# Analysis and Simulation of a Fair Queueing Algorithm

Alan Demers
Srinivasan KeshavÜ
Scott Shenker

Xerox PARC
3333 Coyote Hill Road
Palo Alto, CA 94304

(Originally published in Proceedings SIGCOMM ë89,
CCR Vol. 19, No. 4, Austin, TX, September, 1989, pp. 1-12)

## Abstract

*We discuss gateway queueing algorithms and their role in controlling congestion in datagram networks. A fair queueing algorithm, based on an earlier suggestion by Nagle, is proposed. Analysis and simulations are used to compare this algorithm to other congestion control schemes. We find that fair queueing provides several important advantages over the usual first-come-first-serve queueing algorithm: fair allocation of bandwidth, lower delay for sources using less than their full share of bandwidth, and protection from ill-behaved sources.*

### 1. Introduction

Datagram networks have long suffered from performance degradation in the presence of congestion [Ger80]. The rapid growth, in both use and size, of computer networks has sparked a renewed interest in methods of congestion control [DEC87abcd, Jac88a, Man89, Nag87]. 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. Of course, 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. The second point of implementation is at the gateway. Congestion can be controlled at gateways 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 gatewayís buffer space, do not affect congestion directly, in that they do not change the total traffic on the gatewayís outgoing line. Queueing algorithms do, however, determine the way in which packets from different sources interact with each other which, in turn, affects the collective behavior of flow control algorithms. We shall 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* packets are *discarded* by the gateway). Currently, the most common queueing algorithm is first-come-first-serve (FCFS). FCFS queueing essentially relegates all congestion control to the sources, since the order of arrival completely determines the bandwidth, promptness, and buffer space allocations. Thus, FCFS inextricably intertwines these three allocation issues. There may indeed be flow control algorithms that, when universally implemented throughout a network with FCFS gateways, can overcome these limitations and provide reasonably fair and efficient congestion control. This point is discussed more fully in Sections 3 and 4, where several flow control algorithms are compared. However, with todayís 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. Even if a universal standard such as ISO [ISO86] were adopted, malfunctioning hardware and software could violate the standard, and 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

Ü Current Address: University of California at Berkeley sources. Unfortunately, no matter what flow control algorithm is used by the well-behaved sources, networks with FCFS gateways do not have this property. A single source, sending packets to a gateway 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 [Nag87, Nag85] proposed a *fair queueing* (FQ) algorithm in which gateways 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 delay of other sources. In fact, when a source sends packets too quickly, it merely increases the length of its own queue. Nagleís algorithm, by changing the way packets from different sources interact, does not reward, nor leave others vulnerable to, anti-social behavior. On the surface, this proposal appears to have considerable merit, but we are not aware of any published data on the performance of datagram networks with such fair queueing gateways. In this paper, we will first describe a modification of Nagleís algorithm, and then provide simulation data comparing networks with FQ gateways and those with FCFS gateways.

The three different components of congestion control algorithms introduced above, source flow control, gateway routing, and gateway 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, but it must not require them.

We had three goals in writing this paper. The first was to describe a new fair queueing algorithm. In Section 2.1, we discuss the design requirements for an effective queueing algorithm and outline how Nagleís original proposal fails to meet them. In Section 2.2, we propose a new fair queueing algorithm which meets these design requirements. The second goal was to provide some rigorous understanding of the performance of this algorithm; this is done in Section 2.3, where we present a delay-throughput curve given by our fair queueing algorithm for a specific configuration of sources. The third goal was to evaluate this new queueing proposal in the context of real networks. To this end, we discuss flow control algorithms in Section 3, and then, in Section 4, we present simulation data comparing several combinations of flow control and queueing algorithms on six benchmark networks. Section 5 contains an overview of our results, a discussion of other proposed queueing algorithms, and an analysis of some criticisms of fair queueing.

In circuit switched networks where there is explicit buffer reservation and uniform packet sizes, it has been established that round robin service disciplines allocate bandwidth fairly [Hah86, Kat87]. Recently Morgan [Mor89] 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 DATAKITô queueing algorithm combines round robin service and FIFO priority service, and has been analyzed extensively [Lo87, Fra84]. Also, Luan and Lucantoni present a different form of bandwidth management policy for circuit switched networks [Lua88].

Since the completion of this work, we have learned of a similar *Virtual Clock* algorithm for gateway resource allocation proposed by Zhang [Zha89]. Furthermore, Heybey and Davin [Hey89] have simulated a simplified version of our fair queueing algorithm.

