Supporting Best-Effort Traffic with Fair Service Curve

T.S. Eugene Ng Donpaul C. Stephens Ion Stoica

Hui Zhang

Feburary 2000

CMU-CS-99-169

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

An earlier version of this paper appeared in *Proceedings of IEEE GLOBECOM'99*.

This research was sponsored by DARPA under contract numbers N66001-96-C-8528 and E30602-97-2-0287, and by NSF under grant numbers Career Award NCR-9624979 and ANI-9814929. Additional support was provided by Intel Corp.

Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, Intel, or the U.S. government.

Keywords: Resource management, scheduling, best-effort traffic, delay differentiation, fairness.

Abstract

Packet Fair Queueing (PFQ) algorithms are the most popular and well studied scheduling algorithms for integrated services networks for two reasons: (1) With reservation, they can provide per-flow end-to-end delay guarantees for real-time traffic flows. (2) Without reservation, they can provide protection among competing best-effort flows while allowing dynamic bandwidth sharing. However, PFQ algorithms have two important limitations. The first one is that, since only one parameter (a weight) is used to allocate resource for each flow, there is a coupling between delay and bandwidth allocation. This can result in network under-utilization when real-time flows have diverse delay and bandwidth requirements. The second and less well known limitation is that, due to the instantaneous fairness property of PFQ algorithms, when used for best-effort service, PFQ algorithms favor continuously-backlogged throughput-oriented applications such as FTP over bursty applications such as WWW and telnet.

In a previous study [21], we proposed the Fair Service Curve (FSC) algorithm which enables more flexible delay and bandwidth allocation for real-time traffic through the use of non-linear service curves. In this paper, we show that, when used for best-effort traffic, FSC can improve performance of delay-sensitive bursty applications without negatively affecting the performance of throughput-oriented applications.

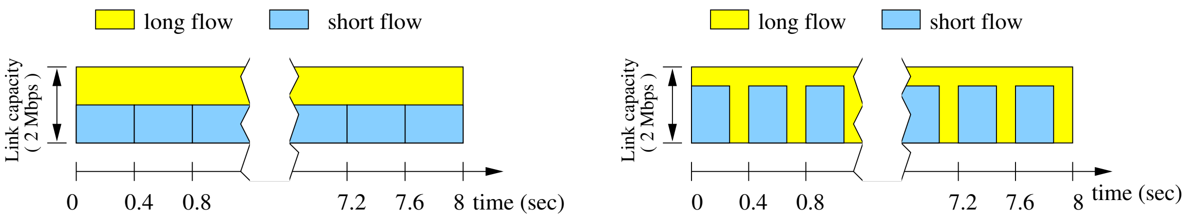
1 Introduction

With the rapid growth of the Internet and the advancement of router technologies, we see two important trends. On one hand, best-effort data traffic continues to account for the majority of the Internet's traffic. On the other hand, advanced routers with sophisticated queue and buffer management capabilities are becoming available. While there is a huge body of literature on using advanced buffer management and packet scheduling algorithms to support real-time continuous media traffic, there is relatively less work on how to exploit these algorithms to better support best-effort data traffic. This paper is aimed to address the latter issue.

Packet Fair Queueing (PFQ) algorithms (i.e., Weighted Fair Queueing [5, 14] and its many variants [1, 7, 8, 18, 16, 20, 23]) have become the most popular algorithms implemented in today's advanced switches and routers [19, 9] because these algorithms provide support for both real-time and best-effort traffic. With bandwidth reservation, PFQ algorithms are able to provide end-to-end delay guarantees. Without reservation, these algorithms can provide best-effort service since they can allocate bandwidth fairly among competing flows, protecting well-behaved flows against ill-behaved ones.

A well known limitation of PFQ is that it couples delay and bandwidth allocation, as there is only one parameter, a weight, that specifies the resource allocated to a flow. This weight affects both the delay and bandwidth properties of the flow. Consequently, under PFQ, it is not possible to differentiate between two flows that have the same bandwidth but different delay requirements without over-reservation. This may result in low network utilization when real-time flows have diverse delay and bandwidth requirements. To address this problem, we proposed the Fair Service Curve (FSC) [21] algorithm which has the ability to decouple delay and bandwidth allocation. This is achieved by properly assigning service curves of different shapes to different flows – concave curves for flows with tight per-packet delay bounds and convex curves for flows with less stringent per-packet delay bounds. This flexibility allows FSC to achieve higher resource utilization for real-time traffic than PFQ.

When used for best-effort service, PFQ favors continuously backlogged traffic over short lived bursty traffic. This is because PFQ is designed to achieve instantaneous bandwidth fairness for all flows, irrespective of their delay requirements. In reality, different types of best-effort data traffic, such as Telnet, FTP, and WWW, have different characteristics and thus performance



(a) Equal Weights for Short and Long Flows

(b) Short Flows with $3\times$ the Long Flow's weight

Figure 1: Improving burst delays

objectives. For example, while the burst delay is the performance index for interactive services, the average throughput is the performance index for bulk transfer applications such as FTP. The key observation is that, since the performance index of bulk-transfer applications is determined over relatively long time scales, we may be able to exploit these applications' insensitivity to short term service variations to improve the performance of delay sensitive bursty applications.

To illustrate how this may be realized, consider a 2 Mbps link shared by one long flow that transfers 1 MB, and several short flows that transfer 50 KB each. Assume that the link is managed by PFQ and each flow has a weight of one. For simplicity, assume that all flows are continuously backlogged, and that once a short flow finishes, another short flow starts immediately. Thus, there are exactly two flows, the long flow and a short flow, backlogged at any given time. As a result each backlogged flow is allocated 1 Mbps. Therefore, as shown in Figure 1 (a), the long flow takes 8 seconds to finish, while a short flow takes 0.4 seconds to complete. Now consider the case where all short flows are assigned three times the weight of the long flow. Each short flow now receives 1.5 Mbps, which consequently reduces its latency by 33% to 0.27 seconds. At the same time, the transfer time of the long flow does *not* change. Thus, by assigning different weights, it is possible to significantly speed-up short transfers without affecting the longer flow.

In order to achieve this performance, a system would either need to estimate the length of a flow when it becomes backlogged, or dynamically reduce the flow's weight after the length of the transfer exceeds a certain threshold. While it is unclear how this could be implemented in a system based on PFQ, the service curve framework in an FSC system enables us to clearly specify the burst threshold and the higher relative share that these bursts should receive. This enables FSC to provide better performance for delay-oriented bursty flows than PFQ, while maintaining the same long term throughput for persistent flows.

In this paper, we show that FSC can out-perform PFQ in supporting best-effort traffic, even in the case when we assign the *same* service curve to *all* flows.[1] We begin with simplified traffic sources in order to more easily illustrate the parameter selection for FSC when used for best effort traffic. We then show that these basic results remain applicable when the sources are more diverse, as is the case in actual data networks. In order to quantify the impact on long-lived throughout-oriented traffic sources, we use an experiment where the short bursty traffic sources are specifically designed to extract the maximal benefit from FSC. Finally, we use a synthetic workload of FTP flows whose lengths are drawn to model the AT&T Internet traffic distributions [6] in order to analyze the tradeoffs in parameter selection for realistic data traffic.

The rest of this paper is organized as follows. In Section 2, we give an overview of PFQ algorithms, the service curve model, and the FSC algorithm, and discuss the use of FSC for best-effort service. We present and discuss our simulation results in Section 3, and expose some implementation issues in Section 4. Related work is discussed in Section 5, and finally we summarize our findings in Section 6.

2 Packet Fair Queueing (PFQ) and Fair Service Curve (FSC) Algorithms

In this section, we first explain the central ideas behind various PFQ algorithms. Then we present the concepts behind service curve based algorithms and describe the Fair Service Curve (FSC) algorithm we use in this paper for supporting best-effort traffic.

2.1 PFQ Algorithms

Packet Fair Queueing (PFQ) algorithms are based on the GPS model [15]. In GPS, each flow i is characterized by its weight, ϕ_i. During any time interval when there are exactly n non-empty queues, the server serves the n packets at the head of the queues simultaneously, in proportion to their weights.

Each PFQ algorithm maintains a system virtual time $v^s(\cdot)$ which represents the normalized fair amount of service that each flow should have received by time t. In addition, it associates to

[1] Although this requires per flow queueing, it does not require the scheduler to distinguish between different types of flows.

each flow i a virtual start time $v_i(\cdot)$, and a virtual finish time $f_i(\cdot)$. Intuitively, $v_i(t)$ represents the normalized amount of service that flow i has received by time t, and $f_i(t)$ represents the sum between $v_i(t)$ and the normalized service that flow i should receive for serving the packet at the head of its queue (determined by the flow's weight ϕ_i). The goal of all PFQ algorithms is then to minimize the discrepancies among $v_i(t)$'s and $v^s(t)$. This is usually achieved by selecting for service the packet with the smallest $v_i(t)$ or $f_i(t)$. The system virtual time is primarily used to reset $v_i(t)$ whenever an unbacklogged flow i becomes backlogged again. More precisely,

$$
\begin{aligned}
v_i(t) &= \begin{cases} \max(v^s(t), v_i(t-)) & i \text{ becomes backlogged} \\ v_i(t-) + \frac{l_i^k}{\phi_i} & p_i^k \text{ finishes} \end{cases} && (1) \\
f_i(t) &= v_i(t) + \frac{l_i^{k+1}}{\phi_i} && (2)
\end{aligned}
$$

where $t-$ is the time instant before time t, p_i^k represents the k-th packet of flow i, and l_i^k represents its length. An example of a system virtual time function $v^s(t)$ is the minimum virtual finish time among all backlogged flows [7]. Various PFQ algorithms differ mainly in their computation of the system virtual time function and the packet selection policy.

