A Control-Theoretic Approach to Flow Control

Srinivasan Keshav

Computer Science Division, Department of EECS,
University of California, Berkeley,
and
International Computer Science Institute
Berkeley, CA 94720, USA.
keshav@tenet.Berkeley.EDU

Abstract

This paper presents a control-theoretic approach to reactive flow control in networks that do not reserve bandwidth. We assume a round-robin-like queue service discipline in the output queues of the network's switches, and propose deterministic and stochastic models for a single conversation in a network of such switches. These models motivate the Packet-Pair rate probing technique, and a provably stable rate-based flow control scheme. A Kalman state estimator is derived from discrete-time state space analysis, but there are difficulties in using the estimator in practice. These difficulties are overcome by a novel estimation scheme based on fuzzy logic. We then present a technique to extract and use additional information from the system to develop a continuous-time system model. This is used to design a variant of the control law that is also provably stable, and, in addition, takes control action as rapidly as possible. Finally, practical issues such as correcting parameter drift and coordination with window flow control are described.

1. Introduction

As networks move towards integrated service, there is a need for network control mechanisms that can provide users with different qualities of service in terms of throughput, delay and delay jitter [1]. Recent work has shown that these guarantees can be provided if the network makes bandwidth reservations on behalf of each conversation (also called channel, circuit or virtual circuit; we use 'conversation' throughout) [2-5]. However, such reservations can reduce the statistical multiplexing in the network, making the system expensive.

An interesting problem arises in the control of conversations that do not reserve bandwidth, and hence are not given any performance guarantees by the network. These conversations must adapt to changing network conditions in order to achieve their data transfer goals. In this paper we present a control-theoretic approach to determine how a conversation can satisfy its throughput and queueing delay requirements by adapting its data transfer rate to changes in network state, and to prove that such adaptations do not lead to instability. This approach can be used to control both transport connections in reservationless networks, and so-called 'best-effort' connections in reservation-oriented networks [2].

A control theoretic approach to flow control requires that changes in the network state be observable. In recent work, we have shown that it is possible to measure network state easily if the servers at the output queues of the switches are of a type called a Rate Allocating Server and the transport protocol uses the Packet-Pair probing technique (described below) [6-8]. Thus, in this paper, we will make the assumption that the queue service discipline is of the RAS type and that sources implement Packet-Pair. Our approach does not extend to First-Come-First-Served (FCFS) networks, where there is no simple way to probe the network state.

The paper is laid out as follows. We first describe rate allocating servers (§2), and present deterministic and stochastic models for networks of such servers (§3). Next, we describe the Packet-Pair state probing technique (§4). This is used as the basis for the design of a stable rate-based flow control scheme (§5). A problem with non-linearity in the system is discussed in §6. We present a Kalman state estimator in §7. However, this estimator is impractical, and so we have designed a novel estimation scheme based on fuzzy logic (§8). A technique to increase the frequency of control based on additional information from the system is presented in §9, and this serves as the basis for a new control law. Practical implementation issues are discussed in §10, and these include correcting for parameter drift, and interaction with window flow control. We conclude with some remarks on

This research was supported by the National Science Foundation and the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives, by AT&T Bell Laboratories, Hitachi, Ltd., the University of California under a MICRO grant, and the International Computer Science Institute. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing official policies, either expressed or implied, of the U.S. Government or any of the sponsoring organizations. Current address: AT&T Bell Laboratories, 600 Mtn. Ave, Murray Hill NJ 07974.

the limitations of the approach (§11) and a review of related work (§ 12).

2. Rate Allocating Servers

In this section, we describe the notion of a Rate Allocating Server. Consider the queue service discipline in the output queues of the routers of a communication network. If packets are scheduled in strict Time-Division-Multiplexing (TDM) order, then whenever a conversation's time slot comes around and it has no data to send, the output trunk is kept idle and some bandwidth is wasted. Suppose packets are stamped with a priority index that corresponds to the time of service of the packet were the server actually to do TDM. It can be shown that service in order of increasing priority index has the effect of approximately emulating TDM without its attendant inefficiencies [9]. This idea lies behind the Fair Queueing service discipline [10]. In recent work it has been shown that Fair Queueing is quite similar to the Virtual Clock [5] discipline [11]. Thus, we will refer to both as Rate Allocating Servers (RASs); the reason for this name will shortly become clear.

While the Virtual Clock scheduling discipline was originally presented in the context of a reservation-oriented network layer, we study its behavior in reservationless networks. This raises an important point. In reservation-oriented networks, during call set-up, a conversation specifies a desired service rate to the servers that lie in its path. This information allows each server to prevent overbooking of its bandwidth and the service rate that a conversation receives is constant. However, in reservationless networks, a server is not allowed to refuse any conversations, and so the bandwidth could be overbooked. We assume that in such a situation, a RAS will divide bandwidth in the same way as a Fair Queueing server, that is, equally among the currently active conversations.

With a RAS, there are two reasons why the perceived rate of service of a specific conversation may change. First, the total number of conversations served can change. Since the service rate of the selected conversation is inversely proportional to the number of active conversations, the service rate of that conversation also changes.

Second, if some conversation has an arrival rate slower that its allocated bandwidth share, or has a bursty arrival pattern, then there are intervals where it does not have any packets to send, and the RAS will treat that conversation as idle. Thus, the effective number of active conversations decreases, and the rate allocated to all the other conversations increases. When its traffic resumes, the service rate again decreases.

Note that even with these variations in the service rate, a RAS provides a conversation with a more consistent service rate than a FCFS server. In a FCFS server the service rate of a conversation is linked in detail to the arrival pattern of every other conversation in the server, and so the perceived service rate varies rapidly.

For example, consider the situation where the number of conversations sending data to a server is fixed, and each conversation always has data to send when it is scheduled for service. In a FCFS server, if any one conversation sends a large burst of data, then the service rate of all the other conversations effectively drops until the burst has been served. In a RAS, the other conversations will be unaffected. Thus, the server allocates a rate of service to each conversation that is, to a first approximation, independent of the arrival patterns of the conversations. So, a source sending data to a RAS server should use a rate-based flow control scheme that determines the allocated service rate, and then sends data at this rate.

3. Choice of network model

In this section, we consider how to model a network of Rate Allocating Servers. A theoretically sound flow control mechanism operating in a network of RASs requires an analytical model for network transients. If the network is modeled as a queueing system, usually the kind of results that can be obtained are those that hold in the average case. Though the Chapman-Kolmogorov differential equations do give the exact dynamics of a M/M/1 queue, the solution of these equations is equivalent to evaluating an infinite sum of Bessel functions [12]. This problem is made more complicated if the network is non-Jacksonian (as in our case). Thus, we feel that by using a queueing network model, transient analysis becomes cumbersome and sometimes impossible. However, flow control depends precisely on such transients. Thus, we would like to use an approach that models network transients explicitly.

We choose to model a network of RASs deterministically since it has been shown that a deterministic modeling of a RAS network allows network transients to be calculated exactly [7]. Waclawsky and Agrawala have developed and analyzed a similar deterministic model for studying the effect of window flow control protocols on virtual circuit dynamics [13, 14]. However, this approach can be too simplistic, since it ignores the variations in the allocated service rate discussed in §2. So, we first summarize a deterministic model similar to the one presented in [7], and then present a stochastic extension.

3.1. Deterministic Model

We model a conversation in a FQ network as a regular flow of packets from a source to a destination (sink) over a series of servers (routers or switches) connected by links. The servers in the path of the conversation are numbered 1,2,3...n, and the source is numbered 0. The destination is assumed to acknowledge each packet. (Strictly speaking, this assumption is not required, but we make it for ease of exposition.) We assume, for ease of analysis, that sources always have data to send. This simplification allows us to ignore start-up transients in our analysis. The start-up costs can, in fact, be significant, and these are analyzed in [7].

The time taken to get service at each server is finite and deterministic. If the *i*th server is idle when a packet arrives, the time taken for service is s_i, and the (instantaneous) service rate is defined to be $\rho_i = 1/s_i$. Note that the time to serve one packet includes the time taken to serve packets from all other conversations in round-robin order. Thus, the service rate is the inverse of the time between consecutive packet services from the same conversation.

If the server is not idle when a packet arrives, then the service time can be more than s_i. This is ignored in the model, but we will consider the implications in section 4. If there are other packets from that conversation at the server, an incoming packet waits for its turn to get service (we assume a FCFS queueing discipline for packets from the same conversation).

The source sending rate is denoted by λ and the source is assumed to send packets spaced exactly $s_0 = 1/\lambda$ time units apart. We define

$$s_b = \max_i (s_i \mid 0 \le i \le n)$$

to be the *bottleneck* service time in the conversation, and *b* is the index of the bottleneck server. μ, the bottleneck service rate, is defined to be $\frac{1}{s_b}$.