## 2. Fair Queueing

**2.1. Motivation** What are the requirements for a queueing algorithm that will allow source flow control algorithms to provide adequate congestion control even in the presence of ill-behaved sources? We start with Nagleís observation that such queueing algorithms 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 gateway should provide service that, at least on average, does not depend discontinuously on a packetís time of arrival (this continuity condition will become clearer in Section 2.2). This requirement attempts to prevent the efficiency of source implementations from being overly sensitive to timing details (timers are the Bermuda Triangle of flow control algorithms). Nagleís 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. In the following section we attempt to correct these defects.

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, but only one is considered here). Consider, for example, the allocation of a single resource among N users. Assume there is an amount $\mu_{total}$ of this resource and that each of the users requests an amount $\rho_i$ and, under a particular allocation, receives an amount $\mu_i$. What is a fair allocation? The max-min fairness criterion [Hah86, Gaf84, DEC87d] 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, $\mu_{total} \leftarrow \mu_{total} - \mu_{min}$. This condition reduces to $\mu_i = MIN(\mu_{fair}, \rho_i)$ in the simple example, with $\mu_{fair}$, 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 [DEC87d]. Note that implicit in the max-min definition of fairness is the assumption that the users have equal *rights* to the resource.

In our communication application, the bandwidth and buffer demands are clearly represented by the packets that arrive at the gateway. (Demands for promptness are not explicitly communicated, and we will 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 gateways could know that some sources deserve more bandwidth than others, but there is no adequate mechanism for establishing that knowledge in todayís networks. Allocation per receiver allows a receiverís 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 avoiding 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 user. With our requirements for an adequate queueing algorithm, coupled with our definitions of *fairness* and *user*, we now turn to the description of our algorithm.

**2.2 Definition of algorithm** 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 $\mu$ is the linespeed of the gatewayís 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 $T(t) = R(t_0) + P$. Let $t_i^\alpha$ be the time that packet $i$ belonging to conversation $\alpha$ arrives at the gateway, and define the numbers $S_i^\alpha$ and $F_i^\alpha$ as the values of $R(t)$ when the packet started and finished service. With $P_i^\alpha$ 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 at the gateway, the ordering of the $F_i^\alpha$ 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 $S_i^\alpha$ and $F_i^\alpha$ depend only on the packet arrival times $t_i^\alpha$ 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 $i = MAX(j | t_j^\alpha \le t)$. 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 $F_i^\alpha$ 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 $F_i^\alpha$. In a preemptive version of this algorithm, newly arriving packets whose finishing number $F_i^\alpha$ 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 was proved by Greenberg and Madras [Gree89]). Thus, over sufficiently long conversations, the packetized algorithms asymptotically approach the fair bandwidth allocation of the BR scheme.

Recall that the userís request for promptness is not made explicit. (The IP [Pos81] protocol 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 gateway. 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 $\delta$, and defining a new quantity, the *bid* $B_i^\alpha$, via $B_i^\alpha = P_i^\alpha + MAX(F_{i-1}^\alpha, R(t_i^\alpha) - \delta)$. The quantities $R(t)$, $N_{ac}(t)$, $F_i^\alpha$, and $S_i^\alpha$ remain as before, but now the sending order is determined by the *B*ís, not the *F*ís. The asymptotic bandwidth allocation is independent of $\delta$, since the *F*ís control the bandwidth allocation, but the algorithm gives slightly faster service to packets that arrive at an inactive conversation. The parameter $\delta$ controls the extent of this additional promptness. Note that the bid $B_i^\alpha$ is continuous in $t_i^\alpha$, so that the continuity requirement mentioned in Section 2.1 is met.

The role of this term $\delta$ 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 $\alpha$ is active (i.e. the corresponding conversation in the BR algorithm would have bits in the queue). In this case, the value of $\delta$ 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 $\alpha$ 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 $\alpha$. With $\delta = \infty$, the bid number is $B_i^\alpha = P_i^\alpha + F_{i-1}^\alpha$ and depends only the previous packetís finishing number, no matter how many rounds ago. For intermediate values of $\delta$, scheduling decisions for packets arriving at inactive conversations depends on the previous packetís finishing round as long as it wasnít too long ago, and $\delta$ controls how far back this dependence goes.

Recall that when the queue is full and a new packet arrives, the last packet from the source currently using the most buffer space is dropped. We have chosen to leave the quantities $F_i^\alpha$ and $S_i^\alpha$ 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.

