A Control-Theoretic Approach to Flow Contro

Srinivasan Keshav[1,2]

TR-91-015

March, 1991

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 utput queues of the network's switches, and propose deterministic and stochastic models for a single conversation in a network of such switches. We then construct a standard time-invariant linear model for the simplified dynamics of the system. This is used to design an optimal (Kalman) state estimator, a heuristic second-order state estimator as well as a provably stable rate-based flow control scheme. Finally, schemes for correcting parameter drift and for coordination with window flow control are described.

---

[1]Supported by AT&T Bell Laboratories, MICRO and Hitachi Corporation.

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

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. We then construct a standard time-invariant linear model for the simplified dynamics of the system. This is used to design an optimal (Kalman) state estimator, a heuristic second-order state estimator as well as a provably stable rate-based flow control scheme. Finally, schemes for correcting parameter drift and for coordination with window flow control are described.

Introduction

As networks move towards integrated service, there is a need for network control mechanisms that can provide users with performance guarantees in terms of throughput, delay and delay jitter [Fer90]. 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) [FeV90, Gol90, KKK90, Zha89]. However, such reservations can reduce the statistical multiplexing in the network, making the system expensive.

An interesting problem arises with the control of conversations that do not reserve any bandwidth, and so are not guaranteed any performance 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 should adapt to changes in network state, and to prove that the adaptations do not lead to instability in the system. This approach can be used to control both transport connections in current reservationless networks, and so-called 'best-effort' connections in reservation-oriented networks.

In order to use control theory to do flow control, it is necessary to observe changes in the network state. In recent work, we have shown that it is possible to measure network state easily if the network is of a type called a Rate Allocating Server (RAS) network and the transport protocol uses the Packet-Pair probing technique (described below) [KAS90, SAK90]. 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 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, and present a deterministic and stochastic models for networks of such servers. Next, we describe the packet-pair state probing technique. This is used as the basis for the design of a rate-based flow control scheme, that is described in the next section. After proving that the scheme is stable, we present a state space description of the system. The state space model is used to design Kalman estimators. Three practical concerns: a heuristic state estimation scheme, correcting for parameter drift, and interaction with window flow control are examined. We conclude with a review of related work. In two appendices, we examine the steady state behavior of the system, and prove that Linear Quadratic Gaussian control of the system is unstable.

Supported by AT&T Bell Laboratories, MICRO and Hitachi Corporation.

Rate Allocating Servers

Consider the queue service discipline in the routers of a communication network. If packets are sent using a strict Time-Division-Multiplexing (TDM) discipline, 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 doing TDM. It can be shown that service in order of increasing priority index has the effect of doing TDM without the attendant inefficiencies. This basic idea lies behind the Virtual Clock service discipline [Zha89]. In recent work it has been shown shown that the Virtual Clock discipline is very similar to the Fair Queueing [DKS90] and Delay-EDD [FeV90] disciplines [ZhK91]. Thus, we will refer to all three 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 consider 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.

In a rate allocating server there are two reasons why the perceived rate of service of a conversation may change.

1) The number of conversations served can change. Since the service rate of the conversation is inversely proportional to the number of active conversations, the service rate of the conversation also changes.

2) If some other conversation has a low arrival rate, or has a bursty arrival pattern, there are intervals where it does not have any packets to send, and the RAS will treat such a conversation as idle. Thus, the effective number of active conversations decreases, and the rate allocated to the conversation increases. When the traffic on that conversation resumes, the conversation's service rate will again decrease.

Note that even with these variations in the service rate, a RAS's behavior is better than that of an FCFS server. In an 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 service rate varies much more rapidly. Consider the situation where the number of conversations is fixed, and each conversation always has data to send when it is scheduled for service. In a FCFS server, if any conversation sends a large burst of data that fills up the output queue, then the service rate of all the other conversations effectively drops until the entire burst has been served. In a RAS, the other conversations will be unaffected. The server allocates to each conversation a rate of service that is, to a first approximation, independent of the arrival patterns of the conversations, thus the name Rate Allocating Server.

Network Models for Transient Analysis

Networks are typically modeled as queueing systems and hence the kind of results that are 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 [TrD78]. This problem is even more complicated if the network is non-Jacksonian. Further, if the service discipline is general, and the arrivals are general (a G/G/1 system), no such differential equations are known. Thus, we feel that using queueing theory for transient analysis is usually cumbersome and sometimes impossible. However, flow control depends precisely on such transients. Thus, one would like to use an approach that models network transients explicitly.

It has been shown that a deterministic modeling of a RAS network allows network transients to be calculated exactly [SAK90]. However, this can be too simplistic. So, we first summarize a deterministic model similar to the one presented in [SAK90], and then present a stochastic extension.