We will henceforth work in discrete time, so the continuous time parameter, t, is replaced by the step index k. One step corresponds to one round-trip, as explained in section 5.2. In addition, let $S(k)$ be the number of unacknowledged packets at the source at time k.

3.2. Stochastic model

In the deterministic model, μ is assumed to be constant. Actually, μ changes due to the creation and deletion of active conversations. If the number of active conversations, N_{ac}, is large, we expect that the change in N_{ac} in one time interval will be small compared to N_{ac}. Hence the change in μ in one interval will be small and $\mu(k+1)$ will be 'close' to $\mu(k)$. One way to represent this would be for μ to be a fluctuation around a nominal value μ_0. However, this does not adequately capture the dynamics of the process, since $\mu(k+1)$ is 'close' to $\mu(k)$ and not to a fixed value μ_0. Instead, we model μ as a random walk where the step is a random variable that has zero mean and has low variance. Thus, for the most part, changes are small, but we do not rule out the possibility of a sudden large change. This model is simple and though it represents only the first order dynamics, we feel that it is sufficient for our purpose. Thus, we define

$$\mu(k+1) \;=\; \mu(k) \;+\; \omega(k),$$

where $\omega(k)$ is a random variable that represents zero-mean gaussian white noise. There is a problem here: when μ is small, the possibility of an increase is larger than the possibility of a decrease. Hence, at this point, the distribution of ω is asymmetric, with a bias towards positive values (making the distribution non-gaussian). However, if μ is sufficiently far away from 0, then the assumption of zero mean is justifiable.

The white noise assumption means that the changes in service rate at time k and time $k+1$ are uncorrelated. Since the changes in the service rate are due to the effect of uncorrelated input traffic, we think that this is valid. However, the gaussian assumption is harder to justify. As mentioned in [15], many noise sources in nature are gaussian. Second, a good rule of thumb is that the gaussian assumption will reflect at least the first order dynamics of any noise distribution. Finally, for any reasonably simple control theoretic formulation (using Kalman estimation) the gaussian white noise assumption is unavoidable. Thus, for these three reasons, we will assume that the noise is gaussian.

These strict assumptions about the system noise are necessary mainly for doing Kalman estimation. We also describe a fuzzy prediction approach (§8) that does not make these assumptions

Note that the queueing theoretic approach to modeling μ would be to define the density function of μ, say $G(\mu)$, that would have to be supplied by the system administrator. Then, system performance would be given by expectations taken over the distribution. In contrast, we explicitly model the dynamics of μ and so our control scheme can depend on the currently measured value of μ, as opposed to only on an asymptotic time average.

4. Detailed Dynamics of Packet-Pair

State probing is done using the Packet-Pair mechanism, shown in Figure 1.
The figure presents a time diagram. Time increases along the vertical axes, and each axis represents a node in a communication network. The parallelograms represent the transmission of a packet and correspond to two kinds of delays: the vertical sides are as long as the transmission delay (the packet size divided by

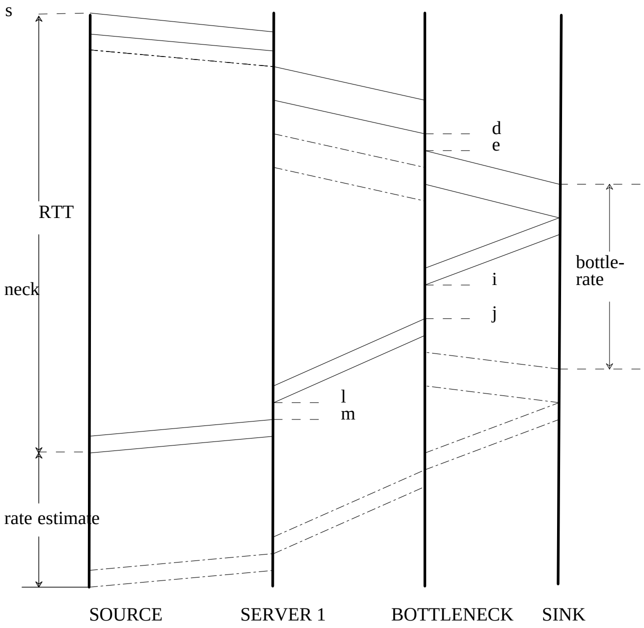


Figure 1: The Packet-Pair probing scheme

the line capacity). Second, the slope of the longer sides is proportional to the propagation delay. Since the network is store-and-forward, a packet cannot be sent till it is completely received. After a packet arrives, it may be queued for a while before it receives service. This is represented by the space between two dotted lines, such as *de*.

In the Packet-Pair scheme, the source sends out two back-to-back packets (at time *s*). These are serviced by the bottleneck; by definition, the inter-packet service time is $1/\mu$, the service time at the bottleneck. Since the acks preserve this spacing, the source can measure the inter-ack spacing to estimate μ.

We now consider possible sources of error in the estimate. The server marked 1 also spaces out the back-to-back packets, so can it affect the measurement of μ? A moment's reflection reveals that as long as the second packet in the pair arrives at the bottleneck before the bottleneck ends service for the first packet, there is no problem. If the packet does arrive after this time, then, by definition, server 1 itself is the bottleneck. Hence, the spacing out of packets at servers before the bottleneck server is of no consequence, and does not introduce errors into the scheme. Another detail that does not introduce error is that the first packet may arrive when the server is serving other packets and may be delayed, for example, by *de*. Since this delay is common to both packets in the pair, this does not matter.

What does introduce an error is the fact that the acks may be spread out more (or less) than s_b due to different queueing delays for each ack along the return path. In the figure note that the first ack has a net queueing delay of *ij* + *lm*, and the second has a zero queueing delay. This has the effect of increasing the estimate of μ.

Note that this source of error will persist even if the inter-ack spacing is noted at the sink and sent to the source using a state exchange scheme [16]. Measuring μ at the sink will reduce the effect of noise, but cannot eliminate it, since any server that is after the bottleneck could also cause a noise in the measurement.

We model this error in measurement as an observation noise. Since the observed value of μ can be either increased or decreased by this noise, with equal probability in either direction, we expect that the noise distribution is symmetric about 0. As a simplification, it is again assumed that the distribution is gaussian, and that the noise is white.

5. Design Strategy

This section describes the strategy used to design the flow-control mechanism, some preliminary considerations, and the detailed design. The design strategy for the flow control mechanism is based upon the Separation Theorem [17]. Informally, the theorem states that for a linear stochastic system where an observer is used to estimate the system state, the eigenvalues of the state estimator and the controller are separate. The theorem allows us to use any technique for state estimation, and then implement control using the estimated state $\hat{x}$ instead of the actual state x. Thus, we will derive a control law assuming that all required estimators are available; the estimators are derived in section 7. We first discuss our assumptions and a few preliminary considerations.

5.1. Choice of Setpoint

The aim of the control is to maintain the number of packets in the bottleneck queue, n_b at a desired setpoint. Since the system has delay components, it is not possible for the control to stay at the setpoint at all times. Instead, the system will oscillate around the setpoint value. The choice of the setpoint reflects a tradeoff between mean packet delay, packet loss and bandwidth loss (which is the bandwidth a conversation loses because it has no data to send when it is eligible for service). This is discussed below.

Let B denote the number of buffers a switch allocates per conversation (in general, B may vary with time, and this can be accounted for. In this paper, we assume that B is static). Consider the distribution of n_b for the controlled system, given by $N(x) = Pr(n_b = x)$ (strictly speaking, $N(x)$ is a Lebesgue measure, since we will use it to denote point probabilities). $N(x)$ is sharply delimited on the left by 0 and on the right by B and tells us three things:

1) Pr(loss of bandwidth) = Pr (RAS server schedules the conversation for service $|$ $n_b = 0$). Assuming that these events are independent, which is a reasonable assumption, we find that Pr(loss of bandwidth) is proportional to $N(0)$.

2) Similarly, Pr (loss of packet) = Pr (packet arrival $|$ $n_b = B$), so that the density at B, $N(B)$ is proportional to the probability of a packet loss.

3) the mean queuing delay is given by

$$\frac{s_b}{\int_0^B N(x)\,dx} \int_0^B xN(x)\,dx,$$

where, on average, a packet takes s_b units of time to get service at the bottleneck.

If the setpoint is small, then the distribution is driven towards the left, the probability of bandwidth loss increases, the mean packet delay is decreased, and the probability of packet loss is decreased. Thus, we trade off bandwidth loss for lower mean delay and packet loss. Similarly, if we choose a large setpoint, we will trade off packet loss for a larger mean delay and lower probability of bandwidth loss. In the sequel, we assume a setpoint of $B/2$. The justification is that, since the system noise is symmetric, and the control tracks the system noise, we expect $N(x)$ to be symmetric around the setpoint. In that case, a setpoint of $B/2$ balances the two tradeoffs. Of course, any other setpoint can be chosen with no loss of generality.

Recent work by Mitra *et al* has shown that asymptotic analysis of product form queueing networks can be used to derive an optimal value of the setpoint [18, 19]. The application of their ideas to this problem is explored in reference [20].