**2.3 Properties of Fair Queueing** 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 on the promptness, or delay, performance of a single FQ gateway, 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 gateways. 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 gateway 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 gateway 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 $\delta$ is only through the combination $P_T - \delta$. We will denote the queueing delay of this packet by $\varphi(t)$, which is a periodic function with period $NP_F$. We are interested in the average queueing delay $\Delta$

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

The finishing numbers $F_i^\alpha$ for the N FTPís can be expressed, after perhaps renumbering the packets, by $F_i^\alpha = (i + l^\alpha) P_F$ where the *l*ís obey $0 \leq l^\alpha < 1$. The queueing delay of the Telnet packet depends on the configuration of *l*ís whenever $P_T < P_F$. One can show that the delay is bounded by the extremal cases of $l^\alpha = 0$ for all $\alpha$ 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*, $0 \leq \varepsilon < 1$, defined via $P_T = P_F (k + \varepsilon)/N$.

Preemptive

$$A(P) = N(P - \frac{P_F}{2}) \quad \text{for} \quad P \geq P_F$$

$$N(P - \frac{P_F}{2}) \leq A(P) \leq \frac{NP^2}{2P_F} \quad \text{for} \quad P_F \geq P \geq \frac{P_F}{2}(1 + \frac{1}{N})$$

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

$$0 \leq A(P) \leq \frac{NP^2}{2P_F} \quad \text{for} \quad \frac{\mathrm{P_F}}{2}(1 - \frac{1}{N}) \geq P$$

Nonpreemptive

$$A(P) = N(P - \frac{P_F}{2}) \quad \text{for} \quad P \geq P_F$$

$$N(P - \frac{P_F}{2}) \leq A(P) \leq (\frac{P_F}{2})\left\{1 + \frac{1}{\mathrm{N}}[k^2 + k(2\varepsilon - 1)]\right\} \quad \text{for} \quad P_F \geq P \geq \frac{P_F}{2}(1 + \frac{1}{N})$$

$$\frac{P_F}{2} \leq A(P) \leq (\frac{P_F}{2})\left\{1 + \frac{1}{N}[k^2 + k(2\varepsilon - 1)]\right\} \quad \text{for} \quad \frac{P_F}{2}(1 + \frac{1}{N}) \geq P$$

Now consider a general Telnet packet generation process (ignoring the effects of flow control) and characterize this generation process by the function $D_0(P_T)$ which denotes the queueing delay of the Telnet source when it is the sole source at the gateway. In the BR algorithm, the queueing delay of the Telnet source in the presence of N FTP sources is merely $D_0((N + 1)P_T)$. For the packetized preemptive algorithm with $\delta = 0$, we can express the queueing delay in the presence of N FTP sources, call it $D_N(P_T)$, in terms of $D_0$ via the relation (averaging over all relative synchronizations between the FTPís and the Telnet):

$$D_N(P_T) = D_0((N + 1)P_T) + A(P_T)$$

where the term $A(P_T)$ reflects the extra delay incurred when emulating the BR algorithm by the preemptive packetized algorithm.

For nonzero values *δ*, the generation process must be further characterized by the quantity $I_0(P_t, t)$ which, in a system where the Telnet is the sole source, is the probability that a packet arrives to a queue which has been idle for time *t*. The delay is given by,

$$D_N(P_T) = D_0((N + 1)P_T) + A(P_T) -$$

$$\int_0^\infty I_0((N + 1)P_{T,}, t)\left\{A(P_t) - A(P_T - MIN(\tfrac{t}{N}, \delta)))\right\}dt$$

where the last term represents the reduction in delay due the nonzero $\delta$. These expressions for $D_N$, which were derived for the preemptive case, are also valid for the nonpreemptive algorithm when $P_T \le P_F$.

What do these forbidding formulae mean? Consider, for concreteness, a Poisson arrival process with arrival rate $\lambda$, packet sizes $P_T = P_F \equiv P$, a linespeed $\mu = 1$, and an FTP synchronization described by $l^\alpha = \alpha / N$ for $\alpha = 0,1,\ldots,N-1$. Define $\rho$ to be the average bandwidth of the stream, measured relative to the fair share of the Telnet; $\rho = \lambda P(N+1)$. Then, for the nonpreemptive algorithm,

$$\frac{D_N(P)}{P} = \frac{\rho}{2(1-\rho)} + \frac{N\rho}{2} + N(1-\rho)\left[\frac{1}{2} - \frac{(N+1)}{N\rho} \times \left[1 - \exp\left[-\frac{\rho N}{(N+1)} MIN\left(\frac{\delta}{P}, \frac{1}{2}\left(1 - \frac{1}{N}\right)\right)\right]\right]\right]$$

