Chapter 2: Fair Queueing

2.1. Introduction

Datagram networks have long suffered from performance degradation in the presence of congestion (44). The rapid growth, in both use and size, of computer networks has sparked a renewed interest in methods of congestion control. These methods have two points of implementation. The first is at the source, where flow control algorithms vary the rate at which the source sends packets. Flow control algorithms are designed primarily to ensure the presence of free buffers at the destination host, but we are more concerned with their role in limiting the overall network traffic, and in providing users with maximal utility from the network.

The second point of implementation is at the switches. Congestion can be controlled at the switches through routing and queueing algorithms. Adaptive routing, if properly implemented, lessens congestion by routing packets away from network bottlenecks. Queueing algorithms, which control the order in which packets are sent and the usage of the switch's buffer space, determine the way in which packets from different sources interact with each other. This, in turn, affects the collective behavior of flow control algorithms. We argue that this effect, which is often ignored, makes queueing algorithms a crucial component in effective congestion control.

Queueing algorithms can be thought of as allocating three nearly independent quantities: bandwidth (*which* packets get *transmitted*), promptness (*when* do those packets get *transmitted*), and buffer space (*which* and *when* packets get *discarded* by the switch). Currently, the most common queueing algorithm is first-come-first-serve (FCFS). In this scheme, the order of arrival completely determines the bandwidth, promptness, and buffer space allocations, inextricably intertwining these three allocation issues. Since each user may have a different preference for each allocated quantity, FCFS queueing cannot provide adequate congestion management.

There may be flow control algorithms that can, when universally implemented in a network with FCFS switches, overcome these limitations and provide reasonably fair and efficient congestion control. However, with today's diverse and decentralized computing environments, it is unrealistic to expect universal implementation of any given flow control algorithm. This is not merely a question of standards, but also one of compliance. First, even if a universal standard such as ISO (12) were adopted, malfunctioning hardware and software could violate the standard. Second, there is always the possibility that individuals would alter the algorithms on their own machine to improve their performance at the expense of others. Consequently, congestion control algorithms should function well even in the presence of ill-behaved sources.

Unfortunately, irrespective of the flow control algorithm used by the well-behaved sources, networks with FCFS switches do not have this property. A single source, sending packets to a switch at a sufficiently high speed, can capture an arbitrarily high fraction of the bandwidth of the outgoing line. Thus, FCFS queueing is not adequate; more discriminating queueing algorithms must be used in conjunction with source flow control algorithms to control congestion effectively in noncooperative environments.

Following a similar line of reasoning, Nagle (101) proposed a *fair queueing* (FQ) algorithm in which switches maintain separate queues for packets from each individual source. The queues are serviced in a round-robin manner. This prevents a source from arbitrarily increasing its share of the bandwidth or the queueing delay received by the other sources. In fact, when a source sends packets too quickly, it merely increases the length of its own queue. Nagle's algorithm, by changing the way packets from different sources interact, does not reward, nor leave sources vulnerable to, anti-social behavior. This proposal appears to have considerable merit, and this chapter describes a modification of Nagle's scheme and explores its implications in some depth.

The three different components of congestion control algorithms introduced above, source flow control, switch routing, and switch queueing algorithms, interact in interesting and complicated ways. It is impossible to assess the effectiveness of any algorithm without reference to the other components of congestion control in operation. We will evaluate our proposed queueing algorithm in the context of static routing and several widely used flow control algorithms. The aim is to find a queueing algorithm that functions well in current computing environments. The algorithm might, indeed it should, *enable* new and improved routing and flow control algorithms (as shown in Chapters 4 and 5), but it must not require them.

In circuit switched networks, where there are explicit buffer reservation and uniform packet sizes, it has been established that round robin service disciplines allocate bandwidth fairly (55,72). Recently Morgan (98) has examined the role such queueing algorithms play in controlling congestion in circuit switched networks; while his application context is quite different from ours, his conclusions are qualitatively similar. In other related work, the Datakit queueing algorithm combines round robin service and FIFO priority service, and has been analyzed extensively (42,86). Similar work has been presented by Zhang (154); her *Virtual Clock* switch queueing algorithm is essentially identical to the fair queueing algorithm presented here (157). Zhang analyzes this algorithm in the context of a proposed resource reservation scheme, the *Flow Network*, whereas we do not consider resource reservation.

