SANE: Stable Agile Network Estimation

Minkyong Kim and Brian Noble
Department of Electrical Engineering and Computer Science
The University of Michigan Ann Arbor, MI 48109

Abstract

Distributed systems are becoming increasingly dependent on *network estimation*, the ability to determine performance along one or more network paths. Producing quality estimates is challenging because network observations are noisy; this is particularly true in wide-area or mobile settings. Current systems depend on simple exponentially weighted moving average filters. These filters are either able to detect true changes quickly or to mask transient changes, but cannot do both. In this paper, we present four filters designed to react quickly to persistent changes while tolerating transient ones. These filters are evaluated in a variety of networking scenarios through three metrics: agility, stability and accuracy. While no single filter dominates, one based on techniques from *statistical process control* shows promise relative to the others.

1 Introduction

Many systems have come to depend on estimates of network latency and bandwidth. Examples include adaptive multimedia systems, mobile applications, load-balancing, prefetching, and distributed query processing. These all require high-quality estimates of network performance in order to provide the best possible service.

Unfortunately, providing such estimates is difficult at best. Network observations are noisy, particularly over wide-area [1] or mobile [2] paths. Such networks also suffer from persistent changes in performance due to vertical handoff [3], wireless fading [4], or routing changes [5]. Good network estimators should ignore transient noise conditions, but react quickly to persistent changes in performance. We call the former property *stability*, and the latter *agility*.

Typically, network estimators — in the form of exponentially weighted moving average (EWMA) filters — provide either of these properties, but not both. This is because they are constructed with static *gain*: the parameter that determines how aggressively an EWMA filter will track changing observations. This gain biases the estimator either towards past history — stability — or current observations — agility.

In this paper, we present the design and evaluation of four filters that strive to be agile when possible and stable when necessary. We first outline a simple model for observing network behavior in Section 2. This model depends only on passive observations made at the end hosts, and requires neither infrastructural support nor the active insertion of traffic to test network quality.

Section 3 presents the candidate filters that yield estimates of end-to-end latency and available bandwidth. Three of them are based on EWMA filters, but use heuristics to vary the gain, selecting for agility or stability as circumstances warrant. The fourth is an application of the commonly-used Kalman filter [6].

These filters are evaluated in Section 4. This evaluation subjects the filters to various networking scenarios, and collects three metrics for each. These measure the agility, stability, and accuracy of the filters. While no single filter is superior on all counts, we find that the *flip-flop filter*, which uses techniques from statistical process control [7], has some attractive properties and no serious disadvantages.

2 Making Observations

A client estimates the latency and bandwidth between it and a remote server by observing exchanges between the two. The client makes these observations only with the cooperation of the server; the intermediate infrastructure need not be modified. Furthermore, we assume that these observations are entirely passive; the client injects no traffic specifically for observing the network. This is particularly important in mobile networks, which are subject to sudden decreases in quality. Using probe traffic to detect decreases in performance only makes a bad situation worse. This section describes how observations are made, and how one can use them to infer instantaneous measurements of latency and bandwidth.

Each observation comprises a single request-response pair between a client and a server. This exchange is illustrated in Fig. 1. The client measures the elapsed time between issuing the request and receiving a response. The server returns, in the response, the time spent between receiving the request and issuing the response: the *service time*. This allows clients to consider networking costs separately from server-imposed delays. If the server does not report service time, the client must assume it is negligible.

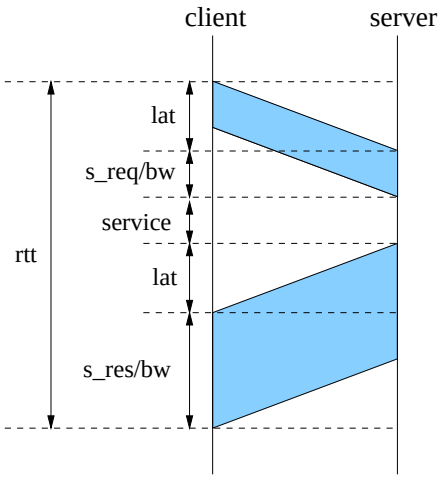


Figure 1: **Round Trip Time.**

Using a simple, single-queue model, the networking costs can be further decomposed into fixed, per packet costs and variable costs that depend on the size of the packet. The former is the latency, lat, imposed on a packet, and the latter is determined by the available bandwidth, bw along the path[1]. Thus, the round trip time is:

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

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

In order to separate latency and bandwidth, we must observe two separate request-response pairs between the client and server. This produces a single set of observations, called the *spot values*.

Networking costs of an individual packet vary based on the underlying characteristics of the path as well as the queuing delay it experiences. Such queueing delay can come from two sources. The first is cross traffic along the networking path taken by the packet. The second source stems from the fact that the packet may queue up behind others that are part of the same conversation. Since our goal is to measure the overall performance that can be realized by a conversation, we want to explicitly account for the second source of queueing delay, called *self-interference* [1].

Fig. 2 illustrates the queueing delay caused by self interference. Message m_t arrives at the modeled queue at time a_t, and departs the bottleneck queue at d_t. The quantity q_t is the self-interference queueing delay that m_t experiences. bw_{t-1} is the bandwidth estimation for the time between $t-2$ and $t-1$, generated at $t-2$.

