Chapter 4: The Packet Pair Flow Control Protocol

4.1. Introduction

Most current packet switched data networks have routers and switches that obey a first-come-first-served (FCFS) queueing discipline, and existing transport layer flow control protocols have been optimized for such networks. If the service discipline is changed to Fair Queueing, then flow control protocols can improve their performance. This chapter presents the design and analysis of a flow control scheme, called the Packet-Pair flow control protocol, enabled by the Fair Queueing discipline. We first present a deterministic model for networks of Fair Queueing servers to motivate the design of Packet-pair. We then give implementation details. Subsequent sections deterministically analyze the transient behavior of Packet-pair.

4.2. Fair Queueing servers

Consider the queue service discipline in the output queues of packet routers. 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 packet's service time were the server actually TDM. It can be shown that service in order of increasing priority index approximately emulates TDM without its attendant inefficiencies (50). This idea lies behind the Fair Queueing (FQ) service discipline.

With a FQ server, there are two reasons why the rate of service perceived by 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 a low arrival rate, or has a bursty arrival pattern, then there are intervals where it does not have packets to send, and the FQ server treats that conversation as idle. Thus, the effective number of active conversations decreases, and the rate allocated to all the other conversations increases. When the traffic resumes, the service rate again decreases.

Note that even with these variations in the service rate, a FQ server 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 FQ server, 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 conversations' arrival patterns. This motivates the use of a rate-based flow control scheme that determines the allocated service rate, and then sends data at this rate.

Choice of network model

We would like to design the flow control mechanism for a source in a network of FQ servers on a sound theoretical basis. This requires an analytic model for network transients. The standard network analysis technique is stochastic queueing analysis, where, for tractability, the usual assumptions are that the network consists of M/M/1 servers, the sources inject Poisson traffic, and the sources generate traffic independently. There are three problems with this approach. First, the strong assumptions regarding servers and sources are not always justifiable in practice. Second, the kind of results that can be obtained are those that hold in the average case, for example, expected queueing delays, and expected packet loss rates. Though the Chapman-Kolmogorov differential equations describe the exact dynamics (and thus the transient behavior) of a single M/M/1 queue, the solution of these equations is as hard as evaluating an infinite sum of Bessel functions (135); besides, extending this analysis to a network of M/M/1 servers is difficult. Third, even if an exact analysis of transients in the network is obtained by an extension of the Chapman-Kolmogorov equations, if the servers are not M/M/1, no such differential equations are known.

Thus, using stochastic queueing analysis, transient analysis of a network of FQ servers (which are not M/M/1) is cumbersome, and perhaps impossible. However, flow control depends precisely on such transients. Thus, we prefer an approach that models network transients explicitly, but without these complications.

We model a single conversation in a network of FQ servers using deterministic queueing analysis. This model, formally defined in the next section, makes a major assumption that the service time per packet, defined as the time between consecutive packet services from a conversation, is assumed to be constant at each server. This is true if all the packets in a given conversation are of the same size and if the number of active conversations (conversations that have data to send when their turn in round-robin order comes by) at each FQ server is constant. The packet size assumption is borne out by studies of data traffic in current networks (13, 51), and will certainly hold in ATM networks of the near future. The other assumption is harder to justify. A FQ server isolates a conversation from other conversations if they are not too bursty, but this is not sufficient justification. We treat this assumption as a necessary crutch to aid deterministic analysis. We do not expect this assumption to hold in practice, and later in this chapter, the assumption is relaxed to allow infrequent, single sharp changes in the number of active conversations. However, note that in the important case of a network of FCFS servers, the deterministic service time assumption is wrong. Hence, for FCFS networks, our analysis is incorrect, and the Packet-pair flow control protocol is infeasible.

With these caveats in mind, it is nevertheless interesting that a deterministic modeling of a FQ server network, though naive, allows network transients to be calculated exactly (129). Waclawsky and Agrawala have developed and analyzed a similar deterministic model for studying the effect of window flow control protocols on virtual circuit dynamics (144, 145).

Model

We model a conversation in a FQ network as a regular flow of packets 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 (notations is summarized in the Appendix to this chapter). The source sends packets to a destination, and 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 that sources always have data to send (an infinite-source assumption). 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 (129). However, for simplicity of exposition, we assume infinite sources from now on.

The service time at each server is 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 for the same conversation. The time taken to traverse a link is assumed to be zero (if it is not, it can always be added to the service time at the previous server).

If the server is not idle when a packet arrives, then the service time may be more than s_i. This is ignored in the model, but we consider its implications in a later section. 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 for packets of the same conversation). We assume a work-conserving discipline, which implies that a server will never be idle whenever a packet is ready.

The source sending rate is denoted by ρ_0 and the source is assumed to send packets spaced exactly $s_0 = 1/\rho_0$ 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 is defined to be $\frac{1}{s_b}$, and is the bottleneck service rate.

We now introduce the notion of a rate-throttle, by means of a recursive definition. To start with, the first server in the path of a conversation is a rate-throttle. Consider the servers along the path from the source to the destination. A server on the path is a rate-throttle if it is slower than some previous rate-throttle. Let SL, the ordered set of rate-throttles, be the set of strictly slower servers from the source to the bottleneck.

We now prove several lemmas about the properties of such conversations. Similar results and a more detailed analysis can be found in (143, 145).

Lemma 1 : (Basic lemma)

Consider data arriving at an initially idle server j at a rate r.

(a) If $r \le \rho_j$, there is no queueing at j, and the departure rate from server j is r.

(b) If $r > \rho_j$, there is queueing at j, and the departure rate from server j is ρ_j.