2.2. Fair Queueing

2.2.1. Motivation

What are the requirements for a queueing algorithm that will allow source flow control algorithms to provide users with adequate utility even in the presence of ill-behaved sources? We start with Nagle's observation that a queueing algorithm must provide *protection*, so that ill-behaved sources can only have a limited negative impact on well-behaved sources. Allocating bandwidth and buffer space in a *fair* manner, to be defined later, automatically ensures that ill-behaved sources can get no more than their fair share. This led us to adopt, as our central design consideration, the requirement that the queueing algorithm allocate bandwidth and buffer space fairly. Ability to control the promptness, or delay, allocation somewhat independently of the bandwidth and buffer allocation is also desirable. Finally, we require that the switch should provide service that, in some sense, does not depend discontinuously on a packet's time of arrival (this continuity condition will be made precise when we define our algorithm). This continuity requirement attempts to prevent the efficiency of source flow control implementations from being overly sensitive to timing details (timers are the Bermuda Triangle of flow control algorithms).

Nagle's proposal does not satisfy these requirements. The most obvious flaw is its lack of consideration of packet lengths. A source using long packets gets more bandwidth than one using short packets, so bandwidth is not allocated fairly. Also, the proposal has no explicit promptness allocation other than that provided by the round-robin service discipline. In addition, the static round robin ordering violates the continuity requirement. These defects are corrected in our version of fair queueing, which we define after first discussing our definition of fairness.

In stating our requirements for queueing algorithms, we have left the term *fair* undefined. The term *fair* has a clear colloquial meaning, but it also has a technical definition (actually several, as discussed in Chapter 1, but only one is considered here). Consider, for example, the allocation of a single resource among N users. Assume there is an amount μ_{total} of this resource, and that each of the users requests an amount ρ_i and, under a particular allocation, receives an amount μ_i. What is a fair allocation? The max-min fairness criterion (43,55,115) states that an allocation is fair if (1) no user receives more than its request, (2) no other allocation scheme satisfying condition 1 has a higher minimum allocation, and (3) condition 2 remains recursively true as we remove the minimal user and reduce the total resource accordingly, i.e. $\mu_{total} \leftarrow \mu_{total} - \mu_{\min}$. This condition reduces to $\mu_i = MIN(\mu_{fair}, \rho_i)$ in the simple example, with μ_{fair}, the *fair share*, being set so that $\mu_{total} = \sum_{i=1}^{N} \mu_i$. This concept of fairness easily generalizes to the multiple resource case (115). Note that implicit in the max-min definition of fairness is the assumption that the users have equal *rights* to the resource.

In this application, the bandwidth and buffer demands are clearly represented by the packets that arrive at the switch. (Demands for promptness are not explicitly communicated, and we return to this issue later.) However, it is not clear what constitutes a *user*. The user associated with a packet could refer to the source of the packet, the destination, the source-destination pair, or even refer to an individual process running on a source host. Each of these definitions has limitations. Allocation per source unnaturally restricts sources such as file servers, which typically consume considerable bandwidth. Ideally, the switches could know that some sources deserve more bandwidth than others, but there is no adequate mechanism for establishing that knowledge in today's networks. Allocation per receiver allows a receiver's useful incoming bandwidth to be reduced by a broken or malicious source sending unwanted packets to it. Allocation per process on a host encourages human users to start several processes communicating simultaneously, thereby evading the original intent of fair allocation. Allocation per source-destination pair allows a malicious source to consume an unlimited amount of bandwidth by sending many packets all to different destinations. While this does not allow the malicious source to do useful work, it can prevent other sources from obtaining sufficient bandwidth.

Overall, allocation on the basis of source-destination pairs, or *conversations*, seems the best tradeoff between security and efficiency and will be used here. However, our treatment will apply to any of these interpretations of the notion of user. Given the requirements for an adequate queueing algorithm, coupled with the definitions of *fairness* and *user*, we now turn to the description of our fair queueing algorithm.