This is the throughput/delay curve the FQ gateway offers the Poisson Telnet source (the formulae for different FTP synchronizations are substantially more complicated, but have the same qualitative behavior). This can be contrasted with that offered by the FCFS gateway, although the FCFS results depend in detail on the flow control used by the FTP sources and on the surrounding network environment. Assume that all other communications speeds are infinitely fast in relation to the outgoing linespeed of the gateway, and that the FTPís all have window size *W*, so there are always *NW* FTP packets in the queue or in transmission. Figure 1 shows the throughput/delay curves for an FCFS gateway, along with those for a FQ gateway with $\delta = 0$ and $\delta = P$. For $\rho \to 0$, FCFS gives a large queueing delay of $(NW - \frac{1}{2})P$, whereas FQ gives a queueing delay of $NP/2$ for $\delta = 0$ and *P/2* for $\delta = P$. This ability to provide a lower delay to lower throughput sources, completely independent of the window sizes of the FTPís, is one of the most important features of fair queueing. Note also that the FQ queueing delay diverges as $\rho \to 1$, reflecting FQís insistence that no conversation gets more than its fair share. In contrast, the FCFS curve remains finite for all $\rho < (N+1)$, showing that an ill-behaved source can consume an arbitrarily large fraction of the bandwidth.

What happens in a network of FQ gateways? There are few results here, but Hahne [Hah86] has shown that for strict round robin service gateways 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 communication path) result in unfair allocations. We believe, but have been unable to prove, that both of these properties hold for our fair queueing scheme.

**Figure 1: Delay vs. Throughput.**

This graph describes the queueing delay of a single Telnet source with Poisson generation process of strength $\lambda$, sending packets through a gateway with three FTP conversations. The packet sizes are $P_T = P_F \equiv P$, the throughput is measured relative to the Telnetís fair share, $\rho \equiv 4\lambda P / \mu$ where $\mu$ is the linespeed. The delay is measured in units of $P/\mu$. The FQ algorithm is nonpreemptive, and the FCFS case always has 15 FTP packets in the queue.

### 3. Flow Control Algorithms

Flow control algorithms are both the benchmarks against which the congestion control properties of fair queueing are measured, and also the environment in which FQ gateways will operate. We already know that, when combined with FCFS gateways, these flow control algorithms all suffer from the fundamental problem of vulnerability to ill-behaving sources. Also, there is no mechanism for separating the promptness allocation from the bandwidth and buffer allocation. The remaining question is then how fairly do these flow control algorithms allocate bandwidth. Before proceeding, note that there are really two distinct problems in controlling congestion. Congestion *recovery* allows a system to recover from a badly congested state, whereas congestion *avoidance* attempts to prevent the congestion from occurring. In this paper, we are focusing on congestion *avoidance* and will not discuss congestion *recovery* mechanisms at length.

A generic version of source flow control, as implemented in XNSís SPP [Xer81] or in TCP [USC81], has two parts. There is a timeout mechanism, which provides for congestion recovery, whereby packets that have not been acknowledged before the timeout period are retransmitted (and a new timeout period set). The timeout periods are given by $\beta rtt$ where typically $\beta \sim 2$ and *rtt* is the exponentially averaged estimate of the round trip time (the *rtt* estimate for retransmitted packets is the time from their first transmission to their acknowledgement). The congestion avoidance part of the algorithm is sliding window flow control, with some set window size. This algorithm has a very narrow range of validity, in that it avoids congestion if the window sizes are small enough, and provides efficient service if the windows are large enough, but cannot respond adequately if either of these conditions is violated.

The second generation of flow control algorithms, exemplified by Jacobson and Karelsí (JK) modified TCP [Jac88a] and the original DECbit proposal [DEC87a-c], are descendants of the above generic algorithm with the added feature that the window size is allowed to respond dynamically in response to network congestion (JK also has, among other changes, substantial modifications to the timeout calculation [Jac88a,b, Kar87]). The algorithms use different signals for congestion; JK uses timeouts whereas DECbit uses a header bit which is set by the gateway on all packets whenever the average queue length is greater than one. These mechanisms allocate window sizes fairly, but the relation *Throughput = Window/RoundTrip* implies that conversations with different paths receive different bandwidths.