[1]This is an oversimplification, since bandwidth constraints due to congestion are felt only at packet granularity. We make this simplification to avoid MTU discovery and the attendant complications.

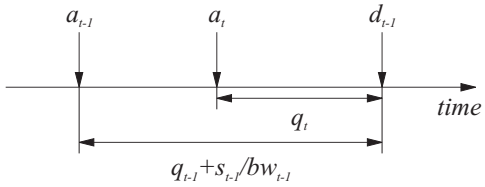


Figure 2: **Self-Interference Queueing Delay.**

This delay, q_t, can be written in recursive form:

$$q_t = \max \begin{pmatrix} q_{t-1} + \frac{s_{t-1}}{bw_{t-1}} - (a_t - a_{t-1}), \\ 0 \end{pmatrix} \qquad (2)$$

where size, s, is sum of s_{req} and s_{res}.

There are two situations where a pair of observations cannot be used to generate spot values. The first is when the total sizes of the two request-response packets are the same. In this case, we hold the new spot value of either latency or bandwidth to be the same as the previous spot value; we choose the one that has been more stable recently. The second case is when the two request-response pairs are subject to substantially different networking conditions. In this case, one of the computed spot values often is negative. We ignore such observations. In the event of a less severe difference, the spot values will be incorrect, and the filter must compensate.

3 Filtering Observations

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

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

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; such filters are biased towards stability. TCP's round trip time estimator is an example of such a filter with a gain of 7/8 [8]. In contrast, filters with low gain will tend to be agile. For example, the network estimator in Odyssey [9] uses filters with gains as low as 1/8 to detect changes as quickly as possible.

Unfortunately, both of these static filters suffer from their biases. The RTT estimator in TCP cannot track widely varying performance adequately, 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. It occasionally is fooled into tracking transient changes in bandwidth and adapting too aggressively as a result. It is left to applications to introduce hysteresis to filter transient changes.

In the remainder of this section, we present four filters designed to ignore transient changes while quickly following persistent ones. The first three are variants of the EWMA filter that use heuristics to vary the filter's gain based on prevailing conditions. The fourth is an application of the commonly-used Kalman filter.

3.1 Flip-Flop Filter

The first filter uses a controller to select between two EWMA filters, one agile and the other stable. 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* [7] 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; this is just the population standard deviation over the square root of the sample size. When a sample exceeds the control limits, the process is judged to be out of control. This is called the *3-sigma rule* [10].

We apply this idea to filter selection, but must make some 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 and use the *moving range* to approximate the standard deviation. The center line is set to a moving average of the estimated value, $\overline{x}$, to account for mean shifts. The control limits use the moving range as a substitute for the sample standard deviation. The moving range, or $\overline{MR}$, is the average of the difference between adjacent points, $|x_i - x_{i-1}|$. However, like the mean, the true value of $\overline{MR}$ may also change over time. Therefore, we keep a moving average of it as well. 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 MR, the value of d_2 is approximately 1.128 [7]. In the process control literature, this type of control chart is called the *individual-x chart* [11].

As long as spot estimates fall within the 3-sigma limits, we use an agile filter with gain 0.1. If the estimates fall outside the limits, we adjust the center line and limits on demand, and fall back to the stable filter with gain of 0.9. In other words, when spot observations are unusually variable the filter dampens its estimates.

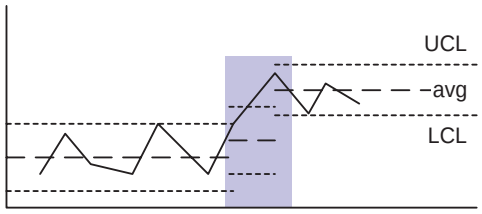


Figure 3: **Flip-Flop Filter Example**

The behavior of the flip-flop filter is illustrated in Fig. 3. The dashed lines show the value of $\overline{x}$ over time, the dotted lines show the control limits, and the solid line plots the estimate over time. During the shaded region, the controller selects the stable filter; at other times, the agile filter is used.

3.2 Stability-Based Filter

Like the flip-flop filter, the *stability-based 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.

The goal of the stability filter is to dampen estimates when the network exhibits unstable behavior. As instability increases, so does gain. Our measure of this instability, U, is similar to the moving range, $\overline{MR}$ used in the flip flop filter. Rather than a simple moving average, we use a second EWMA filter to compute instability:

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

where β is 0.6; this value was chosen empirically to minimize estimation error under varying network performance.

We then set the gain to be:

$$\alpha_t = \frac{U_t}{U_{max}} \tag{6}$$

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

An example of the stability filter is shown in Fig. 4. The dotted line shows the changing spot values, and the solid line tracks estimated values. Notice that the filter is relatively robust against large changes in performance, but tracks small changes well.

