Chapter 5: A Control-Theoretic Approach to Flow Control

5.1. Introduction

If networks implement Fair Queueing at each server, then new flow control protocols are enabled. In Chapter 4, we presented the 2P protocol, which was enabled in this way. The 2P protocol is strongly motivated by a deterministic model for the network. Since, in practice, this assumption cannot always be justified, this chapter presents the design of a flow control protocol that works well even in the presence of stochastic changes in the network state. We use 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.

A control-theoretic approach to flow control requires that changes in the network state be observable. We have shown in Chapter 4 that it is possible to measure network state easily if the servers at the output queues of the switches do Fair Queueing, and the transport protocol uses the Packet-Pair probing technique. Thus, in this chapter, we will make the assumption that the queue service discipline is FQ and that the 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 chapter is laid out as follows. We first present a stochastic model for FQ networks (§5.2). Next, we use the Packet-Pair state probing technique as the basis for the design of a stable rate-based flow control scheme (§5.3). A problem with non-linearity in the system is discussed in §5.4. We present a Kalman state estimator in §5.5. However, this estimator is impractical, and so we have designed a novel estimation scheme based on fuzzy logic (§5.6). A technique to increase the frequency of control based on additional information from the system is presented in §5.7, and this serves as the basis for a new, stable control law. Practical implementation issues are discussed in §5.8, and these include correcting for parameter drift, and interaction with window flow control. We conclude with some remarks on the limitations of the approach (§5.9) and a review of related work (§5.10).

5.2. Stochastic model

In the deterministic model of Chapter 4, μ, the service rate at the bottleneck of a conversation, 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 the value of 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 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 should be 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 assumption is valid. However, the gaussian assumption is harder to justify. As mentioned in (2), 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 (§5.6) that does not require any of these assumptions

Note that the queueing-theoretic approach to modeling μ would be to define the density function of μ, say $G(\mu)$, which would have to be supplied by the system administrator. Then, system performance would be given by expectations on the distribution. For example, a metric $X(\mu)$, that depends on the service rate μ would be described by $E(X(\mu)) = X(\hat{\mu})$, where $\hat{\mu}$, the average value of μ is given by $\hat{\mu} = \int \mu G(\mu)\, d\mu$. Now, if $G(\mu)$ is unknown, so is $\hat{\mu}$. Further, $E(X)$ depends on $\hat{\mu}$, an asymptotic average. In contrast, we explicitly model the dynamics of μ and so our control scheme can depend on the currently measured value of μ, instead of an asymptotic time average.

5.3. 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 (3). 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 of 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 a subsequent section. We first discuss our assumptions and a few preliminary considerations.

5.3.1. 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, this may vary with time; in our work, 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 (FQ 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.

Queueing-theoretic choice of setpoint

In recent work, Mitra et al (96,97) have studied asymptotically optimal choices of window size for window based flow control, when the scheduling discipline at a switch is either FCFS or processor sharing (PS). With some caveats as to its generality, their approach is complementary to ours, and can provide insight into the choice of setpoint.

The basis for their study is product-form queueing network theory, where asymptotic network behavior is studied as the bandwidth-delay product tends to infinity. In this regime, the analysis of virtual circuit dynamics indicates that the network behaves almost deterministically (which reinforces our earlier claim). Further, if a source wishes to optimize the throughput to queueing delay ratio, or power, then the optimal window size K is given by

$$K = \lambda + \alpha\sqrt{\lambda}$$

where λ is the bandwidth delay product, and α is approximately $\frac{1}{2\sqrt{M}}$, M being the number of hops over which the circuit sends data. Thus, the optimal choice of the setpoint is simply $\alpha\sqrt{\lambda}$. Since α can be precomputed for a circuit, and changes in λ can be determined using the packet-pair protocol, the setpoint can be dynamically adjusted to deliver maximum power using the earlier control model. Thus, the two theoretical approaches can be used in conjunction to determine the bandwidth delay product, the optimal setpoint, and, also, a mechanism to keep the system at the optimal setpoint.

However, there are several difficulties with the queueing-theoretic approach that limit its generality. First, the analysis assumes that the scheduling discipline is either FCFS or PS. While Fair Queueing can be approximated by Head-of-Line Processor Sharing (50), approximating it by PS does not seem to be reasonable. Second, the analysis assumes that the cross traffic is strictly Poisson. Since there is no empirical evidence that the assumption is valid, it may be inaccurate. Third, the packet service time is assumed to be exponentially distributed. This assumption is almost certainly incorrect, since numerous studies have indicated that the packet size distribution is usually strongly multi-modal (usually bi-modal), so that the service times will follow a similar distribution (13,51).

Nevertheless, even with these caveats, the queueing approach is a strong basis to justify our intuitions, and, to a first approximation, these results can be used to determine the choice of the setpoint in the control system.

5.3.2. Frequency of control

We initially restrict control actions to only once per round trip time (RTT) (this restriction is removed in §5.7). For the purpose of exposition, we divide time into *epochs* of length RTT (= R + queueing delays) (Figure 5.1). 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, a control action is taken at the end of every epoch.

5.3.3. 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 the 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, which 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 §5.7.