2.2.2. Definition

It is simple to allocate buffer space fairly by dropping packets, when necessary, from the conversation with the largest queue. Allocating bandwidth fairly is less straightforward. Pure round-robin service provides a fair allocation of packets sent, but fails to guarantee a fair allocation of bandwidth because of variations in packet sizes. To see how this unfairness can be avoided, we first consider a hypothetical service discipline where transmission occurs in a bit-by-bit round robin (BR) fashion (as in a head-of-queue processor sharing discipline). This service discipline allocates bandwidth fairly since at every instant in time each conversation is receiving its fair share. Let $R(t)$ denote the number of rounds made in the round-robin service discipline up to time t ($R(t)$ is a continuous function, with the fractional part indicating partially completed rounds). Let $N_{ac}(t)$ denote the number of active conversations, i.e. those that have bits in their queue at time t. Then, $\frac{\partial R}{\partial t} = \frac{\mu}{N_{ac}(t)}$, where μ is the speed of the switch's outgoing line (we will, for convenience, work in units such that $\mu = 1$). A packet of size P whose first bit gets serviced at time t_0 will have its last bit serviced P rounds later, at time t such that $R(t) = R(t_0)+P$. Let t_i^α be the time that packet i belonging to conversation α arrives at the switch, and define the numbers S_i^α and F_i^α as the values of $R(t)$ when the packet started and finished service. With P_i^α denoting the size of the packet, the following relations hold: $F_i^\alpha = S_i^\alpha + P_i^\alpha$ and $S_i^\alpha = MAX(F_{i-1}^\alpha, R(t_i^\alpha))$. Since $R(t)$ is a strictly monotonically increasing function whenever there are bits waiting to be sent at the switch, the ordering of the F_i^α values is the same as the ordering of the finishing times of the various packets in the BR discipline.

Sending packets in a bit-by-bit round robin fashion, while satisfying our requirements for an adequate queueing algorithm, is obviously unrealistic. We hope to emulate this impractical algorithm by a practical packet-by-packet transmission scheme. Note that the functions $R(t)$ and $N_{ac}(t)$ and the quantities S_i^α and F_i^α depend only on the packet arrival times t_i^α and not on the actual packet transmission times, as long as we define a conversation to be active whenever $R(t) \le F_i^\alpha$ for $i = MAX(j \mid t_j^\alpha \le t)$. We are thus free to use these quantities in defining our packet-by-packet transmission algorithm. A natural way to emulate the bit-by-bit round-robin algorithm is to let the quantities F_i^α define the sending order of the packets. Our packet-by-packet transmission algorithm is simply defined by the rule that, whenever a packet finishes transmission, the next packet sent is the one with the smallest value of F_i^α. The continuity requirement mentioned earlier can be restated precisely as demanding that the relative transmission priorities depend continuously on the packet arrival times. The fact that the F_i^α's depend continuously on the t_i^α's means that our algorithm satisfies this continuity requirement.

In a preemptive version of this algorithm, newly arriving packets whose finishing number F_i^α is smaller than that of the packet currently in transmission preempt the transmitting packet. For practical reasons, we have implemented the nonpreemptive version, but the preemptive algorithm (with resumptive service) is more tractable analytically. Clearly the preemptive and nonpreemptive packetized algorithms do not give the same instantaneous bandwidth allocation as the BR version. However, for each conversation the total bits sent at a given time by these three algorithms are always within P_{max} of each other, where P_{max} is the maximum packet size (this emulation discrepancy bound is proved in reference (50)). Thus, over sufficiently long conversations, the packetized algorithms asymptotically approach the fair bandwidth allocation of the BR scheme.

Recall that a user's request for promptness is not made explicit. The IP protocol (108) does have a field for type-of-service, but not enough applications make intelligent use of this option to render it a useful hint. Consequently, promptness allocation must be based solely on data already available at the switch. One such allocation strategy is to give more promptness (less delay) to users who utilize less than their fair share of bandwidth. Separating the promptness allocation from the bandwidth allocation can be accomplished by introducing a nonnegative parameter δ, and defining a new quantity, the *bid* B_i^α, as $B_i^\alpha = P_i^\alpha + MAX(F_{i-1}^\alpha, R(t_i^\alpha) - \delta)$. The quantities $R(t)$, $N_{ac}(t)$, F_i^α, and S_i^α remain as before, but now the sending order is determined by the B's, not the F's. The asymptotic bandwidth allocation is independent of δ, since the F's control the bandwidth allocation, but the algorithm gives slightly faster service to packets belonging to an inactive conversation. The parameter δ controls the extent of this additional promptness. Note that the bid B_i^α is continuous in t_i^α, so that the aforementioned continuity requirement is met.