3.3 Error-Based Filter

The *error-based filter* takes a different approach from the first two. Rather than vary gain based on the variance in

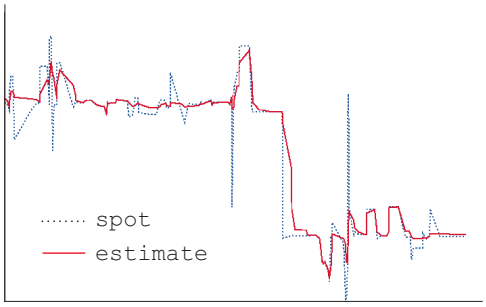


Figure 4: **Stability Filter Example**

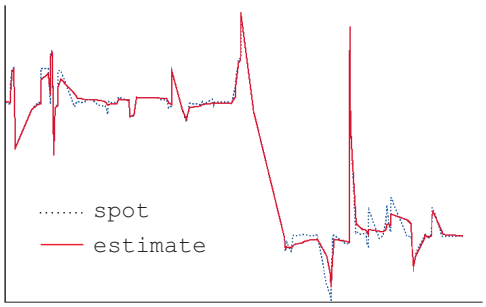


Figure 5: **Error Filter Example**

network observations, this filter bases gain on the quality 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 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:

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

where γ is 0.6. This value was chosen empirically in the same manner as β. We then set the gain of the error 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 filter is shown in Fig. 5. The dotted line shows the changing spot values, and the solid line tracks the estimated value. In contrast to the stability filter, the error filter tracks large changes quickly, but is robust against small fluctuations in performance.

3.4 Kalman Filter

The fourth and final filter we have explored is an application of the *Kalman filter* [6]. 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. In order to apply a filter to the system state, one must measure it:

$$\mathbf{Z}(t) = \mathbf{H}(t)\mathbf{X}(t) + \mathbf{V}(t) \tag{10}$$

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 measurement error.

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:

$$\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} lat \\ \frac{1}{bw} \end{bmatrix} \tag{11}$$

The state equations are then written as:

$$x_1(t+1) = x_1(t) + w_1(t) \tag{12}$$
$$x_2(t+1) = x_2(t) + w_2(t) \tag{13}$$

where w_1 and w_2 are the system noise in latency and bandwidth, respectively. The 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:

$$z(t) = 2x_1(t) + s(t)x_2(t) + v(t) \tag{14}$$

where $s(t)$ is the sum of the sizes of the request and response pair at time t.

Putting these together, we have the following matrices:

$$\Phi = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{15}$$
$$\mathbf{W} = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} \tag{16}$$
$$\mathbf{H} = \begin{bmatrix} 2 & s(t) \end{bmatrix} \tag{17}$$

To apply a Kalman filter, the covariance matrices for $\mathbf{W}$ and $\mathbf{V}$ must be known. This is not generally available, so we

assume that the noise in latency and bandwidth is independent, making their products zero:

$$\mathbf{Q}(t) = \begin{bmatrix} {w_1}^2 & 0 \\ 0 & {w_2}^2 \end{bmatrix} \quad (18)$$

$$\mathbf{R}(t) = [v^2] \quad (19)$$

Here, $\mathbf{Q}$ and $\mathbf{R}$ are the covariance matrices for $\mathbf{W}$ and $\mathbf{V}$, respectively.

Intuitively, $\mathbf{Q}$ represents the system noise, the degree of variability in latency and bandwidth. The terms w_1 and w_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.

The real impact of $\mathbf{Q}$ and $\mathbf{R}$ on the filter's performance is determined by the relative magnitudes of each. Unfortunately, we do not know the relative variances in latency and bandwidth, nor do we have an accurate picture of measurement noise. Instead, we assume that the variances are the same, and arbitrarily set them to 1. With $\mathbf{Q}$ fixed, we then empirically determine $\mathbf{R}$ to be $[10]$, much as for β and γ in the stability-based and error-based filters. The resulting matrices are:

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

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

4 Evaluating Filters

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 latency and bandwidth? How short can those changes be and still be detectable?
- How stable are they in the face of idealized, transient changes in latency and bandwidth? How long can those changes be and still be tolerated?
- How do they behave in the face of more realistic networking conditions, including the presence of cross traffic and mobile nodes?

To answer these questions, we subjected our filters to several synthetic networks that varied latency, bandwidth, cross traffic, or node topology over time. In order to generate such idealized networking conditions, we used ns [12], a packet level network simulator, with extensions for mobile, ad hoc networking [13]. We further modified ns to implement links whose performance can change over time 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.

We apply three metrics to the filters in each setting: agility, stability, and accuracy. We measure agility with the settle time. Settle time is the time it takes a filter to generate the first estimate within 10% of the nominal value after that value changes. Lower settle times are better.

For stability, we report one of two metrics: coefficient of variation (CV), or mean squared error (MSE). They are applied in different circumstances. CV — the ratio of standard deviation to mean — is used during periods when the network is assumed to be stable. Typically, such periods start when a filter settles after a persistent change and end when the next change occurs. Lower CV values are better.

The MSE metric is used to to quantify the degree to which a filter follows a transient. Filters that follow the transient's full magnitude for a short duration are penalized more by MSE than those that do not follow the full magnitude, but are disturbed for a longer time. Lower MSE values are better.

To measure accuracy, we compare each filter's average estimate to the true value. As with stability, we measure accuracy only between the time a filter settles and the next persistent change. We include the accuracy metric to identify estimators that quickly converge and stabilize to an incorrect value.