5.3.4. 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 §5.7.

Figure 5.1 shows the time diagram for the control. The vertical axis on the left represents the time of the source, and the axis on the right that of 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 those at 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.

We now make a few observations regarding Figure 5.1. 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 epoch, all the packets sent in epoch $k-1$ have been acknowledged. So, the only unacknowledged packets are those sent during 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,

$$S(k) = \lambda(k)RTT(k) \tag{5.1}$$

The number of the conversation's 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 §5.5.6). Since $\lambda(k)$ packets were sent in, and $\mu(k)RTT(k)$ packets were serviced in this interval, we have

$$n_b(k+1) = n_b(k) + \lambda(k)RTT(k) - \mu(k)RTT(k) \tag{5.2}$$

Equations (5.1) and (5.2) are the fundamental equations in this analysis. They can be combined to give

$$n_b(k+1) = n_b(k) + S(k) - \mu(k)RTT(k) \tag{5.3}$$

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

$$n_b(k+2) = n_b(k+1) + (\lambda(k+1) - \mu(k+1))RTT(k+1) \tag{5.4}$$

From (5.3) and (5.4):

$$\begin{aligned} n_b(k+2) = n_b(k) + S(k) - \mu(k)RTT(k) + \\ \lambda(k+1)RTT(k+1) - \mu(k+1)RTT(k+1) \end{aligned} \tag{5.5}$$

The control should set this to $B/2$. So, set (5.5) to $B/2$, and obtain $\lambda(k+1)$.

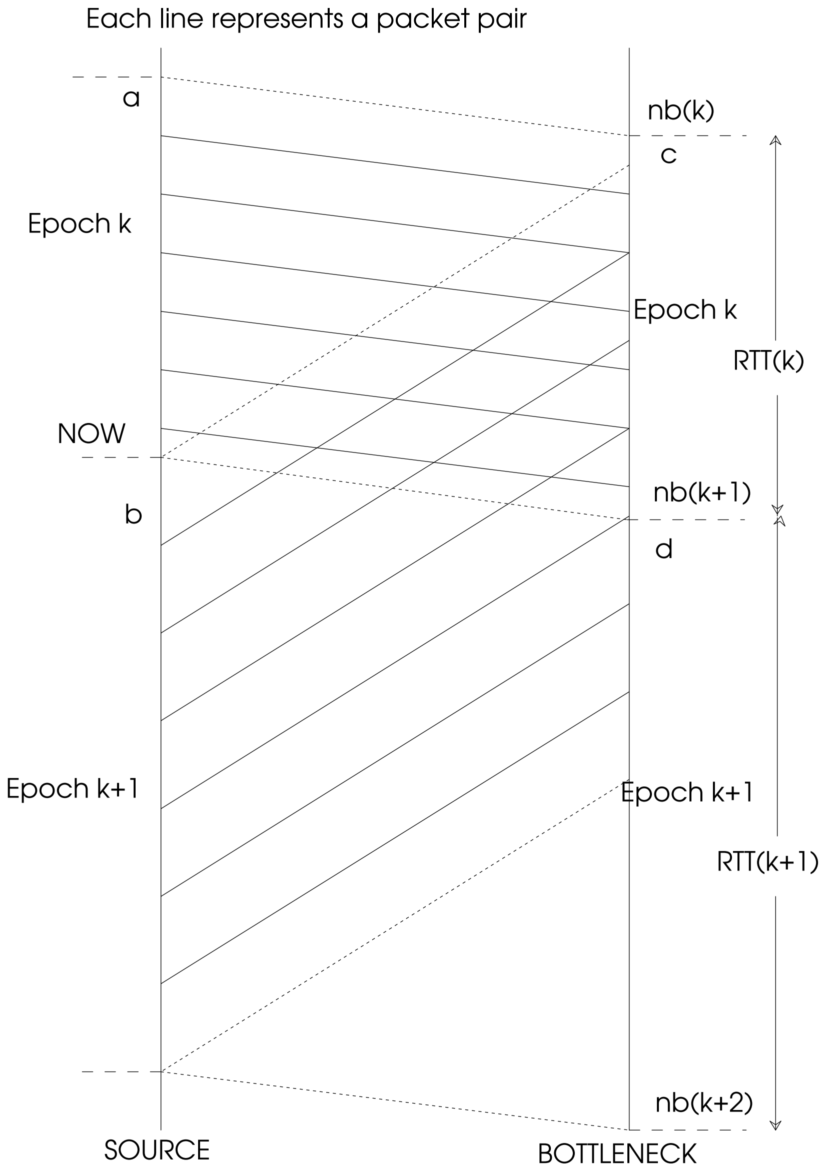


Figure 5.1: Time scale of control

$$n_b(k+2) = B/2 = n_b(k) + S(k) - \mu(k)RTT(k) + (\lambda(k+1) - \mu(k+1))RTT(k+1) \quad 5.6$$

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

$$\lambda(k+1) = \frac{1}{RTT(k+1)} (B/2 - n_b(k) - S(k) + \mu(k)RTT(k) + \mu(k+1)RTT(k+1)) \quad 5.7$$

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

$$\lambda(k+1) = \frac{1}{\hat{RTT}(k+1)} (B/2 - \hat{n}_b(k) - S(k) + \hat{\mu}(k)RTT(k) + \hat{\mu}(k+1)\hat{RTT}(k+1)) \quad 5.8$$

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 $\hat{RTT}(k+1)$ to $RTT(k)$. This gives us:

$$\lambda(k+1) = \frac{1}{RTT(k)}(B/2 - \hat{n}_b(k) - S(k) + 2\hat{\mu}(k)RTT(k)) \qquad 5.9$$

This is the control law. The control always tries to obtain a queue length in the bottleneck equal 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 5 and 6.