Deterministic Model

We model a conversation in a RAS network as a series of servers (routers or switches) connected by links. A packet starts out from the source and traverses the links and servers until it reaches the destination. The time taken to get service at each server is finite but deterministic. The time taken to traverse a link is assumed to be zero (this can always be added to the service time at the previous server).

We number the servers in the path of the conversation as 1,2,3..., and the source is numbered 0. If the ith 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 time between consecutive packet services from the same conversation.

If there are other packets from that conversation at the server, the packet waits for its turn to get service (we assume a FCFS queueing discipline per conversation, so the server is not processor-sharing). We assume a work-conserving discipline, which implies that a server will never be idle whenever it has a packet ready to be served.

The source sending rate at time t is denoted by $\lambda(t)$ and the source is assumed to send data spaced apart 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. $\mu(t)$ is defined to be $\frac{1}{s_b}$, and is the bottleneck service rate. To complete the picture, we assume another set of links and servers that constitute a return path from the destination back to the server. This is the path taken by acknowledgment packets (acks). We assume that every packet is acknowledged, so the destination is just another server, and the returning ack is modeled as the same packet looping back to the source. Strictly speaking, this assumption is not required, but we make it for the sake of convenience.

We will henceforth work in discrete time, so the continuous time parameter, t, is replaced by the step index k. In addition, let $S(k)$ be the number of packets outstanding at time k.

Stochastic model

In the deterministic model, μ is assumed to be constant. Actually, μ changes, but only due to the arrival and departure 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 $\mu(k)$. Mathematically speaking, the correlation between $\mu(k+1)$ and $\mu(k)$ is positive. 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 autocorrelation structure. 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 also preserves the autocorrelation structure. While this models only the first order dynamics, we feel that it is sufficient.

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 an implicit assumption here. In reality, 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 we assume that μ is sufficiently far away from 0, then the symmetricity assumption 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 a valid assumption. The gaussian assumption is harder to justify. As mentioned in [AnM79], many noise sources in nature are gaussian. Second, a good rule of the 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 (in terms of Kalman filtering) the gaussian white noise assumption is unavoidable. Thus, for these three reasons, we will assume that the noise is gaussian.

Note that the queueing theoretic approach to the modeling of μ would be to define the density function of μ, say $G(\mu)$, that would have to be supplied by the system administrator. Then, the 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.

Detailed Dynamics of Packet Pair

In order to understand the way state probing works, it is necessary to consider the detailed dynamics of a packet-pair as shown in Figure 1 (attached). The figure presents a time diagram. Time increases down the vertical axis, and each axis represents a node in a communication network. Lines from the source to the server show the transmission of a packet. The shaded parallelograms represent two kinds of delays: the vertical sides are as long as the transmission delay (the packet size divided by the line capacity). 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 the 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 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 $\frac{1}{\mu}$ on the way back due to the different queueing delays along the return path for each ack. 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 reducing 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. 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 can 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.

Design Strategy

The design strategy is based upon the Separation Theorem [AnM90]. 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 and the estimators are derived in a subsequent section.

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 gateway allocates per conversation. Consider the distribution of n_b for the controlled system, $N(x) = Pr(nb = x)$ (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 packet delay is given by $\frac{1}{\int_0^B N(x)dx} \int_0^B xN(x)dx$.

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 will be fair, and choose 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.

Controller Design

For simplicity, we restrict ourselves to taking control action only once per round trip time (RTT). We also 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.

For the purpose of exposition, divide time into *epochs* of length RTT (= R + queueing delay) (see Figure 2, attached). 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.

Consider the situation at the end of the kth epoch. At this time we know RTT, and the value of $S(k)$, the number of packets outstanding at that time. We also *construct* $\hat{\mu}(k+1)$, which is the estimator for the average service rate during the next (k+1th) epoch. If the service rate is 'bursty', then considering a time average for μ may not be enough. 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. This scenario is most likely when propagation delay is much larger than the service time per packet. In such cases, we should engineer the system so that the control time versus the time scale over which the service rate varies is moderate. So, we could take control action not every RTT, but at some other time scale that matches the service rate dynamics. This involves simple extensions to the control schemes presented below, and does not add any new insights, while it complicates the exposition. Hence we will ignore this problem in the sequel.