Intuitively, PFQ allocates to each backlogged flow a share of service in proportion to its weight. This way PFQ achieves instantaneous fairness for backlogged flows. In addition, if a flow previously received service beyond its (weighted) fair share, it will not be punished in the future. For real-time traffic using reservation, this enables PFQ to provide a bandwidth guarantee to these flows. For best-effort traffic, this enables PFQ to provide fair service among the flows while protecting them against potentially malicious flows.

While the instantaneous fairness property is the basis of how PFQ provides these features, it directly couples the delay and bandwidth allocation of the flows PFQ schedules among. More precisely, if flow i is assigned a rate ϕ_i, then it can be shown that the worst case queueing delay incurred by a packet p_i^k is

$$\frac{l_i^k}{\phi_i} + \frac{l_{max}}{C}, \tag{3}$$

where l_{max} represents the maximum size of a packet, and C represents the capacity of the output link. Thus, the only way to reduce the worst case delay is to increase the reservation ϕ_i. However, this may lead to inefficient resource utilization in the presence of low-bandwidth low-delay flows.

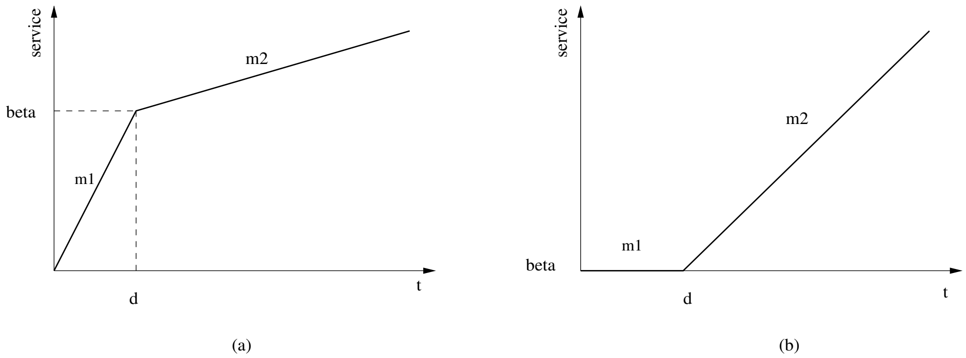


Figure 2: Sample service curves.

As an example, consider a 64 Kbps audio flow with 160 byte packets. To achieve a worst case delay of 5 ms, according to Eq. (3), one should reserve[2] 256 Kbps, which is four times more than the flow's bandwidth requirements!

2.2 Service Curve Model

To address this problem, Cruz has proposed a new service model, called service curve (SC) [2, 3], in the context of real-time guaranteed traffic. In this model, each flow is associated with a service curve S_i, which is a continuous non-decreasing function. A flow i is said to be guaranteed a service curve $S_i(\cdot)$, if for any time t_2 when the flow is backlogged, there exists a time $t_1 < t_2$, which is the beginning of one of flow i's backlogged periods (not necessarily including t_2), such that the following holds

$$S_i(t_2 - t_1) \leq w_i(t_1, t_2), \tag{4}$$

where $w_i(t_1, t_2)$ is the amount of service received by flow i during the time interval $(t_1, t_2]$. For packet systems, we restrict t_2 to be packet departure times. One algorithm that supports service curve guarantees is the Service Curve Earliest Deadline first (SCED) algorithm [17]. SCED can guarantee all the service curves in a system if and only if $\sum_i S_i(t) \leq C \cdot t$ holds for any $t \geq 0$, where C is the output link capacity.

Even though any continuous non-decreasing function can be used as a service curve, for simplicity, usually only two types of non-linear service curves are considered: two-piece linear concave curves (Figure 2(a)), and two-piece linear convex curves (Figure 2(b)). A two-piece

[2]Note that here we ignore the second term $\frac{l_{max}}{C}$, as C is usually very large.

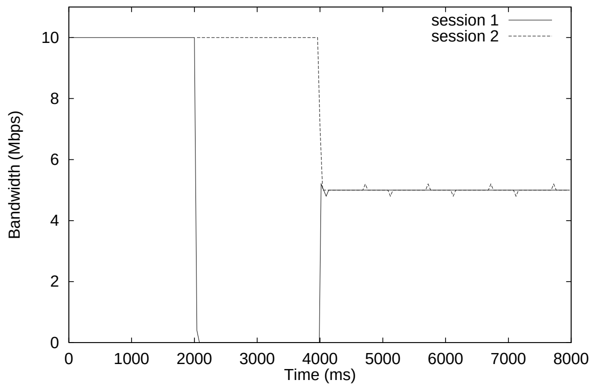


Figure 3: Measured bandwidth of two TCP sessions, startup 2 seconds apart under SCED.

linear service curve is characterized by four parameters: $m1$, the slope of the first segment; $m2$, the slope of the second segment; β, the y-projection of the intersection point of the two segments; d, the x-projection of the intersection point of the two segments. Intuitively, $m2$ specifies the long term throughput guaranteed to a flow, while $m1$ specifies the rate at which a burst of size β is served. Note that a real-time flow served by PFQ can be thought of as having a straight-line service curve that passes through the origin and have a slope of the guaranteed rate r_i.

By using two-piece linear service curves, both delay and bandwidth allocation are taken into account in an *integrated* fashion, yet the allocation policies for these two resources are decoupled. This increases the resource management flexibility and the resource utilization inside the network. To illustrate, consider again the example described in Section 2.1. In SCED, the audio flow can be assigned a service curve with the following parameters: $m1 = 256$ Kbps, $m2 = 64$ Kbps, $\beta = 160$ bytes, and $d = 5$ ms. If the packet arrival process is periodic, then it can be shown by using Eq. (4) that this service curve guarantees a worst case delay of 5 ms. However, unlike PFQ which requires 256 Kbps of bandwidth to be reserved to achieve the same delay, with SCED the long term reserved bandwidth is only 64 Kbps. This creates the opportunity to allocate the remaining bandwidth to other delay-tolerant traffic, such as FTP.

The main drawback of SCED is that it punishes a flow that has received service beyond its service curve. While the SCED algorithm can guarantee all the service curves simultaneously, it does not have the fairness property. As an example, consider two TCP sessions sharing a 10Mbps link scheduled by SCED which start up two seconds apart. Both sessions are assigned the same service curve with $m1$ four times larger than $m2$ and the inflection point occurs at $\beta = 6000$ bytes.

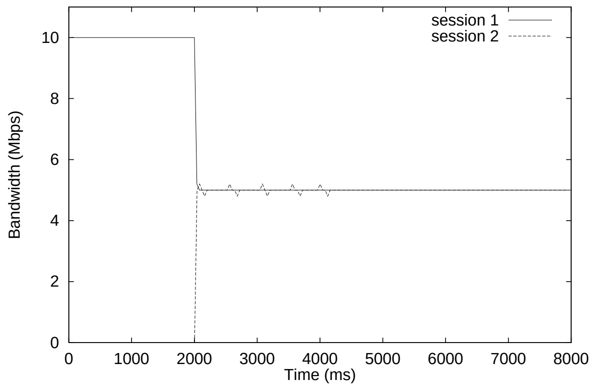


Figure 4: Measured bandwidth of two TCP sessions, startup 2 seconds apart under FSC.

Figure 3 plots the bandwidth received by these two sessions under SCED. Under SCED, once the second session starts up, the first session is denied any service for approximately 2 seconds. Such behavior clearly discourages adaptive flows from sharing the available link capacity. This is the same type of behavior as that exhibited by the well known Virtual Clock (VC) service discipline [24]. In fact, if $m1 = m2$, SCED reduces to VC.

A related problem is that, in SCED, the service curve is defined in terms of *absolute* rates and real time. This makes sense only in a system that employs admission control. In a best effort system, what matters is *relative* performance. However, in SCED, the relation between two service curves does not uniquely determine the service received by each flow. Thus, given the same arrival process, scaling the service curves of the flows will result in different service schedules. As a result the absolute values of the weights or reservations cannot be arbitrarily set. Furthermore, the fact that these values have no special meaning in the context of best effort traffic makes their choice particularly difficult. In contrast, in PFQ and Fair Service Curve (FSC), scaling the parameters of each flow by the same amount does *not* change the service received by each flow. This characteristic simplifies significantly the process of assigning service curves for best effort traffic.

2.3 Fair Service Curve Algorithm

To address these problems, we proposed a new service discipline, called Fair Service Curve (FSC) in [21]. The main difference between FSC and SCED is that under FSC a flow that has received excess service is not punished when other flows become backlogged. As noted above, this is

receive_packet(i, p) $/*$ *flow i has received packet p* $*/$

  enqueue$(queue_i, p)$;

  if $(i \notin \mathcal{A})$ $/*$ *if i was not backlogged* $*/$

    update_v(i, p); $/*$ *update* $V(\cdot)$ *for* i $*/$

    $\mathcal{A} = \mathcal{A} \cup \{i\}$; $/*$ *mark* i *backlogged* $*/$

get_packet() $/*$ *get next packet to send* $*/$

  $i = \min_{v_i} \mathcal{A}$; $/*$ *select backlogged flow with minimum virtual time* $*/$

  $p =$**dequeue**(i);

  update_v(i, p)

  if $(queue_i = \emptyset)$

    $\mathcal{A} = \mathcal{A} \setminus \{i\}$;

  send_packet(p);

Figure 5: The Fair Service Curve (FSC) algorithm. The **receive_packet** function is executed every time a packet arrives; the **get_packet** function is executed every time a packet departs (to select the next packet to send).

also what differentiates PFQ and VC algorithms.[3] To illustrate the difference in fairness between SCED and FSC, consider again the scenario of two TCP with staggered start times sharing a 10Mbps link. Figure 4 plots the bandwidth received by these two sessions under FSC. Contrasted with SCED (Figure 3), FSC fairly allocates bandwidth to both sessions once the second session has started up.