5.3.5. Stability analysis

The state equation is given by (5.2)

$$n_b(k+1) = n_b(k) + \lambda(k)RTT(k) - \mu(k)RTT(k) \qquad 5.10$$

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 (5.2) instead (which is just one step forward in time). This gives

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

Substituting (5.8) in (5.10), we 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 approximation, 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

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

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

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

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

This leads to a characteristic equation

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

so that the roots are

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

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

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

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

$$\Rightarrow \frac{1}{\sqrt{\alpha}} > 1 \Rightarrow \alpha < 1$$

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. Similar studies are mentioned in (30).

5.4. System non-linearity

This section discusses a non-linearity in the system, and how it can be accounted for in the analysis. The state equation (5.9) is correct when $n_b(k+1)$ lies in the range 0 to B. Since the system is physically incapable of having less than zero 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:

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

The introduction of the inequalities 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 inside 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 ((102) page 558).

However, this is only 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, since n_b can never be less than 0, and the window flow control protocol ensures that it never exceeds B, 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.

5.5. Kalman state estimation

A practical scheme is presented in §5.6. Having derived the control law, and proved its stability, we now need to determine stable estimators for the system state. §5.5 presents a Kalman state estimator, and shows that Kalman estimation is impractical. We choose to use Kalman estimation, since it is a well known and robust technique (49). Before the technique is applied, a state-space description of the system is necessary.

5.5.1. State space description

We will use the standard linear stochastic state equation given by

$$\mathbf{x}(k+1) = \mathbf{G}\mathbf{x}(k) + \mathbf{H}\mathbf{u}(k) + \nu_1(k)$$
$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) + \nu_2(k)$$

$\mathbf{x}$, $\mathbf{u}$ and $\mathbf{y}$ are the state, input and output vectors of sizes n, m, and r, respectively. $\mathbf{G}$ is the nxn state matrix, $\mathbf{H}$ is an nxm matrix, and $\mathbf{C}$ 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, $\mathbf{u}$ 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, the source 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 $y(k)$ is to be derived from the state vector $\mathbf{x}$ 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 bottleneck's 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 $\mathbf{G}$, $\mathbf{H}$, ν_1, ν_2 and $\mathbf{C}$ 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 kth 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.

5.5.2. 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 (49). 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 $\mathbf{S} = \mathbf{0}$. However, we still need to know $E(\omega^2)$ and $E(\nu_2^{\,2})$.

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 $E(\omega^2)$ and $E(\nu_2^{\,2})$ 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 values of the 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 (49) (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 Riccati 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 generally 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.

5.6. 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.

5.6.1. 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.

5.6.2. 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, these variances are unknown. It would be useful to automatically determine a 'good' value of α, and to be able to change this value on-line if the system behavior changes. Our approach uses fuzzy control to effect this tuning (82, 152, 158).

5.6.3. Fuzzy exponential averaging

Fuzzy exponential averaging is based on the heuristic that a system can be thought of as belonging to a spectrum of behaviors that ranges from 'steady' to 'noisy'. In a 'steady' system ($\omega \ll \xi$), 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' ($\omega \approx \xi$ or $\omega > \xi$), $\{\theta_k\}$ 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.

5.6.4. 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 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.

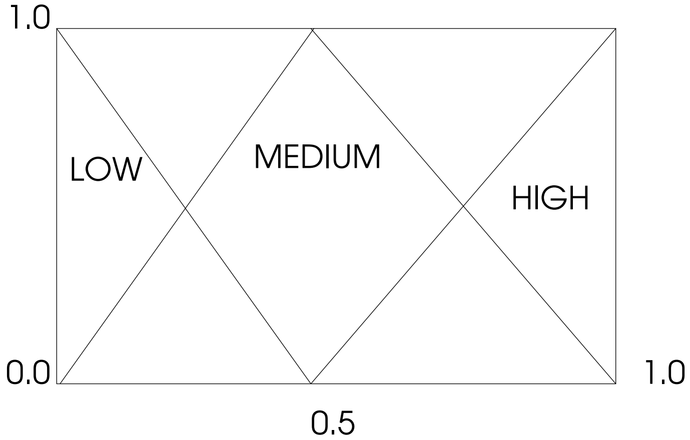
To summarize, we have observed that, if the predictor error is large, then α should be small, and vice versa. Treating 'small' and 'large' as fuzzy linguistic variables (151), we can build a fuzzy controller for the estimation of α.

