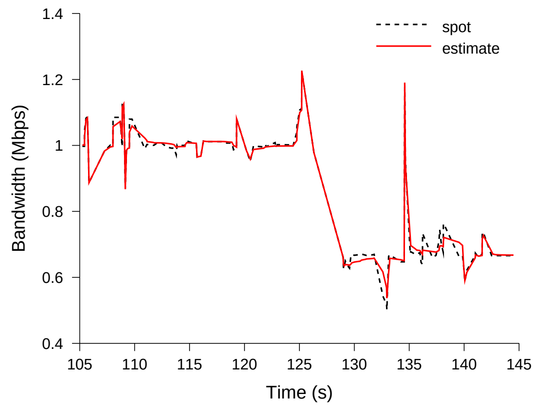


Figure 6: Error-Based Filter Example

filter to be:

$$\alpha_t = 1 - \frac{\Delta_t}{\Delta_{max}} \tag{8}$$

where Δ_{max} is computed the same way as U_{max}.

An example of the Error-based filter is shown in Figure 6. The dotted line shows the changing spot values, and the solid line tracks the estimated value. In contrast to the Stability filter, the Error-based filter tracks large changes quickly, but is robust against small fluctuations in performance.

4.4 Kalman Filter (KF)

The final filter we have explored is an application of the Kalman filter. Kalman filters, if properly applied to a linear system, are *optimal* in that they minimize mean squared estimation error. While an optimal Kalman filter requires significant knowledge of the system—knowledge that is not available when estimating network performance—one can employ reasonable guesses that give a good result.

Kalman filters describe a system in terms of *state space notation*. For our model, this is:

$$\mathbf{X}(t+1) = \Phi(t)\mathbf{X}(t) + \mathbf{W}(t) \tag{9}$$

where $\mathbf{X}$ is the system state vector, Φ is a constant matrix combining the state variables, and $\mathbf{W}$ is a matrix representing system noise.

Our filter estimates latency and bandwidth given round-trip time measurements; latency and bandwidth are the state variables. Recall that the round-trip time is proportional to $1/bw$. Therefore, in order to make the system linear, we use $1/bw$ rather than bw as the second state variable. So, the system state vector is written as:

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} lat \\ \frac{1}{bw} \end{bmatrix} \tag{10}$$

The state equations are then written as:

$$x_1(t+1) = x_1(t) + w_1(t) \tag{11}$$
$$x_2(t+1) = x_2(t) + w_2(t) \tag{12}$$

where w_1 and w_2 are the (unknown) system noise in latency and bandwidth, respectively. Putting these together, we have:

$$\Phi = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{13}$$
$$\mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \tag{14}$$

In order to apply a filter to the system state, one must measure it:

$$\mathbf{Z}(t) = \mathbf{H}(t)\mathbf{X}(t) + \mathbf{V}(t) \tag{15}$$

Here, $\mathbf{Z}$ is a matrix containing the measured state values, $\mathbf{H}$ is a constant matrix combining the system state, and $\mathbf{V}$ is a matrix representing the (unknown) measurement error.

Our measurement, RTT, is a scalar, so $\mathbf{Z}$ is the scalar z, and $\mathbf{V}$ is the scalar v. Our network model says that $RTT = 2lat + s/bw$, so the measurement equation is:

$$z(t) = 2x_1(t) + s(t)x_2(t) + v(t) \tag{16}$$

where $s(t)$ is the sum of the sizes of the request and response pair at time t. Hence, matrix H is:

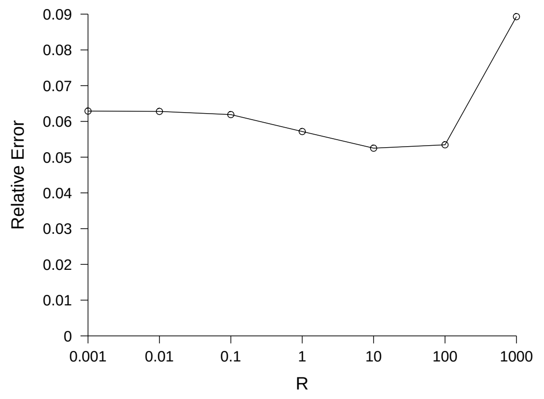
$$\mathbf{H} = \begin{bmatrix} 2 & s(t) \end{bmatrix} \tag{17}$$

To apply a Kalman filter, we must know the process noise covariance matrix $\mathbf{Q}$ and the measurement error covariance matrix $\mathbf{R}$. $\mathbf{Q}$ is the covariance matrix of $\mathbf{W}$; $\mathbf{R}$ is the covariance matrix of $\mathbf{V}$. Since $\mathbf{W}$ and $\mathbf{V}$ are not known, their covariances are unavailable. Therefore, we assume that the noise in latency and bandwidth is independent, making their products zero. This assumption may not be correct, so we plan to explore it in future work.