The pseudocode for FSC is shown in Figure 5 and 6. Overall, FSC is very similar to PFQ in that it also uses the concept of virtual time and a set of virtual start and finish times for each flow. FSC uses the smallest virtual start time ($v_i(t)$) as the selection criterion, and $v^s(t) = (v_{i,min}(t) + v_{i,max}(t))/2$ as the system virtual time function, where $v_{i,min}(t)$ and $v_{i,max}(t)$ are the minimum and maximum virtual start times among all backlogged flows at time t.

However, the difference between FSC and PFQ is in the computation of the time stamps. In

[3] However, note that while both PFQ and VC can provide the same real-time guarantees, this is not true for FSC and SCED. A detailed discussion and a variant of FSC that is able to provide the same real-time guarantees as SCED is given in [21].

update_v(i, p)

if ($i \notin \mathcal{A}$) /* *is flow i backlogged ?* */

$v_i = max(v_i, v^s)$

update_VC(i);

if (**backlogged**(i) = *TRUE*)

return;

else

$w_i = w_i + \mathbf{length}(p)$;

$v_i = V_i^{-1}(w_i)$;

Figure 6: The function which updates the virtual time curves and the virtual times in FSC.

PFQ, ϕ_i can be viewed as the slope of a straight line service curve. In FSC, however, since service curves can be non-linear, we cannot compute the timestamps based on the slope of a service curve only. To compute the timestamps, we need to remember what part of the service curve was used to compute the timestamp of the previous packet. We call the remainder of the service curve the virtual curve $V_i(\cdot)$, it is defined such that $v_i(t) = V_i^{-1}(w_i(t))$, where $w_i(t)$ is the total amount of service received by flow i by time t. When a flow i first becomes backlogged, $V_i(v)$ is initialized to the service curve $S_i(t)$. Thereafter, every time a flow becomes backlogged, the **update_VC** function is called in which $V_i(v)$ is updated as follows:

$$V_i(v) = \min(V_i(v), S_i(v - v^s(t)) + w_i(t)), \;\; \forall\, v \geq v^s(t), \tag{5}$$

This update process is illustrated graphically in Figure 7. Note that when the service curve $S_i(t)$ is a straight line with slope ϕ_i, from Eq. (5) we have $V_i(v) = \phi_i v$. Then, the virtual time $v_i(t)$ is simply $V^{-1}(w_i(t)) = w_i(t)/\phi_i$, which is exactly the virtual time of flow i in PFQ algorithms.

2.4 Fair Service Curve for Best-Effort Service

The service curve model can easily be extended for best-effort service when no reservation or admission control is used. In this case, the absolute values of $m1$ and $m2$ are not important, as they specify only the relative service priorities between bursts of size less than β and the continu-

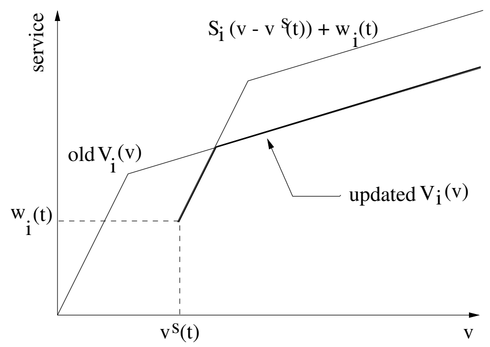


Figure 7: Illustration of the update of the virtual curve.

ously backlogged traffic in the system. We denote the ratio $m1/m2$ as the *Burst Preference Ratio* (BPR) and β as the *Preferred Burst Size* (PBS).

Since admission control is not necessary for best effort service, we can assign every flow in the system the *same* service curve $S(t)$, a concave curve similar to the one in Figure 2(a). The key performance tuning parameters are the burst preference ratio (BPR) $m1/m2$, and the preferred burst size (PBS) β. Intuitively, if a flow has idled for a long enough period of time, when it becomes backlogged again its first β bytes are served at a rate proportional to $m1$. However, if the flow remains backlogged for more than β bytes, its remaining bytes are served at a rate proportional to $m2$, i.e., BPR times lower than $m1$. Thus, if we set β to accommodate the most common burst sizes generated by applications such as WWW, we can provide a significantly lower delay for these applications than it is possible with PFQ.

Note that, unlike PFQ, FSC has “memory” in the sense that it can differentiate between flows that have previously idled and flows that are continuously backlogged and treat them differently. Also, when the system is congested, the long term rate of a flow, bursty or not, is still bounded by the fair share rate because in the long run every flow is serviced at a rate proportional to $m2$. Thus, while packet delay for bulk transfer type applications such as FTP may be increased momentarily, they always receive at least their fair share in the long run. Finally, it is interesting to note that when BPR = 1, or when PBS = 0, FSC degenerates to PFQ.

To give some intuition on how FSC behaves, consider a link shared by 15 constant-bit-rate UDP flows and one ON-OFF flow with a burst size of 32 packets. Figure 8 plots the arrival and departure times for each packet belonging to two consecutive burst periods of the ON-OFF flow. The plot shows the impact of the preferred burst size (PBS) in packets on the departure times,

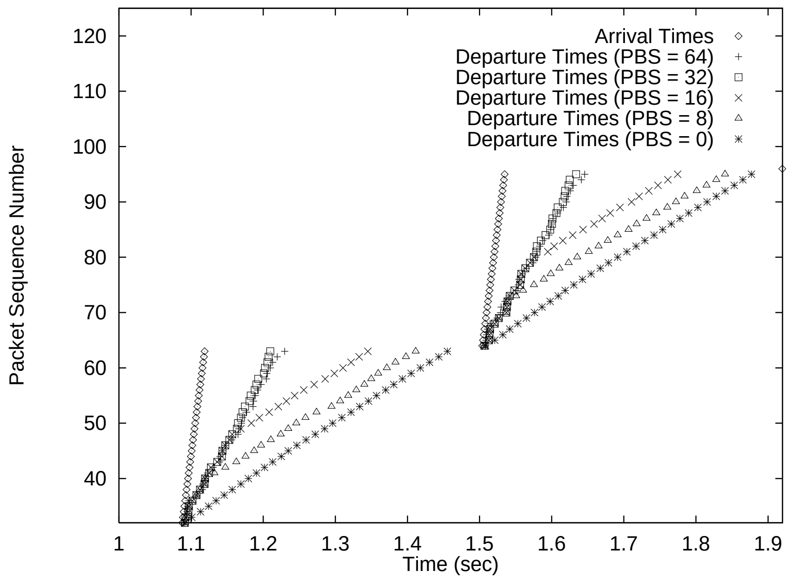


Figure 8: The packet arrival and departure times of a flow, with of 32 packet bursts, for various service curves.

and implicitly on the packet queueing delay, which is given by the horizontal distance between a packet's arrival time and its departure time. We associate to all flows the same service curve. In all cases the burst preference ratio (BPR) is 5. As expected, the delay decreases as PBS increases. Note that the packet departure times follow accurately the shape of the service curve associated with the flows.

3 Simulation Results

In this section we evaluate the FSC algorithm through extensive simulations. All simulations are performed in ns-2 [13], which provides accurate packet-level implementation of various network protocols, buffer management and scheduling algorithms. We examine the behavior of FSC under a taxonomy of transport protocol and traffic model combinations. For transport protocols, we use both TCP[4] and UDP. For traffic models, we use periodic ON-OFF sources, exponentially distributed ON-OFF sources, pseudo WWW traffic sources (a periodic ON-OFF source feeding into TCP), pseudo video (an ns-2 packet trace generated from a MPEG-1 video stream), Telnet, FTP and continuously backlogged UDP sources. The ON-OFF sources are based on our own

[4] Specifically, we use ns-2's implementation of TCP Reno without any protocol hand-shake.

implementation. We have extended ns-2 to support arbitrary traffic sources on top of TCP, and to dynamically create and destroy flows.

Different traffic sources have different performance indices. We measure the performance of ON-OFF sources and Telnet using average burst delay, which is define as the difference between the time when the last packet of the burst arrives at the destination and the time when the first packet of the burst is sent by the source. For continuously backlogged sources we use the overall throughput to measure performance, and for video traffic we use the frame delay distribution. A potential problem when measuring the burst delay under UDP is that some packets may be dropped. For this reason, in the case of UDP sources we report both the average burst delay and the packet dropping rate.

In all simulations, we distinguish between foreground flows which are bursty, and background flows which are persistent. The actual number of foreground and background flows may vary. Unless otherwise specified, the following parameters are used in all simulations. The capacity of each link is 10 Mbps with a latency of 10 ms, and the output buffer size is 128 KB. We use a per-flow buffer management scheme which drops the second packet from the longest queue when the buffer overflows [22]. In addition, the size of all packets is 1000 bytes except for Telnet, which uses 64 byte packets. The simulation time is 20 seconds.

Each set of the results presented in the following sub-sections is aimed to illustrate certain aspects of the FSC algorithm. First, to illustrate the behavior of FSC and to show how variations in the system parameters affect its performance, we use a simple network topology consisting of one link, and traffic sources such as ON-OFF, FTP, and continuously backlogged UDP. In these simulations we also draw comparisons between FSC and PFQ. Next, to show how FSC performs under more realistic load, we use more complex network topologies and traffic sources, such as exponentially distributed ON-OFF, pseudo video, and Telnet. To quantify the impact on long lived throughout oriented traffic sources, we use an experiment where the short bursty traffic sources are specifically designed to extract the maximal benefit from FSC. To explain the tradeoffs in optimizing FSC's parameters for realistic network traffic, we generate a synthetic workload of FTP flows using data from AT&T Labs' recent Internet traffic analysis [6].