Figure 2 shows the time diagram for the control. The vertical axis on the left is the source, the axis on the right is the bottleneck, and we ignore the sink for the moment. Each horizontal line represents a packet pair. The epochs are marked for the source and the bottleneck. Note that the epochs at the bottleneck are time delayed d_1 with respect to the source. We use the convention that the end of the kth epoch is called 'time k'. $n_b(k)$ 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 second ack is received). By an

earlier assumption, the delays due to propagation for the next special pair are the same as for the the earlier one (i.e. the kth and $k+1$th control pair have the same propagation delays). 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 the shaded area 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 $S(k)$. Clearly this is the sending rate multiplied by the sending interval, $\lambda(k)RTT(k)$. Since we expect $\hat{\mu}(k)RTT(k)$ packets to be serviced in this interval, we expect that at the end of the kth epoch, the bottleneck will have $S(k) - \hat{\mu}(k)RTT(k)$ more packets. In other words, we have

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

Similarly

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

Let $\hat{RTT}$ be the estimator for RTT. We already know $RTT(k)$ at time k, so $\hat{RTT}(k) = RTT(k)$. $\hat{RTT}(k+1)$ is the estimator for $RTT(k+1)$. In terms of the estimator, then, equation 2 becomes

$$\hat{n}_b(k+2) = \hat{n}_b(k+1) + S(k+1) - \hat{RTT}(k+1)\hat{\mu}(k+1) \quad (3)$$

We know that

$$S(k+1) = \lambda(k+1)\hat{RTT}(k+1) \quad (4)$$

Substitute (4) in (3)

$$\hat{n}_b(k+2) = \hat{n}_b(k+1) + \lambda(k+1)\hat{RTT}(k+1) - \hat{RTT}(k+1)\hat{\mu}(k+1) \quad (5)$$

$n_b(k+1)$ 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 $n_b(k+2)$. Substitute (1) in (5):

$$\hat{n}_b(k+2) = \hat{n}_b(k) + S(k) - \hat{\mu}(k)RTT(k) + \lambda(k+1)\hat{RTT}(k+1) - \hat{RTT}(k+1)\hat{\mu}(k+1) \quad (6)$$

The control should set this to $B/2$. So, set (6) to $B/2$, and estimate $\lambda(k+1)$.

$$\hat{n}_b(k) + S(k) - \hat{\mu}(k)RTT(k) + \lambda(k+1)\hat{RTT}(k+1) - \hat{RTT}(k+1)\hat{\mu}(k+1) = B/2 \quad (7)$$

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

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

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 four estimators:

$$\hat{RTT}(k+1),\ \hat{\mu}(k),\ \hat{\mu}(k+1),\ \hat{n}_b(k) \quad (9)$$

The effectiveness of the control depends on the choice of the estimators. This is considered later in the paper.

Stability Analysis

The state equation can be derived from (1)

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

For the stability analysis of the 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

$$n_b(k+2) = n_b(k+1) + \lambda(k+1)RTT(k+1) - \mu(k)RTT(k+1) \quad (10)$$

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

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

$$\frac{RTT(k+1)}{\hat{RTT}(k+1)}[B/2 - \hat{n}_b(k) - S(k) + \hat{\mu}(k)RTT(k) + \hat{\mu}(k+1)\hat{RTT}(k+1)] \quad 11$$

Or, moving back two steps in time,

$$n_b(k) = n_b(k-1) - \mu(k)RTT(k) + \frac{RTT(k)}{\hat{RTT}(k)}[B/2 - \hat{n}_b(k-2) - S(k-1) + \hat{\mu}(k-1)RTT(k-1) + \hat{\mu}(k)\hat{RTT}(k)] \quad 12$$

The best estimate of $RTT(k+1)$ depends on the exact time series of $RTT(k)$. One guess is that it should be the close to the value at time k. If we make this assumption, $\hat{RTT}(k)$ can be set to $RTT(k)$, and

$$n_b(k) = n_b(k-1) - \mu(k)RTT(k) + B/2 - \hat{n}_b(k-2) - S(k-1) + \hat{\mu}(k-1)RTT(k-1) + \hat{\mu}(k)\hat{RTT}(k) \quad 13$$

Taking the Z transform of both sizes, we get

$$n_b(z) = z^{-1}n_b(z) - \mu(z)RTT(z) + B/2 - z^{-2}n_b(k) - z^{-1}S(k) + z^{-1}\hat{\mu}(z)z^{-1}RTT(z) + \hat{\mu}(z)\hat{RTT}(z) \quad 14$$

Considering n_b as the output variable, it is clear that the characteristic equation is

$$z^{-2} - z^{-1} + 1 = 0 \quad 15$$

Solving for z^{-1}, we get

$$z^{-1} = \frac{1 \pm \sqrt{1-4}}{2} = \frac{1 \pm 3i}{2} \quad 16$$

The distance from 0 is hence

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

Hence the eigenvalue lies on the unit circle, and the controlled system is *not* asymptotically stable.