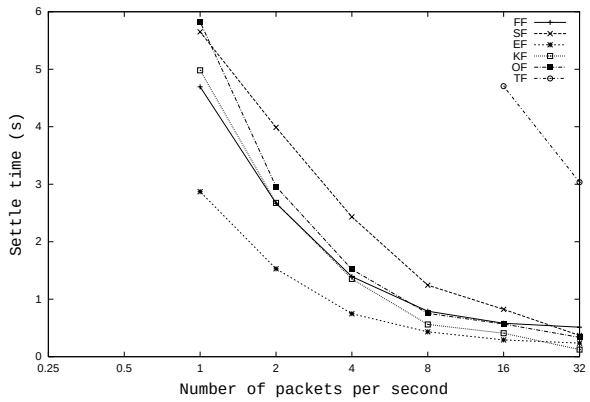
We compare our four filters — flip-fop (FF), stability-based (SF), error-based (EF), and Kalman (KF) — to two static-gain EWMA filters. The first is the agile filter used by Odyssey (OF) with a gain of 1/8. The second is the stable filter used in TCP to estimate RTT (TF) with a gain of 7/8.

4.1 Periodic Variations

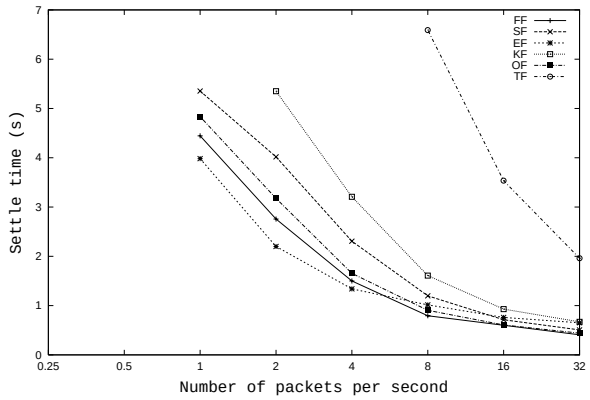
In the first experiment, we subject each filter to two different *square waves*, periodic, ideal variations. In the first, a client and server are connected by a network that provides a connection with constant latency, but bandwidth that instantaneously changes between 1 Mb/s and 10 Mb/s every 10 seconds. The second square wave holds bandwidth constant, and varies latency between 20 and 200 milliseconds every 10 seconds. Our goal in this experiment is to test each filter's *agility*; how long does it take each filter to recognize a change in bandwidth or latency?

Since the filters rely only on passive measurements, their agility depends heavily on the arrival rate of the underlying network traffic. To explore this, our traffic generator uses a Poisson process with means varying between 2.5 and 320 packets per 10 second period. Note that varying the mean of the arrival process for a given square wave is the dual of varying the period of the square wave for a given arrival process. Each request packet is small, while each response packet is randomly chosen to be either small (512 bytes) or large (8KB) with equal probability; with this distribution, the

fastest Poisson process will saturate the link when it is at low bandwidth. Five trials of each experiment were taken with differing random seeds.



(a) bandwidth



(b) latency

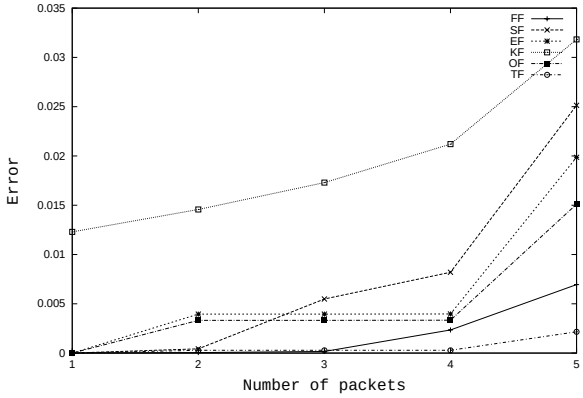
Figure 6: **Agility: Settle Time under Varying Rates**

Fig. 6 show the results averaged over the five trials for each filter at each arrival rate. Fig. 6(a) shows results for changing bandwidth, and Fig. 6(b) depicts the performance over the latency square wave.

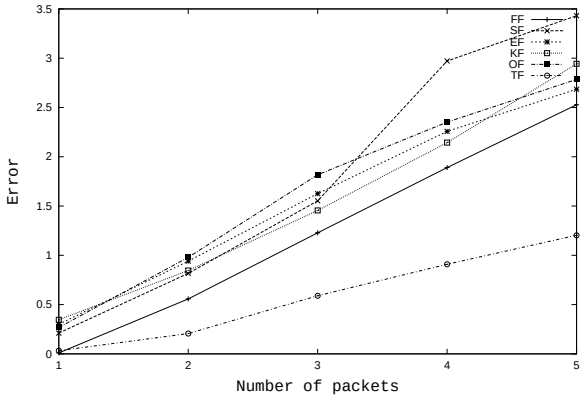
As expected, the stable filter used by TCP is the least agile of all. It cannot reliably detect latency changes below 8 packets per second, or bandwidth changes below 16 pps. Surprisingly, the Odyssey filter is not the most agile. This is because the other filters are able to select lower gains when they are biased towards agility. The error-based filter is the most agile with respect to bandwidth changes, followed by the flip-flop, Kalman, and Odyssey filters as a group; the stability filter is far off. For latency, the results are similar.