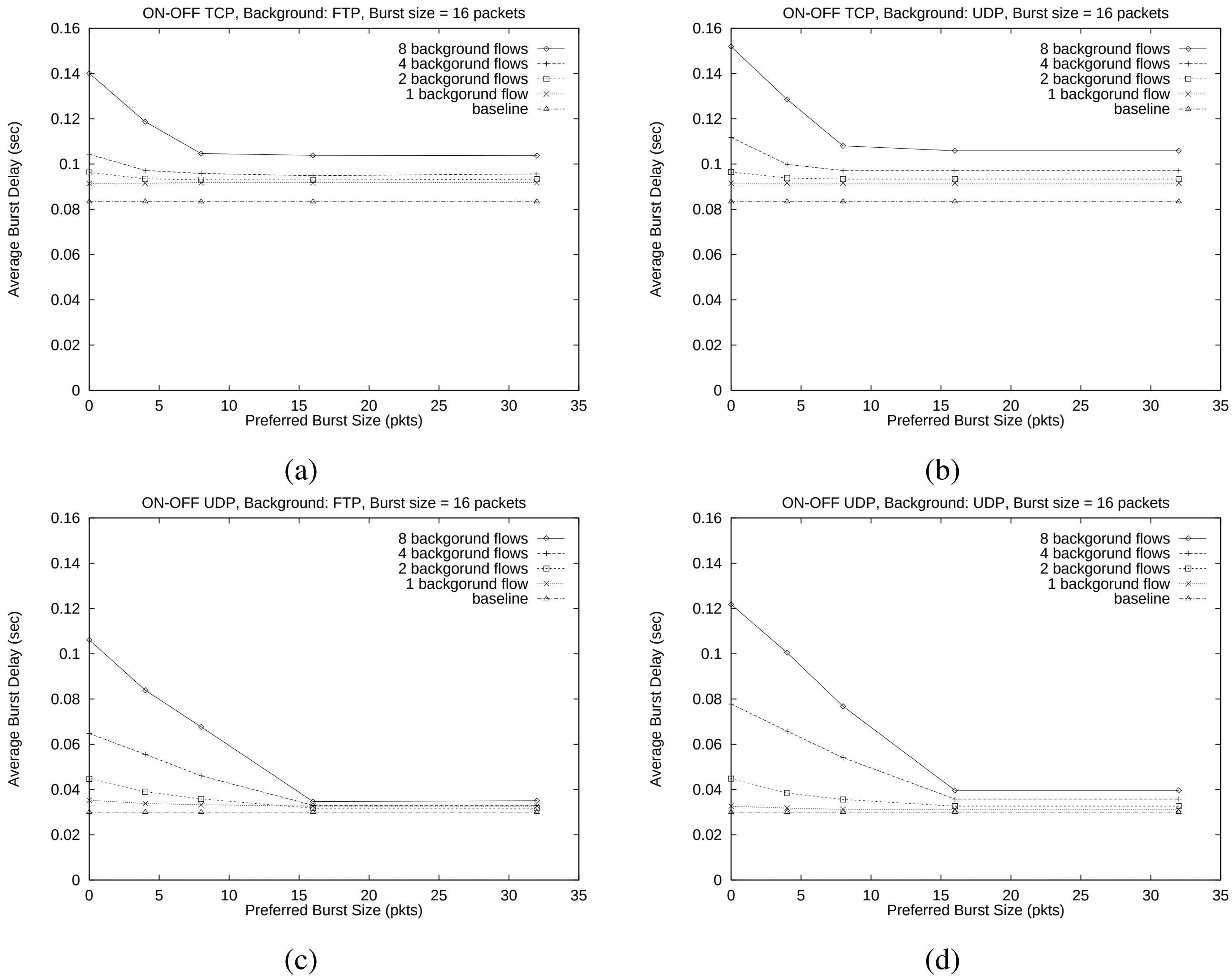


Figure 9: The average burst delay vs PBS for four simulation scenarios, each having 16 flows. Note that the average delay decreases as the PBS increases, up to the burst size of 16 packets.

3.1 Basic Demonstrations

All simulations presented in this sub-section use periodic ON-OFF foreground sources with a period of one second and a peak rate of 4 Mbps. Since the packet size is 1000 bytes, the inter-packet arrival time is 2 ms. All flows within the same simulation have the same burst size and the bursts occur at the beginning of each period. The average burst delay is used as the performance index. To introduce some randomness, the starting times of the flows are drawn from an exponential distribution. Although such a simplistic traffic pattern might not be an accurate simulation of Internet traffic, it makes it easier to understand and analyze the interactions between various parameters, such as the preferred burst size (PBS), the burst preference ratio (BPR), and the percentage of the background persistent traffic. In Sections 3.3, 3.4 and 3.7, we show that the results obtained by using simple periodic ON-OFF traffic are consistent with the ones obtained by using more realistic traffic sources.

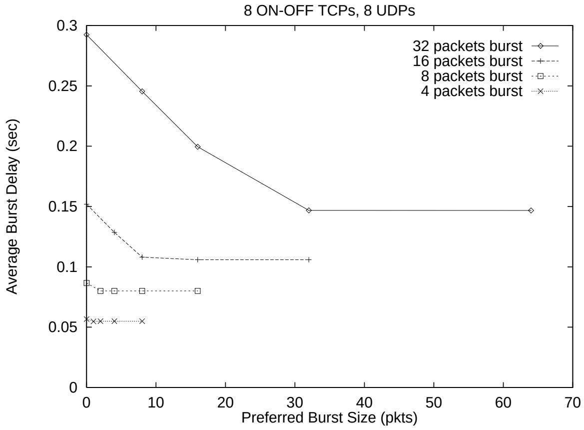
3.1.1 Impact of Preferred Burst Size (PBS)

In this section we study the impact of the preferred burst size (PBS) and the number of background flows on the behavior of FSC. Note that when PBS is zero, FSC is equivalent to PFQ, as the service curve reduces to a straight line intersecting the origin. We consider 16 flows sharing a congested link. The number of persistent background flows varies from 1 to 8. Figure 9 plots the average burst delay as a function of PBS in four different scenarios using all possible combinations of foreground TCP and UDP ON-OFF traffic, and background FTP and constant bit rate UDP traffic. In the scenarios where UDP background is used, the aggregate rate of the background flows is set at twice the link capacity in order to create congestion. In all cases the burst size is 16 packets, and the burst preference ratio (BPR) is 5. As a baseline comparison, in each figure we also plot the average burst delay of an ON-OFF flow that uses an unloaded link.

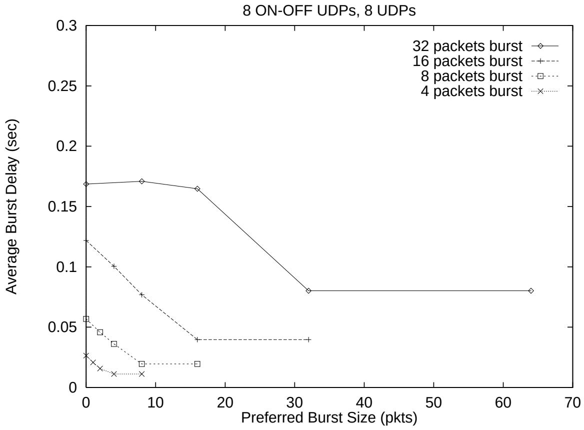
As can be seen in Figure 9, in all scenarios the average burst delay decreases as PBS increases. This is to be expected since a larger PBS results in a larger percentage of the burst of each flow being served at a higher priority. This is because the packets' deadlines are computed based on the first slope $m1$ of their service curves. Moreover, in all four graphs, the data points for PBS equals zero is the corresponding performance points of PFQ under the same scenarios. Clearly, FSC out-performs PFQ in providing low burst delay.

There are three other points worth noting. First, the average delay does not decrease after PBS exceeds the burst size of 16 packets. This is because when PBS reaches the burst size, all packets are already served at the highest priority. We defer a discussion of the implications of setting the PBS too *large* to Section 3.7.

Second, as the number of background flows increases, the relative amount of improvements in the average burst delay also increases. This is because the background flows are continuously backlogged and therefore the deadlines of their packets are computed based on the second slope $m2$ of their service curves most of the time. This increases the relative priority of the ON-OFF flows as the deadlines of their packets are computed based on the first slope $m1$ of their service curves, which is greater than $m2$. Intuitively, as the percentage of background traffic



(a)



(b)

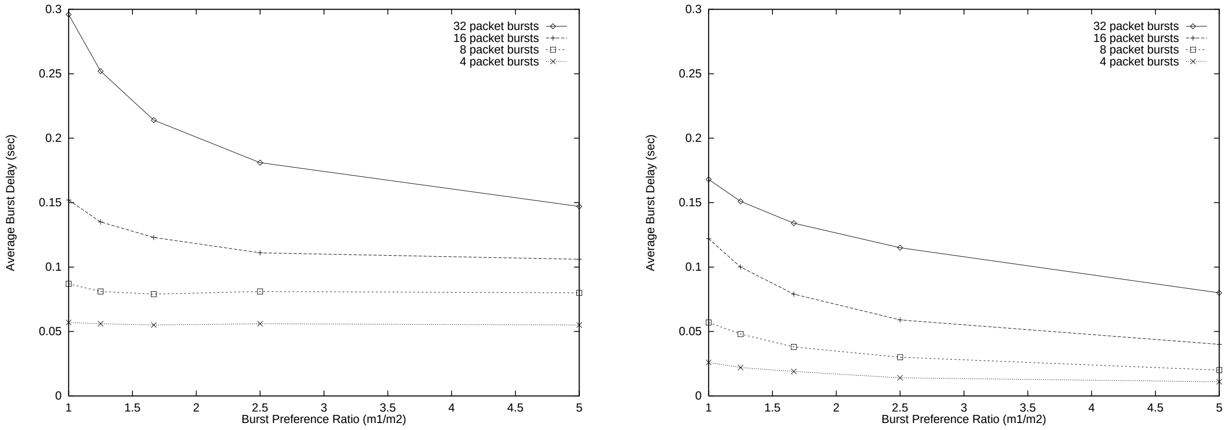
Figure 10: The average burst delay of eight ON-OFF TCP/UDP flows as a function of PBS for various burst sizes.

increases, there are more "opportunities" to shift the delay from the ON-OFF traffic towards the continuously backlogged traffic.