We place the pole of the characteristic equation so that the system is asymptotically stable. Consider the control law

$$\lambda(k+1) = \frac{\alpha}{\hat{RTT}(k+1)}[B/2 - \hat{n}_b(k) - S(k) + \hat{\mu}(k)RTT(k) + \hat{\mu}(k+1)\hat{RTT}(k+1)] \quad 17$$

This leads to a characteristic equation

$$\alpha z^{-2} - z^{-1} + 1 = 0 \quad 18$$

so that the roots are

$$z^{-1} = \frac{1 \pm \sqrt{4\alpha - 1}\,i}{2\alpha} \quad 19$$

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

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

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

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

This means that if $\alpha < 1$, the system is provably asymptotically stable (from 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 $B/2$ the source should send exactly at the rate computed by (8). If it does so, the system may be unstable. Instead, it sends slightly slower and this ensures that the system is asymptotically stable. Note that α is a constant that is independent of the system dynamics and can be chosen in advance to be any desired value < 1. The exact value chosen for α will

control the rise time of the system, and for adequate responsiveness, it should not be too small. We plan to study the effect of α on packet loss and delay jitter by simulation.

State Space Formulation

We now present the state space formulation corresponding to the once-per-RTT control model. This is motivated by the fact that most optimal filtering and control approaches require a state space representation.

We will use the standard linear stochastic state equation given by

20

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

x, **u** and **y** are the state, input and output vectors of sizes n, m and r. **G** is the nxn state matrix, **H** is a nxm matrix and **C** is a 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-1 th epoch, $\mu(k-1)$. This is the only observation it has about the system state and so $y(k)$ is a scalar, $y(k) = \mu(k-1)$. If this is to be derived from the state vector **x** then the state must contain this element. 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 $\mu(k-1)$, $\mu(k)$ 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 component of μ.

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

$$n_b(k+1) = n_b(k) + \lambda(k)RTT(k) - \mu(k)RTT(k) \qquad 21$$
$$\mu(k+1) = \mu(k) + \omega(k)$$
$$\mu_{-1}(k+1) = \mu(k)$$

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

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

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

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

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

This completes the state space description of the flow control system.

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 [GoS84]. 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 :

$$E\left\{\begin{bmatrix} v_1(k) \\ v_2(k) \end{bmatrix} [v_1^T(\theta) v_2(\theta)]\right\} = \begin{bmatrix} Q & S \\ S^T & R \end{bmatrix} \delta(t-\theta) \qquad 22$$

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

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

$$\mathbf{R} = \mathrm{E}(v_2^2)$$

$$\mathbf{S} = \mathrm{E}\begin{bmatrix} 0 \\ \omega v_2 \\ 0 \end{bmatrix}$$

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 $E(\omega^2)$ and $E(v_2^2)$.

From the state equation,

$$\mu(k+1) = \mu(k) + \omega(k) \qquad 23$$

Also,

$$\mu_{observed}(k+1) = \mu(k+1) + v_2(k+1) \qquad 24$$

Substituting (23) in (24),

$$\mu_{observed}(k+1) = \mu(k) + \omega(k) + v_2(k+1) \qquad 25$$

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 $E(v_2^2)$ must be extraneously supplied, either by simulation or by measurement of the actual system. Practically speaking, even if a good guess for these two values is 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 hence matrices **Q**, **R** and **S** are known. It is now straightforward to apply Kalman filtering to the resultant system. We follow the derivation in [GoS84] (pg 249).

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

$$\hat{\mathbf{x}}(k+1) = \hat{\mathbf{G}}\mathbf{x}(k) + \mathbf{K}(k)[y(k) - \hat{\mathbf{C}}\mathbf{x}(k)] + \mathbf{H}u(k) \qquad 26$$

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

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

$$\mathbf{K}(k) = [\mathbf{G}\Sigma(k)\mathbf{C}^T + \mathbf{S}][\mathbf{C}\Sigma(k)\mathbf{C}^T + \mathbf{R}]^{-1} \qquad 27$$

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

$$\Sigma(k+1) = \mathbf{G}\Sigma(k)\mathbf{G}^T + \mathbf{Q} - \mathbf{K}(k)[\mathbf{C}\Sigma(k)\mathbf{C}^T + \mathbf{R}]\mathbf{K}(k)^T \qquad 28$$

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

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

Note that the 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.

The major problem is that the estimator requires the system administrator to supply values of the variance of the system and observation noise. This may not be available in practice. One possibility is for the system to automatically determine these values, using some form of dual control, coupled with a self tuning regulator. However, such control is hard to implement.

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.