5.6.5. Fuzzy controller

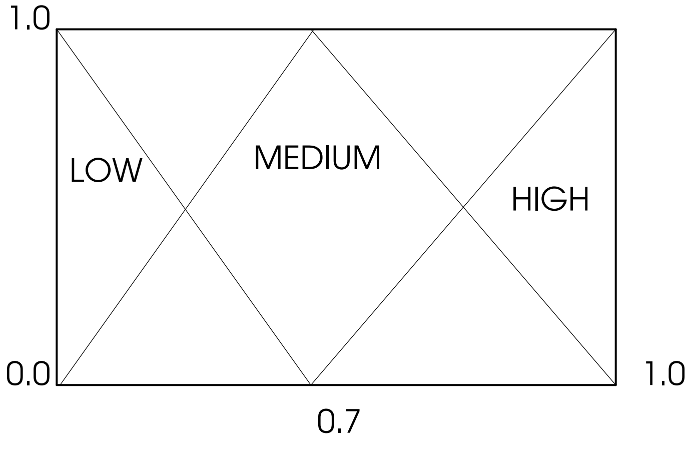
The controller implements three fuzzy laws:

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 5.2.



Linguistic variables to describe α



Linguistic variables to describe the error

Figure 5.2: Definition of linguistic variables

The input to the fuzzy controller is a value of the error, and the controller 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 5.3. Then, the control rules are used to determine the applicability of each outcome to the resultant control. Finally, the fuzzy set representing the control is defuzzified using a 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 relative value given by error $= \frac{|\tilde{\theta}_k - \hat{\theta}_k|}{\tilde{\theta}_k}$. It is not a good idea to use the relative error value directly, since spikes in $\tilde{\theta}_k$ can cause the error to be large, α would drop to 0, and all past history would be lost. So, in the second step, the relative error is smoothed using another exponential averager. The constant for this averager, β, is obtained from another fuzzy controller that links the change in the 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 the change in error are defined by the same linguistic variables, 'low', 'medium' and 'high', and these are defined exactly like the corresponding variables for α. With these changes, the assumption that the variance in the observation noise is small can now be removed. The resulting system is shown in Figure 5.3.