The third generation of flow control algorithms are similar to the second, except that now the congestion signals are sent selectively. For instance, the selective DECbit proposal [DEC87d] has the gateway measure the flows of the various conversations and only send congestion signals to those users who are using more than their fair share of bandwidth. This corrects the previous unfairness for sources using different paths (see [DEC87d] and section 4), and appears to offer reasonably fair and efficient congestion control in many networks. The DEC algorithm controls the delay by attempting to keep the average queue size close to one. However, it does not allow individual users to make different delay/throughput tradeoffs; the collective tradeoff is set by the gateway.

## 4. Simulations

In this section we compare the various congestion control mechanisms, and try to illustrate the interplay between the queueing and flow control algorithms. We simulated these algorithms at the packet level using a network simulator built on the Nest network simulation tool [Nes88]. In order to compare the FQ and FCFS gateway algorithms in a variety of settings, we selected several different flow control algorithms; the generic one described above, JK flow control, and the selective DECbit algorithm. To enable DECbit flow control to operate with FQ gateways, we developed a bit-setting FQ algorithm in which the congestion bits are set whenever the sourceís queue length is greater than $\frac{1}{3}$ of its fair share of buffer space (note that this is a much simpler bit-setting algorithm than the DEC scheme, which involves complicated averages; however, the choice of $\frac{1}{3}$ is completely *ad hoc*). The Jacobson/Karels flow control algorithm is defined by the 4.3bsd TCP implementation. This code deals with many issues unrelated to congestion control. Rather than using that code directly in our simulations, we have chosen to model the JK algorithm by adding many of the congestion control ideas found in that code, such as adjustable windows, better timeout calculations, and fast retransmit to our generic flow control algorithm. The various cases of test algorithms are labeled in table 1.

| Label | Flow Control | Queueing Algorithm |
|---|---|---|
| G/FCFS | Generic | FCFS |
| G/FQ | Generic | FQ |
| JK/FCFS | JK | FCFS |
| JK/FQ | JK | FQ |
| DEC/DEC | DECbit | Selective DECbit |
| DEC/FQbit | DECbit | FQ with bit setting |

**Table 1: Algorithm Combinations**

Rather than test this set of algorithms on a single *representative* network and load, we chose to define a set of *benchmark* scenarios, each of which, while somewhat unrealistic in itself, serves to illuminate a different facet of congestion control. The load on the network consists of a set of Telnet and FTP conversations. The Telnet sources generate 40 byte packets by a Poisson process with a mean interpacket interval of 5 seconds. The FTPís have an infinite supply of 1000 byte packets that are sent as fast as flow control allows. Both FTPís and Telnetís have their maximum window size set to 5, and the acknowledgement (ACK) packets sent back from the receiving sink are 40 bytes. (The small size of Telnet packets relative to the FTP packets makes the effect of $\delta$ insignificant, so the FQ algorithm was implemented with $\delta = 0$). The gateways have finite buffers which, for convenience, are measured in packets rather than bytes. The system was allowed to stabilize for the first 1500 seconds, and then data was collected over the next 500 second interval. For each scenario, there is a figure depicting the corresponding network layout, and a table containing the data. There are four performance measures for each source: total throughput (number of packets reaching destination), average round trip time of the packets, the number of packet retransmissions, and number of dropped packets. We do not include confidence intervals for the data, but repetitions of the simulations have consistently produced results that lead to the same qualitative conclusions.

We first considered several single-gateway networks. The first scenario has two FTP sources and two Telnet sources sending to a sink through a single bottleneck gateway. Note that, in this underloaded case, all of the algorithms provide fair bandwidth allocation, but the cases with FQ provide much lower Telnet delay than those with FCFS. The selective DECbit gives an intermediate value for the Telnet delay, since the flow control is designed to keep the average queue length small.

**Scenario 1: Underloaded Gateway**

Scenario 2 involves 6 FTP sources and 2 Telnet sources again sending through a single gateway. The gateway, with a buffer size of only 15, is substantially overloaded. This scenario probes the behavior of the algorithms in the presence of severe congestion.