$$\mathbf{Q}(t) = \begin{bmatrix} {w_1}^2 & 0 \\ 0 & {w_2}^2 \end{bmatrix} \tag{18}$$
$$\mathbf{R}(t) = [v^2] \tag{19}$$

Intuitively, $\mathbf{Q}$ represents the degree of variability in latency and bandwidth. The terms ${w_1}^2$ and ${w_2}^2$ describes the degree to which one is more volatile than the other. $\mathbf{R}$ describes the measurement uncertainty. If measurements are uncertain, system state estimates should not change drastically with individual measurements. In other words, the filter becomes stable when $\mathbf{R}$ is large.

Unfortunately, we do not know the relative variances in latency and bandwidth, nor do we have an accurate picture of measurement noise. Although good guesses for $\mathbf{Q}$ and $\mathbf{R}$ will not provide optimal performance, they will make the filter perform reasonably well [15]. The real impact of $\mathbf{Q}$ and $\mathbf{R}$ on the filter's performance is determined by the relative magnitudes of each. For matrix $\mathbf{Q}$, we assume that the variances of noise in latency and bandwidth are the same, and set both ${w_1}^2$ and ${w_2}^2$ to 1.



This figure shows relative error of Kalman filter with different R values under the simple ad hoc network simulation described in Section 5.5.

Figure 7: Kalman Parameter

With $\mathbf{Q}$ fixed, we then empirically determine $\mathbf{R}$ using a simple ad hoc network simulation, which is also used to evaluate the performance of filters. There are two reasons that we chose this simulation to determine the best $\mathbf{R}$ value. First, the nominal bandwidth values are known, allowing us to compute the relative error, $|estimate - true|/true$. Second, the noise endemic to mobile networks tests the stability of the filter. In other words, the error values are not only affected by how fast each filter settles, but also how stable it is once settled. In contrast, if we tune the parameter in an environment that does not have much noise after each persistent change, the error is mainly determined by how fast the filter settles to a new value. In this case, a smaller $\mathbf{R}$, which makes the filter agile, would always be preferable.

Figure 7 shows the relative error of the Kalman filter with $\mathbf{R}$ values varied from 0.001 to 1000. The results show that $[10]$ is the best value for $\mathbf{R}$, and that this value is relatively insensitive to the performance of the Kalman filter. In other words, any value except 1000 does not make a large difference in performance. This holds true for all of our experiments. The resulting matrices are:

$$\mathbf{Q}(t) = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{20}$$

$$\mathbf{R}(t) = [10] \tag{21}$$

Given matrices $\mathbf{Q}$ and $\mathbf{R}$ with the state equations, applying the Kalman filter is straightforward [5].

5 Evaluation

In evaluating the candidate filters, we set out to answer the following questions:

- How agile are the filters in the face of idealized, persistent changes in network capacity?
- How stabile are the filters in reaction to transient changes in network capacity?
- How do filters react in the presence of changing network congestion?
- How do filters react in the presence of topology changes in a mobile network?
- How do filters compare in a complex, ad hoc network?

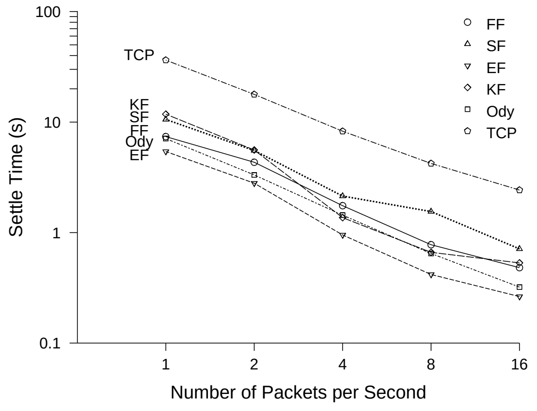
To answer these questions, we subjected each filter to a variety of networking scenarios. Many of these are based on simulations that vary latency, bandwidth, cross traffic, or node topology over time. For the simulations, we used ns [3], a packet level network simulator, with the Monarch extensions for mobile, ad hoc networking [4]. The Monarch extensions include near/far propagation models, packet capture, and the complete IEEE 802.11 MAC implementation [10]. The 802.11 MAC layer incorporates collision avoidance and link-level acknowledgement and retransmission. The nominal transmission range is 250 m. For the ad hoc network routing protocol, we used dynamic source routing [12].