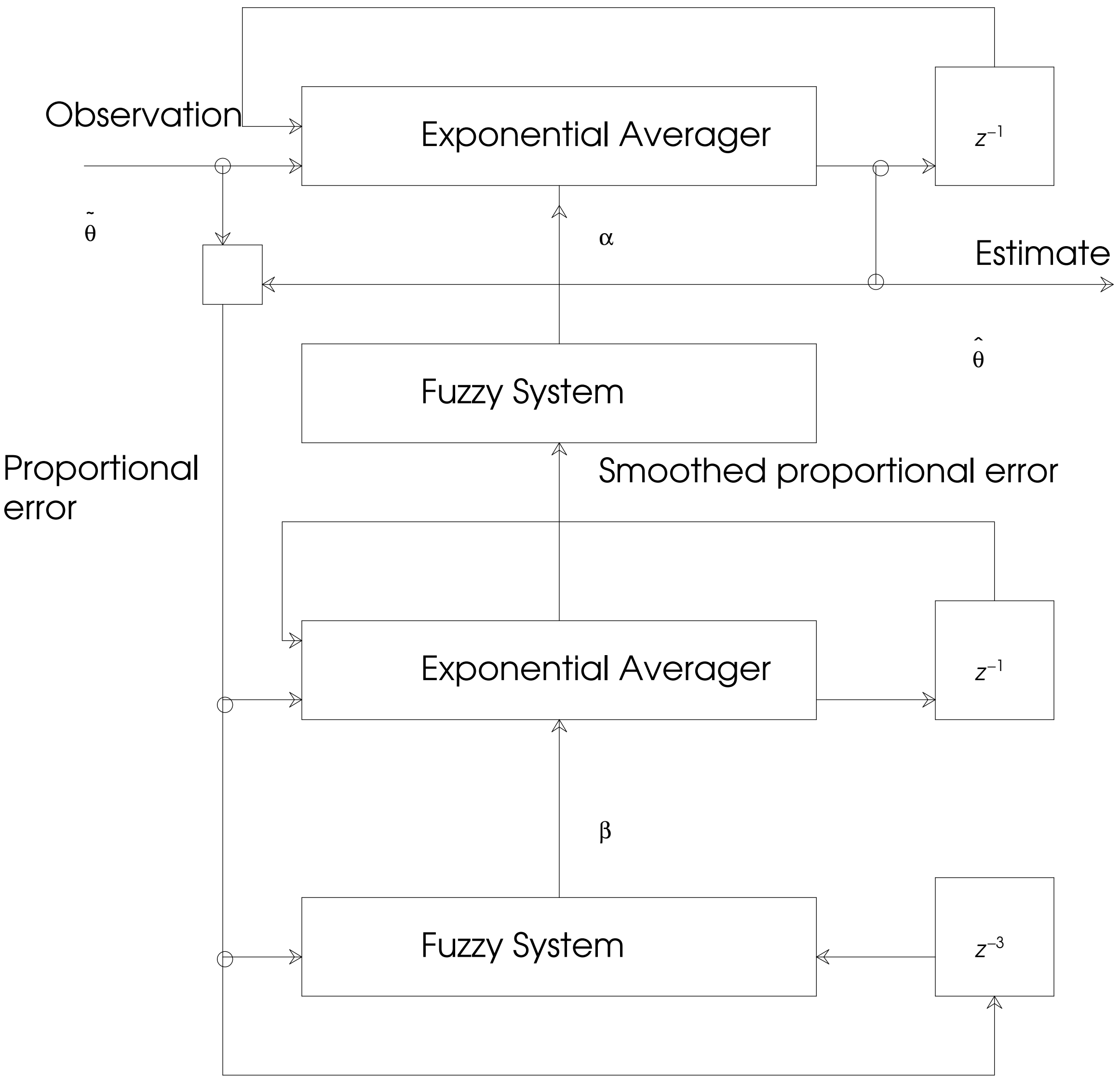


Figure 5.3: Fuzzy prediction system

Details of the prediction system, and a performance analysis can be found in reference (75).

5.7. 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 includes delays due to three causes:

- the propagation delay due to the finiteness of the speed of light and the 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 process 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 by the probe packet, 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 (though it has a component due to phase delay). 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.

5.7.1. 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 allows us to use 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 that 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 the 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.

We will work in continuous time, since this makes the analysis easier. We also make the fluid approximation (1), so packet boundaries are ignored, and the data flow is like that of a fluid in a hydraulic system. This approximation is commonly used (8, 133), and both analysis (97) and simulations show that the approximation is a close one.

Let us assume that λ, the sending rate, is held fixed for some duration J, starting from time t. Then,

$$n_b(t+J) = n_b(t) + \lambda(t)J - \mu(t)J \qquad 5.11$$

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

$$n_b(t) = S(t) - R\mu(t) \qquad 5.12$$

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

$$n_b(t+J) = n_b(t) + \lambda(t)J - \mu(t)J = B/2 \qquad 5.13$$

So,

$$\lambda(t) = \frac{B/2 - S(t) + R\hat{\mu}(t) + J\hat{\mu}(t)}{J} \qquad 5.14$$

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

$$\dot{n}_b(t) = \lim_{\delta \to 0} \frac{n_b(t+\delta) - n_b(t)}{\delta} = \lambda(t) - \mu(t) \qquad 5.15$$

From equation (5.13),

$$\dot{n}_b = \frac{B/2 - n_b(t)}{J} \qquad 5.16$$

If we define the state of the system by

$$x = n_b(t) - B/2 \qquad 5.17$$

then the equilibrium point is given by

$$x = 0 \qquad 5.18$$

and the state equation is

$$\dot{x} = \frac{-x}{J} \qquad 5.19$$

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.

5.8. Practical issues

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

5.8.1. 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 from 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.

5.8.2. 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 (67).

The choice of window size is critical. Using fixed size 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* (54) is attractive. In that 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 chapter.

5.9. Limitations of the control-theoretic 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 the 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 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, FQ state can be probed using a packet pair, and so FQ networks are amenable to a formal treatment.

5.10. 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 (6, 9, 10, 27, 127). 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, and take into account the sending rate of all the users explicitly. 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 (116) and Jacobson (63), but the approaches of these authors are 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 §5.9.1 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* (79) 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; hence they are not able to prove the stability of their system. Further, we believe that our fuzzy scheme is a better way to predict service rates than their straightforward moving-average approach.

A control-theoretic approach to individual optimal flow control was described originally by Agnew (1), and since extended by Filipiak (38) and Tipper *et al* (133). 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 global queue length (over all conversations), so that $\mu = G(n_b)$, where $G(.)$ is some non-linear function. This is not true for a FQ server, where the service rate is independent of the queue length. Hence, we cannot apply their techniques to our problem.

Vakil, Hsiao and Lazar (137) 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 instead of 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 (138) have considered the design of optimal traffic filters when the state is not fully observable, but the filters are specialized for voice traffic.

Robertazzi and Lazar (119) and Hsiao and Lazar (60) 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 §5.9.1 has been extensively simulated in a number of scenarios and the results are presented in Chapter 6. The results can be summarized as

- The performance of the flow control with Fair Queueing servers in the benchmark suite described in reference (23) is comparable to that of the DECbit scheme (117), 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 TCP (63, 117)), 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 (156).

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.

5.11. Future work

This chapter 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 (69). Other possible extensions are to design a minimum variance controller and a non-linear controller.

5.12. Appendix 5.A - Steady state

As a sanity check, consider the steady state where μ does not change. We prove that in this case λ is set to μ, and that the number of packets in the bottleneck will be $B/2$.

In the steady state, any sensible estimator $\hat{\mu}(k)$ will converge, so that $\hat{\mu}(k) = \hat{\mu}(k+1) = \mu$. We will assume that we are starting at time 0, that $n_b(0) = 0$ and that $\alpha = 1$.