Third, the relative amount of improvements in the average burst delay is larger when the foreground traffic uses UDP (see Figure 9(c),(d)) than when it uses TCP (see Figure 9(a),(b)). This is because the TCP protocol makes use of acknowledgements, which add a fixed overhead, in terms of round-trip-time, to the burst delay. This is evident from the "baseline" plots, where only one flow is backlogged. It takes roughly three times longer to send the same burst under TCP than under UDP.

Since the simulation scenarios that employ the same foreground traffic exhibit similar trends, in the remaining of this section we will limit our study to two scenarios: ON-OFF TCP foreground with UDP background, and ON-OFF UDP foreground with UDP background. The reason for choosing UDP over FTP as the background traffic is to factor out the variations due to FTP dynamics. Finally, unless otherwise specified, we only consider the 8 foreground flows / 8 background flows case.

The next experiment illustrates the impact of the burst size on the behavior of FSC (see Figure 10). As a general trend the average burst delay decreases as PBS increases. As shown in Figure 10(a), when the ON-OFF traffic is TCP, the decrease in the average burst delay is more significant for larger bursts. When the burst size is small, the burst delay is dominated by the round-trip-time as the sender waits for acknowledgements, thus a reduction in the packet



(a) (b)

Figure 11: The average burst delay versus BPR for eight flows when the ON-OFF traffic is (a) TCP, and (b) UDP, respectively.

queueing delay due to FSC does not translate into a significant reduction of the overall burst delay. When the ON-OFF traffic is UDP, it is interesting to note that for a burst size of 32 packets, there is little improvement in the average burst delay between PBS $= 0$ and PBS $= 4$ packets (see Figure 10(b)). The reason is that, when PBS $= 0$, 32.5% of the packets are dropped, while when PBS $= 4$ packets, only 15 % of the packets are dropped. Thus, although the average burst delay does not change between PBS $= 0$ and PBS $= 4$ packets, there are actually more packets delivered when PBS $= 4$ packets. The percentage of dropped packets reduces to 1.5 % for PBS $= 8$ packets, and no packet is dropped when PBS ≥ 16 packets.

3.1.2 Impact of Burst Preference Ratio (BPR)

In this section we study the effects of the Burst Preference Ratio (BPR) on the behavior of FSC. We consider two simulation scenarios: UDP foreground with UDP background, and TCP foreground with UDP background. For each experiment we set PBS to be the same as the burst size of the flows and vary the BPR. As shown in Figure 11, in both cases the average burst delay decreases as the BPR increases. This is expected since increasing BPR results in an increase of the relative priority of the bursty traffic. Also, similar to the previous experiment (see Figure 10), FSC is more effective for larger burst sizes, especially when the ON-OFF traffic is TCP. Again, notice that the data points for BPR$= 0$ are the corresponding performance points of PFQ under the same scenarios. The advantage of FSC over PFQ can be seen clearly.

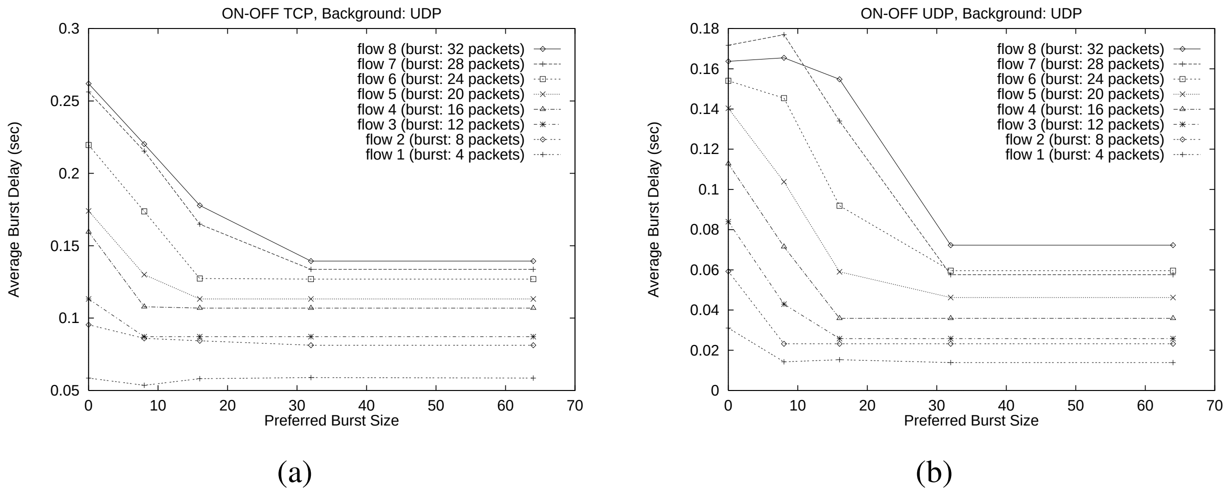


Figure 12: The average burst delay versus PBS for eight flows, with burst sizes between 4 and 32 packets, when the ON-OFF traffic is (a) TCP, and (b) UDP, respectively.

3.2 Non-homogeneous ON-OFF Sources

Now that we have demonstrated the basic features of FSC, we begin to consider more complex traffic sources. In this section, we consider again a congested link shared by eight ON-OFF flows and eight background UDP flows. However, unlike the previous experiments in which all flows have bursts of the same size, in this experiment each flow has a different burst size. More precisely, the burst size of flow i is $4 \times i$ packets, where $1 \leq i \leq 8$. Our goal is to study how the average burst delay of each flow is affected by the preferred burst size (PBS). The results for both TCP and UDP ON-OFF foreground traffic are shown in Figure 12.

In the first scenario (see Figure 12(a)) the average burst delay of each flow decreases as PBS increases. As expected, the average burst delay of a flow no longer decreases once PBS exceeds the flow's burst size. However, in the second scenario when all flows are UDPs (see Figure 12(b)), the average burst delay for flows with large burst sizes actually increases initially as PBS increases. This is because more packets are being transmitted as PBS increases. For example, when PBS $= 0$, 33.5% of the packets of flow 8 are being dropped, when PBS $= 8$ packets, the dropping rate reduces to 15%. Finally, for PBS ≥ 16 packets, no packet is dropped.

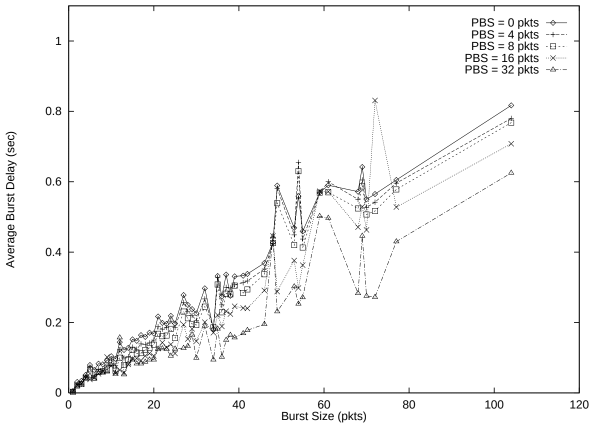


Figure 13: The average burst delay versus burst size for eight ON-OFF TCP flows, which have burst sizes exponentially distributed with a mean of 16 packets. The flows compete with other eight constant-bit-rate UDP flows.

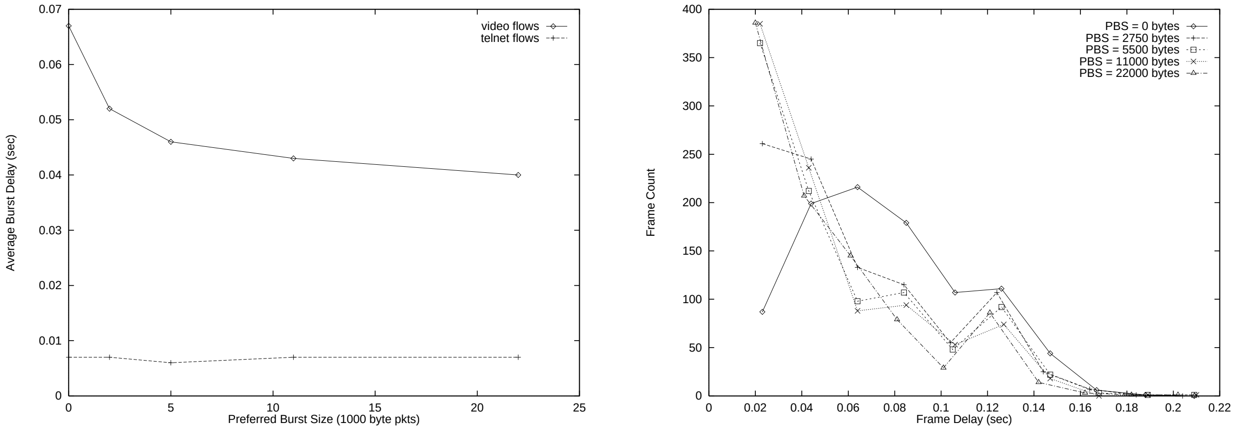
3.3 Exponential ON-OFF Sources

In the previous section we have assumed that the ON-OFF traffic is periodic and that each flow has a fixed burst size. While this setting makes it easier to understand the behavior of FSC, it is not realistic. For this reason, we consider a more realistic ON-OFF traffic source whose burst size is exponentially distributed. Since we intend to model WWW-like traffic, we assume only TCP ON-OFF foreground traffic. Again we consider eight foreground and eight background flows sharing the same link. The mean of the burst size is 16 packets. In order to obtain more data points we increase the simulation time to 100 seconds.