4.2 Transient Changes

The second experiment gauges the filters' resistance to short-term drops in performance. As with the agility experiment, we subject each filter to two different networks, one where the bandwidth drops from 10 Mb/s to 1 Mb/s for a short time, and the other where latency increases from 20 ms to 200 ms. 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 from 1 to 5 packets. The sizes of these packets are chosen as before; since they are random, we perform five trials at each length.



(a) bandwidth



(b) latency

Figure 7: **Stability: Resistance to Transients**

Fig. 7 reports the mean squared error for the filters at each transient duration; Fig. 7(a) depicts the results for bandwidth drops, and Fig. 7(b) shows results for latency changes. As expected, the stable TCP filter is the most resistant to change. The flip-flop filter, while more susceptible, is the second-best performer for both latency and bandwidth. The Kalman filter is eager to follow bandwidth changes, while the stability filter is the most sensitive to latency changes. The stability filter also follows longer transients more aggressively than shorter ones; as the transient's duration increases, the stability filter judges the new value to be stable, decreasing gain.

4.3 Congestion

The third experiment gauges the filters' ability to react to changing congestion along a network path. The network

for this experiment contains six nodes, and is illustrated in Fig. 8. In this topology, the client and server exchange packets with two different Poisson processes: one at 140 Kb/s and the other at 1 Mb/s. Each trial lasts 100 seconds. During this time, the congestion source periodically sends a constant bit rate stream of 5 Mb/s to the congestion sink. There are two periods of congestion, each 20 seconds long: the first starts at 20 seconds and the second starts at 60 seconds. During periods of congestion, the estimated bandwidth should be 5 Mb/s; otherwise, it should be 10 Mb/s.

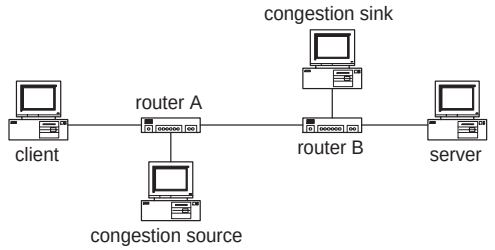


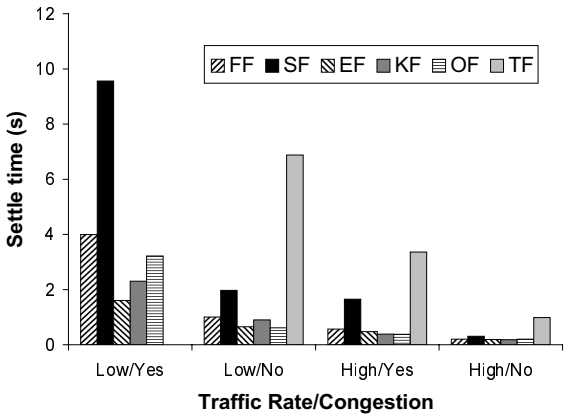
Figure 8: **Topology for Congestion Experiments**

Fig. 9 presents the settle time, average estimate, and coefficient of variation for the filters in each situation; the results reflect five trials with different random seeds. The situations are distinguished by whether the client-server traffic has a high or low rate, and whether or not there is congestion. For example, the result for *Low/Yes* are for experiments with 140 Kb/s client-server traffic in the presence of the 5 Mb/s congestion stream. We exclude results from the first period to avoid startup transients.

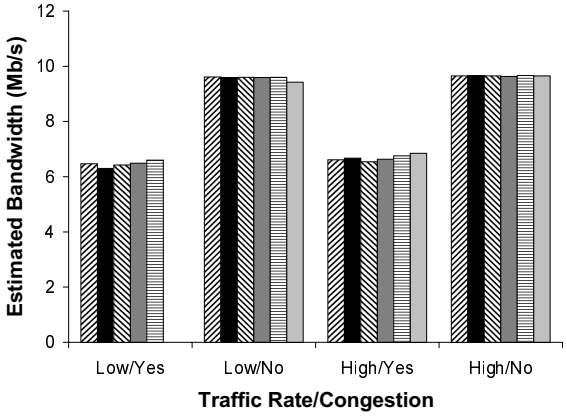
Fig. 9(a) shows the average time each filter needs to settle within 10% of the expected value. The TCP filter is clearly the least agile of any of the filters. The TCP filter is never able to converge when there is congestion interfering with low-rate traffic, and often did not settle in other situations with congestion. Since all of our measurements use the settle time in some way, we report no results for the TCP filter under low traffic with congestion.

The stability filter performs poorly compared to the others in all but the most favorable conditions: high client-server traffic with no congestion. In the worst set of circumstances — low client-server traffic with congestion — the error-based and Kalman filters hold an advantage over the flip-flop and Odyssey filters. In all other cases the four are roughly equal.