The role of this term δ can be seen more clearly by considering the two extreme cases $\delta = 0$ and $\delta = \infty$. If an arriving packet has $R(t_i^\alpha) \le F_{i-1}^\alpha$, then the conversation α is active (i.e. the corresponding conversation in the BR algorithm would have bits in the queue). In this case, the value of δ is irrelevant and the bid number depends only on the finishing number of the previous packet. However, if $R(t_i^\alpha) > F_{i-1}^\alpha$, so that the α conversation is inactive, the two cases are quite different. With $\delta = 0$, the bid number is given by $B_i^\alpha = P_i^\alpha + R(t_i^\alpha)$, and is completely independent of the previous history of user α. With $\delta = \infty$, the bid number is $B_i^\alpha = P_i^\alpha + F_{i-1}^\alpha$ and depends only the previous packet's finishing number, no matter how many rounds ago. For intermediate values of δ, scheduling decisions for packets of inactive conversations depends on the previous packet's finishing round as long as it was not too long ago, and δ controls how far back this dependence goes.

Recall that when the queue is full and a new packet arrives, the last packet from the conversation currently using the most buffer space is dropped. We have chosen to leave the quantities F_i^α and S_i^α unchanged when we drop a packet. This provides a small penalty for ill-behaved hosts, in that they will be charged for throughput that, because of their own poor flow control, they could not use. Recent work (57) raises questions about the desirability of this aspect of our algorithm.

2.2.3. Performance

The desired bandwidth and buffer allocations are completely specified by the definition of fairness, and we have demonstrated that our algorithm achieves those goals. However, we have not been able to characterize the promptness allocation for an arbitrary arrival stream of packets. To obtain some quantitative results about the promptness, or delay, performance of a single FQ switch, we consider a very restricted class of arrival streams in which there are only two types of sources. There are FTP-like file transfer sources, which always have ready packets and transmit them whenever permitted by the source flow control (which, for simplicity, is taken to be sliding window flow control), and there are Telnet-like interactive sources, which produce packets intermittently according to some unspecified generation process. What are the quantities of interest? An FTP source is typically transferring a large file, so the quantity of interest is the transfer time of the file, which for asymptotically large files depends only on the bandwidth allocation. Given the configuration of sources this bandwidth allocation can be computed *a priori* by using the fairness property of FQ switches. The interesting quantity for Telnet sources is the average delay of each packet, and it is for this quantity that we now provide a rather limited result.

Consider a single FQ switch with N FTP sources sending packets of size P_F, and allow a single packet of size P_T from a Telnet source to arrive at the switch at time t. It will be assigned a bid number $B = R(t)+P_T-\delta$; thus, the dependence of the queueing delay on the quantities P_T and δ is only through the combination $P_T-\delta$. We will denote the queueing delay of this packet by $\phi(t)$, which is a periodic function with period NP_F. We are interested in the average queueing delay Δ

$$\Delta \equiv \frac{1}{NP_F} \int_0^{NP_F} \phi(t)\,dt$$

The finishing numbers F_i^α for the N FTP's can be expressed, after perhaps renumbering the packets, by $F_i^\alpha = (i+l^\alpha)P_F$ where the l's obey $0 \le l^\alpha < 1$. The queueing delay of the Telnet packet depends on the configuration of l's whenever $P_T < P_F$. One can show that the delay is bounded by the extremal cases $l^\alpha = 0$ for all α and $l^\alpha = \alpha/N$ for $\alpha = 0,1,\ldots,N-1$. The delay values for these extremal cases are straightforward to calculate; for the sake of brevity we omit the derivation and merely display the result below. The average queueing delay is given by $\Delta = A(P_T-\delta)$, where the function $A(P)$, the delay with $\delta = 0$, is defined below (with integer k and small constant ε, $0 \le \varepsilon < 1$, defined via $P_T = P_F(k+\varepsilon)/N$).