5.2. Frequency of Control

We initially restrict control actions to only once per round trip time (RTT) (this restriction is removed in section 9). For the purpose of exposition, divide time into *epochs* of length RTT (= R + queueing delays) (Figure 2). This is done simply by transmitting a specially marked packet-pair, and when it returns, taking control action, and sending out another marked pair. Thus, the control action is taken at the end of every epoch.

5.3. Assumptions Regarding Round Trip Time Delay

We assume that the propagation delay, R, is constant for a conversation. This is usually true, since the propagation delay is due to the speed of light in the fiber and hardware switching delays. These are fixed, except for rare rerouting.

We assume that the round trip time is large compared to the spacing between the acknowledgments. Hence, in the analysis, we treat the arrival of the packet pair as a single event, that measures both the round trip time and the bottleneck service rate.

Finally, we assume that the measured round trip time in epoch k, denoted by $RTT(k)$, is a good estimate for the round trip time in epoch $k+1$. The justification is that when the system is in equilibrium, the queue lengths are expected to be approximately the same in successive epochs. In any case, for wide area networks, the propagation delay will be much larger than the additional delay caused by a change in the queueing delay. Hence, to a first approximation, this change can be ignored. This assumption is removed in section 9.

5.4. Controller Design

Consider the situation at the end of the kth epoch. At this time we know $RTT(k)$, the round trip time in the kth epoch, and $S(k)$, the number of packets outstanding at that time. We also *predict* $\hat{\mu}(k+1)$, which is the estimator for the average service rate during the $(k+1)$th epoch. If the service rate is 'bursty', then using a time average for μ may lead to problems. For example, if the average value for μ is large, but during the first part of the control cycle, the actual value is low, then the bottleneck buffers could overflow. In such cases, we can take control action with the arrival of every probe, as discussed in section 9.

Figure 2 shows the time diagram for the control. The vertical axis on the left is the source, and the axis on the right is the bottleneck. Each line between the axes represents a packet pair. Control epochs are marked for the source and the bottleneck. Note that the epochs at the bottleneck are time delayed with respect to the source. We use the convention that the end of the kth epoch is called 'time k', except that $n_b(k)$ refers to the number of packets in the bottleneck at the *beginning* of the kth epoch. Estimators are marked with a hat.

We now make a few observations regarding Figure 2. The distance ab is the RTT measured by the source (from the time the first packet in the pair is sent to the time the first ack is received). By an earlier assumption, the propagation delay for the $(k+1)$th special pair is the same as for the kth pair. Then $ab = cd$, and the length of epoch k at the source and at the bottleneck will be the same, and equal to $RTT(k)$.

At the time marked 'NOW', which is the end of the kth

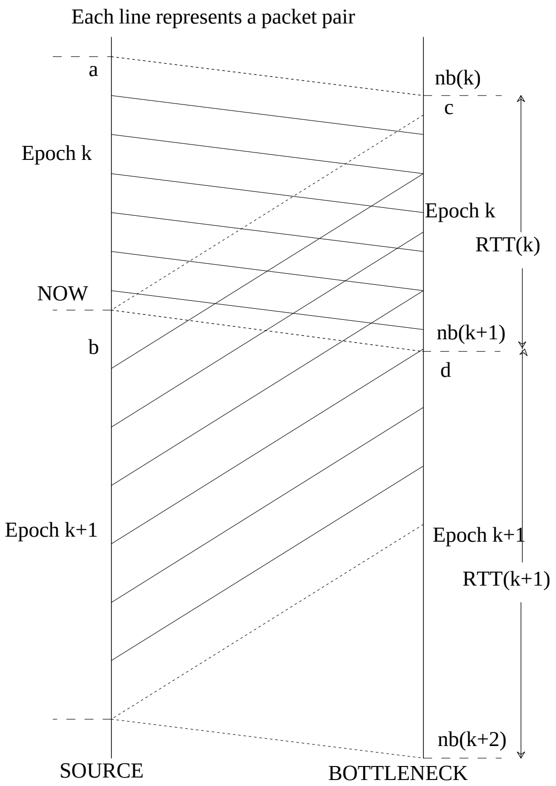


Figure 2: Time scale of control

epoch, all the packets sent in epoch $k-1$ have been acknowledged. So, the only unacknowledged packets are those sent in the kth epoch itself, and this is the same as the number of outstanding packets $S(k)$. This can be approximated by the sending rate multiplied by the sending interval, $\lambda(k) RTT(k)$. So,

$$S(k) = \lambda(k) RTT(k) \qquad 1$$

The number of packets in the bottleneck at the beginning of the $(k+1)$th epoch is simply the number of packets at the beginning of the kth epoch plus what came in minus what went out in the kth epoch (ignoring the non-linearity at $n_b = 0$, discussed in §5.6). Since $\lambda(k)$ packets were sent in, and $\mu(k) RTT(k)$ packets were serviced in this interval, we have

$$n_b(k+1) = n_b(k) + \lambda(k) RTT(k) - \mu(k) RTT(k) \qquad 2$$

Equations (1) and (2) are the fundamental equations in this analysis. They can be combined to give

$$n_b(k+1) = n_b(k) + S(k) - \mu(k) RTT(k) \qquad 3$$

Now, $n_b(k+1)$ is already determined by what we sent in the kth epoch, so there is no way to control it. Instead, we will try to control $n_b(k+2)$. We have

$$n_b(k+2) = n_b(k+1) + (\lambda(k+1) - \mu(k+1)) RTT(k+1) \qquad 4$$

From (3) and (4):

$$n_b(k+2) = n_b(k) + S(k) - \mu(k) RTT(k) + \qquad 5$$
$$\lambda(k+1) RTT(k+1) - \mu(k+1) RTT(k+1)$$

The control should set this to $B/2$. So, set (5) to $B/2$, and obtain $\lambda(k+1)$.

$$n_b(k+2) = B/2 = n_b(k) + S(k) - \mu(k) RTT(k) \qquad 6$$
$$+ (\lambda(k+1) - \mu(k+1)) RTT(k+1)$$

This gives $\lambda(k+1)$ as

$$\lambda(k+1) = \frac{1}{RTT(k+1)} \qquad 7$$
$$[B/2 - n_b(k) - S(k) + \mu(k) RTT(k) + \mu(k+1) RTT(k+1)]$$

Replacing the values by their estimators (which will be derived later), we have

$$\lambda(k+1) = \frac{1}{\hat{RTT}(k+1)} \qquad 8$$
$$[B/2 - \hat{n}_b(k) - S(k) + \hat{\mu}(k) RTT(k) + \hat{\mu}(k+1) \hat{RTT}(k+1)]$$

Since both $\hat{\mu}(k)$ and $\hat{\mu}(k+1)$ are unknown, we can safely assume that $\hat{\mu}(k) = \hat{\mu}(k+1)$. Further, from an earlier assumption, we set $RTT(k+1)$ to $RTT(k)$. This gives us:

$$\lambda(k+1) = \frac{1}{RTT(k)} [B/2 - \hat{n}_b(k) - S(k) + 2\hat{\mu}(k) RTT(k)] \qquad 9$$

This is the control law. The control always tries to get the buffer to $B/2$. It may never reach there, but will always stay around it.

Note that the control law requires us to maintain two estimators: $\hat{\mu}(k)$ and $\hat{n}_b(k)$. The effectiveness of the control depends on the choice of the estimators. This is considered in sections 7 and 8.

5.5. Stability Analysis

The state equation is given by (2)

$$n_b(k+1) = n_b(k) + \lambda(k) RTT(k) - \mu(k) RTT(k) \qquad 10$$

For the stability analysis of the controlled system, $\lambda(k)$ should be substituted using the control law. Since we know $\lambda(k+1)$, we use the state equation derived from (2) instead (which is just one step forward in time). This gives

$$n_b(k+2) = n_b(k+1) + (\lambda(k+1) - \mu(k+1)) RTT(k+1)$$

Substitute (8) in (10) to find the state evolution of the controlled system.

$$n_b(k+2) = n_b(k+1) - \mu(k+1) RTT(k+1) + \frac{RTT(k+1)}{RTT(k)}$$
$$[B/2 - \hat{n}_b(k) - S(k) + 2\hat{\mu}(k) RTT(k)]$$

By assumption, $RTT(k)$ is close to $RTT(k+1)$. So, to first order, canceling $RTT(k)$ with $RTT(k+1)$ and moving back two steps in time,

$$n_b(k) = n_b(k-1) - \mu(k-1) RTT(k-1) +$$
$$B/2 - \hat{n}_b(k-2) - S(k-2) + 2\hat{\mu}(k-2) RTT(k-2)$$

Taking the Z transform of both sizes, and assuming $n_b(k-2) = \hat{n}_b(k-2)$, we get

$$n_b(z) = z^{-1} n_b(z) - z^{-2} \mu(z) * RTT(z) +$$
$$B/2 - z^{-2} n_b(z) - z^{-2} S(z) + 2z^{-4} \hat{\mu}(z) * RTT(z)$$

Considering n_b as the state variable, it can be easily shown that the characteristic equation is

$$z^{-2} - z^{-1} + 1 = 0$$

If the system is to be asymptotically stable, then the roots of the characteristic equation (the eigenvalues of the system), must lie inside the unit circle on the complex Z plane. Solving for z^{-1}, we get

$$z^{-1} = \frac{1 \pm \sqrt{1-4}}{2} = \frac{1 \pm \sqrt{3}\, i}{2}$$

The distance from 0 is hence

$$\sqrt{\frac{1}{2}^2 + \frac{\sqrt{3}}{2}^2} = 1$$

Since the eigenvalues lie on the unit circle, the controlled system is *not* asymptotically stable.