A Heuristic Estimation Technique

The heuristic is based on the exponential average filter, described by:

$$\hat{\theta}(k+1) = \alpha\, \hat{\theta}(k) + (1-\alpha)\, \theta(k) \qquad 29$$

where $\hat{\theta}(k)$ is the estimator at step k+1, θ is the current observation, and α is a constant chosen to control the effect of past history on the estimator. If α is small, then the estimator reacts quickly to changes in θ, else, the previous history slows down the response of $\hat{\theta}$ to changes in θ. This filter is commonly used in communication systems, since it is easy to compute, and the choice of α can be used to control its behavior. If α is large, $\hat{\theta}$ can be considered to be θ passed through a low pass filter.

The heuristic we use is to relate the choice of α to the system state. We note that variation of μ is due to observation noise and system noise. Since the observation noise is high-frequency, it can be removed by choosing a reasonably large α. How should α react to system noise?

Consider two types traffic arrival patterns and the corresponding system noise distribution. In one case, the traffic arrival pattern is bursty, and hence the past history of μ will probably not reflect the future state. In this system, the variation in μ will be large, and to adapt quickly, α should be smaller. In contrast, if the system state is relatively unchanging, then the system noise is nearly zero. μ will not vary much, and α should be larger.

Since the variance in μ is linked to the desirable choice for α, we propose to link the two. Essentially, α is chosen to be inversely proportional to the inverse of the variance of μ scaled to lie between 0 and 1. We intend to test this heuristic through extensive simulation.

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.

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. This is not a big problem. Our argument was that strict *bandwidth* reservation leads to a loss of statistical multiplexing. Buffer reservation does not have this problem. As long as no conversation is refused admission due to a lack of buffers, it is clear that statistical multiplexing of bandwidth is not affected, 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 believe that the two forms of flow control are *not* diametrically opposed, as has been claimed by Jain [Jai90], but in fact can work together.

The question of what window size to use 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 [HKM90] is very attractive. Briefly, in this scheme, some fixed number of users, say 10, are given a window corresponding to the largest possible bandwidth delay product. Now, the 11th conversation must have a bandwidth allocation no larger than a tenth of the largest possible bandwidth, and hence can be allocated a window size that is a tenth of the previous value. Thus, the next hundred conversations can be given a full window's worth of buffers at the same cost as for the first ten conversations. This argument is quite general, and a scheme that sets up these windows and dynamically adjusts them is described by the authors. We believe that such a window based flow control scheme can be used in conjunction with the rate based flow control scheme proposed in this paper.

Limitations of the Control-Theoretic Approach

The main limitation of the control theoretic approach is that it restricts the form of the system model. Since most control theoretic approaches assume a linear system, the system model must be cast in this form. This might lead to the loss of some information. 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 controller design requires that 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, RAS state can be probed using a packet pair, and so RAS networks are amenable to a formal treatment.

Contributions and Related Work

We believe that this paper is amongst the first to propose a formal control theoretic approach to flow control. While such control principles have been appealed to in work by Jain [RaJ87] and Jacobson [Jac88], their approach is quite informal.

Our use of a packet-pair to estimate the state is unique. and this estimation is critical in enabling the control scheme. We have described the first provably stable rate-based flow control scheme. Practical

concerns in implementing the scheme have also been discussed.

Another control theoretic approach was described originally by Agnew [Agn76] and since extended by Filipiak [Fil88] and Tipper et al [TiS90]. In their approach, the system is approximated by a first order differential equation based on the flow model. The system parameters are tuned so that in the steady state, the differential equation solution and the solution of a corresponding queueing model agree. This approach is rather different from ours. 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. To be precise, they assume that $\mu = G(n_b)$, where $G(.)$ is some nonlinear function. This is not true for a RAS server. Hence, we cannot apply their techniques to our system.

Future Work

This paper makes several simplifications and assumptions. In future work, we will present simulation results to back up some of these claims. It is also 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. Finally, as mentioned earlier, other possible extensions are to design a minimum variance controller and a non-linear controller.

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. The helpful advice of Dr. R.P. Singh, Prof. D. Ferrari, Dr. D. Mitra, Prof. M. Tomizuka and Prof. P.P. Varaiya on several aspects of the work is much appreciated.

References

[Agn76] Agnew, C., Dynamic Modeling and Control of Congestion-prone Systems, *Operations Research 24*, 3 (1976), 400-419.

[AnM79] Anderson, B. D. O. and Moore, J. B., Optimal Filtering, *Prentice Hall*, 1979.

[AnM90] Anderson, B. D. O. and Moore, J. B., Linear Quadratic Methods, *Prentice Hall*, 1990.

[DKS90] Demers, A., Keshav, S. and Shenker, S., 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..

[FeV90] Ferrari, D. and Verma, D., A Scheme for Real-Time Channel Establishment in Wide-Area Networks, *IEEE J. on Selected Areas in Communications*, April 1990.