In the steady state, the state equation (2) becomes

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

We assume that the estimators converge to the correct value, since there is no stochastic variation in the system. Hence,

$$\hat{RTT} = RTT \text{ for } all\ k$$

This makes the control law

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

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

The first packet pair estimates μ, and, for simplicity, we assume that this is exactly correct. Hence,

$$\lambda(1) = 2\mu - \mu + \frac{1}{RTT(1)}(B/2)$$

$$\lambda(1) = \mu + \frac{B/2}{RTT(1)}$$

The buffer at the end of epoch 1 is given by

$$n_b(1) = 0 + (\mu + \frac{B/2}{RTT(1)})RTT(1) - \mu RTT(1)$$

$$n_b(1) = B/2$$

To check further, at time 2, we get

$$\lambda(2) = 2\mu - \mu + \frac{1}{RTT(2)}(B/2 - B/2)$$

and, since $\hat{n}_b(k+1)(1) = B/2$.

$$\Rightarrow \lambda(2) = \mu$$

And,

$$n_b(2) = B/2 + \mu RTT(2) - \lambda(2)RTT(2)$$

$$n_b(2) = B/2 \qquad 5.A1$$

So, the buffer is B/2 at time 2. Let us see what $\lambda(3)$ is when $\lambda(2)$ is μ.

$$\lambda(3) = 2\mu - \mu + \frac{1}{RTT(3)}(B/2 - B/2)$$

$$\lambda(3) = \mu$$

So, if $\lambda(k) = \mu$ and $n_b(k) = B/2$, $\lambda(k+1) = \mu$, and λ is fixed from time 3 onwards. From (5.A1) we see that if $\lambda = \mu$ and $n_b(k) = B/2$ then $n_b(k+1) = B/2$. Thus, both the recurrences reach the stable point at $(\mu, B/2)$ at time 3. Thus, if the system is steady, so is the control. This is reassuring.

5.13. Appendix 5.B - Linear quadratic gaussian optimal control

This section considers an approach to optimal control, and shows that it is infeasible for our system. We consider optimal control of the flow control system described in §5.5. One optimal control technique that is popular in the control theoretic literature is Linear Quadratic Gaussian (LQG) control. This technique provides optimal control for a linear system where the goal is to optimize a quadratic metric in the presence of gaussian noise. We follow the description of LQG presented in (102) (pg. 835).

The quadratic performance index to minimize is given by

$$J = \mathbf{x}^T \mathbf{Q}\, \mathbf{x}$$

where the state vector **x** is modified so that it reflects a setpoint of n_b at $B/2$. Thus, minimizing the expected value of J will keep the state close to the setpoint, so that the system is close to optimal.

There is a problem with this formulation. Classical LQG demands that J include a term that minimizes u (the control effort). In our case, we are not interested in reducing u, since a) increasing u is not costly and b) in any case, the goal is to send at the maximum possible rate, and this corresponds to *maximizing* u rather than minimizing it! If we impose this restriction, then we can no longer do standard LQG.

We can get around this problem by modifying the criterion so that

$$J = \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\varepsilon\mathbf{I}\mathbf{u}$$

and then considering the control as $\varepsilon \to 0$ (assume for the moment that the limit of the series converges to the value of the control at the limit). However, note that the Kalman criterion for stability of the optimal control is that:

$$rank\ (\mathbf{Q}^{\frac{1}{2}^*}\ |\ \mathbf{G}^*\mathbf{Q}^{\frac{1}{2}^*}\ |\ (\mathbf{G}^*)^2\mathbf{Q}^{\frac{1}{2}^*}) = 3 \qquad 5.B1$$

where $\mathbf{Q}^{\frac{1}{2}}$ is defined by $\mathbf{Q} = \mathbf{Q}^{\frac{1}{2}}\mathbf{Q}^{\frac{1}{2}}$, and the * denoted the conjugate transpose operator. If we want to minimize $(n_b - B/2)^2$, then

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Thus, the rank of the matrix in (5.B1) is 1, which is less than 3. Hence, the Kalman stability criterion is not satisfied, and the LQG optimal control is *not stable*.

Note that the problem with the control is not due to our assumption about **R** being εI. Rather, this is because of the nature of the matrix **Q**. However, the nature of **Q** is determined completely by the form of **x** and the nature of the control problem itself. We conclude that for this system, stable LQG control is not feasible.