Figure 13 shows the average burst delay versus burst size. In general, the average burst delay improves as PBS increases, with the improvements being more significant for larger burst sizes. These results are consistent with the ones presented in Sections 3.1.1 and 3.2. The prominent peaks in the delays are likely caused by TCP timeouts as a result of packet loss.

3.4 Mixed Application Traffic

In this section we study how effective FSC is in dealing with a mix of traffic sources. For this we consider a more complex simulation scenario in which 20 flows share the same link. Out of these 20, two are MPEG-1 video flows sending at their fair rate, three are Telnet flows, five are FTP flows, and the last 10 are background UDP flows. The video flows have a maximum frame



(a) Entire Range (b) Short Flows

Figure 14: (a) The average frame delay for MPEG-1 video traffic and the average packet delay for Telnet traffic versus PBS. In this experiment we consider two video flows and three Telnet flows that share a link with five FTP flows and 10 background UDP flows. (b) The frame delay distribution for the video traffic for different PBS values.

size of 11 packets. The packet size for all flows is 1000 bytes, except for Telnet which uses 64 byte packets.

Figure 14(a) shows the average burst delay versus preferred burst size. For the video flows, we assume that a burst consists of one frame, while for the Telnet flows, we assume that a burst consists of one packet. FSC is able to significantly reduce the average frame delay for the video traffic. When PBS exceeds the maximum frame size, we obtain up to 50% improvement. However, PBS does not affect the packet delay of the Telnet traffic. This is because the Telnet sources are sending at an extremely low rate with very small packet size compared to the other flows. Therefore their packets are immediately sent regardless of the value of PBS. We expect FSC to have a more significant impact on Telnet when the fair share rate is closer to the Telnet session's rate. Finally, Figure 14(b) shows the distribution of the frame delay for the video traffic. As expected, the tail of the distribution decreases as PBS increases.

3.5 Multiple Links

In this experiment we study the behavior of FSC when a flow traverses multiple congested links. We use the topology in Figure 15. Each congested link is traversed by 10 background UDP flows which send at their fair rate. To reduce the effects of the number of hops on the round-trip

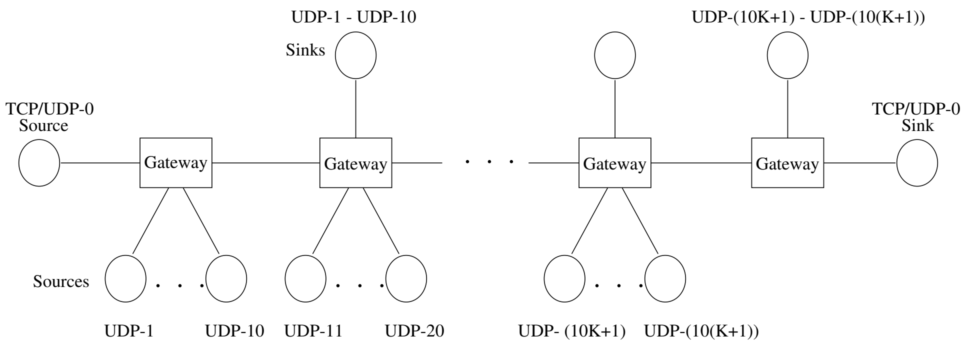
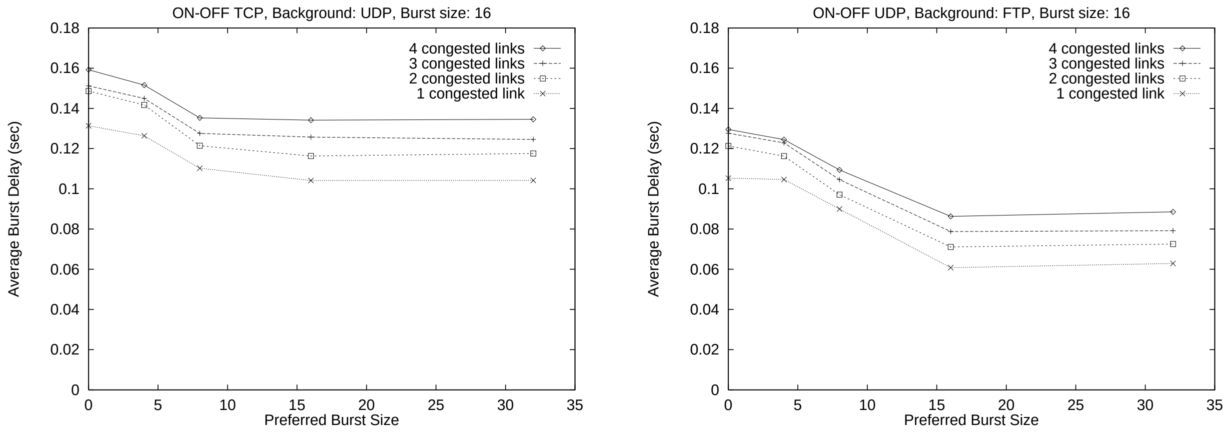


Figure 15: Topology for analyzing the effects of multiple congested links on the throughput of a flow. Each congested link is traversed by ten cross flows (all UDPs).



(a) (b)

Figure 16: (a) The average burst delay versus PBS as the number of bottleneck links varies. The foreground traffic consists of one ON-OFF TCP. (b) The same experiment when the foreground traffic consists of one ON-OFF UDP.

time we fix the end-to-end propagation delay at 10 ms. The link latencies are then computed by dividing the end-to-end propagation delay by the number of links. However, note that due to the queueing delay the round-trip time will still increase with the number of hops. Finally, we assume a periodic ON-OFF flow with 16 packet bursts that traverses all congested links.

Figure 16 shows the average burst delay of the ON-OFF flow versus PBS for 1, 2, 3 and 4 congested links. We consider two scenarios where the foreground flow is TCP or UDP. In both cases the average burst delay decreases as PBS increases. However, as we have observed in previous experiments (see for example Section 3.1.1), the improvements are larger when the foreground traffic is UDP. Finally, note that the absolute improvements in the average burst delay

	Average burst delay	TCP throughput
FSC	21.81 ms	1.66 Mbps
PFQ	54.79 ms	1.67 Mbps

Table 1: Comparison of background TCP throughput using a FSC worst case flow arrival.

are basically independent of the number of congested links.

3.6 Impact on Background Traffic

In the previous experiments, we have shown that FSC is effective in reducing the average burst delay of bursty traffic. A natural question to ask is, could this improvement in bursty traffic performance negatively affect the background persistent traffic?

To answer this question, we construct a simulation scenario in which the bursty traffic flows take *maximal* advantage of the benefit provided by FSC and thus put the persistent background traffic in the worst possible position under FSC. To achieve maximal benefit in FSC, each bursty flow should send exactly as much data as the PBS, and the bursty flows should be back-to-back so that each and every burst is served at the highest priority (along the first slope) under FSC.

In this experiment, we use one persistent background TCP flow and a series of bursty foreground UDP flows, each sending 10 packets of 1000 bytes each. Under FSC, we choose the PBS to be 10 packets and the BPR to be 5. With a 10 Mbps link, this implies that a burst can be served at the maximal rate of 8.33 Mbps under FSC. Therefore, to make the bursty flows back-to-back under FSC, the inter-flow arrival time needs to be 9.6 ms. Using this traffic arrival pattern, we compare the performance of FSC against PFQ (FSC with PBS = 0). Table 1 shows the performance of the bursty flows and the TCP flow under the two different algorithms.

Under FSC, the throughput of the TCP flow is exactly as expected (one-sixth of 10 Mbps) since the UDP bursty flows are always being served at the highest priority and consume 8.33 Mbps. What is somewhat surprising is that the TCP throughput is essentially unchanged under PFQ even though the bursty flows are served at the same priority as the TCP flow. This seemingly contradictory result is simple to explain. Under FSC, only one bursty UDP flow is backlogged at any given time; in contrast, under PFQ, five bursty UDP flows are simultaneously backlogged throughout the simulation (except during the very beginning) because the bursty flows are no longer served at a special high priority and they take five times longer to finish. Therefore, with 5 UDP flows and one TCP flow, the TCP flow simply gets its fair share under PFQ, which is 1.67 Mbps. The reason why this is slightly larger than the TCP throughput under FSC is that during the beginning of the simulation, there are less than 6 flows simultaneously backlogged. This result shows that even under a worst case bursty flow arrival scenario, FSC provides virtually the same performance to a persistent TCP flow as PFQ.