However, we can place the pole of the characteristic equation so that the system is asymptotically stable. Consider the control law

$$\lambda(k+1) = \frac{\alpha}{RTT(k)}$$

$$[B/2 - \hat{n}_b(k) - S(k) + 2\hat{\mu}(k)RTT(k)]$$

$$[B/2 - \hat{n}_b(k) - S(k) + 2\hat{\mu}(k)RTT(k)]$$

This leads to a characteristic equation

$$\alpha z^{-2} - z^{-1} + 1 = 0$$

so that the roots are

$$z^{-1} = \frac{1 \pm \sqrt{4\alpha - 1\, i}}{2\alpha}$$

The poles are symmetric about the real axis, so we need only ensure that

$$|z^{-1}| > 1$$

$$=> \sqrt{(\frac{1}{2\alpha})^2 + (\frac{\sqrt{4\alpha - 1}}{2\alpha})^2} > 1$$

$$=> \frac{1}{\sqrt{\alpha}} > 1 \quad => \quad \alpha < 1$$

This means that if $\alpha < 1$, the system is provably asymptotically stable (by the Separation Theorem, since the system and observer eigenvalues are distinct, this stability result holds irrespective of the choice of the estimators).

The physical interpretation of α is simple: to reach $B/2$ at the end of the next epoch, the source should send exactly at the rate computed by (9). If it does so, the system may be unstable. Instead, it sends at a slightly lower rate, and this ensures that the system is asymptotically stable. Note that α is a constant that is independent of the system's dynamics and can be chosen in advance to be any desired value smaller than 1.0. The exact value chosen for α controls the rise time of the system, and for adequate responsiveness, it should not be too small. Our simulations indicate that a value of 0.9 is a good compromise between responsiveness and instability [20]. Similar studies are mentioned in [21].

6. System non-linearity

This section discusses a non-linearity in the system, and how it can be accounted for in the analysis. Note that the state equation (9) is correct when $n_b(k+1)$ lies in the range 0 to B. Since the system is physically incapable of having less than 0, and more than B packets in the bottleneck queue, the equation actually is incorrect at the endpoints of this range. The correct equation is then:

$$n_b(k+1) = \begin{cases} \text{if } n_b(k) + S(k) - RTT(k)\mu(k) < 0 \textit{ then } 0 \\ \text{if } n_b(k) + S(k) - RTT(k)\mu(k) > B \textit{ then } B \\ \quad \textit{otherwise} \;\; n_b(k) + S(k) - RTT(k)\mu(k) \end{cases}$$

The introduction of the MAX and MIN terms in the state equation makes the system nonlinear at the boundaries. This is a difficulty, since the earlier proof of stability is valid only for a linear system. However, note that if the equilibrium point (setpoint) is chosen to lie in the interior of the range [0,B], then the system is linear around the setpoint. Hence, for small deviations from the setpoint, the earlier stability proof, which assumes linearity, is sufficient. For large deviations, stability must be proved by other methods, such as the second method of Liapunov [22] page 558.

However, this is only as an academic exercise. In practice, the instability of the system means that n_b can move arbitrarily away from the setpoint. In section 10.2, we show how window-based flow control can be used in conjunction with a rate-based approach. Then, n_b can never be less than 0, and the window flow control protocol ensures that it never exceeds B, and so true instability is not possible.

Nevertheless, we would like the system to return to the setpoint, whenever it detects that it has moved away from it, rather than operating at an endpoint of its range. This is automatically assured by equation (9), which shows that the system chooses $\lambda(k+1)$ such that $n_b(k+2)$ is $B/2$. So, whenever the system detects that it is at an endpoint, it immediately takes steps to ensure that it moves away from it.

Thus, the non-linearity in the system is of no practical consequence, except that the flow control mechanism has to suitably modify the state equations when updating $\hat{n}_b(k+1)$. A rigorous proof of the stability of the system using Liapunov's second method is also possible, but the gain from the analysis is slight.

7. Kalman State Estimation

This section presents a Kalman state estimator, and shows that Kalman estimation is impractical. A practical scheme is presented in §8.

Having derived the control law, and proved its stability, we now need to determine stable estimators for the system state. We choose to use Kalman estimation, since it is a well known and robust technique [23]. Before the technique is applied, a state-space description of the system is necessary.

7.1. State Space Description

We will use the standard linear stochastic state equation given by

$$\mathbf{x}(k+1) = \mathbf{Gx}(k) + \mathbf{Hu}(k) + \nu_1(k)$$
$$\mathbf{y}(k) = \mathbf{Cx}(k) + \nu_2(k)$$

x, **u** and **y** are the state, input and output vectors of sizes n, m and r, respectively. **G** is the nxn state matrix, **H** is an nxm matrix, and **C** is an rxn matrix. $\nu_1(k)$ represents the system noise vector, which is assumed to be zero-mean, gaussian and white. $\nu_2(k)$ is the observation noise, and it is assumed to have the same characteristics as the system noise.

Clearly, **u** is actually u, a scalar, and $u(k) = \lambda(k)$. At the end of epoch k, the source receives probes from epoch k-1. (To be precise, probes can be received from epoch k-1 as well as from the beginning of epoch k. However, without loss of generality, this is modeled as part of the observation noise.) So, at that time, it knows the average service time in the k-1th epoch, $\mu(k-1)$. This is the only observation it has about the system state and so $y(k)$ is a scalar, $y(k) = \mu(k-1) + \nu_2$. If this is to be derived from the state vector **x** by multiplication with a constant matrix, then the state must contain $\mu(k-1)$. Further, the state must also include the number of packets in the buffer, n_b. This leads to a state vector that has three elements, n_b, $\mu(k)$ and $\mu(k-1)$, where $\mu(k)$ is needed since it is part of the delay chain leading to $\mu(k-1)$ in the corresponding signal flow graph. Thus,

$$\mathbf{x} = \begin{bmatrix} n_b \\ \mu \\ \mu_{-1} \end{bmatrix}$$

where μ_{-1} represents the state element that stores the one step delayed value of μ.

We now turn to the **G**, **H**, ν_1, ν_2 and **C** matrices. The state equations are

$$n_b(k+1) = n_b(k) + \lambda(k) RTT(k) - \mu(k) RTT(k)$$

$$\mu(k+1) = \mu(k) + \omega(k)$$

$$\mu_{-1}(k+1) = \mu(k)$$

Since $RTT(k)$ is known at the end of the *k*th epoch, we can represent it by a pseudo-constant, *Rtt*. This gives us the matrices

$$\mathbf{G} = \begin{bmatrix} 1 & -Rtt & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\mathbf{H} = \begin{bmatrix} Rtt \\ 0 \\ 0 \end{bmatrix}$$

$$\nu_1 = \begin{bmatrix} 0 \\ \omega \\ 0 \end{bmatrix}$$

$$\mathbf{C} = [\ 0\ 0\ 1\ ]$$

ν_2 is simply the (scalar) variance in the observation noise. This completes the state space description of the flow control system.

7.2. Kalman Filter Solution to the Estimation Problem

A Kalman filter is the minimum variance state estimator of a linear system. In other words, of all the possible estimators for **x**, the Kalman estimator is the one that will minimize the value of $E([\ \hat{x}(t) - x(t)]^T[\hat{x}(t) - x(t)])$, and in fact this value is zero. Moreover, a Kalman filter can be manipulated to yield many other types of filters [23]. Thus, it is desirable to construct a Kalman filter for **x**.

In order to construct the filter, we need to determine three matrices, **Q, S** and **R**, which are defined implicitly by :

$$E\left\{ \begin{bmatrix} \nu_1(k) \\ \nu_2(k) \end{bmatrix} [\nu_1^T(\theta)\,\nu_2(\theta)] \right\} = \begin{bmatrix} Q & S \\ S^T & R \end{bmatrix} \delta\,(t - \theta)$$

where δ is the Kronecker delta defined by $\delta(k)$ = if (k = 0) then 1 else 0. Expanding the left hand side, we have

$$\mathbf{Q} = \mathrm{E} \begin{bmatrix} 0 & 0 & 0 \\ 0 & \omega^2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$\mathbf{R} = \mathrm{E}\,({\nu_2}^2)$$

$$\mathbf{S} = \mathrm{E} \begin{bmatrix} 0 \\ \omega\nu_2 \\ 0 \end{bmatrix}$$

If the two noise variables are assumed to be independent, then the expected value of their product will be zero, so that **S** = **0**. However, we still need to know $\mathrm{E}(\omega^2)$ and $\mathrm{E}({\nu_2}^2)$.