Preemptive

$$A(P) = N\left(P - \frac{P_F}{2}\right) \qquad \text{for} \qquad P \ge P_F$$

$$N\left(P - \frac{P_F}{2}\right) \le A(P) \le \frac{NP^2}{2P_F} \quad \text{for} \quad P_F \ge P \ge \frac{P_F}{2}\left(1 + \frac{1}{N}\right)$$

$$\frac{1}{2P_F}\left(\frac{P_F}{2} + N\left(P - \frac{P_F}{2}\right)\right)^2 \le A(P) \le \frac{NP^2}{2P_F} \quad \text{for} \quad \frac{P_F}{2}\left(1 + \frac{1}{N}\right) \ge P \ge \frac{P_F}{2}\left(1 - \frac{1}{N}\right)$$

$$0 \le A(P) \le \frac{NP^2}{2P_F} \qquad \text{for} \qquad \frac{P_F}{2}\left(1 - \frac{1}{N}\right) \ge P$$

Nonpreemptive

$$A(P) = N\left(P - \frac{P_F}{2}\right) \qquad \text{for} \qquad P \ge P_F$$

$$N\left(P - \frac{P_F}{2}\right) \le A(P) \le \left(\frac{P_F}{2}\right)\left\{1 + \frac{1}{N}\left(k^2 + k(2\varepsilon-1)\right)\right\} \quad \text{for} \quad P_F \ge P \ge \frac{P_F}{2}\left(1 + \frac{1}{N}\right)$$

$$\frac{P_F}{2} \le A(P) \le \left(\frac{P_F}{2}\right)\left\{1 + \frac{1}{N}\left(k^2 + k(2\varepsilon-1)\right)\right\} \quad \text{for} \quad \frac{P_F}{2}\left(1 + \frac{1}{N}\right) \ge P$$

A more detailed analysis of the single server case can be found in reference (23). What happens in a network of FQ switches? There are few analytical results here, but Hahne (55) has shown that for strict round robin service switches and only FTP sources there is fair allocation of bandwidth (in the multiple resource sense) when the window sizes are sufficiently large. She also provides examples where insufficient window sizes, but much larger than the pipeline depth of the communication path, can result in unfair allocations. It can be shown that both of these properties hold for our fair queueing scheme.

Chapters 4 and 5 analyze networks of FQ switches from the point of view of a single conversation. While this analysis cannot explain overall network dynamics, it is sufficient to model and control a single conversation. Simulation results for networks of FQ switches are presented in Chapter 6.

2.3. Discussion

In an FCFS switch, the queueing delay of packets is, on average, uniform across all sources and directly proportional to the total queue size. Thus, achieving ambitious performance goals, such as low delay for Telnet-like sources, or even mundane ones, such as avoiding dropped packets, requires coordination among all sources to control the queue size. Having to rely on source flow control algorithms to solve this control problem, which is extremely difficult even in a maximally cooperative environment and impossible in a noncooperative one, merely reflects the inability of FCFS switches to distinguish between users and to allocate bandwidth, promptness, and buffer space independently.

In the design of the fair queueing algorithm, we have attempted to address these issues. The algorithm does allocate the three quantities separately. Moreover, the promptness allocation is not uniform across users and is somewhat tunable through the parameter δ. Most importantly, fair queueing creates a firewall that protects well-behaved sources from their uncouth brethren. Not only does this allow the current generation of flow control algorithms to function more effectively, but it creates an environment where users are rewarded for devising more sophisticated and responsive algorithms. The game-theoretic issue first raised by Nagle, that one must change the rules of the switch's game so that good source behavior is encouraged, is crucial in the design of switch algorithms (101). A formal game-theoretic analysis of a simple switch model (an exponential server with N Poisson sources) suggests that fair queueing algorithms make self-optimizing source behavior result in fair, protective, nonmanipulable, and stable networks; in fact, they may be the only reasonable queueing algorithms to do so (124). Our calculations show that the fair queueing algorithm is able to deliver low delay to sources using less than their fair share of bandwidth, and that this delay is insensitive to the window sizes being used by the FTP sources.

The protection property of the FQ algorithm enables a new class of flow control algorithm. Since each user gets a fair share of the network bandwidth, the stream of acknowledgments received by the source, is, to a first approximation, dependent only on that source's own behavior. Hence, by monitoring the acknowledgment stream, the source can optimize its sending rate regardless of the behavior of the other sources. This idea is explained in greater detail in Chapters 4 and 5.