Proof :

(a) Initially, since the server is idle, its queue is empty. If the first packet arrives at time t_0, it will depart at time $t_0 + s_j$. Packets in the arriving stream are spaced $1/r$ time units apart. Thus, the next packet arrives at time $t_0 + 1/r$. Since $\rho_j \ge r$, $1/r \ge 1/\rho_j$ and $t_0 + 1/r \ge t_0 + s_j$, so the next packet arrives only after the first one has left. Thus, there is no queueing at the server. Simple induction on the sequence number of the arriving packet gives us the result on queueing.

The departure rate of the packets is constrained only by the arrival rate, and hence the output stream from the server has a rate r.

(b) Since the departure of the first packet happens after the arrival of the next packet, the second packet will be queued in the server. If there is a queue already, and a packet arrives before the departure of the previous packet, it will only add to the queue. Induction on the packet sequence number gives us the queueing result.

Since the departure stream from the server has an inter-packet spacing of s_j, the output stream is at rate ρ_j.

Lemma 2 : (Composition)

Consider two adjacent servers j and $j+1$. If data enters server j at a rate r such that $\rho_j \ge r > \rho_{j+1}$ queueing occurs only at server $j+1$.

Proof :

Since $r \le \rho_j$, there is no queueing at server j (Lemma 1). Hence, the departure rate of packets from server j, as well as the arrival rate at server $j+1$ is r. Since $r > \rho_{j+1}$, there is queueing at server $j+1$ (Lemma 1).

Lemma 3 : (Single bottleneck)

If data enters a segment of the VC numbered k, k+1, .. L, at a rate r such that $\rho_L < r < \rho$, $\rho \in \{\rho_k, \rho_{k+1}, \cdots, \rho_{L-1}\}$, then queueing occurs only at L.

Proof :

Since $r < \rho_k$, there is no queueing at server k and the departure rate from server k is r (Lemma 1). We can thus delete server k from the chain, and repeat the argument for the servers k+1, k+2, ... L. For the servers L-1, L, we use Lemma 2 to get the desired result.

Lemma 4 : (Chain of rate-throttles)

Queueing can happen only in elements of SL, the set of strictly slower servers.

Proof:

Break up the server chain 1,2, ... , b into sub-chains 1,2, ... sl_1; sl_1, sl_1+1, $\cdots$, sl_2; ... , such that only $sl_i \in$ SL. Consider the first such chain. If $\rho_0 < \rho_{sl_1}$, there is no queueing at sl_1. Hence, to get the worst possible scenario, we assume that $\rho_0 > \rho_{sl_1}$. In that case, from Lemma 3, the only queueing at the first chain will be at sl_1 (if ρ_0 is very large, sl_1 could just be 1). By definition of SL, the departure rate from sl_1, ρ_{sl_1}, satisfies the requirements for Lemma 3, so there will be queueing at sl_2, and at no other node in that subchain. From induction on the sequence number of the subchain, we get the desired result.

Lemma 5 : (Probing)

If a source sends packets spaced s_0 time units apart, and $\rho_0 \geq \rho_b$, the acks will be received at the source at intervals of s_b time units.

Proof :

By definition of the bottleneck, and Lemmas 1 and 4, the departure rate of packets at the bottleneck is μ. Since acks are created for each packet instantaneously, the acks will be spaced apart by s_b.

Define Δ_j to be $\rho_{sl_{j-1}} - \rho_{sl_j}$.

Lemma 6 : (Burst dynamics)

If a source sends a burst of K packets at a rate $s_0 \gg \rho_i$, for all i, then the queue at sl_j builds up at the rate Δ_j, reaches its peak at time $t_j = \sum_{i=0}^{j-1} s_i + \frac{K}{\rho_{sl_{j-1}}}$, and decays at the rate ρ_{sl_j}.

Proof :

Consider the situation at sl_j. This server receives packets at a rate $\rho_{sl_{j-1}}$, and serves them at the rate of ρ_{sl_j}. Thus, the queue builds up at the rate Δ_j. The queue reaches the maximum size when the last packet from the previous rate-throttle arrives. Since this is at a rate $\rho_{sl_{j-1}}$, the time to receive K packets is $K/\rho_{sl_{j-1}}$. To this we add $\sum_{i=0}^{j-1} s_i$, which is the time that the first packet arrived, to get the desired result. Finally, the queue will decay at the service rate of the rate-throttle, i.e., ρ_{sl_j}.

Note that in our model, it is not possible to have more than one bottleneck. While queueing may occur at more than one node, the service rate of the circuit is determined by the lowest indexed server with a service rate of μ, and this will be the bottleneck.

4.3. Rate probing schemes

How should we design a flow control scheme for a FQ network? Since the network allocates each conversation a service rate at its bottleneck server, a simple flow control scheme would be to *probe* the server to determine its current service rate for that conversation, and then send data at that rate (note that each conversation has its *own* bottleneck server). Sending it any slower would result in loss of throughput, and any faster would result in queueing at the bottleneck. Thus, it is clear that we should use a rate-based flow control scheme (17). Note that rate-based flow control is explicitly enabled by FQ networks.

Rate based flow control

Our first attempt at designing a rate-based flow control scheme modified an idea described by Clark et al. for NETBLT (17), but as shown below, it was not successful. If a source sends data at a rate ρ_0, and receives acknowledgments at a rate ρ_b, then a reasonable control scheme is: if $\rho_0 > \rho_b$, decrease ρ_0, else increase it. The idea is that the rate at which acknowledgments are received is approximately the rate which the FQ server has allocated to the conversation. This should match the sending rate.