From the state equation,

$$\mu(k+1) = \mu(k) + \omega(k)$$

Also,

$$\mu_{observed}(k+1) = \mu(k+1) + \nu_2(k+1)$$

Combining,

$$\mu_{observed}(k+1) = \mu(k) + \omega(k) + \nu_2(k+1)$$

which indicates that the observed value of μ is affected by both the state and observation noise. As such, each component cannot be separately determined from the observations alone. Thus, in order to do Kalman filtering, the values of $\mathrm{E}(\omega^2)$ and $\mathrm{E}({\nu_2}^2)$ must be extraneously supplied, either by simulation or by measurement of the actual system. Practically speaking, even if good guesses for these two values are supplied, the filter will have reasonable (but not optimal) performance. Hence, we will assume that the value of noise variances are supplied by the system administrator, and so matrices **Q, R** and **S** are known. It is now straightforward to apply Kalman filtering to the resultant system. We follow the derivation in [23] (pg 249).

The state estimator $\hat{x}$ is derived using

$$\hat{\mathbf{x}}(k+1) = \mathbf{G}\hat{\mathbf{x}}(k) + \mathbf{K}(k)[y(k) - \mathbf{C}\hat{\mathbf{x}}(k)] + \mathbf{H}u(k)$$

$$\hat{\mathbf{x}}(0) = \mathbf{0}$$

where **K** is the Kalman filter gain matrix, and is given by

$$\mathbf{K}(k) = [\mathbf{G}\Sigma(k)\mathbf{C}^T + \mathbf{S}\ ][\mathbf{C}\Sigma(k)\mathbf{C}^T + \mathbf{R}]^{-1}$$

$\Sigma(k)$ is the error state covariance, and is given by the Riccatti difference equation

$$\Sigma(k+1) = \mathbf{G}\Sigma(k)\mathbf{G}^T + \mathbf{Q} - \mathbf{K}(k)[\mathbf{C}\Sigma(k)\mathbf{C}^T + \mathbf{R}]\mathbf{K}(k)^T$$

$$\Sigma(0) = \Sigma_0$$

where Σ_0 is the covariance of **x** at time 0, and can be assumed to be **0**.

Note that a Kalman filter requires the Kalman gain matrix **K**(k) to be updated at each time step. This computation involves a matrix inversion, and appears to be expensive. However, since all the matrices are at most 3x3, in practice this is not a problem.

To summarize, if the variances of the system and observation noise are available, Kalman filtering is an attractive estimation technique. However, if these variances are not available, then Kalman filtering cannot be used. In the next section, we present a heuristic estimator that works even in the absence of knowledge about system and observation noise.

8. Fuzzy Estimation

This section presents the design of a fuzzy system that predicts the next value of a time series. Consider a scalar variable θ that assumes the sequence of values

$$\{\theta_k\} = \theta_1, \theta_2, \cdots, \theta_k$$

where

$$\theta_k = \theta_{k-1} + \omega_{k-1}$$

and ω_k (called the ‘system perturbation’) is a random variable from some unknown distribution.

Suppose that an observer sees a sequence of values

$$\tilde{\theta}_1, \tilde{\theta}_2, \ldots.., \tilde{\theta}_{k-1}$$

and wishes to use the sequence to estimate the current value of θ_k. We assume that the observed sequence is corrupted by some observation noise ξ, so that the observed values $\{\tilde{\theta}_k\}$ are not the actual values $\{\theta_k\}$, and

$$\tilde{\theta}_k = \theta_k + \xi_k$$

where ξ_k is another random variable from an unknown distribution.

Since the perturbation and noise variables can be stochastic, the exact value of θ_k cannot be determined. What is desired, instead, is $\hat{\theta}_k$, the predictor of θ_k, be optimal in some sense.

8.1. Assumptions

We model the parameter θ_k as the state variable of an unknown dynamical system. The sequence $\{\theta_k\}$ is then the sequence of states that the system assumes. We make three weak assumptions about the system dynamics. First, the time scale over which the system perturbations occur is assumed to be an order of magnitude slower than the corresponding time scale of the observation noise.

Second, we assume that system can span a spectrum ranging from 'steady' to 'noisy'. When it is steady, then the variance of the system perturbations is close to zero, and changes in $\{\tilde{\theta}_k\}$ are due to observation noise. When the system is noisy, $\{\theta_k\}$ changes, but with a time constant that is longer than the time constant of the observation noise. Finally, we assume that ξ is from a zero mean distribution.

Note that this approach is very general, since there are no assumptions about the exact distributions of ω and ξ. On the other hand, there is no guarantee that the resulting predictor is optimal: we only claim that the method is found to work well in practice.

8.2. Exponential averaging

The basis of this approach is the predictor given by:

$$\hat{\theta}_{k+1} = \alpha\hat{\theta}_k + (1-\alpha)\tilde{\theta}_k$$

The predictor is controlled by a parameter α, where α is the weight given to past history. The larger it is, the more weight past history has in relation to the last observation. The method is also called exponential averaging, since the predictor is the discrete convolution of the observed sequence with an exponential curve with a time constant α

$$\hat{\theta}_k = \sum_{i=0}^{k-1}(1-\alpha)\tilde{\theta}_i\alpha^{k-i-1} + \alpha^k\hat{\theta}_0$$

The exponential averaging technique is robust, and so it has been used in a number of applications. However, a major problem with the exponential averaging predictor is in the choice of α. While in principle, it can be determined by knowledge of the system and observation noise variances, in practice, the variances are unknown. It would be useful to automatically determine a 'good' value of α, and to be able to change this value online if the system behavior changes. Our approach uses fuzzy control to effect this tuning [24-26].

8.3. Fuzzy exponential averaging

Fuzzy exponential averaging is based on the assumption that a system can be thought of as belonging to a spectrum of behavior that ranges from 'steady' to 'noisy'. In a 'steady' system, the sequence $\{\theta_k\}$ is approximately constant, so that $\{\tilde{\theta}_k\}$ is affected mainly by observation noise. Then, α should be large, so that the past history is given more weight, and transient changes in $\tilde{\theta}$ are ignored.

In contrast, if the system is 'noisy', $\{\theta_k\}$ itself could vary considerably, and $\tilde{\theta}$ reflects changes both in θ_k and the observation noise. By choosing a lower value of α, the observer quickly tracks changes in θ_k, while ignoring past history which only provides old information.

While the choice of α in the extremal cases is simple, the choice for intermediate values along the spectrum is hard to make. We use a fuzzy controller to determine a value of α that gracefully responds to changes in system behavior. Thus, if the system moves along the noise spectrum, α adapts to the change, allowing us to obtain a good estimate of θ_k at all times. Moreover, if the observer does not know α *a priori*, the predictor automatically determines an appropriate value.

8.4. System identification

Since α is linked to the 'noise' in the system, how can the amount of 'noise' in the system be determined? Assume, for the moment, that the variance in ω is an order of magnitude larger than the variance in ξ. Given this assumption, if a system is 'steady', the exponential averaging predictor will usually be accurate, and prediction errors will be small. In this situation, α should be large. In contrast, if the system is 'noisy', then the exponential averaging predictor will have a large estimation error. This is because when the system noise is large, past history cannot predict the future. So, no matter what the value of α, it will usually have a large error. In that case, it is best to give little weight to past history by choosing a small value of α, so that the observer can track the changes in the system.

To summarize, the observation is that if the predictor error is large, then α should be small, and vice versa. Treating 'small' and 'large' as fuzzy linguistic variables [27], this is the basis for a fuzzy controller for the estimation of α.

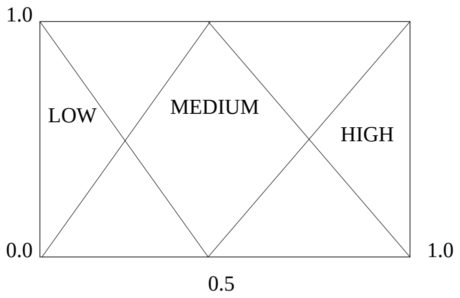
8.5. Fuzzy Controller

The controller implements three fuzzy laws:

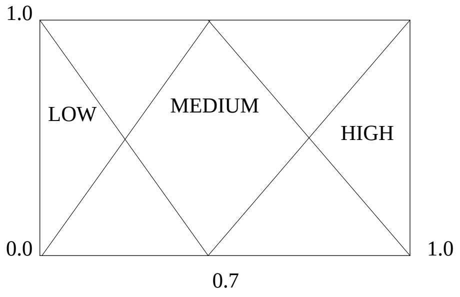
If error is low, then α is high
If error is medium, then α is medium
If error is high, then α is low

The linguistic variables 'low', 'medium' and 'high' for α and error are defined in Figure 3.
The input to the fuzzy controller is a value of error, and it outputs α in three steps. First, the error value is mapped to a membership in each of the fuzzy sets 'low', 'medium', and 'high' using the definition in Figure 3. Then, the control rules are used to determine the applicability of each outcome to the resultant control. Finally, the fuzzy set expressing the control is defuzzified using the centroid defuzzifier.