We further modified ns to implement links whose performance can change according to a profile we provide. For each experiment, we generate a profile that specifies the topology, link characteristics, traffic, and how each of these change over time. This gives us two important benefits. First, since we know the objective state of the network, we can precisely quantify the behavior of our estimator. Second, these changes can be made arbitrarily taxing, stressing the adaptive estimators.

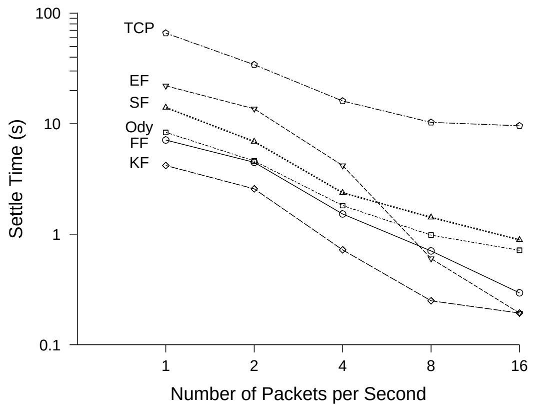
In the situations we examined, the long-term estimates for latency and bandwidth produced by all the filters are roughly the same. The filters differ only in how quickly they arrive at this estimate, and how well they hold to that estimate once there. In other words, they differ in agility and stability. We use *settle time* as a measure of agility. Settle time measures the elapsed time it takes a filter to produce an estimate within 10% of some new nominal value. Lower settle times are better. We use two different measures to describe stability, depending on the context. The first, *coefficient of variation* (CV), is used during times when network performance is nominally stable. It is the ratio of standard deviation to mean. It measures the degree to which measurement noise affects a filter's estimates; lower CV values are better. The second, *mean squared error*, is used to measure resistance to transients. We use this metric because it penalizes filters that are disturbed by large amounts for a short time more than those disturbed by small amounts for a longer time. An error that is large in magnitude is more likely to cause an adaptive system to make a poor decision.

5.1 Detecting Persistent Changes

In the first experiment, we subject each filter to two *step functions*: an immediate change from one bandwidth to another. A client and server are connected by a single link, with per-



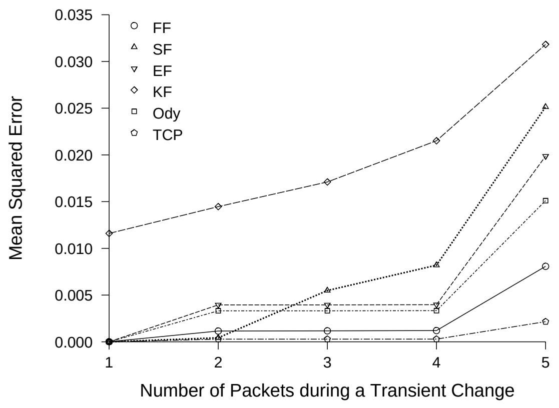
(a) Step-up



(b) Step-down

This figure depicts the agility of each filter under a sharp, ideal change in bandwidth. The X axis shows the average packets per second generated by the Poisson request process. The Y axis gives the settle time, in seconds, and is in log scale. In both cases, the Odyssey filter and the Flip-Flop filter perform well.

Figure 8: Agility: Settle Time under Varying Rates



This figure depicts each filter's stability in the face of a transient drop in bandwidth. The X axis shows the number of packets each filter observes during the performance drop; the Y axis gives the mean squared error. The TCP filter and the Flip-Flop filter are the least perturbed.

Figure 9: Stability: Resistance to Transient Change

formance that varies over time. We explored two different changes. In the first, called *step-up*, we increase the available link bandwidth from 1 Mb/s to 10 Mb/s. In the second, called *step-down*, the change is in the other direction. Our goal in this experiment is to test each filter's agility; how long it takes each filter to recognize a persistent change in bandwidth.

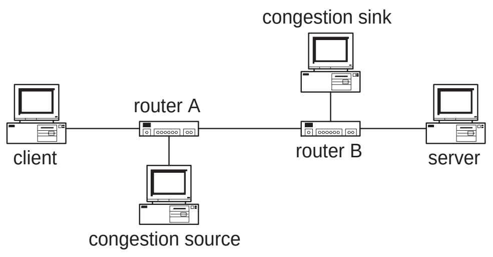
Since the filters rely only on passive measurements, their agility depends heavily on the rate of the underlying network traffic. To explore this, our traffic generator uses a Poisson process with means varying between one and sixteen packets per second; the fastest Poisson process will saturate the link when it is at low bandwidth. All the measurements are done at the UDP layer, so the sizes of requests and responses can be larger than the MTU, which is 1500 bytes for Ethernet. For our experiment, each request is small, while each response is randomly chosen to be either small (512 bytes) or large (8 KB) with equal probability; we used 8 KB responses to represent NFS traffic. Large responses will be fragmented by IP. Five trials of each experiment were taken with differing random seeds for the request generator.