In this chapter we have compared our fair queueing algorithm with only the standard first-come-first-serve queueing algorithm. We know of three other widely known queueing algorithm proposals. The first two were not intended as a general purpose congestion control algorithms. Prue and Postel (113) have proposed a type-of-service priority queueing algorithm, but allocation is not made on a user-by-user basis, so fairness issues are not addressed. There is also the Fuzzball selective preemption algorithm (94,95) whereby the switches allocate buffers fairly (on a source basis, over all of the switch's outgoing buffers). This is very similar to our buffer allocation policy, and so can be considered a subset of our FQ algorithm. The Fuzzballs also had a form of type-of-service priority queueing but, as with the Prue and Postel algorithm, allocations were not made on a user-by-user basis. The third policy is the Random-Dropping (RD) buffer management policy in which the service order is FCFS, but when the buffer is overloaded, the packet to be dropped is chosen at random (90). This algorithm tends to allocate bandwidth to cooperative sources more or less evenly. However, it has been shown that the RD algorithm does not provide max-min fair bandwidth allocation, is vulnerable to ill-behaved sources, and is unable to provide reduced delay to conversations using less than their fair share of bandwidth (39,56,125,154).

There are two objections that have been raised in conjunction with fair queueing. The first is that some source-destination pairs, such as file server or mail server pairs, need more than their fair share of bandwidth. This can achieved by weighted fair queueing, described below. Assign each source-destination pair a number n_α which represents how many queue slots that conversation gets in the bit-by-bit round robin. We now redefine N_{ac} as $N_{ac} = \sum n_\alpha$ with the sum over all active conversations, and P_i^{α} as $1/n_{\alpha}$ times the true packet length. With these changes, the earlier algorithm allocates each user a share of the bandwidth proportional to its weight. Of course, the truly vexing problem is the politics of assigning the n_{α}. Note that, while we have described an extension that provides for different relative shares of bandwidth, one could also define these shares as absolute fractions of the bandwidth of the outgoing line. This would guarantee a minimum level of service for these sources, and is very similar to the *Virtual Clock* algorithm of Zhang (154).

The other objection is that fair queueing requires the switches to be smart and fast. There is the technological question of whether or not one can build FQ switches that can match the bandwidth of fibers. If so, are these switches economically feasible? Work by Restrick and Kalmanek at AT&T Bell Laboratories has shown that it is possible to build fair queueing servers that switch ATM cells at 1.2 Gbps (69). This indicates that building smarter switches does not necessarily have to make them slower.