The increase and decrease policies are multiplicative, that is the algorithm is

if ($\rho_0 > \rho_b$) **then** $\rho_0 = \alpha\rho_0$ **else** $\rho_0 = \beta\rho_0$

where $\alpha < 1$ and $\beta > 1$. As the service rate changes, this adaptive scheme should converge on the new rate, and the system should stabilize at the correct rate.

However, there are four problems. First, a source cannot determine an increase in available capacity except by sending at a slightly increased rate and looking at the stream of acknowledgments (acks). Thus, a sudden large increase in the service rate can be adjusted for only after several round trip times. This is undesirable, particularly in high speed networks, where the bandwidth delay product can be large. Second, it takes a few round trip times to adjust to a sharp decrease in service rate. In the meantime, the bottleneck queue builds up. Third, after a decrease, the source sends at very nearly the service rate, so the built up queues never shrink, and the network becomes more prone to packet loss. Finally, the rate probe tends to push the network towards congestion, since the source always tries an increased sending rate, until the rate can no longer be supported. These problems point to a need for a better rate control algorithm, such as Packet-pair.

4.4. The Packet-pair scheme

Packet-pair is described in three stages. First, we motivate the algorithm. This is followed by a complete description and implementation details.

Motivation

Packet-pair is based on three observations:

(1) *The probing lemma allows a source to determine the bottleneck service rate by sending two packets at a rate faster than the bottleneck service rate, and measuring the inter-ack spacing.*

Consider a packet pair as it travels through the system, as shown in Figure 4.1. The figure presents a time diagram. Time increases down the vertical axis, and each axis represents a node along the path of a conversation. Lines from the source to the server show the transmission of a packet. The 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. After a packet arrives, it may be queued for a while before it receives service. This is represented by the space between the horizontal dotted lines, such as *de*.

In the packet-pair scheme, the source emits two back-to-back packets (at time s). These are serviced by the bottleneck; by definition, the inter-packet service time is s_b, the service time at the bottleneck. Since the acks preserve this spacing, the source can measure the inter-ack spacing to estimate s_b.

We now consider possible sources of error in the estimate. Server 1 also spaces out the back-to-back packets, so can it affect the measurement of s_b? 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 bottleneck server is serving other packets and may be delayed by a time interval such as *de*. Since this delay is shared by both packets in the pair, this does not affect the observation.

However, errors can be introduced by the fact that the acks may be spread out more (or less) than s_b due to differing queueing delays 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 reducing the estimate of s_b.