The results for this experiment are shown in Figure 8. Figure 8(a) shows the filters' ability to detect increases in bandwidth; Figure 8(b) quantifies detection of decreases. As one would expect, each filter detects changes more quickly with more opportunities for observation. The static EWMA filters also behave as expected. The stable TCP filter is the slowest to converge, while the agile Odyssey filter is among the fastest. Only the Flip-flop filter is comparable to the Odyssey filter in both cases.

The relatively slow response of EWMA-based filters to detect decreases compared to increases is not surprising, and agrees with the evaluation of the Odyssey filter [21]. For example, the Odyssey filter settles within 11.25% of the goal (10 Mb/s) with one accurate observation for the increase in bandwidth, while it settles within 112.5% of the goal (1 Mb/s) for the decrease. The Kalman filter does not follow this, and detects decreases faster than increases for all packet numbers.

5.2 Resisting Transient Changes

The second experiment gauges the filters' resistance to short-term drops in performance. In this experiment, each filter is subjected to a short decrease in bandwidth from 10 Mb/s to 1 Mb/s. In order to fairly compare the filters, we must ensure that the same number of packets experience each transient change. So, unlike the agility experiments, we use a constant transmission rate, and vary the length of the transient



This figure illustrates the network topology for the congestion experiments. The client and server exchange requests and responses, through routers A and B. During the experiment, the congestion source begins a traffic stream to the congestion sink.

Figure 10: Topology for Congestion Experiments

from 1 to 5 packets. The sizes of these packets are chosen as before; requests are small, with responses randomly small or large. Since they are random, we perform five trials at each duration.

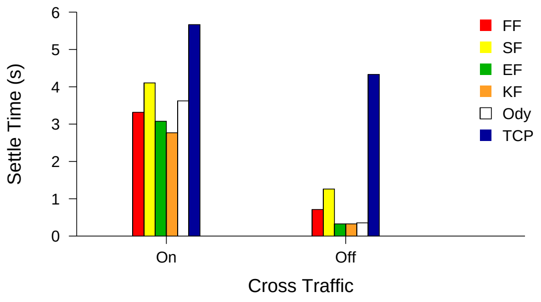
To evaluate stability of filters, we compute the mean squared error with a constant nominal value of 10 Mb/s. The results for bandwidth estimation are shown in Figure 9. As expected, the TCP filter is the most resistant to change. The Flip-flop filter is also very stable. The Kalman, Error-based and Odyssey filters are susceptible to changes, while the Stability filter follows longer transients more aggressively than shorter ones. As the duration of the transient increases, the Stability filter judges the transient value to be stable, and follows it.

5.3 Detecting Congestion

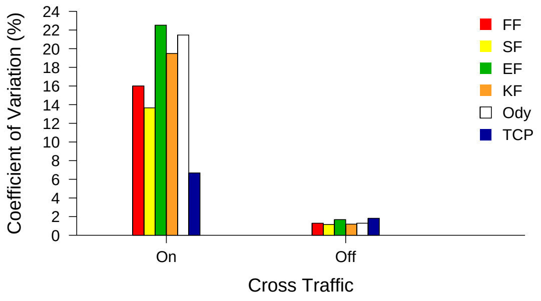
The previous two experiments explored changes in link capacity, as might happen during a vertical handoff. However, link congestion also determines the end-to-end bandwidth that is ultimately available to a client and server. This experiment characterizes each filter's ability to detect and estimate the effects of congestion.

The client and server are part of a six-node network, depicted in Figure 10; each link has a capacity of 10 Mb/s. They exchange request-response traffic with a Poisson process at an average rate of 140 Kb/s. The experiment lasts a total of 150 seconds. 50 seconds into the experiment, the congestion source sends a constant bit rate (CBR) stream of 5 Mb/s in 8 KB chunks to the congestion sink, reducing the available bandwidth along the client-server path. This congestion traffic lasts for 50 seconds, and then stops. The results reflect five trials with different random seeds.