[Fer90] Ferrari, D., Client Requirements for Real-Time Communications Services, *IEEE Communications Magazine 28*, 11 (November 1990).

[Fil88] Filipiak, J., Modelling and Control of Dynamic Flows in Communication Networks, *Springer-Verlag*, 1988.

[Gol90] Golestani, S. J., A Stop-and-Go Queueing Framework for Congestion Management, *Proc. ACM SigComm 1990*, September 1990, 8-18.

[GoS84] Goodwin, G. C. and Sin, K. S., Adaptive Filtering Prediction and Control, *Prentice Hall*, 1984.

[HKM90] Hahne, E. L., Kalmanek, C. R. and Morgan, S. P., Fairness and Congestion Control on a Large ATM Data Network, *Submitted for ITC 13* , Jan 1990.

[Jac88] Jacobson, V., Congestion Avoidance and Control, *ACM SigComm Proceedings*, August 1988, 314-329.

[Jai90] Jain, R., Myths About Congestion Management in High-Speed Networks, *Dec. Technical Report-726*, October 1990, Digital Equipment Corporation.

[KKK90] Kalmanek, C. R., Kanakia, H. and Keshav, S., Rate Controlled Servers for Very High Speed Networks, *Proc. Globecom 1990*, December 1990.

[KAS90] Keshav, S., Agrawala, A. K. and Singh, S., Design and Analysis of a Flow Control Algorithm for a Network of Rate Allocating Servers, *Proc. IFIP TC6/WG6.1/WG6.4 Workshop on Protocols for High Speed Networks*, November 1990.

[Oga87] Ogata, K., Discrete Time Control Systems, *Prentice Hall*, 1987.

[RaJ87] Ramakrishnan, K. K. and Jain, R., Congestion avoidance in Computer Networks with a Connectionless Network Layer - Part II - An Explicit Binary Feedback Scheme, *Dec. Technical Report-508*, April 1987, Digital Equipment Corporation.

[SAK90] Singh, S., Agrawala, A. K. and Keshav, S., Deterministic Analysis of Flow and Congestion Control Policies in Virtual Circuits, *University of Maryland Tech Report 2490*, June 1990.

[TiS90] Tipper, D. and Sundareshan, M. K., Numerical Methods for Modeling Computer Networks under Nonstationary Conditions, *JSAC 8*, 9 (December 1990).

[TrD78] Tripathi, S. and Duda, A., Time-dependent Analysis of Queueing Systems, *INFOR 24*, 3 (1978), 334-346.

[Zha89] Zhang, L., A New Architecture for Packet Switching Network Protocols, *PhD thesis*, Massachusetts Institute of Technology, July 1989.

[ZhK91] Zhang, H. and Keshav, S., Comparision of Rate-Based Service Disciplines, *Submitted to ACM SigComm 1991*, March 1991.

Appendix 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, and that $n_b(0) = 0$. Assume $\alpha = 1$ for the moment.

In the steady state, the state equation (10) becomes

$$n_b(k+1) = n_b(k) + \lambda(k+1)RTT(k+1) - \mu(k+1)RTT(k+1) \qquad \text{A1}$$

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

$$\hat{RTT} = RTT \text{ for } all\ k \qquad \text{A2}$$

This makes the control law to be

$$\lambda(k+1) = \frac{1}{RTT(k)}[B/2 - \hat{n}_b(k) - S(k) + \mu RTT(k) + \mu RTT(k)] \qquad \text{A3}$$

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

$$\lambda(k+1) = 2\mu - \lambda(k) + \frac{1}{RTT(k)}[B/2 - \hat{n}_b(k)] \qquad \text{A4}$$

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

$$\lambda(1) = 2\mu - \mu + \frac{1}{RTT(1)}(B/2) \qquad \text{A5}$$

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

The buffer at the end of epoch 1 is given by

$$n_b(1) = 0 + (\mu + \frac{B/2}{RTT}(1))RTT(1) - \mu RTT(1) \qquad \text{A6}$$

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

To check further, at time 2, we get

$$\lambda(2) = 2\mu - \mu + \frac{1}{RTT}(2)[B/2 - B/2] \quad \text{A7}$$

since (5) tells us that $\hat{n}_b(k+1)(1) = B/2$.

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

And,

$$n_b(2) = B/2 + \mu RTT(2) - \lambda(2)RTT(2) \quad \text{A8}$$

$$n_b = B/2 \quad \text{A9}$$

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 (A9) 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 (μ, $B/2$) at time 3. Thus, if the system is steady, so is the control. This is reassuring.

Appendix B - Linear Quadratic Gaussian Optimal Control

We consider optimal control of the flow control system described above. 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 [Oga87] (pg. 835).