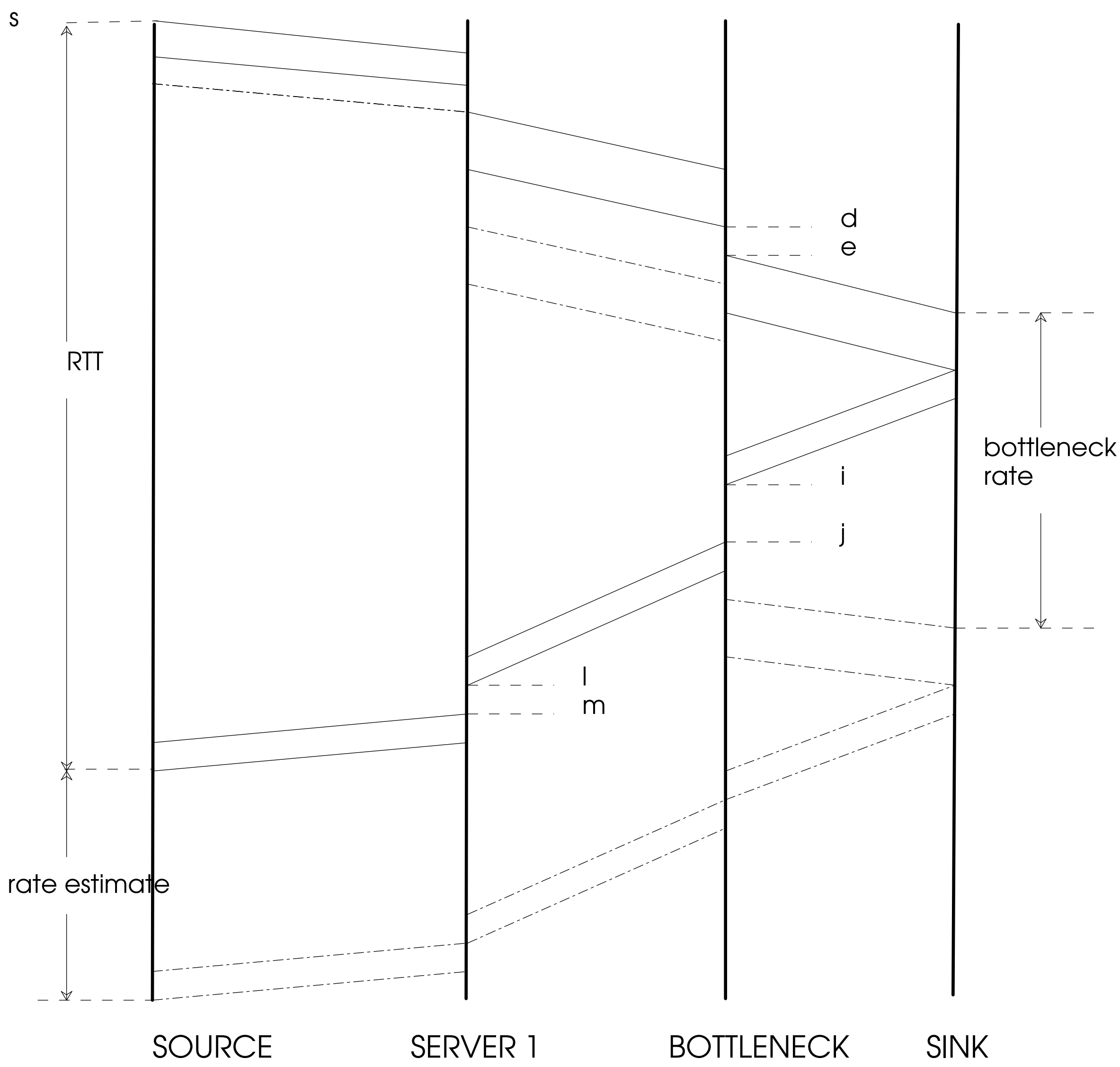

Figure 4.1: The packet-pair probing scheme

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 protocol (120). Measuring s_b 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 perturbation in the measurement.

The conclusion is that the estimate of the service rate made by the sender can be corrupted by noise. In the deterministic model described earlier, even if the server is busy when a packet arrives, queueing delays are assumed to be zero, and thus Lemma 5 proves that the source observes the service rate exactly. In reality, these small queueing delays can cause observation noise. In a later section, we show how the flow control mechanism accounts for this.

(2) *If a source has a rate allocation $1/s_b$ and a round trip propagation delay R, it operates optimally when it has R/s_b packets outstanding.*

The packets sent from a source and not yet acknowledged constitute a pipeline, in the sense that they are being 'processed' in parallel by the network. Then V, the pipeline depth, is given by $V = R/s_b$. A source should keep exactly V packets outstanding to fully utilize the bottleneck bandwidth, and simultaneously have zero queueing delay (64,97). V depends on R and s_b. In the model presented earlier, the values of these quantities are fixed, but, in reality, they could change with time, and so it is necessary periodically to measure them. The source can measure s_b using the packet-pair method described above. R, the propagation delay, is the time between sending out a packet and receiving an ack when all the queues along the path are empty. This can be approximated by measuring r_t, the round trip time, though r_t will have a component due to the queueing delay.

(3) *If the pipeline depth V can increase or decrease by at most ΔV in any interval of time r_t, then keeping ΔV packets in the bottleneck queue's buffers will ensure that the bottleneck will not be idle.*

If an increase in R or s_b increases the pipeline depth by ΔV, some bottleneck bandwidth will be unutilized until the source reacts to the change. Since a source takes at least r_t time units to react, the source should have enough packets in the buffer to take up any transients. If the bottleneck queues ΔV packets, when V increases, the buffer will be drained, and no loss of throughput will occur. Thus, Packet-pair tries to ensure that, at any given time, at least ΔV packets are present in the bottleneck queue. We assume a buffer capacity of at least $2\Delta V$ per conversation at every switching node.

Note that this scheme avoids wasted bandwidth but adds a queueing delay (on average $\Delta V s_b$) to every packet served. A user can adjust the targetted bottleneck queue size to obtain a range of delay versus bandwidth loss tradeoffs. To get a lower average queueing delay, the bottleneck queue size should be kept small, but this introduces the possibility of a bandwidth loss when the pipeline depth increases. If this loss is to be avoided, then ΔV packets should be kept in the queue, but this will also increase the average queueing delay. We denote the target bottleneck queue size by n_b, and in the rest of the chapter, n_b is assumed to be ΔV. In practice, users who desire low queueing delays should choose n_b to be close to zero, while those who desire bulk throughput should choose a larger value. This is discussed in Chapter 5.

Algorithm

There are three phases in the operation of Packet-pair: start-up, queue priming and normal transmission.

At start-up, the source does not know the value of s_b. Since it should not overload the bottleneck with packets, some sort of 'slow-start' is desirable. This can be combined this with an initial measurement of the conversation parameters by sending a *packet-pair*, two packets sent as fast as possible (back-to-back). The round-trip time of the first packet gives us R_e, an estimator for R and the inter-arrival time of the two packets gives us s_e, an estimator for s_b.

Once the source computes $V_e = R_e / s_e$, an estimator of V, it can decide what n_b should be. n_b should be chosen depending on the value of ΔV for the network. This value can be determined empirically or the administrator can choose this to tune protocol performance. Deciding n_b a priori is possible, but not desirable, since an administrator might want n_b to be some fraction of V_e. Thus, the decision about the value of n_b is deferred till the end of the first round-trip-time. During queue priming, the source sends out a burst of n_b back-to-back packets so that the n_b packets accumulate in the bottleneck queue.

During normal transmission the source transmits packet-pairs every $2s_e$ time units and updates s_e based on the inter-arrival time between paired acks. R_e, the estimate for R, is updated to $r_t - n_b s_e$, which accounts for the queueing delay. To react immediately to changes in V, the source recomputes V_e on the arrival of every pair.

Let V_{new}, V_{old} be the new and old values of V_e using the new and old estimates, respectively. If $V_{new} < V_{old}$, the source calculates

$$n_{skip} = \max(\lceil (V_{old} - V_{new})/2 \rceil, 0),$$

where $\lceil z \rceil$ is the smallest integer greater than or equal to z. The source then skips n_{skip} transmission slots with a duration of the new value of s_e, and then continues to send pairs of packets at regular intervals of $2s_e$. If $V_{new} > V_{old}$, the source immediately transmits a burst of $V_{new} - V_{old}$ back-to-back packets (these are specially marked as non-paired packets).

Implementation

Figure 4.2 is the state diagram for a Packet-pair implementation.