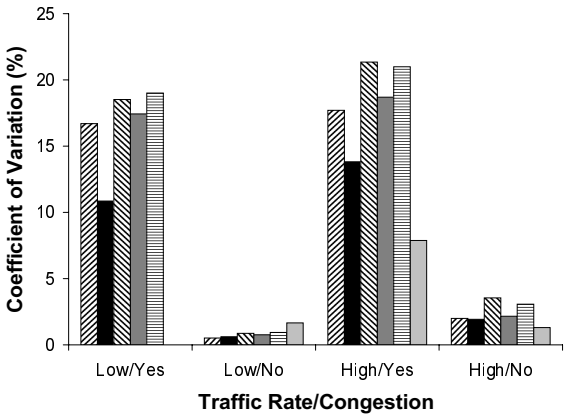
Fig. 9(b) depicts the average estimate produced by each filter between the settle time and the next transition. All of the filters perform similarly; however, none of them are very accurate in detecting bandwidth during congestion; average estimates range between 6.3 and 6.8 Mb/s. To see why, consider how the client-server traffic interacts with the congestion traffic. In order to detect the influence congestion has on observed traffic, observed packets must be delayed by congestion packets. Unfortunately, whether or not this occurs is



(a) Agility: settle times



(b) Accuracy: average estimate



(c) Stability: coefficient of variation

Figure 9: **Filter Performance under Changing Congestion**

at the discretion of the router. In particular, in a router with FCFS forwarding, congestion will not perturb observed traffic if the observed traffic arrives closely spaced without intervening congestion packets. So, the client-server traffic is able to observe the congestion traffic only some of the time.

The fact that observed traffic is not always perturbed by congestion also has unfortunate implications for filter stabil-

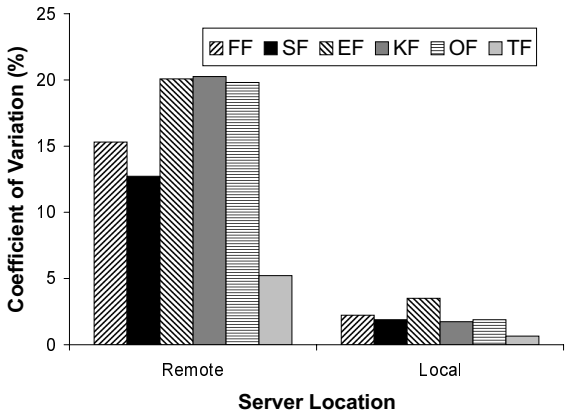


Figure 10: **Stability over Real Networks**

ity, as shown in Fig. 9(c). While the TCP filter shows modest variation in situations when it is able to settle, the other filters are all noisy. The stability filter offers the next best performance, followed by the flip-fop filter. The error-based filter exhibits the worst stability of the adaptive filters. Note that the TCP filter yields the highest coefficient of variation under low client-server traffic with no congestion. This is an artifact of long settle times; the filter does not have enough time to converge, and its gentle rise is mistaken for instability.

4.4 Stability over Real Networks

To gauge whether low stability is endemic to these filters in the presence of other traffic, we subjected them to two different, real-world networking scenarios. In them, a dummy client and server exchanged ICMP ECHO request and response packets, where the requests and responses were each either 512 bytes or 8 KB with equal probability. These 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 200 seconds long. The raw RTT observations from each trial were recorded, and each filter was subjected to precisely the same set of observations.

Since we did not know the amount of cross traffic or — in the case of the distant server — the congestion-free bandwidth, we cannot present results for accuracy or settle time. Furthermore, without settle time, we cannot measure stability in the same way as for other experiments. Instead, we ignore the first half of each trial, and report results only for the last half. As expected, the filters produced similar average bandwidths to each server: 9.3 Mb/s to the local server and 5.8 Mb/s to the remote host.

The stability results for this experiment are shown in Fig. 10. These numbers should be taken with a grain of salt, as we do not know that the underlying conditions during this

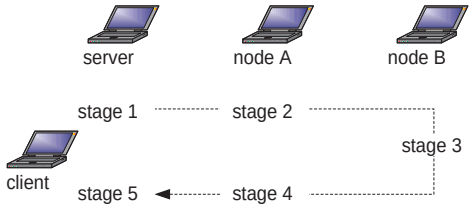


Figure 11: **Topology of Mobile Experiment**

time were in fact stable. Thus, these numbers may slightly understate the case.

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 larger number of hops combined with the greater likelihood of encountering cross traffic to the latter. As expected, the TCP filter is very stable across wide-area observations, but the flip-flop and stability filter also perform reasonably well compared to their peers. In fact, flip-flop is somewhat more stable over real networks than in the artificial environment of Fig. 9. We suspect that the more bursty congestion in real networks prevents the control limits from growing as large. All of the filters are comparable in the local-area case.

4.5 Mobility

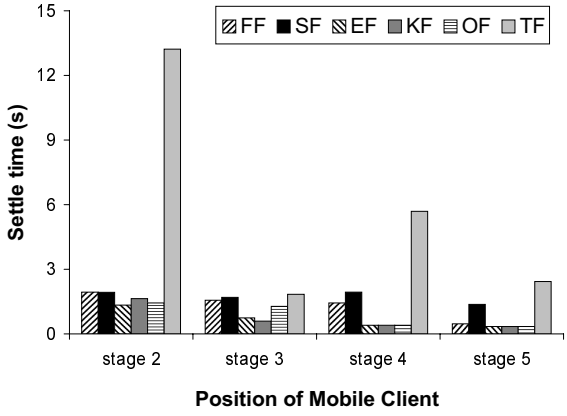
The final experiment explores how our filters react to changing topologies in mobile, ad hoc networks. We take full advantage of the Monarch extensions [13] to ns, which include near/far propagation models, packet capture, and the complete IEEE 802.11 MAC implementation [14]. The 802.11 MAC layer incorporates collision avoidance and link-level acknowledgement with retransmission.