The quadratic performance index to minimize is given by

$$J = \mathbf{x}^T \mathbf{Q}\, \mathbf{x} \quad \text{B1}$$

where the state vector $\mathbf{x}$ 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 approach. 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

$$J = \mathbf{x}^T \mathbf{Q} \mathbf{x} + \mathbf{u}^T \varepsilon \mathbf{I} \mathbf{u} \quad \text{B2}$$

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:

$$rank\ [\mathbf{Q}^{\frac{1}{2}^*} \mid \mathbf{G}^*\mathbf{Q}^{\frac{1}{2}^*} \mid (\mathbf{G}^*)^2\mathbf{Q}^{\frac{1}{2}^*}] = 3 \quad \text{B3}$$

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 (B3) 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 $\mathbf{R}$ being εI. Rather, this is because of the nature of the matrix $\mathbf{Q}$. However, the nature of $\mathbf{Q}$ is determined completely by the form of $\mathbf{x}$ and the nature of the control problem itself. We conclude that for this system, LQG control is not feasible.

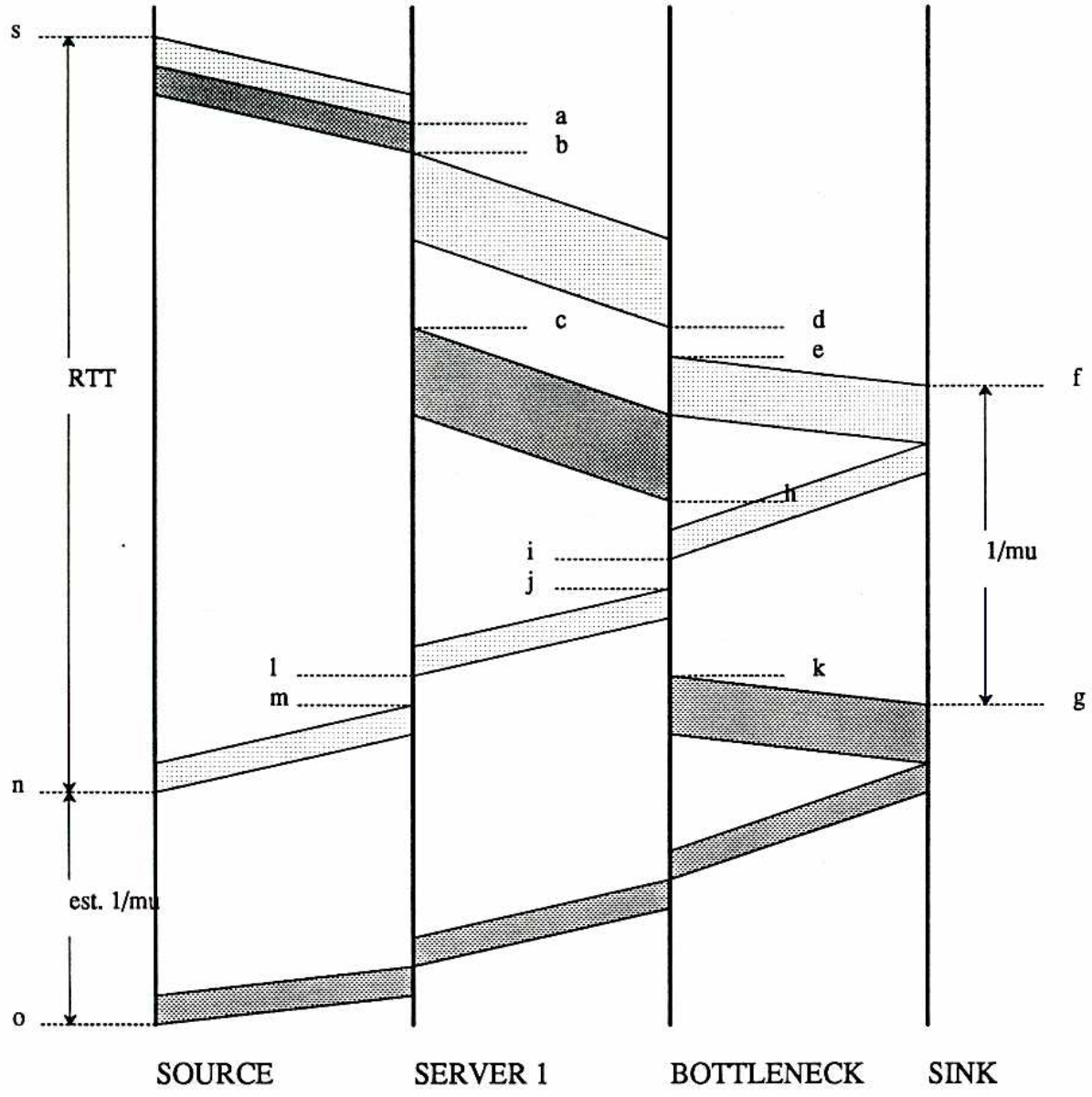
s
a
b
c
d
e
f
RTT
h
i
j
1/mu
l
m
k
g
n
est. 1/mu
o
SOURCE
SERVER 1
BOTTLENECK
SINK