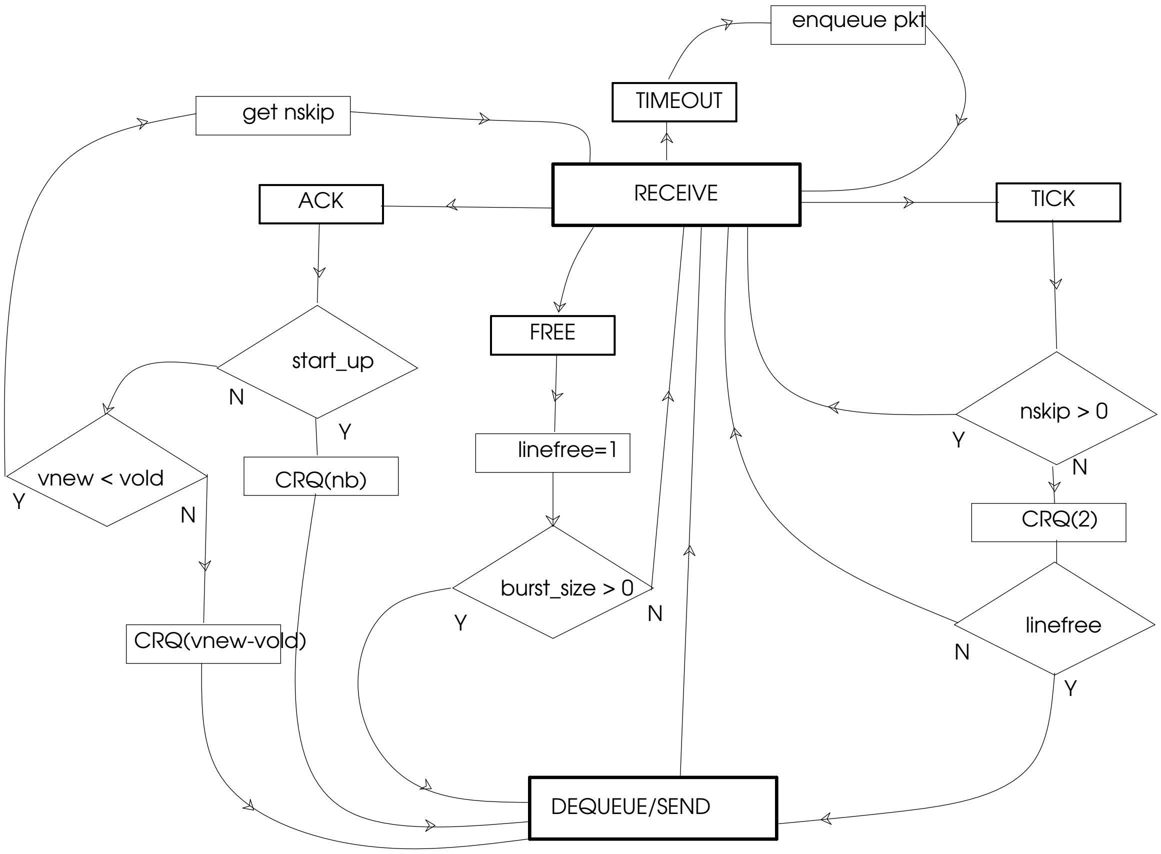

Figure 4.2: State diagram for implementing Packet-Pair
(CRQ(x) = accept x packets from user and enqueue)

A Packet-pair source usually is in the 'receive' state, waiting for an interrupt, one of

a) A signal indicating receipt of an acknowledgment packet. (ACK)
b) A 'tick' indicating that at the current sending rate, the next packet is due to be sent. (TICK)
c) A signal indicating that the output line is now free. (FREE)
d) A signal indicating that the last packet has timed out. (TIMEOUT)

There are two important state variables. `linefree` indicates that the output line from the source is free. `num_in_burst` is the number of packets enqueued in the output queue that belong to a burst. As long as `num_in_burst` is positive, a packet is dequeued and sent on the arrival of every FREE signal.

When an ACK signal arrives, if the ack is the first of a pair, R_e, the estimate for R, is updated. If it is the second of a pair, s_e is updated, and the source computes V_e and n_{skip}. If $V_{new} > V_{old}$, a burst of $V_{new} - V_{old}$ packets are queued on the output queue.

When an acknowledgment is received, some of the packets it acknowledges may have been timed out, and may be enqueued waiting to be sent out. So, at this point, all queued retransmissions that have become invalid are discarded. This implicitly assumes that retransmitted packets are queued at the transport layer. If a retransmitted packet has already been passed to a lower protocol layer, it will have to be retrieved from that layer, possibly violating layering. If this is a concern, this step can be ignored, since discarding retransmitted packets is not essential for the correct operation of the protocol.

If a TICK is received and n_{skip} is non-zero, it is decremented, the TICK timer is reloaded with $2s_e$, and we return to the receive state. Else, two packets (from the client) are enqueued on the output queue. If the line is free, one of the packets is dequeued and sent, else the source waits for an FREE to arrive. When an FREE arrives, if there are burst packets to be sent, one of them is dequeued and transmitted, else the source marks the line as free and returns to the receive state.

In current versions of TCP, there is a single retransmission timer, that is set on each packet transmission to $\hat{r}_t + 2MDM$ (ignoring some minor details). Here $\hat{r}_t$ is a moving average of the measured round trip time, and MDM is a moving average of the absolute difference between the measured round trip time and $\hat{r}_t$. When the timer goes off, the last unacknowledged packet is retransmitted. While we find this algorithm suitable, note that Packet-pair detects impending congestion using packet-pair probes, so, unlike TCP (63, 153), it is rather insensitive to the exact choice of the retransmission timer value. Thus, as a simplification, assuming one retransmission timer per packet, the retransmission timer value is simply Xr_t, where X is some small integer, and can be used as a tuning parameter (we used $X = 3$). On a TIMEOUT the timed out packet is placed in the output queue, waiting to be retransmitted. The new timeout value for the packet is twice the old value.