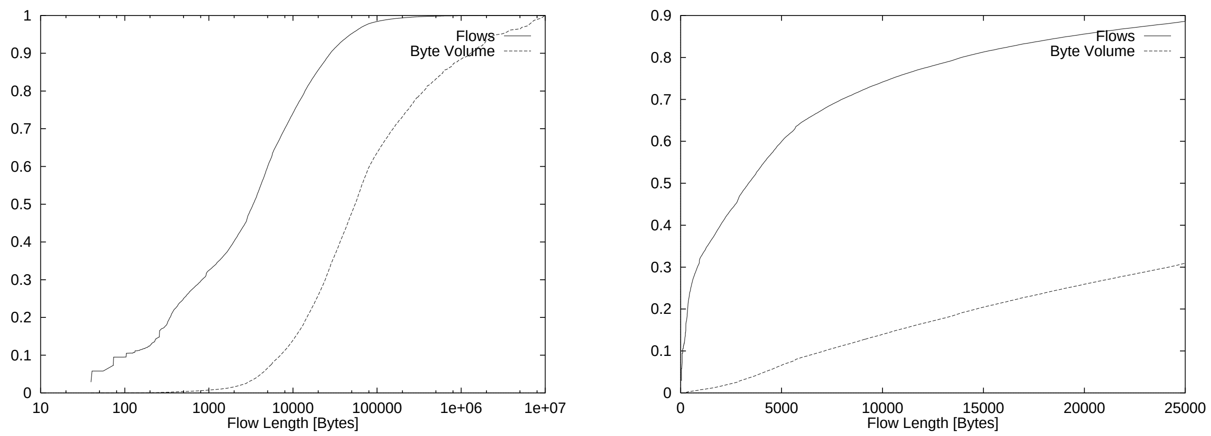
Without any adverse effect on the performance of the TCP flow, FSC is again able to bring the average burst delay of the UDP flows down to 22 ms, of which 10 ms is the link propagation delay. In other words, the queueing delay is reduced by almost a factor of 4 compared to PFQ.

3.7 Performance for WWW traffic

So far, we have shown that FSC can reduce the average delay of bursty traffic without adversely affecting the background persistent traffic. The improvements are most pronounced when the number of background sessions is large and when the PBS corresponds to the burst size of the sessions in the foreground. However, they leave the question of how to configure FSC for realistic traffic largely unanswered.

As we increase the preferred burst size (PBS), we increase the percentage of flows and bytes that will be completely covered by the PBS. The byte-volume of traffic that is *not* covered by the PBS determines the amount of background traffic. As we have shown earlier, the delays of short bursts are reduced as the amount of background traffic increases. Thus, increasing the PBS will reach a point of diminishing and then negative returns when less traffic exists in the "background". At the limit, if we set the PBS to be greater than or equal to the length of the longest flow, all data will be serviced along the first slope of the service curve and FSC will again be equivalent to PFQ. An analogous problem exists for the BPR. As in the limit, if we set the BPR very large, background traffic could see no service while bursts are being served. Thus, to maximize the benefit of FSC, we would like to choose a PBS that encompasses a relatively large percentage of the flows while covering a relatively small percentage of the byte volume and choose a BPR that can significantly reduce the delays of these bursts without adverely affecting background traffic.

In order to answer these questions, we employ flow length data from AT&T Labs' recent



(a) Entire Range (b) Short Flows

Figure 17: Cumulative probability distribution of flow lengths and their portion of the total byte volume.

Internet traffic analysis [6]. Figure 17 shows the probability that a host-level flow has up to x bytes, and the contribution of these flows to the cumulative byte count. For example, while 60% of the host-level flows are less than 5000 bytes in length, these flows constitutes approximately only 7% of the byte volume of the trace. For this traffic distribution, choosing a PBS of x bytes will completely cover all the flows up to x bytes in length, and their corresponding byte volume. The actual coverage will be larger than this, as longer lived flows that consist of periodic short bursts may transmit at a low enough sustained rate so that their entire transmission is transmitted along the first slope.

To determine how to configure FSC's parameters for WWW traffic, we generate a synthetic workload of FTP traffic, whose flow lengths are chosen to model this distribution. We divide the flows into 10 groups, each representing 10 % of the flows, and compute the average flow length within each group, as shown in Table 2. Based on the average flow length of $13,666$ bytes, we generate a synthetic workload of FTP traffic via a Poisson process with a mean flow arrival rate corresponding to 95% of the link capacity and select among the 10 groups uniformly to determine the flow length. We run these simulations for 1 minute of simulation time over a 10Mbps link with a 2 ms latency while setting the maximum segment size of the TCP sources to 576 bytes. Figure 18 plots the average transfer time experienced by flows in groups 7 through 10 as we vary the BPR from 1 to 10 and the PBS from 0 KB to 100KB.

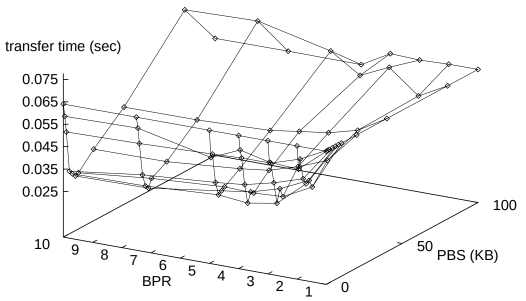
Note that all points with PBS = 0 and/or BPR = 1 correspond to PFQ. Groups 0 through 3 are

Group	Average Flow Length
0	61
1	239
2	539
3	1349
4	2739
5	4149
6	6358
7	10910
8	19878
9	90439

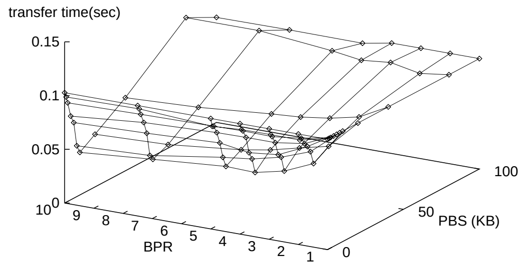
Table 2: Average flow length per group (bytes).

sufficiently small and short lived that PFQ and FSC have roughly equivalent performance, while groups 4 through 6 have analogous improvements to those shown here. While our earlier results have shown minimal impact on background traffic, Figure 18 (d) shows that Group 10 in fact sees a noticable impact with large PBS settings. The reason is that the buffer resources in this system, while shared, are finite. In this study, when the buffer resources are depleted, a packet is pushed out from the longest queue [4]. Thus, the longest flows (Group 10) will incur the losses when the link becomes congested. As our measurements include all packets required to complete the FTP transfer, this explains the impact. Because this practical constraint cannot be avoided in actual systems, this encourages us to configure the system with conservative settings.

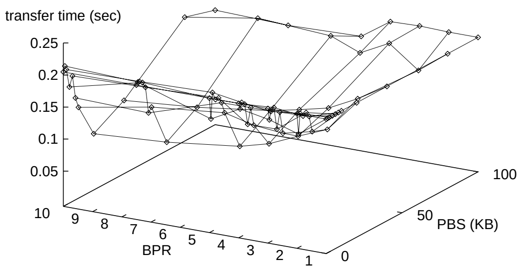
While a flow's delay is minimal when its length corresponds to the PBS, minimal additional improvements are seen with BPR greater than 4. The larger the PBS, the higher the percentage of flows that are entirely covered by the first slope and the performance returns to that of PFQ. For this simulation set, setting BPR = 4 and PBS = 6000 bytes reduces transfer times of most groups (some by over 50%) while only increasing the transfer time of the largest group by 1%.



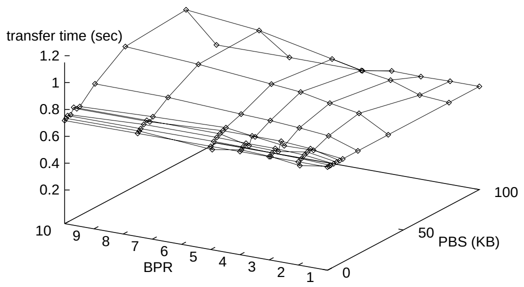
(a) 60 to 70th Percentile Flows



(b) 70 to 80th Percentile Flows



(c) 80 to 90th Percentile Flows



(d) 90 to 100th Percentile Flows

Figure 18: The average flow transfer time as a function of PBR and PBS for simulated WAN groups

4 Implementation Issues

The implementation complexity of a per-flow queueing system can be divided into three separate tasks: classification, buffer management, and the scheduling itself. While these components are usually discussed in the context of networks that provide performance guarantees to individual flows, the complexity is significantly reduced when they are applied for best-effort traffic. Traffic requiring bandwidth and/or delay guarantees need to employ a signaling mechanism to request resource capacity from the network. In order to identify these guaranteed flows, the classification module needs to uniquely map traffic into their allocated queues. As individual guaranteed flows have unique demands for system resources, the scheduling and buffer management options for the queues need to be individually programmed.

While this flexibility is required for reserved traffic, best effort applications have much looser constraints. Since our goal is equal treatment for individual flows, a mechanism should be employed to persistently map each individual flow into its own queue. Perfect isolation and fairness between flows, while desirable, is not as critical for best effort applications. McKenney proposed identifying individual flows by means of a probabilistic hashing algorithm in [12]. If the number of queues is less than the number of active flows, collisions will occur. The hash key can periodically be changed to alleviate the unfairness caused by these collisions. Content Addressable Memories (CAMs) or other devices capable of dynamically binding flows to queues could also be employed. However, it is preferable that the binding be persistent. Otherwise a user could exploit the fact that bursts up to the PBS receive an increased share when arriving into a newly assigned queue. We note that these or other mechanisms used for flow identification could also be used for the FRED [11] scheme, and vice versa.

For any per-flow management scheme, there is an $O(N)$ space requirement to maintain the state for each flow. As noted in [22], with today's memory prices, and historic trends, it appears that this is an insignificant factor in the cost of products for the features that can be provided. A more relevant concern is the amount of computational resources required per-packet to perform the scheduling. The implementation cost of a scheduling algorithm can be divided into three components: (1) computation of the system virtual time function, (2) calculation of the session timestamps used for scheduling, and (3) sorting the timestamps by the specified criteria. The FSC system virtual time function is simply the average of the largest and smallest start time of any flow in the system. The timestamps of the sessions in FSC can be sorted using the packet scheduling architecture presented by Stephens et al [19]. The one remaining issue we must address is the calculation of the timestamps used for scheduling.

