A Control-Theoretic Approach to Flow Control

Srinivasan Keshav

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

Abstract

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

1. Introduction

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

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

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

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

---

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

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

2. Rate Allocating Servers

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

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

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

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

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

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

3. Choice of network model

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

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

3.1. Deterministic Model

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

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

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

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

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

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

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

3.2. Stochastic model

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

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

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

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

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

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

4. Detailed Dynamics of Packet-Pair

State probing is done using the Packet-Pair mechanism, shown in Figure 1.

The figure presents a time diagram. Time increases along the vertical axes, and each axis represents a node in a communication network. The parallelograms represent the transmission of a packet and correspond to two kinds of delays: the vertical sides are as long as the transmission delay (the packet size divided by the line capacity). Second, the slope of the longer sides is proportional to the propagation delay. Since the network is store-and-forward, a packet cannot be sent till it is completely received. After a packet arrives, it may be queued for a while before it receives service. This is represented by the space between two dotted lines, such as *de*.

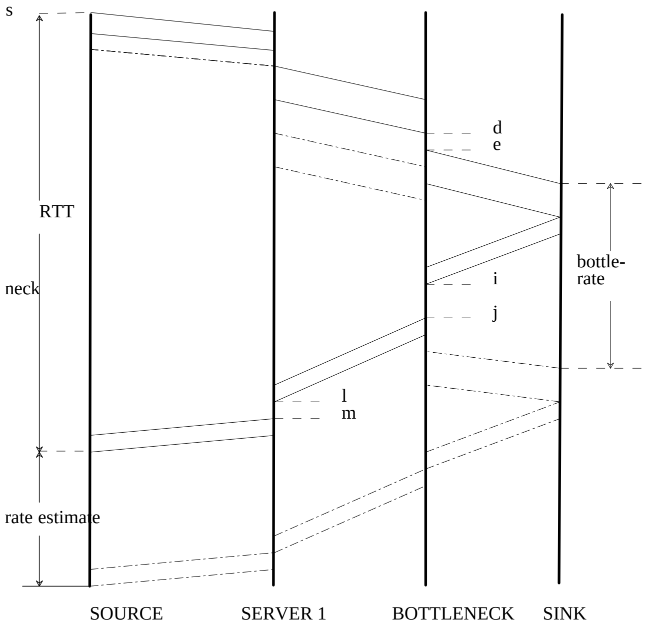

Figure 1: The Packet-Pair probing scheme

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

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

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

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

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

5. Design Strategy

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

5.1. Choice of Setpoint

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

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

> 1) Pr(loss of bandwidth) = Pr (RAS server schedules the conversation for service $\mid n_b = 0$). 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 $\mid n_b = B$), so that the density at B, $N(B)$ is proportional to the probability of a packet loss.
>
> 3) the mean queuing delay is given by
>
> $$\frac{s_b}{\int_0^B N(x)\,dx} \int_0^B xN(x)\,dx,$$
>
> where, on average, a packet takes s_b units of time to get service at the bottleneck.

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

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

5.2. Frequency of Control

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

5.3. Assumptions Regarding Round Trip Time Delay

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

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

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

5.4. Controller Design

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

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

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

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

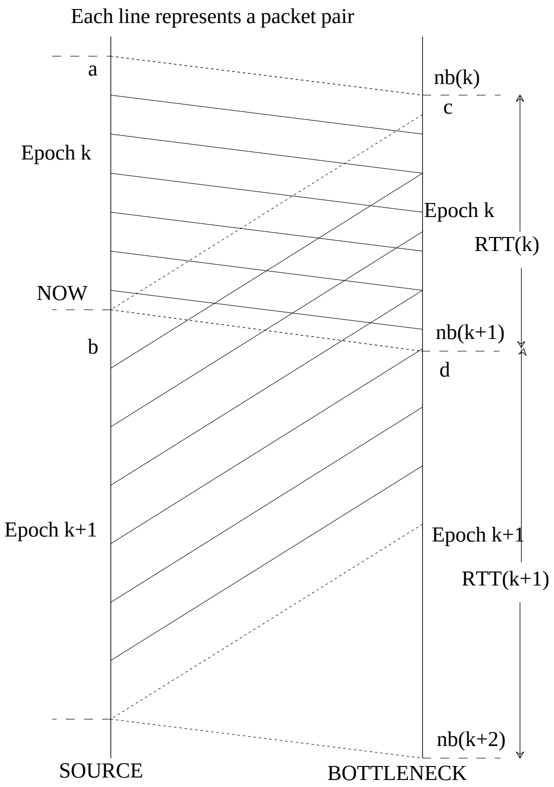

Figure 2: Time scale of control

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

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

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

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

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

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

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

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

From (3) and (4):

$$n_b(k+2) = n_b(k) + S(k$$