4.5. Analysis

We will analyze the the behavior of Packet-pair in the steady state (that is, when R and s_b do not change), and its response to transient changes in the virtual circuit. We will make four simplifying assumptions:

- Flow control is being done on behalf of an infinite source, that always has some data ready to send.
- Changes in V are bounded from above by ΔV, and the source knows or can estimate this value.
- Each server reserves $B \geq 2\Delta V$ buffers for each source.
- Transients are assumed to be due to a sharp, rather than a gradual, change in the system state. We assume that the value of a parameter, such as R, is constant until time t_0, at which point it changes discontinuously to its new value. We denote the value of $R(t)$ before the change as $R(t_0-\varepsilon)$, and after as $R(t_0+\varepsilon)$. We define $s_b(t_0-\varepsilon)$ and $s_b(t_0+\varepsilon)$ similarly.

Optimal flow control

We introduce the notion of optimal flow control and show that, in the steady state, Packet-pair is optimal. An optimal transmission flow control scheme should *always* operate at the knee of the load-throughput curve, so that maximum throughput is achieved with minimum delay (64). As the conditions at the server change, the source flow control must adapt itself to the change. However, if we consider the speed-of-light propagation delay in this control loop for wide-area networks, it is clear that no realizable flow control scheme can always operate at the knee. Hence, we propose a weaker definition of optimality that is suitable for high throughput applications.

Let the bottleneck have B buffer spaces available for each source. Then, a flow control scheme is optimal in the interval (T_0, T_1) if in every time interval $(t_1, t_2) \in (T_0, T_1)$, there are no buffer overflows, and there is no loss of bandwidth at the bottleneck node. To be precise, at the bottleneck node, if the buffer occupancy at time t_1 is k,

$$0 \le \int_{t_1}^{t_2} (\rho_0(t - d_1) - \rho_b(t))\, dt \le B - k$$

where $\rho_0(t)$ is the source sending rate at time t, $\rho_b(t)$ is the bottleneck service rate at time t and d_1 is the propagation delay from the source to the bottleneck.

Steady state behavior of Packet-pair

In the steady state, Packet-pair will keep n_b packets in the bottleneck queue, and send packets at exactly the service rate, μ. Proposition 1 proves the optimality of Packet-pair in the steady state.

Proposition 1:

Let the transmission at the source start at time T_0 and end at time T_1. If V is constant in (T_0, T_1), then Packet-pair is an optimal flow control scheme in $(T_0 + 2R(0), T_1)$.

Proof: At time $T_0 + R(0)$, the source knows μ. Since $\Delta V = 0$, we can set $n_b = 0$, and priming the queue is not necessary. Thus, the source will immediately start to send a packet-pair every $2s_b$ time units. The first pair reaches the bottleneck at the latest by $T_0 + 2R(0)$. Since service is at rate μ, there is no build up of the queue. Clearly, no bandwidth is lost, and optimality conditions are trivially satisfied in $(T_0 + 2R(0), T_1)$.

Remark

Note that many schemes described in the literature do not satisfy this weak notion of optimality even in the steady state. For example, the Jacobson-Karels version of TCP (63) drops packets if the maximum possible window size is larger than the buffer capacity at the bottleneck queue. The DECbit scheme keeps the queues at an average queue length of 1, and so will lose bandwidth when V increases (64). As we mentioned earlier, NETBLT causes queues to build up whenever V decreases, and they are never adjusted for. Hence, a sequence of decreases in V will cause NETBLT to drop packets. Jain's delay based scheme (65) will respond poorly if V decreases due to a decrease in R, since it interprets the decrease in delay as a signal to increase the window size, which will cause further queueing, and possible packet loss. A more detailed analysis of these schemes can be found in (129).

Response to transients

In our model, the only network parameters visible to a source are μ and r_t. Thus, a flow control scheme can react to a change only in either of these variables, and we will study the response of Packet-pair to these changes. For each change, we study the packet or bandwidth loss, and the time taken to return to steady state.

Note that r_t itself depends on R and on the queueing delay in the bottleneck node. In steady state, queueing delay is constant, and r_t changes only if R or μ change. Thus, we need only consider changes in R and μ. In either case, the effect is to change $V = R.\mu$. We will denote the time at which the change occurred by t_0, and the change in V by $\delta V \le \Delta V$.

4.5.1. Increase in propagation delay

4.5.1.1. Loss of bandwidth

R could increase if the VC is rerouted, and some new servers are added to the VC's path. There are two cases: either the path increase occurs before the bottleneck node, or after. If the increase happens after the bottleneck, then some server downstream of the bottleneck will have an increased idle time, and there is no loss in bottleneck bandwidth.

If the increase happens before the bottleneck, then for some period of time, the bottleneck could be idle, since packets that should have arrived at the bottleneck will now be sent to the new servers instead. Since we bound the increase in V by ΔV, if the bottleneck queue has ΔV packets, the buffered packets will be transmitted in the interim, and there will be no loss of bandwidth.

Another subtle possibility for bandwidth loss is if the first packet of a packet-pair reaches the bottleneck after time delay D_1 while the second one reaches there after a delay D_{1new}. The inter-arrival time of the pair at the bottleneck is then $\xi = D_{1new} - D_1$. If this is larger than s_b, the protocol will react by skipping some pairs of packets. However, Packet-pair will recover as soon as the next pair of acks arrive.