(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 (35,154). endpoints, for example, two BSD sockets (21,23). *Virtual Circuits* (VCs) (such as in Datakit (41)). not make such demands (35). (68), increasing the load on the bottleneck (101). (63,114). standstill (132). The first definition is used in references (66,117), the second in reference (63), and the third in reference (85). over the resources (140). (36,47). (35,68). 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), (1,38,66,79,133,137,138). (63,117), (126,137,138). Examples are the Virtual Clock scheme (154), Delay-EDD (36) and Stop-and-Go queueing (46). game (124). (6), Sanders (121-123), Douligeris and Majumdar (25-27) and Lazar *et al* (59). several orders of magnitude larger (84). and perhaps have catastrophic effects (101,132). (112) themselves overload the congested network. of the recent past (45000 kbps to 56 kbps) (69). network allocates its resources (23). (43,55,106,107,115,139,149). criterion to adopt (44). Nevertheless, it is necessary to negotiation and enforcement mechanism (20,155). e.g. (63). the switches (117). Still others assume complete control but no control over source traffic (54). having hardware support for rapid switching (100) optimizing for the average case (18) removing signaling information from the data path (41) choice of scheduling algorithm (157) an efficient call processing architecture (131). peer entities (for example, (117)) For example, a Fair Queueing (23) switch has the responsibility for (66)). 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 (66). 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 (75), retransmission timers (153), throughput monitoring (147), or delay monitoring (66) such as a hardware problem (153). (14,110). The Loss-load curve scheme (148) and the selective DECbit scheme (115) seek to attention in the literature, and is surveyed in (44). Other schemes, such as Input Buffer Limit (81), Channel Queue Limit (70), Buffer Class (45) and Matsumoto's X.75 proposal (91) packet switches (100). reservation-oriented networks (157). Some exceptions are in the DEC scheme (115), the Loss-load curve approach (148) and the Random dropping scheme (90). scheme does not reduce congestion (90). (63,88). either using a window adjustment scheme (117) or a rate adjustment scheme (17,75). 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), (36), by Topolcic *et al* in the ST-II proposal (134) and by Cidon *et al* for the PARIS network (16). In a hybrid scheme described in reference (24) The two-window scheme proposed by Hahne *et al* (54) this requirement can be reduced even further (96). user to obey some traffic profile (37). at the time of call set-up, and doing admission control (36). (36). (35). 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 (36). Ferrari and Verma (34) 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). (46,48,68). (36,142). 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 (157). The max-min fairness criterion (43,55,115) (115). (this emulation discrepancy bound is proved in reference (50)). The IP protocol (108) does have a field for type-of-service, Recent work (57) raises questions about the { 1~+~{1 over {N}} ({k sup 2} ~+~ k(2 epsilon -1)) right }~~for~~{P sub F} left { 1~+~{1 over {N}} ({k sup 2} ~+~ k(2 epsilon -1)) right } found in reference (23). There are few analytical results here, but Hahne (55) is crucial in the design of switch algorithms (101). queueing algorithms to do so (124). Prue and Postel (113) (94,95) the packet to be dropped is chosen at random (90). (39,56,125,154). (154). at 1.2 Gbps (69). (50,57,89,93,124,126). such as NetBlt (17), and the technology for this is well known (65). (93). since N_{ac} is strictly non-increasing in (t_{chk}, t). has been proposed (57). (58,78). traffic in the sense of (57), TCP protocol (109) with the modifications made by Jacobson (63). we used the REAL network simulator (73) Along with the work done by McKenney (93) and Heybey et al (57), this work provides the practitioner Queueing, such as the Virtual Clock algorithm (154). `i_min[k]` is the position in `max` of the swap(min(a), min(b)); swap(i_max(i_min(a)), i_max(i_min(b))); swap(i_min(a), i_min(b)); pointer consistency, i.e. that `i_min[i_max[a]] = a` and `i_max[i_min[a]] = a`. `i_min[n] = i_max[n] = n` . min(ptr) = data; for (; (ptr/2 >= 1) &&(min(ptr) < min (ptr/2)); ptr /=2) min(1) = INFINITY; save = i_min(1); max(save) = -1; if (min(a) < min(a/2)) smaller = a/2; smaller_son = (min(lson(a)) < min(rson(a))) ? lson(a) : rson(a); if (min(smaller_son) < min(a)) smaller = smaller_son; attendant inefficiencies (50). of Bessel functions (135); of data traffic in current networks (13,51), (129). (143,145). analyzed in (129). (144,145). flow control scheme (17). scheme modified an idea described by Clark et al. for NETBLT (17), and sent to the source using a state exchange protocol (120). (64,97). so, unlike TCP (63,153), (64). (T_0, T_1) if in every time interval $(t_1, t_2) \in (T_0, T_1)$, $(T_0 + 2R(0), T_1)$. conditions are trivially satisfied in $(T_0 + 2R(0), T_1)$. For example, the Jacobson-Karels version of TCP (63) and so will lose bandwidth when V increases (64). Jain's delay based scheme (66) will respond (129). In recent work (157), we have shown that the FQ discipline is similar to the Virtual Clock (154) and Delay-EDD (36) service disciplines. though not using packet pairs, is described in reference (53), (63,109) or Jain's delay-based approach (66). 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 (76). 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 (156). experimental high speed network testbed (69). lk1 eq 1 over { RTT(k)} (B/2 mi nhat-1 mi sk pl mu RTT(k) pl mu RTT (k)) lk1 eq 2 mu mi lk pl 1 over { RTT(k)} (B/2 mi nhat-1) lambda (2) eq 2 mu mi mu pl 1 over { RTT (2)} (B/2 mi B/2) lambda (3) eq 2 mu mi mu pl 1 over { RTT(3)} (B/2 - B/2) We follow the description of LQG presented in (102) (pg. 835). bold "Q" eq left (pile { { 1 ˜0 ˜0 ˜ } above { 0 ˜0 ˜0 ˜ } above { 0 ˜0 ˜0 ˜ } } right) (99) (93,117,154), (33). as in reference (12). traffic (13,51,105). (33,103), (4,28) (73,74). transport flow control protocol (109,150), (63,127,154,156), the DECbit scheme (114-117). as implemented in XNS's SPP (150) or in TCP (before 4.3 Tahoe BSD) (109), Jacobson and Karels' (JK) modified TCP (63) (15,115-117), (63,71)). protocols such as NETBLT (17) congestion avoidance scheme (66). (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].