When FCFS gateways are paired with generic flow control, the sources segregate into *winners*, who consume a large amount of bandwidth, and *losers*, who consume very little. This phenomenon develops because the queue is almost always full. The ACK packets received by the *winners* serve as a signal that a buffer space has just been freed, so their packets are rarely dropped. The *losers* are usually retransmitting, at essentially random times, and thus have most of their packets dropped. This analysis is due to Jacobson [Jac88b], and the segregation effect was first pointed out to us in this context by Sturgis [Stu88]. The combination of JK flow control with FCFS gateways produces fair bandwidth allocation among the FTP sources, but the Telnet sources are almost completely shut out. This is because the JK algorithm ensures that the gatewayís buffer is usually full, causing most of the Telnet packets to be dropped.

When generic flow control is combined with FQ, the strict segregation disappears. However, the bandwidth allocation is still rather uneven, and the useful bandwidth (rate of nonduplicate packets) is 12% below optimal. Both of these facts are due to the inflexibility of the generic flow control, which is unable to reduce its load enough to prevent dropped packets. This not only necessitates retransmissions but also, because of the crudeness of the timeout congestion recovery mechanism, prevents FTPís from using their fair share of bandwidth. In contrast, JK flow control combined with FQ produced reasonably fair and efficient allocation of the bandwidth. The lesson here is that fair queueing gateways by themselves do not provide adequate congestion control; they must be combined with intelligent flow control algorithms at the sources.

**Scenario 2: Overloaded Gateway**

**Scenario 3 : Ill-Behaved Sources**

The selective DECbit algorithm manages to keep the bandwidth allocation perfectly fair, and there are no dropped packets or retransmissions. The addition of FQ to the DECbit algorithm retains the fair bandwidth allocation and, in addition, lowers the Telnet delay by a factor of 9. Thus, for each of the three flow control algorithms, replacing FCFS gateways with FQ gateways generally improved the FTP performance and dramatically improved the Telnet performance of this extremely overloaded network.

In scenario 3 there is a single FTP and a single Telnet competing with an ill-behaved source. This ill-behaved source has no flow control and is sending packets at twice the rate of the gatewayís outgoing line. With FCFS, the FTP and Telnet are essentially shut out by the ill-behaved source. With FQ, they obtain their fair share of bandwidth. Moreover, the ill-behaved host gets much less than its fair share, since when it has its packets dropped it is still charged for that throughput. Thus, FQ gateways are effective *firewalls* that can protect users, and the rest of the network, from being damaged by ill-behaved sources.

## Scenario 4: Mixed Protocols

We have argued for the importance of considering a heterogeneous set of flow control mechanisms. Scenario 4 has single gateway with two pairs of FTP sources, employing generic and JK flow control respectively. With a FCFS gateway, the generic flow controlled pair has higher throughput than the JK pair. However, with a FQ gateway, the situation is reversed (and the generic sources have segregated). Note that the FQ gateway has provided incentive for sources to implement JK or some other intelligent flow control, whereas the FCFS gateway makes such a move sacrificial.

Certainly not all of the relevant behavior of these algorithms can be gleaned from single gateway networks. Scenario 5 has a multinode network with four FTP sources using different network paths. Three of the sources have short nonoverlapping conversations and the fourth source has a long path that intersects each of the short paths. When FCFS gateways are used with generic or JK flow control, the conversation with the long path receives less than 60% of its fair share. With FQ gateways, it receives its full fair share. Furthermore, the selective DECbit algorithm, in keeping the average queue size small, wastes roughly 10% of the bandwidth (and the conversation with the long path, which should be helped by any attempt at fairness, ends up with less bandwidth than in the generic/FCFS case).

### Scenario 5: Multihop Path

Scenario 6 involves a more complicated network, combining lines of several different bandwidths. None of the gateways are overloaded so all combinations of flow control and queueing algorithms function smoothly. With FCFS, sources 4 and 8 are not limited by the available bandwidth, but by the delay their ACK packets incur waiting behind FTP packets. The total throughput increases when the FQ gateways are used because the small ACK packets are given priority.

### Scenario 6: Complicated Network

For the sake of clarity and brevity, we have presented a fairly clean and uncomplicated view of network dynamics. We want to emphasize that there are many other scenarios, not presented here, where the simulation results are confusing and apparently involve complicated dynamic effects. These results do not call into question the efficacy and desirability of fair queueing, but they do challenge our understanding of the collective behavior of flow control algorithms in networks.

## 5. Discussion

In an FCFS gateway, 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 in a maximally cooperative environment and impossible in a noncooperative one, merely reflects the inability of FCFS gateways 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 $\delta$. 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 gatewayís game so that good source behavior is encouraged, is crucial in the design of gateway algorithms. A formal game-theoretic analysis of a simple gateway 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 [She89a].