(132). This problem was recognized quite early (22), references (44,110) provide a detailed review of these. control scheme (117). Control Protocol (TCP) (21,109) and to recover from it (63). and interactive video (36,154). endpoints, for example, two BSD sockets (21,23). *Virtual Circuits* (VCs) (such as in Datakit (41)). not make such demands (36). (68), increasing the load on the bottleneck (101). (63,114). standstill (132). The first definition is used in references (65,117), the second in reference (63), and the third in reference (85). over the resources (140). (34,48). (36,68). in a common pool (62). a day (13,105). data transfer (20,155), peak hours than during off-peak hours (7,111,130). the recent past (32,146), For example, the SPAWN system (146) users to declare their resource needs accurately (20). (80). been extensively studied by Sanders (121-123). (29,77,92). (29), determine the effect of its sending rate on the network (117). (54,63,96,117). (44), (51,52,104), (1,38,65,79,133,137,138). (63,117), (127,137,138). Examples are the Virtual Clock scheme (154), Delay-EDD (34) and Stop-and-Go queueing (47). game (125). (6), Sanders (121-123), Douligeris and Majumdar (25-27) and Lazar *et al* (59). several orders of magnitude larger (84). and perhaps have catastrophic effects (101,132). (112) themselves overload the congested network. of the recent past (45000 kbps to 56 kbps) (69). network allocates its resources (23). (43,55,106,107,115,139,149). criterion to adopt (44). Nevertheless, it is necessary to negotiation and enforcement mechanism (20,155). e.g. (63). the switches (117). Still others assume complete control but no control over source traffic (54). having hardware support for rapid switching (100) optimizing for the average case (18) removing signaling information from the data path (41) choice of scheduling algorithm (157) an efficient call processing architecture (131). peer entities (for example, (117)) For example, a Fair Queueing (23) switch has the responsibility for (65)). very bursty (52,104). and high bandwidth off-line bulk data transfer (12). Gerla and Kleinrock (44) and Pouzin (110) transient spikes in queue occupancy (116,117). (as in CIGALE or Cyclades) (61,88). and possible congestion (65). not acknowledged 'in time' (109). When the alarm goes off, congestion is suspected (153). the switch sends information in packet headers (115) or Source Quench packets (112), choke packets (88), state-exchange packets (79,120), rate-control messages (148), or throttle packets (70) probe values (76), retransmission timers (153), throughput monitoring (147), or delay monitoring (65) such as a hardware problem (153). (14,110). The Loss-load curve scheme (148) and the selective DECbit scheme (115) seek to attention in the literature, and is surveyed in (44). Other schemes, such as Input Buffer Limit (81), Channel Queue Limit (70), Buffer Class (45) and Matsumoto's X.75 proposal (91) packet switches (100). reservation-oriented networks (157). Some exceptions are in the DEC scheme (115), the Loss-load curve approach (148) and the Random dropping scheme (90). scheme does not reduce congestion (90). (63,88). either using a window adjustment scheme (117) or a rate adjustment scheme (17,76). of the future, such cooperation can no longer be assumed (118). For example, in the DECbit scheme (117), The same is true for the Jacobson-Karels TCP scheme (63). (90), and completely by the Loss-load curve method (148). developed for the Datakit network (41). This is the approach taken by Zhang in the Flow Network (154), (34), by Topolcic *et al* in the ST-II proposal (134) and by Cidon *et al* for the PARIS network (16). In a hybrid scheme described in reference (24) The two-window scheme proposed by Hahne *et al* (54) this requirement can be reduced even further (96). user to obey some traffic profile (37). at the time of call set-up, and doing admission control (34). (34). (36). their data can be dropped or violation-tagged (5). (87,128,136,142). In the Flow network, the Virtual Clock mechanism is used (154), distributed flow control (34). Ferrari and Verma (35) and Kanakia (68) have proposed mechanisms based on calendar queues (11) that have a similar effect. though this was not studied in any depth (113). (46,47,68). (34,141). performance guarantee (83). is also relevant (19,31). simulator (73). congestion (44). even if a universal standard such as ISO (12) Nagle (101) proposed a *fair queueing* (FQ) fairly (55,72). Recently Morgan (98) has examined the role such queueing and has been analyzed extensively (42,86). Similar work has been presented by Zhang (154); the fair queueing algorithm presented here (157). The max-min fairness criterion (43,55,115) (115). (this emulation discrepancy bound is proved in reference (50)). The IP protocol (108) does have a field for type-of-service, Recent work (57) raises questions about the { 1~+~{1 over {N}} ({k sup 2} ~+~ k(2 epsilon -1)) right }~~for~~{P sub F} left { 1~+~{1 over {N}} ({k sup 2} ~+~ k(2 epsilon -1)) right } found in reference (23). There are few analytical results here, but Hahne (55) is crucial in the design of switch algorithms (101). queueing algorithms to do so (125). Prue and Postel (113) (94,95) the packet to be dropped is chosen at random (90). (40,56,124,154). (154). at 