The error $|\tilde{\theta} - \hat{\theta}|$ is processed in two steps before it is input to the fuzzy system. First, it is converted to a proportional value, error $= \frac{|\tilde{\theta}_k - \hat{\theta}_k|}{\tilde{\theta}_k}$. Second, it is not a good idea to use the absolute error value directly, since spikes in $\tilde{\theta}_k$ can cause the error to be large, so that α drops to 0, and all past history is lost. So, the absolute error is smoothed using another exponential averager. The constant for this averager, β, is obtained from another fuzzy controller that links the change in error to the value of β. The idea is that if the change in error is large, then β should be large, so that spikes are ignored. Otherwise, β should be small. β and change in error are defined by the same linguistic variables, 'low' and 'high', and these are defined exactly like the corresponding variables for α. With these changes, the assumption



Linguistic variables to describe α



Linguistic variables to describe error

Figure 3: Definition of linguistic variables

that the variance in the observation noise is small can now be removed. The resulting system is shown in Figure 4.

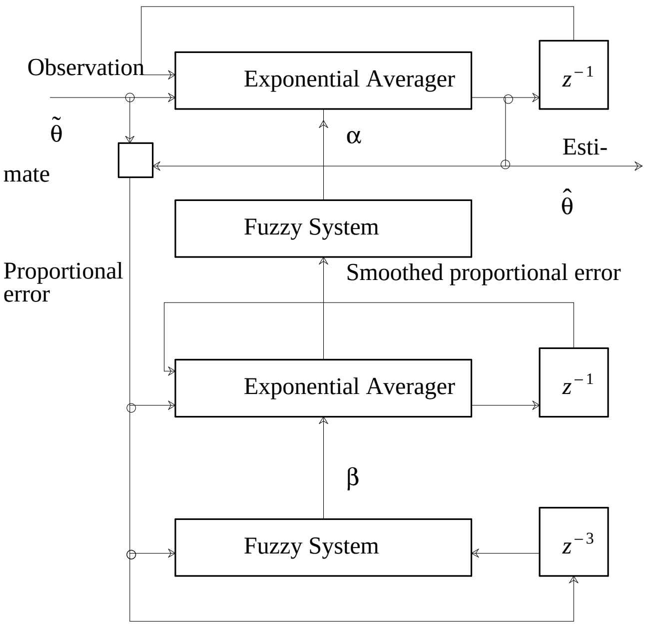


Figure 4: Fuzzy prediction system

Details of the prediction system and a performance analysis can be found in reference [28].

9. Using additional information

This section describes how the frequency of control can be increased by using information about the propagation delay. Note that $\hat{n}_b(k+1)$, the estimate for the number of packets in the bottleneck queue, plays a critical role in the control system. The controller tracks changes in $\hat{n}_b(k)$, and so it is necessary that $\hat{n}_b(k)$ be a good estimator of n_b. $\hat{n}_b(k)$ can be made more accurate if additional information from the network is available. One such piece of information is the value of the propagation delay.

The round-trip time of a packet has delays due to three causes:

- the propagation delay from the speed of light and processing at switches and interfaces
- the queueing delay at each switch, because previous packets from that conversation have not yet been serviced
- the phase delay, introduced when the first packet from a previously inactive conversation waits for the server to finish service of packets from other conversations

The propagation delay depends on the geographical spread of the network, and for WANs, it can be of the order of a few tens of milliseconds. The phase delay is roughly the same magnitude as the time it takes to send one packet each from all the conversations sharing a server, the round time. The queueing delay is of the order of several round times, since each packet in the queue takes one round time to get service. For future high speed networks, we expect the propagation and queueing delays to be of roughly the same magnitude, and the phase delay to be one order of magnitude smaller. Thus, if queueing delays can be avoided, the measured round-trip time will be approximately the propagation delay of the conversation.

An easy way to avoid queueing delays is to measure the round-trip time for the first packet of the first packet-pair. Since this packet has no queueing delays, we can estimate the propagation delay of the conversation from this packet's measured round trip time. Call this propagation delay R.

The value of R is useful, since the number of packets in the bottleneck queue at the beginning of epoch $k+1$, $n_b(k+1)$, can be estimated by the number of packets being transmitted ('in the pipeline') subtracted from the number of unacknowledged packets at the beginning of the epoch, $S(k)$. That is,

$$\hat{n}_b(k+1) = S(k) - R\hat{\mu}(k)$$

Since S, R and $\hat{\mu}(k)$ are known, this gives us another way of determining $\hat{n}_b(k+1)$. This can be used to update $\hat{n}_b(k+1)$ as an alternative to equation (2). The advantage of this approach is that equation (2) is more susceptible to parameter drift. That is, successive errors in $\hat{n}_b(k+1)$ can add up, so that $\hat{n}_b(k+1)$ could differ substantially from n_b. In the new scheme, this risk is considerably reduced: the only systematic error that could be made is in μ, and since this is frequent sampled, as well as smoothed by the fuzzy system, this is of smaller concern.

There is another substantial advantage to this approach: it enables control actions to be taken much faster than once per round trip time. This is explained in the following section.

9.1. Faster than once per RTT control

It is useful to take control actions as fast as possible so that the controller can react immediately to changes in the system. In the system described thus far, we limited ourselves to once per RTT control because this enables the simple relationship between $S(k)$ and $\lambda(k)$ given by equation (1). If control actions are taken faster than once per RTT, then the epoch size is smaller, and the relationship is no longer true. The new relationship is much more complicated, and it is easily shown that the state and input vectors must expand to include time delayed values of μ, λ and n_b. It is clear that the faster that control actions are required, the larger the state vector, and this complicates both the analysis and the control.

In contrast, with information about the propagation delay R, control can be done as quickly as once every packet-pair with no change to the length of the state vector. This is demonstrated below.

If control is done once every probe, then it is easier to work in continuous time. We also make the fluid approximation [29], so packet boundaries are ignored, and the data flow is like that of a fluid in a hydraulic system. This approximation is commonly used [30, 31], and both analysis [19] and our simulations show that the approximation is a close one, particularly when the bandwidth-delay product is large [20].

Let us assume that λ is held fixed for some duration J. Then,

$$n_b(t+J) = n_b(t) + \lambda(t)J - \mu(t)J \qquad 11$$

where μ is the average service rate in the time interval [t, t+J], and n_b is assumed to lie in the linear region of the space. Also, note that

$$n_b(t) = S(t) - R\mu(t) \qquad 12$$

The control goal is to have $n_b(t+J)$ be the setpoint value $B/2$. Hence,

$$n_b(t+J) = n_b(t) + \lambda(t)J - \mu(t)J = B/2 \qquad 13$$

So,

$$\lambda(t) = \frac{B/2 - S(t) + R\hat{\mu}(t) + J\hat{\mu}(t)}{J} \qquad 14$$

which is the control law. The stability of the system is easily determined. Note that $\dot{n}_b(t)$ is given by

$$\dot{n}_b(t) = \underset{\delta \to 0}{limit} \frac{n_b(t+\delta) - n_b(t)}{\delta} = \lambda(t) - \mu(t) \qquad 15$$

From equation (13),

$$\dot{n}_b = \frac{B/2 - n_b(t)}{J} \qquad 16$$

If we define the state of the system by

$$x = n_b(t) - B/2 \qquad 17$$

then, the equilibrium point is given by

$$x = 0 \qquad 18$$

and the state equation is

$$\dot{x} = \frac{-x}{J} \qquad 19$$

Clearly, the eigenvalue of the system is -1/J, and since J is positive, the system is both Lyapunov stable and asymptotically stable. In this system, J is the pole placement parameter, and plays exactly the same role as α in the discrete time system. When J is close 0, the eigenvalue of the system is close to $-\infty$ and the system will reach the equilibrium point rapidly. Larger values of J will cause the system to move to the equilibrium point more slowly. An intuitively satisfying choice of J is one round trip time, and this is easily estimated as $R + S(k)\mu(t)$. In practice, the values of R and $S(k)$ are known, and $\mu(t)$ is estimated by $\hat{\mu}$, which is the fuzzy predictor described earlier.

10. Practical Issues

This section considers two practical considerations: how to correct for parameter drift; and how to coordinate rate-based and window-based flow control.