During congestion, observed round-trip times are exceptionally noisy. This is because cross traffic may or may not interfere with any particular request-response pair. In other words, the observed traffic is only partially perturbed by congestion. If a router has cross-traffic queued when a request or response arrives, it will be delayed, but otherwise



(a) Agility: settle times



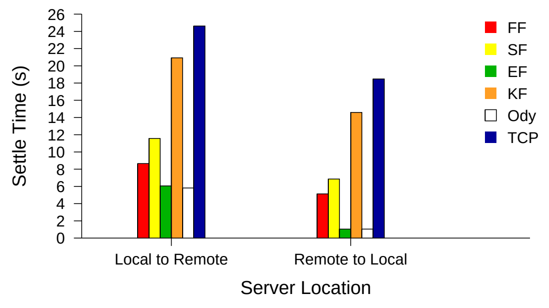
(b) Stability: coefficient of variation

This figure depicts the filters' reaction to congestion. There are two categories across each X axis; when congestion is introduced, and when it is removed. Figure 11(a) gives the settle time for each filter; the Y axis is given in seconds. Figure 11(b) shows the coefficient of variation exhibited by each filter after it has settled. The TCP filter is slow to detect any change; the Stability filter is slow to detect the absence of congestion. The TCP, Stability, and Flip-flop filters are the most stable.

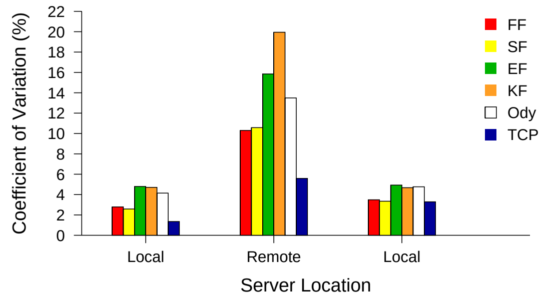
Figure 11: Detecting and Tolerating Congestion

it will not. While each filter produces the same long-term estimates, they are consistently optimistic, at just over 6 Mb/s. This optimism is a direct result of the fact that congestion traffic only partially perturbs the observed round trips.

Figure 11 shows the agility and stability of each filter. The first set of bars shows the result from 50 seconds to 100 seconds when the congestion traffic is present; the second set shows it from 100 seconds to 150 seconds. To compute the settle time, one needs the nominal bandwidth during each period. We determined this value by taking the average estimate produced by all filters over the last half of each 50 second interval. As explained above, this estimate will tend to be optimistic, but is the best one can do without measurement within the network. As shown in Figure 11(a), the TCP filter is clearly the least agile of any of the filters. The other five filters are very agile. As shown in Figure 11(b), all filters except the TCP are not stable when the congestion traffic is on. Among the five, the Stability and Flip-flop filters provide more stable estimates than the others.



(a) Agility: settle times



(b) Stability: coefficient of variation

This figure depicts the filters' reaction to routing changes in wide-area networks. The client first requests data from a co-located server, switches to a remote one, and then switches back. Figure 12(a) gives settle times for the two route changes; the X axis lists the route changes, and the Y axis gives settle time, in seconds. Figure 12(b) presents coefficients of variation in each distinct experiment phase after settling. The Kalman and TCP filters are particularly poor at detecting changes, while the Kalman, Error-based, and Odyssey filters are particularly susceptible to noise.

Figure 12: Performance over Wide-Area Networks

5.4 Wide-Area Networks

To gauge whether instability is endemic to these filters in the presence of cross traffic, we subjected them to two different, real-world networking scenarios. In them, a client and server exchanged ICMP ECHO packets, where each packet was either 512 bytes or 8 KB with equal probability. The requests were generated by a Poisson process with an average rate of one packet per second. We took measurements between one client and two servers. One server was in the same subnet, and the other was located twelve hops and roughly 100 ms away. We repeated this for five trials to each server; each trial was 300 seconds long. In each trial, the client starts communication with the local server, switches to the remote one at 100 seconds, and switches back to the local at 200 seconds. The raw observations from each trial were recorded, and each filter was subjected to precisely the same set of observations.

Since this environment is real, we do not know the nominal bandwidth. However, we need some measure of nominal to compute agility and stability for our filters. We used the the average bandwidth of the second half of each 100-second period as the nominal bandwidth.

Figure 12(a) shows the settle times for each filter. The Odyssey and Error-based filters are the most agile ones. Although the Flip-flop and Stability filters are not as agile as the best performers, they are much better than both the Kalman and TCP filters.

Figure 12(b) shows the stability results for each 100-second period separately. The stability for all filters is much better when observing traffic from the local server than from the remote one. This is most likely due to the smaller number of hops combined with the lower likelihood of encountering cross traffic in the local case. The Kalman filter performs particularly poorly in the communication with the remote server. As expected, the TCP filter is very stable, but the Flip-flop and Stability filters also perform reasonably well.