Recall that in both PFQ and FSC, the virtual time of a flow, $v_i(t)$, is computed as $V_i^{-1}(w_i(t))$, where $w_i(t)$ is the amount of service received by flow i by time t, and V_i is the virtual curve of flow i. The main difference between PFQ and FSC is that while in PFQ the virtual curve is a linear function, i.e., $V_i(t) = \phi_i \cdot t$, in FSC it is a two-piece linear function given by Eq. (5). As a result, computing the virtual time in FSC requires an additional comparison to detect whether $w_i(t)$ has exceeded V_i's inflection point. Another difference is that in FSC we need to update the virtual curve according to Eq. (5). The important thing to note here is that this operation is done only when a flow becomes backlogged, and as shown in [21], it takes constant time. Thus, implementing FSC is just marginally more complex than PFQ.

5 Related Work

Scheduling support for providing low burst delay has been studied in the context of ATM VBR traffic scheduling. In [10], Lam and Xie have proposed burst scheduling, which includes a new VBR traffic flow model based on bursts. Under this model, each burst is described by the burst size and the burst rate. This information is then carried in the first packet of a burst to provide resource requirement hints to the scheduler. Admission is performed at the burst level. In the event of an admission control failure, the entire burst is discarded. For packets admitted into the buffer, the scheduler uses a virtual clock server to serve the current admitted bursts at their burst rates. These mechanisms ensure that admitted bursts experience low delay. In addition, due to statistical multiplexing, it is possible to maintain the burst drop rate to a minimum. When a zero loss rate is required, a reservation can be made using the peak burst rate. Although burst scheduling is aimed to guarantee burst delay for VBR traffic in ATM, the ideas can certainly be applied to other networking technologies such as IP.

The FSC algorithm differs from burst scheduling in several ways. First of all, burst scheduling was designed for supporting real-time traffic in a connection-oriented network. It utilizes signaling information carried in packets and performs admission control. In contrast, FSC does not need explicit signaling to specify the beginning and the end of a burst and can give bursty traffic a lower burst delay. Secondly, burst-scheduling uses the Virtual Clock service discipline. While this may be acceptable for video traffic with certain intrinsic rates, Virtual Clock does not support adaptive bursty data traffic well due to the lack of the fairness property. FSC, on the other hand, provides long term fairness similar to PFQ algorithms. In this way, it allows statistical sharing among competing flows, and at the same time provides protection. Thus, FSC is more suitable for the diverse traffic mix environment in today's Internet.

Implementing per-flow scheduling in isolation does not necessarily lead to improved performance over FIFO forwarding without appropriate buffer management. This is because the scheduler can only operate on packets that are stored in the system buffers. If the buffer management is FCFS, a flow can monopolize the link. To prevent this lockout behaviors, Davin and Heybey [4] proposed to push-out a packet from the longest queue. A recent simulation study [22] evaluated the benefits of per flow scheduling and buffer management. They found that both fair allocation of buffer and link capacity improve application level fairness. Furthermore, they illustrated that the benefits were complementary. The buffer management algorithm with the best performance was a push-out scheme that chose the packet from the front of the longest queue. They picked the “front” packet in order to improve the TCP feedback and eliminate the possibility of losing a later acknowledgement. Since dropping the actual first packet from the queue would require complex interactions with the scheduler, they actually dropped the second packet. We have also adopted this buffer management scheme in our study.

6 Summary

In this paper we investigate the Fair Service Curve (FSC) algorithm in the context of supporting best-effort service. We show that it out-performs PFQ algorithms under a variety of traffic conditions. In particular, FSC can significantly improve the delay performance for bursty traffic without negatively affecting the throughput performance for continuously backlogged traffic. We have conducted extensive simulation experiments that involve a large variety of traffic sources, such as periodic and exponentially distributed ON-OFF sources, MPEG-1 video sources, FTP, and Telnet. As a general trend the average burst delay decreases as:

1. the Preferred Burst Size (PBS) increases. This is because the portion of the burst which is served according to the first slope of the service curve increases. The only exception is when the bursty traffic is UDP and the number of packets successfully transmitted increases with PBS. Note that although the average burst delay might not decrease in this case, the flow still gets better service as fewer packets are being dropped.

2. the Burst Preference Ratio (BPR) increases. This is because the relative priority of the packets served according to the first slope of the service curve increases.

3. the number of background continuously backlogged traffic flows increases. Since the traffic of the backlogged flows is served according to the second slope of the service curve, the larger the number of background flows, the higher the relative priority of the traffic that is served according to the first slope. Intuitively, more background traffic provides more “opportunities” to shift the delay from the bursty traffic towards the background traffic.

In order to determine practical settings for FSC for best-effort traffic, we generated a synthetic workload of WWW traffic. This exposed the limitations of setting the PBS and BPR too large for a given traffic pattern. For this set of web traffic traces, setting BPR = 4 and PBS = 6000 Bytes provides a significant reduction in the transfer time of the majority of flows without noticable impact on the background traffic.

Our results show that it is possible to significantly improve the delay of short and bursty flows without affecting the long term throughput of persistent flows. Since compared to PFQ, the added complexity to implement FSC is minimal, we conclude that FSC represents a viable alternative to support best effort traffic in today's Internet.

7 Acknowledgement

We would like to thank Anja Feldmann, Jennifer Rexford, and Ramon Caceres for providing us with the Internet traffic distribution data from their study.

References

[1] J.C.R. Bennett and H. Zhang. WF^2Q: Worst-case fair weighted fair queueing. In *Proceedings of IEEE INFOCOM'96*, pages 120–128, San Francisco, CA, March 1996.

[2] R.L. Cruz. Service burstiness and dynamic burstiness measures: A framework. *Journal of High Speed Networks*, 1(2):105–127, 1992.

[3] R.L. Cruz. Quality of service guarantees in virtual circuit switched network. *IEEE Journal on Selected Areas in Communications*, 13(6):1048–1056, August 1995.

[4] J. Davin and A. Heybey. A simulation study of fair queueing and policy enforcement. *Computer Communication Review*, 20(5):23–29, October 1990.

[5] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *Journal of Internetworking Research and Experience*, pages 3–26, October 1990. Also in Proceedings of ACM SIGCOMM'89, pp 3-12.

[6] A. Feldmann, J. Rexford, and R. Caceres. Efficient policies for carrying web traffic over flow-switched networks. *IEEE/ACM Transactions on Networking*, pages 673–685, December 1998.

[7] S.J. Golestani. A self-clocked fair queueing scheme for broadband applications. In *Proceedings of IEEE INFOCOM'94*, pages 636–646, Toronto, CA, April 1994.

[8] P. Goyal, H.M. Vin, and H. Chen. Start-time Fair Queuing: A scheduling algorithm for integrated services. In *Proceedings of the ACM-SIGCOMM 96*, pages 157–168, Palo Alto, CA, August 1996.

[9] V.P. Kumar, T.V. Lakshman, and D. Stiliadis. Beyond best effort: Router architectures for the differentiated services of tomorrow's internet. *IEEE Communcations Magazine*, May 1998.

[10] S. S. Lam and G. G. Xie. Burst scheduling networks, June 1998. Technical Report TR-94-20 (3rd revision), University of Texas at Austin.

[11] D. Lin and R. Morris. Dynamics of random early detection. In *Proceedings of ACM SIGCOMM '97*, pages 127–137, Cannes, France, October 1997.

[12] P. McKenney. Stochastic fair queueing. In *Proceedings of IEEE INFOCOM'90*, San Francisco, CA, June 1990.

[13] Ucb/lbnl/vint network simulator - ns (version 2).

[14] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control - the single node case. In *Proceedings of the INFOCOM'92*, 1992.

[15] A. Parekh and R. Gallager. A generalized processor sharing approach to flow control - the single node case. *ACM/IEEE Transactions on Networking*, 1(3):344–357, June 1993.

[16] S. Suri and G. Varghese and G. Chandranmenon. Leap Forward Virtual Clock. In *Proceedings of INFOCOM 97*, Kobe, Japan, April 1997.

[17] H. Sariowan, R.L. Cruz, and G.C. Polyzos. Scheduling for quality of service guarantees via service curves. In *Proceedings of the International Conference on Computer Communications and Networks (ICCCN) 1995*, pages 512–520, September 1995.

[18] M. Shreedhar and G. Varghese. Efficient fair queueing using deficit round robin. In *Proceedings of SIGCOMM'95*, pages 231–243, Boston, MA, September 1995.

[19] D.C. Stephens, J.C.R. Bennett, and H.Zhang. Implementing scheduling algorithms in high speed networks. *IEEE Journal on Selected Areas in Communications:Special Issue on Next-generation IP Switches and Router*, 17(6):1145–1158, June 1999.

[20] D. Stiliadis and A. Verma. Design and analysis of frame-based fair queueing: A new traffic scheduling algorithm for packet-switched networks. In *Proceedings of ACM SIGMETRICS'96*, May 1996.

[21] I. Stoica, H. Zhang, and T.S.E. Ng. A Hierarchical Fair Service Curve algorithm for link-sharing, real-time and priority services. In *Proceedings of the ACM-SIGCOMM 97*, Cannes, France, August 1997.

[22] B. Suter, T.V. Lakshman, D. Stiliadis, and A. Choudhury. Design considerations for supporting tcp with per-flow queueing. In *INFOCOM'98*, San Francisco, CA, March 1998.

[23] G. Xie and S. Lam. Delay guarantee of virtual clock server. *IEEE/ACM Transactions on Networking*, 3(4):683–689, December 1995.

[24] L. Zhang. Virtual clock: A new traffic control algorithm for packet switching networks. In *Proceedings of ACM SIGCOMM'90*, pages 19–29, Philadelphia, PA, September 1990.