Proposition 2

The loss of bandwidth will be $2n + \xi / s_b$ packets, where

$$n = \max\left(\left\lceil \frac{1}{2}\left(\frac{R_{old}}{s_b} - \frac{R_{new}}{\xi}\right)\right\rceil, 0\right)$$

Proof:

The second term, ξ / s_b, is just the bubble size in the pipeline. Since Packet-pair mistakenly estimates that the pipeline depth is R_{new}/ξ, instead of R_{new}/s_b, it skips n slots, and in each slot it loses 2 packets, leading to a net loss of $2n$ packets.

4.5.1.2. Recovery time

Suppose R increased at time t_0. This information reaches the source at the latest by time $t_0 + R(t_0 + \varepsilon)$. The source will immediately send a burst of δV packets. Since the source is sending at rate μ, all but the bottleneck server will be idle when this burst is initiated. By the Burst dynamics lemma, we see that the last packet of the burst will arrive at the bottleneck at time $t_0 + \sum_{i=0}^{b-1} s_i + \frac{\delta V}{\rho_{sl_{b-1}}}$. This replenishes the bottleneck queue and so the steady state is regained. Subsequent increases in R are handled as before.

4.5.2. Increase in service rate

4.5.2.1. Loss of bandwidth

μ could increase if the number of conversations at the bottleneck decreases. The increase in μ could lead to some other node in the VC's path to become the bottleneck. We need to consider two cases, depending on whether the bottleneck migrates or not (that is, whether or not some other node becomes the bottleneck).

Assume that the bottleneck does not migrate. Due to the increase in μ, the bottleneck will serve packets faster than they arrive, and, until the first packet transmitted at the new rate arrives, there could be a loss of bandwidth. Now, the increase in μ becomes known to the source by time $t_0 + \sum_{i=b}^{n} s_i$, and the first packet from the burst reaches the server by $t_0 + R(t_0)$. So, if there are $(s_b(t_0+\varepsilon) - s_b(t_0-\varepsilon))R(t_0) \le \Delta V$ packets in the bottleneck buffer, there will be no loss of bandwidth.

If the bottleneck migrates downstream from the old bottleneck, then packets queued at the old bottleneck will arrive at the new bottleneck and will form a queue there. It is easy to show that the only loss of throughput happens for the brief interval where the first packet from the old bottleneck is in transit to the new bottleneck.

If the bottleneck migrates upstream, then the δV packets queued at the old bottleneck cannot compensate for the bubble in the pipeline. The new bottleneck will be idle till the first packet from the compensatory burst arrive, and the loss could be as large as δV packets. Note that no flow control algorithm can prevent this loss, since the source must take at least $R(t_0)$ time to react to the change in s_b. Thus we have shown that no feasible flow control algorithm can satisfy even our weak notion of optimality in non-steady state operation.

4.5.2.2. Recovery time

The recovery time is just the time for δV packets to accumulate at the bottleneck, and this is the same as in the case of the increase in R, that is $\sum_{i=0}^{b-1} s_i + \frac{\delta V}{\rho_{L_{b-1}}}$.

4.5.3. Decrease in propagation delay

4.5.3.1. Loss of packets

R could decrease if packets are sent through a shorter route, but through the same bottleneck. We will assume that the packets in transit are not lost, but are received at the bottleneck. The effect of this change is to put an additional δV packets in the buffers of the bottleneck queue. Since in the steady state there are ΔV packets in the bottleneck, and we assume that we can buffer $2\Delta V$ packets, there is no packet loss.

4.5.3.2. Recovery time

The source knows the reduced value of R at the latest by time $t_0+R(t_0 + \varepsilon)$. At this point, it will skip $(V_{old}-V_{new})/2$ transmission slots, taking a time equal to $1/2(R(t_0-\varepsilon)/s_b - R(t_0+\varepsilon)/sb)2sb$, $= R(t_0-\varepsilon) - R(t_0+\varepsilon)$. The first packet after resumption of normal transmission reaches the bottleneck latest by time $t_0+R(t_0-\varepsilon) - R(t_0+\varepsilon) + R(t_0-\varepsilon) = t_0 + R(t0-\varepsilon)$. Since the excess packets accumulated at the bottleneck are exactly $(R(t_0-\varepsilon)/s_b - R(t_0+\varepsilon)/s_b)$, they will all be cleared in this time, and the system will reach the steady state at time $t_0 + R(t_0)$

4.5.4. Decrease in service rate

4.5.4.1. Loss of packets

The source knows of the decrease in μ at the latest by $t_0 + R(t_0)$, and will skip $\delta V/2$ transmission slots. The bottleneck could accumulate an additional δV packets in this time. Since there are $2\Delta V$ buffers, there is no packet loss.

4.5.4.2. Recovery time

The source detects the decrease in μ by time $t_0 + \sum_{i=b}^{n} s_i$, and will skip some transmissions. The first packet sent at the new rate is received at the bottleneck after a time $\sum_{i=1}^{b-1} s_i$, so that the first packet arrives at the bottleneck at time $t_0 + R(t_0) + skiptime$. During the first $R(t_0)$ time units, the bottleneck queue will accumulate packets in excess of ΔV, but exactly these many packets will be serviced during $skiptime$. Hence, at $t_0 + R(t_0) + skiptime$ the steady state is attained.