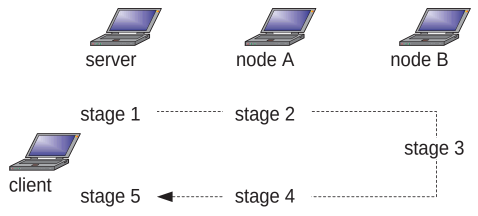


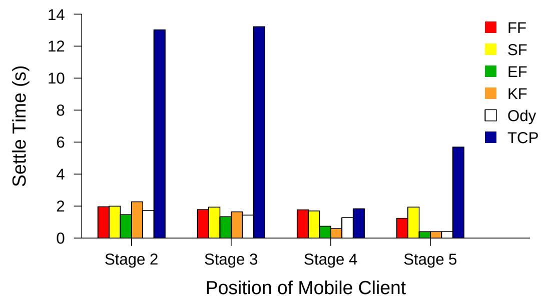
Figure 13: Topology Changes

5.5 Topology Changes in Mobile Network

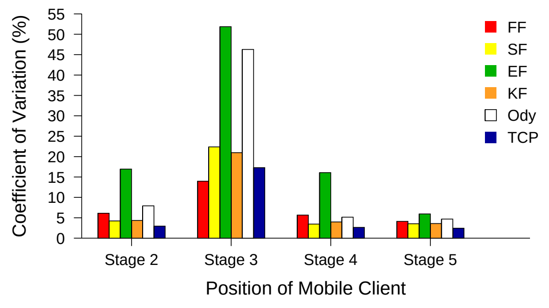
The next experiment explores how our filters react to changing topologies in simple ad hoc networks without the presence of cross traffic. The topology for this simple network is shown in Figure 13. Three wireless nodes—one of them a server—are arranged in a line, each separated by 200 m. The client moves from the server's neighborhood to the vicinity of the far node, and then back, comprising five different stages. The client completes this circuit in five minutes; we use handoff times as the breakpoints between stages. The traffic rate was Poisson, with an average of four requests per second; requests were small and responses were 512 bytes or 8 KB with equal probability. We took five trials for each filter.

There are interesting effects even in this simple topology. First, although there is no cross traffic, the effective bandwidth changes as the client moves through the stages. This is because all nodes share the same physical channel. While the bandwidth of the physical device is 2 Mb/s, the effective bandwidth between client and server is divided by two when routed through node A, and by three through node B.

Second, the wireless MAC protocol allows collisions even when the client and server communicate directly; the rate of collisions goes up with hop count. This leads to substantial variability in RTT observations, increased noise in all of



(a) Agility: settle times



(b) Stability: coefficient of variation

This figure shows the filters' performance when moving through a simple ad hoc network. Figure 14(a) shows the time each filter requires to determine that a handoff has occurred. The X axis lists each handoff to a new stage, and the Y axis gives the settle time. Figure 14(b) presents the stability during each stage after settling. The X axis is as before, and the Y axis gives the coefficient of variation. The TCP filter is the obvious poor performer in agility. The Flip-flop filter does not match the top performer, but is a contender. The Flip-flop filter provides the best stability, particularly in stage 3.

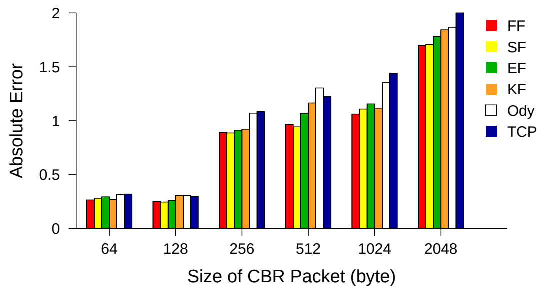
Figure 14: Performance under Changing Topology

the estimators, a higher average latency, and lower average bandwidth.

Figure 14(a) shows the agility of each stage. All of the filters except for TCP settle reasonably well, though the Error-based filter holds a slight advantage. Figure 14(b) shows the stability. All of the filters except the Error-based and Odyssey filters show comparable stability. The Flip-flop filter has an advantage in the worst situation: stage three, where collisions and link-level retransmissions are most frequent. This is because the infrequent retransmissions in the mobile experiment lie outside the Flip-flop control limits, causing the Flip-flop filter to choose the estimates generated by the stable EWMA filter.

5.6 Ad Hoc Wireless Networks

In the final experiment, each filter is given the task of predicting available capacity in an ad hoc network with substantial cross traffic. This is a demanding scenario, as the network topology—and hence route and cross traffic—are constantly changing. In such a network, past observed performance is at best a loose predictor of future results. While no filter can predict perfectly in such an environment, many applications require the best predictions they can get.



This figure shows the filters' ability to correctly predict round trip times in a large ad hoc network. The X axis lists the packet sizes of background traffic. The Y axis shows the average error produced by each filter's estimates, in seconds; lower errors are better. In general, increased background traffic leads to less accurate predictions. At relatively low levels of background traffic, each filter is comparable. As background traffic increases, the Flip-Flop and Stability filters begin to show advantages.