10.1. Correcting for Parameter Drift

In any system with estimated parameters, there is a possibility that the estimators will drift away from the true value, and that this will not be detected. In our case, the estimate for the number of packets in the bottleneck buffer at time k, $\hat{n}_b(k)$, is computed from $\hat{n}_b(k-1)$ and the estimator $\hat{\mu}(k)$. If the estimators are incorrect, $\hat{n}_b(k)$ might drift away from $n_b(k)$. Hence, it is reasonable to require a correction for parameter drift.

Note that if $\lambda(k)$ is set to 0 for some amount of time, then n_b will decrease to 0. At this point, $\hat{n}_b$ can also be set to 0, and the system will resynchronize. In practice, the source sends a special pair and then sends no packets till the special pair is acknowledged. Since no data was sent after the pair, when acks are received, the source is sure that the bottleneck queue has gone to 0. It can now reset $\hat{n}_b$ and continue.

The penalty for implementing this correction is the loss of bandwidth for one round-trip-time. If a conversation lasts over many round trip times, then this loss may be insignificant over the lifetime of the conversation. Alternately, if a user sends data in bursts, and the conversation is idle between bursts, then the value of $\hat{n}_b$ can be resynchronized to 0 one RTT after the end of the transmission of a data burst.

10.2. The Role of Windows

Note that our control system does not give us any guarantees about the shape of the buffer size distribution $N(x)$. Hence, there is a non-zero probability of packet loss. In many applications, packet loss is undesirable. It requires endpoints to retransmit messages, and frequent retransmissions can lead to congestion. Thus, it is desirable to place a sharp cut-off on the right end of $N(x)$, or strictly speaking, to ensure that there are no packet arrivals when $n_b = B$. This can be arranged by having a window flow control algorithm operating simultaneously with the rate-based flow control algorithm described here.

In this scheme, the rate-based flow control provides us a 'good' operating point which is the setpoint that the user selects. In addition, the source has a limit on the number of packets it could have outstanding (the window size), and every server on its path reserves at least a window's worth of buffers for that conversation. This assures us that even if the system deviates from the setpoint, the system does not lose packets and possible congestive losses are completely avoided.

Note that by reserving buffers per conversation, we have introduced reservations into a network that we earlier claimed to be reservationless. However, our argument is that strict *bandwidth* reservation leads to a loss of statistical multiplexing. As long as no conversation is refused admission due to a lack of buffers, statistical multiplexing of bandwidth is not affected by buffer reservation, and the multiplexing gain is identical to that received in a network with no buffer reservations. Thus, with large cheap memories, we claim that it will be always be possible to reserve enough buffers so that there is no loss of statistical multiplexing.

To repeat, we use rate-based flow control to select an operating point, and window-based flow control as a conservative cut-off point. In this respect, we agree with Jain that the two forms of flow control are *not* diametrically opposed, but in fact can work together [32].

The choice of window size is critical. Using fixed sized windows is usually not possible in high speed networks, where the bandwidth-delay product, and hence the required window can be large (of the order of hundreds of kilobytes per conversation). In view of this, the adaptive window allocation scheme proposed by Hahne *et al* [33] is attractive. In their scheme, a conversation is allocated a flow control window that is always larger than the product of the allocated bandwidth at the bottleneck, and the round trip propagation delay. So, a conversation is never constrained by the size of the flow control window. A signaling scheme dynamically adjusts the window size in response to changes in the network state. We believe that their window-based flow control scheme is complementary to the rate-based flow control scheme proposed in this paper.

11. Limitations of the Our Approach

The main limitation of a control-theoretic approach is that it restricts the form of the system model. Since most control-theoretic results hold for linear systems, the system model must be cast in this form. This can be rather restrictive, and certain aspects of the system, such as the window flow control scheme, are not adequately modeled. Similarly, the standard noise assumptions are also restrictive and may not reflect the actual noise distribution in the target system.

These are mainly limitations of linear control. There is a growing body of literature dealing with non-linear control and one direction for future work would be to study non-linear models for flow control.

Another limitation of control theory is that for controller design, the network state should be observable. Since a FCFS server's state cannot be easily observed, it is hard to apply control theoretic principles to the control of FCFS networks. In contrast, RAS state can be probed using a packet pair, and so RAS networks are amenable to a formal treatment.

12. Related Work and Contributions

Several control theoretic approaches to flow control have been studied in the past. One body of work has considered the dynamics of a system where users update their sending rate either synchronously or asynchronously in response to measured round trip delays, or explicit congestion signals, for example in references [34-38]. These approaches typically assume Poisson sources, availability of global information, a simple flow update rule, and exponential servers. We do not make such assumptions. Further, they deal with the dynamics of the entire system, with the sending rate of all the users explicitly taken into account. In contrast, we consider a system with a single user, where the effects of the other users are considered as a system 'noise'. Also, in our approach, each user uses a rather complex flow update rule, based in part on fuzzy prediction, and so the analysis is not amenable to the simplistic approach of these authors.

Some control principles have been appealed to in work by Jain [39] and Jacobson [40], but the approaches of these authors is quite informal. Further, their control systems take multiple round trip times to react to a change in the system state. In contrast, the system in §9.1 can take control action multiple times per RTT. In a high bandwidth-delay product network, this is a significant advantage.

In recent work, Ko *et al* [41] have studied an almost identical problem, and have applied principles of predictive control to hop-by-hop flow control. However, they appeal primarily to intuitive heuristics, and do not use a formal control-theoretic model, and hence are not able to prove stability of their system. Further, we believe that our fuzzy scheme is a better way to predict service rate than their straightforward moving-average approach.

A control theoretic approach to individual optimal flow control was described originally by Agnew [29] and since extended by Filipiak [42] and Tipper *et al* [30]. In their approach, a conversation is modeled by a first order differential equation, using the fluid approximation. The modeling parameters are tuned so that, in the steady state, the solution of the differential equation and the solution of a corresponding queueing model agree. While we model the service rate at the bottleneck μ as a random walk, they assume that the service rate is a non-linear function of the queue length, so that $\mu = G(n_b)$, where $G(.)$ is some nonlinear function. This is not true for a RAS, where the service rate is independent of the queue length. Hence, we cannot apply their techniques to our problem.

Vakil, Hsiao and Lazar [43] have used a control-theoretic approach to optimal flow control in double-bus TDMA local-area integrated voice/data networks. However, they assume exponential FCFS servers, and, since the network is not geographically dispersed, propagation delays are ignored. Their modeling of the service rate μ is as a random variable as opposed to a random walk, and though they propose the use of recursive minimum mean squared error filters to estimate system state, the bulk of the results assume complete information about the network state. Vakil and Lazar [44] have considered the the design of optimal traffic filters when the state is not fully observable, but the filters are specialized for voice traffic.

Robertazzi and Lazar [45] and Hsiao and Lazar [46] have shown that under a variety of conditions, the optimal flow control for a Jacksonian network with Poisson traffic is *bang-bang* (approximated by a window scheme). It is not clear that this result holds when their strong assumptions are removed.

In summary, we feel that our approach is substantially different from those in the literature. Our use of a packet-pair to estimate the system state is unique, and this estimation is critical in enabling the control scheme. We have described two provably stable rate-based flow control schemes as well as a novel estimation scheme using fuzzy logic. Some practical concerns in implementing the scheme have also been addressed.

The control law presented in §9.1 has been extensively simulated in a number of scenarios [20]. While considerations of space do not allow us to present detailed results in this paper, they can be summarized as

- The performance of the flow control with Fair Queueing servers in the benchmark suite described in reference [10] is comparable to that of the DECbit scheme [47], but without any need for switches to set bits.
- The flow control algorithm responds quickly and cleanly to changes in network state.
- Unlike some current flow control algorithms (DECbit and Jacobson's modifications to 4.3 BSD [40, 47]), the system behaves extraordinarily well in situations where the bandwidth-delay product is large, even if the cross traffic is misbehaved or bursty.
- Implementation and tuning of the algorithm is straightforward, unlike the complex and ad-hoc controls in current flow control algorithms.
- Even in complicated scenarios, the dynamics are simple to understand and manage: in contrast the dynamics of

Jacobson’s algorithm are messy and only partially understood [48].

In conclusion, we believe that our decision to use a formal control-theoretic approach in the design of a flow control algorithm has been a success. Our algorithm behaves well even under great stress, and, more importantly, it is simple to implement and tune. These are not fortuitous, rather, they reflect the theoretical underpinnings of the approach.

13. Future Work

This paper makes several simplifications and assumptions. It would be useful to measure real networks to see how far theory and practice agree. We plan to make such measurements in the XUNET II experimental high speed network testbed [49]. Other possible extensions are to design a minimum variance controller and a non-linear controller.