1.2 Gbps (69). (50,57,89,93,125,127). such as NetBlt (17), and the technology for this is well known (66). (93). since N_{ac} is strictly non-increasing in (t_{chk}, t). has been proposed (57). (58,78). traffic in the sense of (57), TCP protocol (109) with the modifications made by Jacobson (63). we used the REAL network simulator (73) Along with the work done by McKenney (93) and Heybey et al (57), this work provides the practitioner Queueing, such as the Virtual Clock algorithm (154). `i_min[k]` is the position in `max` of the swap(min(a), min(b)); swap(i_max(i_min(a)), i_max(i_min(b))); swap(i_min(a), i_min(b)); pointer consistency, i.e. that `i_min[i_max[a]] = a` and `i_max[i_min[a]] = a`. `i_min[n] = i_max[n] = n`. min(ptr) = data; for (; (ptr/2 >= 1) &&(min(ptr) < min (ptr/2)); ptr /=2) min(1) = INFINITY; save = i_min(1); max(save) = -1; if (min(a) < min(a/2)) smaller = a/2; smaller_son = (min(lson(a)) < min(rson(a))) ? lson(a) : rson(a); if (min(smaller_son) < min(a)) smaller = smaller_son; attendant inefficiencies (50). of Bessel functions (135); of data traffic in current networks (13,51), (129). (143,145). analyzed in (129). (144,145). flow control scheme (17). scheme modified an idea described by Clark et al. for NETBLT (17), and sent to the source using a state exchange protocol (120). (64,97). so, unlike TCP (63,153), (64). (T_0, T_1) if in every time interval $(t_1, t_2) \in (T_0, T_1)$, $(T_0 + 2R(0), T_1)$. conditions are trivially satisfied in $(T_0 + 2R(0), T_1)$. For example, the Jacobson-Karels version of TCP (63) and so will lose bandwidth when V increases (64). Jain's delay based scheme (65) will respond (129). In recent work (157), we have shown that the FQ discipline is similar to the Virtual Clock (154) and Delay-EDD (34) service disciplines. though not using packet pairs, is described in reference (53), (63,109) or Jain's delay-based approach (65). As mentioned in (2), Separation Theorem (3). In recent work, Mitra et al (96,97) Processor Sharing (50), (13,51). (B/2 mi nb (k) mi sk pl mu (k) RTT(k) pl mu (k+1) RTT (k + 1)) (B/2 mi n hat sub b (k) mi sk pl mu hat (k) RTT(k) pl mu hat (k+1) RTT hat (k + 1)) (B/2 mi n hat sub b (k) mi sk pl 2 mu hat (k) RTT(k)) (B/2 mi n hat sub b (k) mi sk pl 2 mu hat (k) RTT(k)) (B/2 mi n hat sub b (k) mi sk pl 2 mu hat (k) RTT(k)) Similar studies are mentioned in (30). chosen to lie inside the range (0,B), such as the second method of Liapunov ((102) page 558). known and robust technique (49). bold "x" eq left (pile { nb ˜ above mu ˜ above mu sub -1 ˜} right) $\mathbf{G} = \begin{bmatrix} 1 & -Rtt & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$ $\mathbf{H} = \begin{bmatrix} Rtt \\ 0 \\ 0 \end{bmatrix}$ $\nu_1 = \begin{bmatrix} 0 \\ \omega \\ 0 \end{bmatrix}$ $\mathbf{C} = (\ 0\ 0\ 1\ )$ $E((\hat{x}(t) - x(t))^T(\hat{x}(t) - x(t)))$, to yield many other types of filters (49). right) (nu sub 1 sup T (theta) nu sub 2 (theta)) ˜ right } eq left (pile { { Q ˜ S˜ } above { S sup T ˜ R˜ } } right) ˜delta $\mathbf{Q} = E\begin{bmatrix} 0 & 0 & 0 \\ 0 & \omega^2 & 0 \\ 0 & 0 & 0 \end{bmatrix}$ $\mathbf{S} = E\begin{bmatrix} 0 \\ \omega\nu_2 \\ 0 \end{bmatrix}$ system. We follow the derivation in (49) (pg 249). bold "K" (k) eq (bold "G" SIGMA (k) bold "C" sup T pl bold " S ") (bold (82,152,158). (151), found in reference (75). We also make the fluid approximation (1), This approximation is commonly used (8,133), and both analysis (97) and simulations show time interval (t, t+J), but in fact can work together (67). scheme proposed by Hahne *et al* (54) is attractive. (6,9,10,27,127). work by Jain (116) and Jacobson (63), In recent work, Ko *et al* (79) described originally by Agnew (1), and since extended by Filipiak (38) and Tipper *et al* (133). Vakil, Hsiao and Lazar (137) Vakil and Lazar (138) have considered Robertazzi and Lazar (119) and Hsiao and Lazar (60) in the benchmark suite described in reference (23) is comparable to that of the DECbit scheme (117), (63,117)), messy and only partially understood (156). experimental high speed network testbed (69). lk1 eq 1 over { RTT(k)} (B/2 mi nhat-1 mi sk pl mu RTT(k) pl mu RTT (k)) lk1 eq 2 mu mi lk pl 1 over { RTT(k)} (B/2 mi nhat-1) lambda (2) eq 2 mu mi mu pl 1 over { RTT (2)} (B/2 mi B/2) lambda (3) eq 2 mu mi mu pl 1 over { RTT(3)} (B/2 - B/2) We follow the description of LQG presented in (102) (pg. 835). bold "Q" eq left (pile { { 1 ˜0 ˜0 ˜ } above { 0 ˜0 ˜0 ˜ } above { 0 ˜0 ˜0 ˜ } } right) (99) (93,117,154), (33). as in reference (12). traffic (13,51,105). (33,103), (4,28) (73,74). transport flow control protocol (109,150), (63,126,154,156), the DECbit scheme (114-117). as implemented in XNS's SPP (150) or in TCP (before 4.3 Tahoe BSD) (109), Jacobson and Karels' (JK) modified TCP (63) (15,115-117), (63,71)). protocols such as NETBLT (17) congestion avoidance scheme (65). (115,148), (see reference (115) and the simulation results below), on the description in references (115,116). of 40 and 570 bytes in the Internet (12). Floyd and Jacobson in (39,40)). (39,40). (54). (though global tradeoffs are still possible (6)). and has also been studied exhaustively in references (39,40). (101)). by Prof. S. Singh and Prof. A. Agrawala (129)). laws is a well known problem (49). thesis by D. Verma (142).