Figure 15: Accuracy in an Ad Hoc Network

We placed 50 nodes in a space 1500 meters by 500 meters. Initial node locations are distributed throughout the space uniformly randomly. Each node follows the random waypoint model [12] with a pause time of 20 seconds and a maximum speed of 20 meters per second. Each trial simulated 500 seconds.

The 50 nodes were formed into 25 pairs. One pair served as the estimating client and corresponding server, exchanging requests and responses with Poisson distribution, averaging four requests per second. The remaining pairs are CBR connections, each source transmitting four packets per second. Since the amount of traffic substantially affects noise in observations, we repeated the simulations with six different packet sizes. The size varied from 64 bytes to 2 KB. We ran five different trials for each packet size.

In this experiment, we cannot easily compute the nominal bandwidth. Thus, we cannot use the usual metrics. Instead, we computed average absolute error, the difference between the measured RTT and the estimated RTT based on the bandwidth and latency estimates. Figure 15 shows the results. For every packet size, the Flip-flop and Stability filters work the best. They show advantages more clearly as the background traffic increases.

Interestingly, neither the TCP filter nor the Odyssey filter performed well in the ad hoc network. This is because one was incapable of detecting changes quickly enough, while the other was overly sensitive to noise. Either problem is

	Filter	FF	SF	EF	KF	Ody	TCP
Agility	Step Up	◦	×	⊚	×	-	×
	Step Down	⊚	×	×	⊚	-	×
	Congestion	◦	◦	◦	⊚	-	×
	Wide-Area	×	×	◦	×	-	×
	Mobile	◦	◦	⊚	◦	-	×
Stability	Transient	◦	×	×	×	×	-
	Congestion	×	×	×	×	×	-
	Wide-Area	◦	◦	×	×	×	-
	Mobile	◦	◦	×	◦	×	-

This table summaries performance of six filters. We compared the performance each filter with the reference filter. As the reference filter, we used the Odyssey filter for agility, and used the TCP filter for stability. For each experiment, filters that worked better than the reference are marked ⊚, those that are comparable to the reference are given ◦, and those that performed poorly are marked ×.

Figure 16: Evaluation Summary

enough to disqualify a filter from consideration in this setting.

5.7 Summary

Figure 16 summaries the results of the experiments. Clearly, neither the Odyssey nor TCP filter is a good choice overall. However, the Odyssey filter is always very agile, and the TCP filter is always very stable. So, we use them as the references against which to compare. For agility, the filters that are comparable to Odyssey are marked ◦. Those that work better are marked ⊚, while those that perform substantially worse are marked ×. For stability, we compare against TCP, with the same conventions. The top half of the table shows the agility results, and the bottom half shows the stability results. We did not include the results from the complex ad hoc network experiment presented in Section 5.6 because neither static-gain filter worked well.

The Flip-flop filter is clearly the winner. It often produced agility comparable to the agile Odyssey filter and stability comparable to the stable TCP filter. In the few settings in which it did not compare with the reference filter, it often outperformed its adaptive peers. For the wide-area experiment, the Flip-flop filter was more agile than the Stability and Kalman filters, but not the Error-based one. For the congestion experiment, the Flip-flop and Stability filters were much more stable than the other two adaptive filters.

The Stability filter also worked well in most of the experiments. It provided stability comparable to Flip-flop, but was always slower than Flip-flop in detecting persistent changes. While the Error-based filter provided excellent agility, it was too unstable to be considered useful. The Kalman filter was also very agile, but has a tendency to follow transient changes in bandwidth. We suspect that the Kalman filter could be tuned to behave more reasonably, but do not have confidence that such tuning would apply to all possible circumstances.

6 Conclusion

Adaptation to changing network conditions is critical to the success of mobile systems. However, knowing how those conditions change is a difficult problem. Individual observations of network performance may have little bearing on available capacity. Historically, systems have filtered such observations with static-gain EWMA filters. Such filters are biased either toward agility or stability. Ideally, a network estimator would be agile when possible, but stable when necessary; it should adapt to the prevailing network conditions.

We have developed four candidate filters with this goal in mind. Each one examines the behavior of the network, and tunes itself in an attempt to provide accurate and timely estimates. Two of these filters are modifications of an EWMA filter that use heuristics to vary the filter's gain dynamically. A third is an application of the well-known Kalman filter. The fourth uses techniques from statistical process control to select between two static-gain EWMA filters, one agile and the other stable.