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. Furthermore, simulations indicate that, when combined with currently available flow control algorithms, FQ delivers satisfactory congestion control in a wide variety of network scenarios. The combination of FQ gateways and DECbit flow control was particularly effective. However, these limited tests are in no way conclusive. We hope, in the future, to investigate the performance of FQ under more realistic load conditions, on larger networks, and interacting with routing algorithms. Also, we hope to explore new source flow control algorithms that are more attuned to the properties of FQ gateways.

In this paper 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 general purpose congestion control algorithms. Prue and Postel [Pru87] 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 [Mill87,88] whereby the gateways allocate buffers fairly (on a source basis, over all of the gatewayís 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, when the buffer is overloaded, the packet to be dropped is chosen at random [Per89, Jac88ab]. This algorithm greatly alleviates the problem of segregation. However, it is now generally agreed that the RD algorithm does not provide 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 [She89b, Zha89, Has89].

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. There are several responses to this. First, FQ is no worse than the status quo. FCFS gateways already limit well-behaved hosts, using the same path and having only one stream per source destination pair, to their fair share of bandwidth. Some current bandwidth hogs achieve their desired level of service by opening up many streams, since FCFS gateways implicitly define streams as the unit of *user*. Note that that there are no controls over this mechanism of gaining more bandwidth, leaving the network vulnerable to abuse. If desired, however, this same trick can be introduced into fair queueing by merely changing the notion of user. This would violate layering, which is admittedly a serious drawback. A better approach is to confront the issue of allocation directly by generalizing the algorithm to allow for arbitrary bandwidth priorities. Assign each pair a number $n_\alpha$ which represents how many queue slots that conversation gets in the bit-by-bit round robin. The new relationships are $N_{ac} = \sum n_\alpha$ with the sum over all active conversations, and $P_i^\alpha$ is set to be $1/n_\alpha$

times the true packet length. Of course, the truly vexing problem is the politics of assigning the priorities $n_\alpha$. 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 [Zha89].

The other objection is that fair queueing requires the gateways to be smart and fast. There is technological question of whether or not one can build FQ gateways that can match the bandwidth of fibers. If so, are these gateways economically feasible? We have no answers to these questions, and they do indeed seem to hold the key to the future of fair queueing.

## 6. Acknowledgements

The authors gratefully acknowledge H. Murrayís important role in designing an earlier version of the fair queueing algorithm. We wish to thank A. Dupuy for assistance with the Nest simulator. Fruitful discussions with J. Nagle, E. Hahne, A. Greenberg, S. Morgan, K. K. Ramakrishnan, V. Jacobson, M. Karels, D. Greene and the End-to-End Task Force are also appreciated. In addition, we are indebted to H. Sturgis for bringing the segregation result in section 4 to our attention, and to him and R. Hagmann and C. Hauser for the related lively discussions. We also wish to thank the group at MIT for freely sharing their insights: L. Zhang, D. Clark, A. Heybey, C. Davin, and E. Hashem.

## 7. References

[DEC87a] R. Jain and K. K. Ramakrishnan, ìCongestion Avoidance in Computer Networks with a Connectionless Network Layer, Part I-Concepts, Goals, and Alternativesî, DEC Technical Report TR-507, Digital Equipment Corporation, April 1987.

[DEC87b] K. K. Ramakrishnan and R. Jain, ìCongestion Avoidance in Computer Networks with a Connectionless Network Layer, Part II-An Explicit Binary Feedback Schemeî, DEC Technical Report TR-508, Digital Equipment Corporation, April 1987.

[DEC87c] D.-M. Chiu and R. Jain, ìCongestion Avoidance in Computer Networks with a Connectionless Network Layer, Part III-Analysis of Increase and Decrease Algorithmsî, DEC Technical Report TR-509, Digital Equipment Corporation, April 1987.

[DEC87d] K. K. Ramakrishnan, D.-M. Chiu, and R. Jain ìCongestion Avoidance in Computer Networks with a Connectionless Network Layer, Part IV-A Selective Binary Feedback Scheme for General Topologiesî, DEC Technical Report TR-510, Digital Equipment Corporation, November 1987.

[Fra84] A. Fraser and S. Morgan, ìQueueing and Framing Disciplines for a Mixture of Data Traffic Typesî, AT&T Bell Laboratories Technical Journal,Volume 63, No. 6, pp 1061-1087, 1984.