In this experiment, we simulate the topology shown in Fig. 11. In it, three wireless nodes — one of them a server — are arranged in a line. The client moves from the server's neighborhood to the vicinity of the far node, and then back. The client completes this circuit in five minutes; we use handoff times as the breakpoints between stages. During stage 1, the node can talk directly to the server; at stage 2, it must go through node A to reach the server. The traffic rate was Poisson, with an average of four packets per second; request packets were small and response packets were 512 bytes or 8 KB with equal probability. We took five trials for each filter.

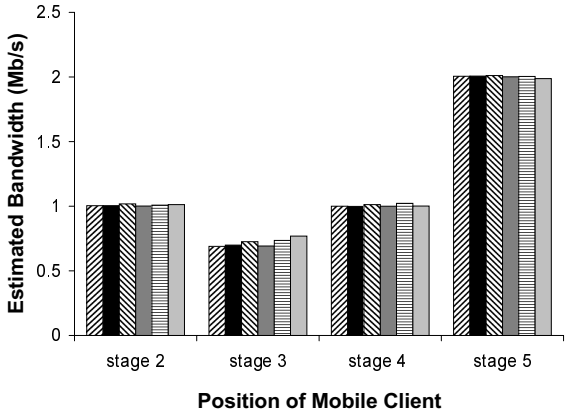
Even in this simple topology, there are interesting effects. First, the effective bandwidth changes as the client moves through the stages because all nodes share the same physical channel. Thus, while the bandwidth of the physical device is 2 Mb/s, the effective bandwidth between client and server is divided by two when routed through node A, and by three through node B.

Nominal latencies between client and server are 1.15 ms directly, 2.5 ms through node A, and 4 ms through node B. However, the wireless MAC protocol allows collisions even

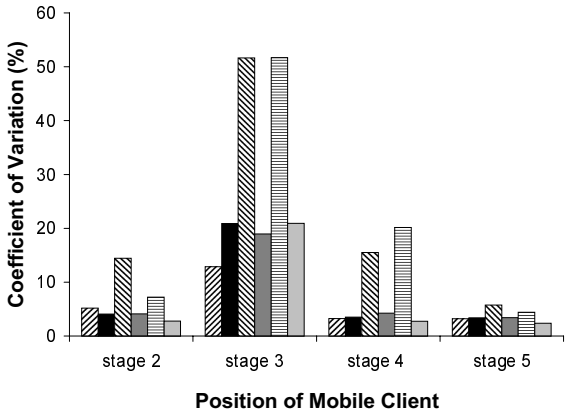
when the client and server communicate directly; the rate of collisions goes up with hop count. This leads to substantial variability in RTT observations, increased noise in all of the estimators, a higher average latency, and lower average bandwidth.



(a) Agility: settle times



(b) Accuracy: average estimate



(c) Stability: coefficient of variation

Figure 12: **Filter Performance in Mobile Environment**

The results for bandwidth estimation are shown in Fig. 12(a)–12(c). For brevity, we omit the latency results, where the relative performance of the filters is similar. However, because retransmissions have a much bigger effect on latency than on bandwidth, the latency estimators are significantly noisier, yield a wider spread in estimated values, and take longer to settle. Given the presence of link-level retransmissions, these differences may be reasonable.

As shown in Fig. 12(a), all of the filters except for TCP settle reasonably well, though the error-based and Kalman filters hold a slight advantage. Interestingly, the stability-based filter settles rather quickly in this environment, but was not particularly agile in the congestion experiments. This is because each individual stage in the mobility experiments is relatively free of noise, allowing the stability filter to quickly decrease its gain after handoff. In contrast, the uncertain interference provided by congestion traffic prevents the stability-based filter from quickly detecting the change.

All of the adaptive filters are reasonably accurate. All of the filters except for Odyssey and error-based show comparable stability, though the flip-flop filter has a slight advantage in the worst situation: stage three, where collisions and link-level retransmissions are most frequent. This is in contrast to the congestion experiments, where the stability-based filter held an advantage. The infrequent retransmissions in the mobile experiment lie outside the flip-flop control limits, and they are incorporated with the stable filter. However, in the congestion case, noise is regularly present; this tends to widen the flip-flop control limits.

4.6 Summary and Discussion

Unfortunately, none of the filters is a clear winner in all situations. However, all filters except flip-flop can be ruled out by extremely poor performance in one or more cases. The stability-based filter is especially resistant to transient changes. While it can track persistent changes reasonably well in the absence of noise, noisy observations force it towards stability. This prevents the stability filter from tracking true changes, as shown in Fig. 9. The error-based filter is especially agile, but it is far too susceptible to noise.

The Kalman filter's main drawback is its willingness to follow transient bandwidth changes — both in the synthetic network of Fig. 7 and the real networks depicted in Fig. 10. This could be due to a poorly-tuned $\mathbf{Q}$, since the relative variances were chosen arbitrarily. While it seems plausible that one could tune the Kalman filter for particular scenarios, it is doubtful that it could be tuned to handle all situations well. This is because $\mathbf{Q}$ fixes the relative variance of latency and bandwidth, but they are likely to change at different rates as circumstances change.