We evaluated these filters in a variety of contexts, including persistent and transient capacity changes, the presence of cross traffic, and changes in topology. We compared each adaptive filter to two static-gain EWMA filters, one agile and the other stable. We find that in the majority of scenarios, the filter based on statistical process control provides agility comparable to the agile Odyssey filter and stability comparable to the stable TCP filter. We believe that this filter presents the best choice for adaptive, mobile systems.

Acknowledgements

Pramod Khargonekar helped us construct a Kalman filter for the network estimation problem. Mark Corner, Landon Cox, Songkuk Kim, and Kevin Lai read earlier drafts of this document, and suggested numerous improvements to the presentation.

This research was supported in part by Novell, Inc., and the National Science Foundation under grant CCR-9984078. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Novell, Inc., the National Science Foundation, or the U.S. Government.

References

[1] H. Balakrishnan, H. S. Rahul, and S. Seshan. An integrated congestion management architecture for Internet hosts. In *Proceedings of ACM SIGCOMM '99*, pages 175–87, Cambridge, MA, USA, August 1999.

[2] J.-C. Bolot. Characterizing end-to-end packet delay and loss behavior in the Internet. *Journal of High Speed Networks*, 2(3):305–23, 1993.

[3] L. Breslau, D. Estrin, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[4] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of MobiCom'98*, pages 85–97, Dallas, TX, USA, October 1998.

[5] R. B. Brown and P. Y. C. Hwang. *Introduction ot Random Signals and Applied Kalman Filtering*. John Wiley & Sons, Inc., 1997.

[6] R. L. Carter and M. E. Crovella. Server selection using dynamic path characterization in wide-area networks. In *Proceedings of INFOCOM '97*, pages 1014–21, Kobe, Japan, April 1997.

[7] A. Downey. Using pathchar to estimate internet link characteristics. In *Proceedings of ACM SIGCOMM '99*, pages 241–250, August 1999.

[8] D. Duchamp. Issues in wireless mobile computing. In *Proceedings of the Third Workshop on Workstation Operating Systems*, pages 2–10, Key Biscayne, FL, USA, April 1992.

[9] A. Gelb. *Applied Optimal Estimation*. M.I.T. Press, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1974.

[10] IEEE. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE Std 802.11-1999.

[11] V. Jacobson. Congestion avoidance and control. In *Proceedings of ACM SIGCOMM '88*, pages 314–329, August 1988.

[12] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In *Mobile Computing*, T. Imielinski and H. Korth, editors. Chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[13] R. H. Katz. Adaptation and mobility in wireless information systems. *IEEE Personal Communications*, 1(1):6–17, 1994.

[14] R. H. Katz and E. A. Brewer. The case for wireless overlay networks. In *Proceedings 1996 SPIE Conference on Multimedia and Networking*, pages 77–88, San Jose, CA, January 1996.

[15] S. Keshav. A control-theoretic approach to flow control. In *Proceedings of ACM SIGCOMM '91*, pages 3–15, September 1991.

[16] C. Labovitz, G. R. Malan, and F. Jahanian. Internet routing instability. *IEEE/ACM Transactions on Networking*, 6(5):515–28, October 1998.

[17] K. Lai and M. Baker. Measuring bandwidth. In *Proceedings of INFOCOM '99*, pages 235–45, New York, NY, USA, March 1999.

[18] K. Lai and M. Baker. Measuring link bandwidths using a deterministic model of packet delay. In *Proceedings of ACM SIGCOMM 2000*, Stockholm, Sweden, August 2000.

[19] S. Low. *Traffic management of ATM networks: service provisioning, routing, and traffic shaping*. PhD thesis, University of California, Berkeley, 1992.

[20] D. C. Montgomery. *Introduction to statistical quality control*. John Wiley & Sons, Inc., 3rd edition, 1997.

[21] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker. Agile application-aware adaptation for mobility. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pages 276–87, Saint-Malo, France, October 1997.

[22] V. Paxson. End-to-end internet packet dynamics. In *Proceedings of ACM SIGCOMM '97*, pages 139–52, Cannes, France, September 1997.

[23] T. S. Rappaport. *Wireless Communications:Principles and Practice*. Upper Saddle River, New Jersey:Prentice Hall, 1996.

[24] S. E. Rigdon, E. N. Cruthis, and C. W. Champ. Design strategies for individuals and moving range control charts. *Journal of Quality Technology*, 26(4):274–87, October 1994.

[25] U. Schmid and W. A. Halang. Synchronized UTC for distributed real-time systems. In *Proceedings of the IFAC Workshop on Real Time Programming*, pages 101–107, Pergamon, Oxford, UK, June 1994.

[26] Western Electric. *Statistical Quality Control Handbook*. Western Electric Corporation, Indianapolis, Inc., 1956.