The source skips transmission for the time during which the bottleneck services the excess packets accumulated in its queue. Hence,

$$skiptime = \max(\lceil \frac{1}{2} \frac{R(t_0)}{s_b(t_0-\varepsilon)} mi \frac{R(t_0)}{s_b(t_0+\varepsilon)} \rceil, 0) sb(t_0+\varepsilon)$$

4.6. Conclusions

This chapter models networks of FQ servers as a sequence of D/D/1 queues. In recent work (156), we have shown that the FQ discipline is similar to the Virtual Clock (154) and Delay-EDD (37) service disciplines. Thus, this modeling approach may be applied to networks of Virtual Clock and Delay-EDD servers as well. The network model initially assumes that the bottleneck service rate is constant. Later, this assumption is relaxed, and infrequent, single sharp changes in the bottleneck service rate are allowed. Some lemmas about the model are proved, which motivates the design of the Packet-pair flow control scheme. The detailed design of Packet-pair, as well as the state diagram of an implementation are presented. The Packet-pair protocol has several advantages over other flow control protocols: it responds quickly to changes in the network state, it takes advantage of FQ routers to probe network state, and it does not require

Change	Bandwidth loss	Packet loss	Recovery time
Increase in propagation delay	None	None	$\leq t_0 + R(t_0 + \varepsilon) + \sum_{i=0}^{b-1} s_i + \frac{\delta V}{\rho_{sl_{b-1}}}$
Increase in bottleneck service time	0, if no bottleneck migration $\leq \delta V$ if bottleneck migrates	None	As above
Decrease in propagation delay	None	None	$\leq t_0 + R(t_0)$
Decrease in bottleneck service time	None	None	$\leq t_0 + R(t_0) + s_b(t_0+\varepsilon)$ $\max(\lceil \frac{R(t_0)}{2s_b(t_0-\varepsilon)} - \frac{R(t_0)}{s_b(t_0+\varepsilon)} \rceil, 0)$

Table 4.1: Summary of analytic results

any assistance from routers, such as bit-setting. A similar approach to passively probing the network, though not using packet pairs, is described in reference (53), where each probe packet is time stamped, and the time series of delays suffered the probes is used to obtain a congestion indicator. Then, the packet sending rate is adjust to be proportional to the inverse of the level of congestion in the network. However, the Packet-pair protocol is not without its limitations. First, it requires that all the packet routers in the network implement Fair Queueing or a similar round-robin-like discipline. This is not necessary for flow control schemes such as the one in TCP (63, 109) or Jain's delay-based approach (65). Since most current networks do not implement FQ, our approach is of limited practical significance, but it is hoped that this situation will change in the future. Second, Packet-pair assumes that changes in the bottleneck service rate happen slower than one round trip time. This is true if the conversations are long lived, and not too bursty. This is plausible if most of the traffic consists of fairly smooth (perhaps, uncompressed video) streams. However, if the number of active conversations can change drastically over the time scale of one round trip time, then Packet-pair is inadequate. We address this in Chapter 5, where we use a formal control-theoretic approach to flow control, and propose a hybrid flow control scheme that integrates window and rate-based flow control. Buffer reservations at each packet switch ensures that even if the flow control scheme incorrectly estimates the bottleneck service rate, there is no packet loss. With these changes, the packet-pair scheme performs well in FQ networks even if the bottleneck service rate changes rapidly and drastically.

4.7. Appendix 4.A : Notation

The following notation is used throughout the paper. Since a single conversation is studied, it is implicitly assumed that the variables are subscripted with the conversation identifier. The time dependencies are usually ignored in the text.

$s_i(t)$: service time at the ith server in the path.
$\rho_i(t)$: service rate at the i server, $\rho_i(t) = 1/s_i(t)$.
$s_b(t)$: service time at the bottleneck server.
$\mu(t)$: bottleneck service rate $= 1/s_b(t)$.
$s_e(t)$: estimator for $s_b(t)$.
sl_i: ith rate throttle in the path.
Δ_j: $\rho_{sl_{j-1}} - \rho_{sl_j}$
$R(t)$: round trip propagation delay (excluding queueing delays).
$r_t(t)$: $R(t)$ + queueing delay
$R_e(t)$: estimator for R.
$V(t)$: pipeline depth $= R/s_b$.
$V_e(t)$: estimator for $V = R_e/s_e$.
δV: actual change in V.
$\rho_0(t)$: source sending rate
$R(t_0-\varepsilon)$: $R(t)$ before a change at time t_0.
$R(t_0+\varepsilon)$: $R(t)$ after a change at time t_0.
$s_b(t_0-\varepsilon)$: $s_b(t)$ before a change at time t_0
$s_b(t_0+\varepsilon)$: $s_b(t)$ after a change at time t_0
n_b: desired number of packets in the bottleneck buffer

(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 (34,154). endpoints, for example, two BSD sockets (21,23). *Virtual Circuits* (VCs) (such as in Datakit (41)). not make such demands (34). (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). (37,48). (34,68). 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), (126,137,138). Examples are the Virtual Clock scheme (154), Delay-EDD (37) 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 (156) 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 (156). 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), (37), 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 (36). at the time of call set-up, and doing admission control (37). (37). (34). their data can be dropped or violation-tagged (5). (87,128,136,141). In the Flow network, the Virtual Clock mechanism is used (154), distributed flow control (37). 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). (37,142). 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 (156). 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). (50,57,89,93,125,126). 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). (144,145). analyzed in (129). (143,145). 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 (156), we have shown that the FQ discipline is similar to the Virtual Clock (154) and Delay-EDD (37) 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. (6,9,10,27,126). 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 (138) Vakil and Lazar (137) 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 (157). 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), (63,127,154,157), 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 (141).