The flip-flop filter, in contrast, only suffers during periods of excessive noise, but no filter other than the overly-conservative TCP performs well in that situation. While it is

rarely the best performer, it handles bad cases relatively well. We conclude from these observations that the flip-flop filter is the best choice for good network performance estimates. We expect that the control limits in the flip-flop filter can be refined through a more sophisticated set of control rules. For example, by differentiating between noise — situations where individual observations fall both above and below the mean — and shifts in the observed mean, the flip-flop filter can reduce its instability under congestion. Exploring these *sensitizing rules for control charts* [7] remains an area of important future work.

5 Related Work

The work most closely related to our own is Keshav's work on flow control [15]. He proposed the *packet pair* technique: the use of two closely spaced packets to elicit self-interference queueing delay and hence 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. Instead, he employed a fuzzy logic estimator based on the same heuristic used by our error-based filter, but with an added mechanism to resist transient spikes. This mechanism would be helpful in the mobile networking environment, but not in the presence of congestion-induced noise. In fact, this estimator was explicitly designed for *rate-allocation servers* that are less severely affected by cross traffic noise. Unfortunately, the current infrastructure is based on FCFS routers; a domain this work explicitly excludes.

There have been several approaches to estimating network performance through active probing. For example, Bolot [16] uses pairs of UDP packets to explore network state, but requires substantial amounts of bandwidth to do so. Downey's application of `pathchar` [17] 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 [18]. These tools rely on bursts of ICMP packets, sent in several phases; they assume that network conditions do not change during this process, limiting the granularity of changes that they can detect.

Several variations on packet pair improve its ability to generate spot observations of network performance. Paxson [1] presents receiver-based packet pair, which takes observations at the receiver that incorporate timing information from the sender for more accurate measurements. Lai [19] presents a further refinement, called receiver only packet pair. It depends only on timing information taken at the receiver and incorporates a mechanism called *packet windows* that increases the agility of measurements, but leaves them susceptible to transient noise. Lai's subsequent work develops a more sophisticated network model [20] that generates accurate spot measurements. He focuses on determining bottleneck bandwidth along a path — ignoring cross traffic — by determining the minimum delay along that path. However, our filters could be used with this model to determine available bandwidth.

Finally, Balakrishnan's congestion manager [21] allows multiple conversations — 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 EWMA filter. We believe that the filters presented here can be used in this system to improve the agility of applications without unduly sacrificing stability.

6 Conclusion

Good-quality estimates of network performance are indispensable for a number of applications. Most systems use simple, exponentially weighted moving average filters to provide estimates. Unfortunately these filters are either too stable or too agile, depending on their gain.

To address this problem, we present four candidate filters, each of which strives to determine whether it should be aggressive or conservative in following changing observations. The flip-flop filter applies techniques from statistical process control to select between an agile EWMA filter and a stable one. The stability-based and error-based filters use heuristics to dynamically set the gain on an EWMA filter; the former increases gain when network observations are stable and the latter increases gain in response to inaccurate estimates. The fourth filter is a practical application of the commonly-used Kalman filter.

Each of these filters is subjected to a variety of idealized and realistic networking conditions to evaluate their efficacy. There are three metrics by which these filters are compared: agility, stability, and accuracy. Common to all of these metrics is the notion of settle time, the time required for a filter to produce an estimate within 10% of the true value. None of the candidate filters dominates in all circumstances. However, the flip-flop filter has some attractive properties with no serious drawbacks, and may be amenable to further refinement. With these improvements, the flip-flop filter can be used in a variety of adaptive systems to improve their performance.

References

[1] V. Paxson, "End-to-end internet packet dynamics," in *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997, pp. 139–52.

[2] B. D. Noble, M. Satyanarayanan, G. T. Nguyen, and R. H. Katz, "Trace-based mobile network emula-

tion," in *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997, pp. 51–61.

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

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

[5] C. Labovitz, G. R. Malan, and F. Jahanian, "Internet routing instability," *IEEE/ACM Transactions on Networking*, vol. 6, no. 5, pp. 515–28, October 1998.

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

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

[8] V. Jacobson, "Congestion avoidance and control," in *Proceedings of ACM SIGCOMM '88*, August 1988, pp. 314–329.

[9] 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*, Saint-Malo, France, October 1997, pp. 276–87.

[10] Western Electric, *Statistical Quality Control Handbook*, Western Electric Corporation, Indianapolis, Ind., 1956.

[11] S. E. Rigdon, E. N. Cruthis, and C. W. Champ, "Design strategies for individuals and moving range control charts," *Journal of Quality Technology*, vol. 26, no. 4, pp. 274–87, October 1994.

[12] 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*, vol. 33, no. 5, pp. 59–67, May 2000.

[13] 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*, Dallas, TX, USA, October 1998, pp. 85–97.

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

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

[16] J.-C. Bolot, "Characterizing end-to-end packet delay and loss behavior in the Internet," *Journal of High Speed Networks*, vol. 2, no. 3, pp. 305–23, 1993.

[17] A.B. Downey, "Using pathchar to estimate internet link characteristics," in *Proceedings of ACM SIGCOMM '99*, August 1999, pp. 241–250.

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

[19] K. Lai and M. Baker, "Measuring bandwidth," in *Proceedings of INFOCOM '99*, New York, NY, USA, March 1999, pp. 235–45.

[20] 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, To appear.

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