14. Acknowledgements

The use of a noise variable to model the bottleneck service rate was suggested by G. Srinivasan. The fact that the system noise distribution is not symmetric when μ is close to 0 was pointed out by Prof. J. Walrand. P.S. Khedkar gave valuable comments on the fuzzy controller. Prof. D. Ferrari was a source of constant encouragement and keen questioning. The helpful advice and criticism of Dr. R.P. Singh, Dr. D. Mitra, Prof. M. Tomizuka, Prof. P.P. Varaiya and the anonymous refrees on several aspects of the work are much appreciated.

15. References

1. D. Ferrari, Client Requirements for Real-Time Communications Services, *IEEE Communications Magazine 28*, 11 (November 1990).
2. C. R. Kalmanek, H. Kanakia and S. Keshav, Rate Controlled Servers for Very High Speed Networks, *Proc. Globecom 1990*, December 1990, 300.3.1-300.3.9.
3. D. Ferrari and D. Verma, A Scheme for Real-Time Channel Establishment in Wide-Area Networks, *IEEE J. on Selected Areas in Communications*, April 1990.
4. S. J. Golestani, A Stop-and-Go Queueing Framework for Congestion Management, *Proc. ACM SigComm 1990*, September 1990, 8-18.
5. L. Zhang, A New Architecture for Packet Switching Network Protocols, *PhD thesis*, Massachusetts Institute of Technology, July 1989.
6. S. Keshav, A. K. Agrawala and S. Singh, Design and Analysis of a Flow Control Algorithm for a Network of Rate Allocating Servers, in *Protocols for High Speed Networks II*, Elsevier Science Publishers/North-Holland, April 1991.
7. S. Singh, A. K. Agrawala and S. Keshav, Deterministic Analysis of Flow and Congestion Control Policies in Virtual Circuits, Tech. Rpt.-2490, University of Maryland, June 1990.
8. S. Keshav, The Packet Pair Flow Control Protocol, Tech. Rpt. 91-028, International Comp. Sci. Institute , Berkeley, CA 94704, May 1991.
9. A. Greenberg and N. Madras, How Fair is Fair Queueing?, *Proc. Performance 90*, 1990.
10. A. Demers, S. Keshav and S. Shenker, Analysis and Simulation of a Fair Queueing Algorithm, *Journal of Internetworking Research and Experience*, September 1990, 3-26;. also Proc. ACM SigComm, Sept. 1989, pp 1-12..
11. H. Zhang and S. Keshav, Comparison of Rate-Based Service Disciplines, *Proc. ACM SigComm 1991*, September 1991. also International Comp. Sci. Institute Tech. Rpt. 91-024, Berkeley, CA..
12. S. Tripathi and A. Duda, Time-dependent Analysis of Queueing Systems, *INFOR 24*, 3 (1978), 334-346.
13. J. G. Waclawsky, Window Dynamics, *PhD Thesis*, University of Maryland, College Park, May 1990.
14. J. G. Waclawsky and A. K. Agrawala, Dynamic Behavior of Data Flow within Virtual Circuits, Comp. Sci.-Tech. Rpt.-2250, University of Maryland , May 1989.
15. B. D. O. Anderson and J. B. Moore, *Optimal Filtering*, Prentice Hall, 1979.
16. K. K. Sabnani and A. N. Netravali, A High Speed Transport Protocol for Datagram/Virtual Circuit Networks, *Proc. ACM SigComm 1989*, September 1989, 146-157.
17. B. D. O. Anderson and J. B. Moore, *Linear Quadratic Methods*, Prentice Hall, 1990.
18. D. Mitra and J. B. Seery, Dynamic Adaptive Windows for High Speed Data Networks: Theory and Simulations , *Proc. ACM SigComm 1990*, September 1990, 30-40.
19. D. Mitra, Asymptotically Optimal Design of Congestion Control for High Speed Data Networks, *To Appear in IEEE Trans. on Communications*, 1991.
20. S. Keshav, Congestion Control in Computer Networks, *PhD thesis* , University of California, Berkeley , August 1991.
21. A. E. Ekberg, D. T. Luan and D. M. Lucantoni, Bandwidth Management: A Congestion Control Strategy for Broadband Packet Networks: Characterizing the Throughput-Burstiness Filter, *Proc. ITC Specialist Seminar*, Adelaide, 1989, paper no. 4.4.
22. K. Ogata, Discrete Time Control Systems, *Prentice Hall*, 1987.
23. G. C. Goodwin and K. S. Sin, *Adaptive Filtering Prediction and Control*, Prentice Hall, 1984.
24. H. J. Zimmerman, in *Fuzzy Set Theory and its Applications*, Kluwer Academic Publishers, 1985.
25. L. A. Zadeh, Outline of a New Approach to the Analysis of Complex Systems and Decision Processes, *IEEE Trans. on Systems, Man and Cybernetics*, 1973, 28-44.
26. G. Langari, Analysis and Design of Fuzzy Control Systems, *PhD thesis (in preparation)*, University of California, Berkeley, 1991.
27. L. A. Zadeh, Fuzzy Sets, *Journal of Information and Control 8* (1965), 338-353.
28. P. S. Khedkar and S. Keshav, Fuzzy Prediction of Time-series, Proc. IEEE Conference on Fuzzy Systems-92, March 1992.
29. C. Agnew, Dynamic Modeling and Control of Congestion-prone Systems, *Operations Research 24*, 3 (1976), 400-419.
30. D. Tipper and M. K. Sundareshan, Numerical Methods for Modeling Computer Networks under Nonstationary Conditions, *JSAC 8*, 9 (December 1990).

31. J. Bolot, Dynamical Behavior of Rate-Based Flow Control Mechanisms, Comp. Sci.-Tech. Rpt. 2279.1, University of Maryland, October 1989.

32. R. Jain, Myths About Congestion Management in High-Speed Networks, Technical Report-726, Digital Equipment Corporation, October 1990.

33. E. L. Hahne, C. R. Kalmanek and S. P. Morgan, Fairness and Congestion Control on a Large ATM Data Network with Dynamically Adjustable Windows, *13th International Teletraffic Congress*, Copenhagen , June 1991.

34. K. Bharath-Kumar and J. M. Jaffe, A New Approach to Performance-Oriented Flow Control, *IEEE Trans. on Communication COM-29*, 4 (April 1981), 427-435.

35. S. Shenker, A Theoretical Analysis of Feedback Flow Control, *Proc. ACM SigComm 1990*, September 1990, 156-165.

36. C. Douligeris and R. Majumdar, User Optimal Flow Control in an Integrated Environment, *Proc. of the Indo-US Workshop on Systems and Signals*, January 1988. Bangalore, India.

37. A. D. Bovopoulos and A. A. Lazar, Asynchronous Algorithms for Optimal Flow Control of BCMP Networks, Tech. Rpt. WUCS-89-10, Washington University, St. Louis, MO, February 1989.

38. A. D. Bovopoulos and A. A. Lazar, Decentralized Algorithms for Optimal Flow Control, *Proc. 25th Allerton Conference on Communications Control and Computing*, October 1987. University of Illinois, Urbana-Champaign.

39. K. K. Ramakrishnan and R. Jain, Congestion avoidance in Computer Networks with a Connectionless Network Layer - Part II - An Explicit Binary Feedback Scheme, Technical Report-508, Digital Equipment Corporation, April 1987.

40. V. Jacobson, Congestion Avoidance and Control, *Proc. ACM SigComm 1988*, August 1988, 314-329.

41. K. Ko, P. P. Mishra and S. K. Tripathi, Predictive Congestion Control in High-Speed Wide-Area Networks, in *Protocols for High Speed Networks II*, Elsevier Science Publishers/North-Holland, April 1991.

42. J. Filipiak, *Modelling and Control of Dynamic Flows in Communication Networks*, Springer-Verlag, 1988.

43. F. Vakil, M. Hsiao and A. A. Lazar, Flow Control in Integrated Local Area Networks, Vol. 7, 1987.

44. F. Vakil and A. A. Lazar, Flow Control Protocols for Integrated Networks with Partially Observed Traffic, *IEEE Transactions on Automatic Control 32*, 1 (1987), 2-14.

45. T. G. Robertazzi and A. A. Lazar, On the Modeling and Optimal Flow Control of the Jacksonian Network, *Performance Evaluation 5* (1985), 29-43.

46. M. Hsiao and A. A. Lazar, Optimal Flow Control of Multi-Class Queueing Networks with Partial Information, *IEEE Transactions on Automatic Control 35*, 7 (July 1990), 855-860.

47. K. K. Ramakrishnan and R. Jain, A Binary Feedback Scheme for Congestion Avoidance in Computer Networks, *ACM ACM Trans. on Comp. Sys. 8*, 2 (May 1990), 158-181.

48. L. Zhang, S. Shenker and D. D. Clark, Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic, *Proc. ACM SigComm 1991*, September 1991.

49. C. R. Kalmanek, Xunet 2: A Nationwide Testbed in High-Speed Networking, *Comp. Sci. Tech. Rpt.*, March 1991, AT&T Bell Labs, 600 Mountain Ave. Murray Hill, NJ 07974.