[Gaf84] E. Gafni and D. Bertsekas, ìDynamic Control of Session Input Rates in Communication Networksî, IEEE Transactions on Automatic Control, Volume 29, No. 10, pp 1009-1016, 1984.

[Ger80] M. Gerla and L. Kleinrock, ìFlow Control: A Comparative Surveyî, IEEE Transactions on Communications, Volume 28, pp 553-574, 1980.

[Gre89] A. Greenberg and N. Madras, private communication, 1989.

[Hah86] E. Hahne, ìRound Robin Scheduling for Fair Flow Control in Data Communication Networksî, Report LIDS-TH-1631, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, Massachusetts, December, 1986.

[Has89] E. Hashem, private communication, 1989.

[Hey89] A. Heybey and C. Davin, private communication, 1989.

[ISO86] International Organization for Standardization (ISO), ìProtocol for Providing the Connectionless Mode Network Serviceî, Draft International Standard 8473, 1986.

[Jac88a] V. Jacobson, ìCongestion Avoidance and Controlî, ACM SigComm Proceedings, pp 314-329, 1988.

[Jac88b] V. Jacobson, private communication, 1988.

[Jai86] R. Jain, ìDivergence of Timeout Algorithms for Packet Retransmissionî, Proceedings of the Fifth Annual International Phoenix Conference on

Computers and Communications, pp 1162-1167, 1987.

[Kar87] P. Karn and C. Partridge, ìImproving Round-Trip Time Estimates in Reliable Transport Protocolsî, ACM SigComm Proceedings,pp 2-7, 1987.

[Kat87] M. Katevenis, ìFast Switching and Fair Control of Congested Flow in Broadband Networksî, IEEE Journal on Selected Areas in Communications,Volume 5, No. 8, pp 1315-1327, 1987.

[Lo87] C.-Y. Lo, ìPerformance Analysis and Application of a Two-Priority Packet Queueî, AT&T Technical Journal,Volume 66, No. 3, pp 83-99, 1987.

[Lua88] D. Luan and D. Lucantoni, ìThroughput Analysis of an Adaptive Window-Based Flow Control Subject to Bandwidth Managementî, Proceedings of the International Teletraffic Conference, 1988.

[Man89] A. Mankin and K. Thompson, ìLimiting Factors in the Performance of the Slo-start TCP Algorithmsî, preprint.

[Mor89] S. Morgan, ìQueueing Disciplines and Passive Congestion Control in Byte-Stream Networksî, IEEE INFOCOM ë89 Proceedings, pp 711-720, 1989.

[Mil87] D. Mills and W.-W. Braun, ìThe NSFNET Backbone Networkî, ACM SigComm Proceedings,pp 191-196, 1987.

[Mil88] D. Mills, ìThe Fuzzballî, ACM SigComm Proceedings,pp 115-122, 1988.

[Nag84] J. Nagle, ìCongestion Control in IP/TCP Networks, Computer Communication Review, Vol 14, No. 4,pp 11-17, 1984.

[Nag85] J. Nagle, ìOn Packet Switches with Infinite Storageî, RFC 896 1985.

[Nag87] J. Nagle, ìOn Packet Switches with Infinite Storageî, IEEE Transactions onCommunications,Volume 35, pp 435-438, 1987.

[Nes88] D. Bacon, A. Dupuy, J. Schwartz, and Y. Yemini, ìNest: A Network Simulation andPrototyping Toolî, Dallas Winter 1988 Usenix Conference Proceedings, pp.71-78, 1988.

[Per89] IETF Performance and Congestion Control Working Group, ìGateway Congestion Control Policiesî, draft, 1989.

[Pos81] J. Postel, ìInternet Protocolî, RFC 791 1981.

[Pru88] W. Prue and J. Postel, ìA Queueing Algorithm to Provide Type-of-Service for IP Linksî, RFC1046, 1988.

[She89a] S. Shenker, ìGame-Theoretic Analysis of Gateway Algorithmsî, in preparation, 1989.

[She89b] S. Shenker, ìComments on the IETF Performance and Congestion Control Working Group Draft on Gateway Congestion Control Policiesî, unpublished, 1989.

[Stu88] H. Sturgis, private communication, 1988.

[USC81] USC Information Science Institute, ìTransmission Control Protocolî, RFC 793, 1981.

[Xer81] Xerox Corporation, ìInternet Transport Protocolsî, XSIS 028112, 1981.

[Zha89] L. Zhang, ìA New Architecture for Packet Switching Network Protocolsî, MIT Ph. D. Thesis, forthcoming, 1989.