FIGURE 1

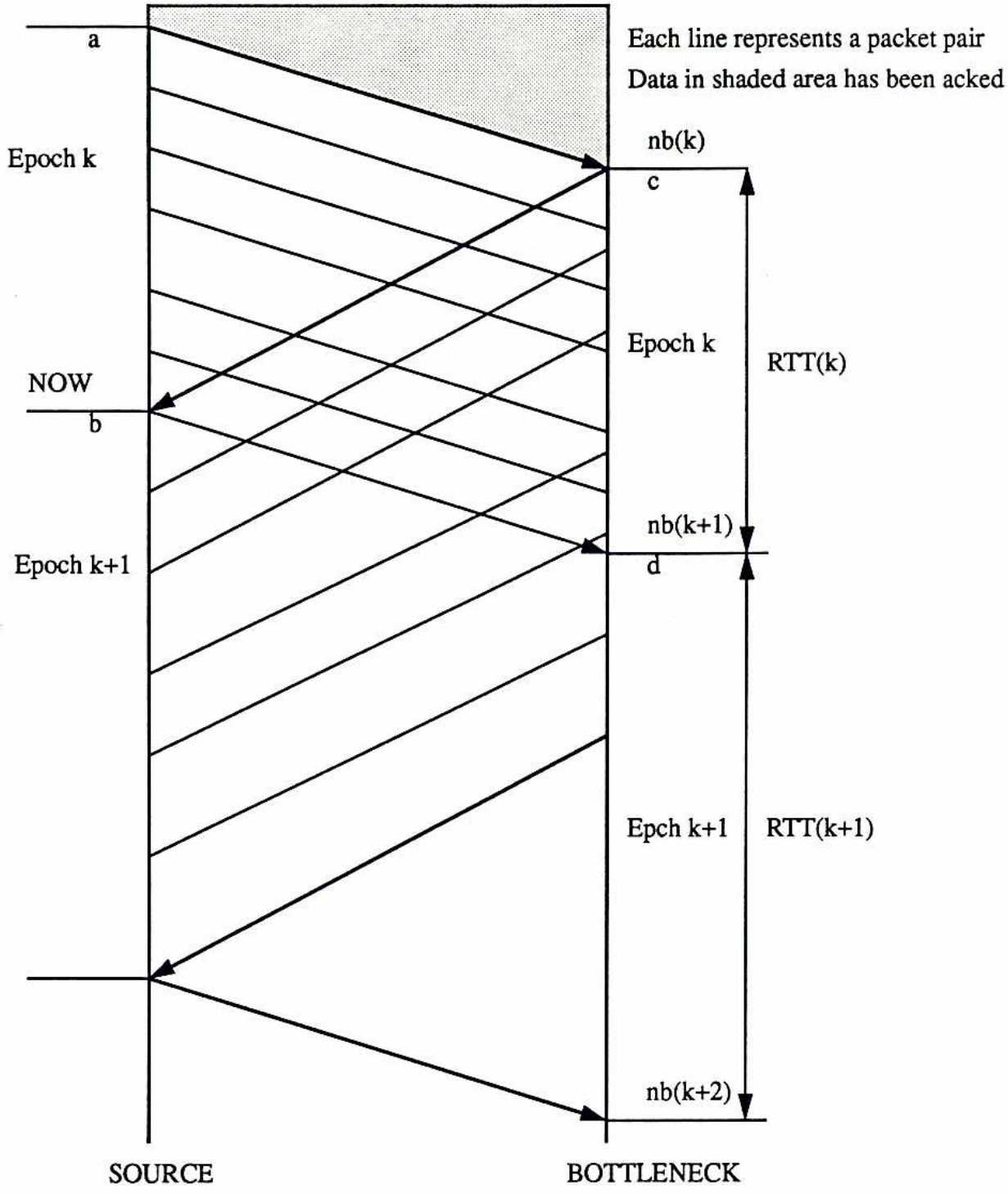
a
Each line represents a packet pair
Data in shaded area has been acked
nb(k)
Epoch k
c
Epoch k
RTT(k)
NOW
b
nb(k+1)
Epoch k+1
d
Epch k+1
RTT(k+1)
nb(k+2)
SOURCE
BOTTLENECK

FIGURE 2