Supporting Best-Effort Traffic with Fair Service Curve*

T.S. Eugene Ng[1], Donpaul C. Stephens[1,2], Ion Stoica[1], Hui Zhang[1]

[1]Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213, USA
{eugeneng, istoica, hzhang}@cs.cmu.edu

[2]Bell Laboratories
101 Crawfords Corner Road
Holmdel, NJ 07733, USA
donpaul@bell-labs.com

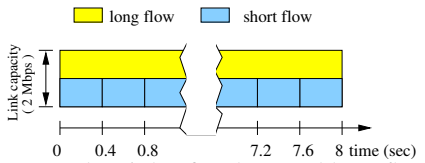
Abstract

While PFQ algorithms can provide per-flow end-to-end delay guarantees for real-time traffic or protection among competing best-effort traffic, they 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. When used for real-time traffic, this can result in network under-utilization. The second and less well known limitation is that, when used for best-effort traffic, PFQ algorithms favor throughput-oriented applications such as FTP over delay-sensitive bursty applications such as WWW, and telnet. This is due to the memory-less instantaneous fairness property of PFQ algorithms. In a previous study [1], 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.

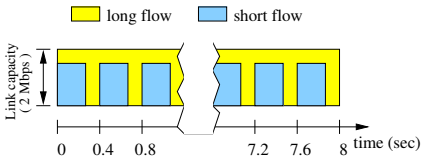
1 Introduction

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 [2, 3] and its many variants [4, 5, 6, 7]) have become ones of the most popular algorithms implemented in today's advanced switches and routers [8, 9] because these algorithms provide support for both real-time and best-effort traffic. Intuitively, PFQ allocates to each backlogged flow a share of service in proportion to its weight. When used with reservation, PFQ can guarantee a flow a minimum bandwidth which can in turn provide end-to-end delay guarantees for constrained flows. 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) Equal weights for short and long flows



(b) Short flows with $3\times$ the long flow's weight

Figure 1: Improving burst delays

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 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. There-

*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.

fore, as shown in Figure 1 (a), the long flow takes 8 seconds to finish, while a short flow takes 0.4 second 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 second. 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.

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]

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

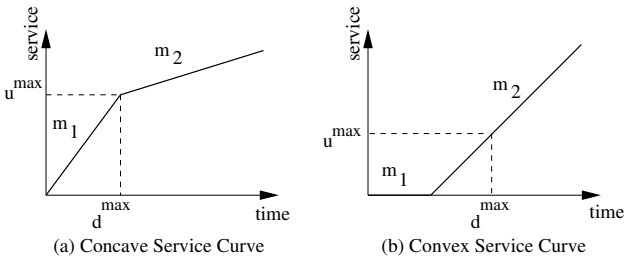
Packet Fair Queueing (PFQ) algorithms are based on the GPS model [3]. In GPS, each flow i is characterized by its weight, ϕ_i. During any time interval when there are exactly n non-empty queues, the server serves the n packets at the head of the queues simultaneously, in proportion to their weights.

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 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. 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.

The main problem with PFQ algorithms is that they couple the delay and bandwidth allocation. More precisely, if flow i is assigned a rate ϕ_i, then it can be shown that the worst case queueing delay per node incurred by a packet p_i^k is

$$\frac{l_i^k}{\phi_i} + \frac{l_{max}}{C}, \tag{1}$$

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. 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. (1), one should reserve[2] 256 Kbps, which is four times more than the flow's bandwidth requirement!



(a) Concave Service Curve (b) Convex Service Curve

Figure 2: Sample service curves.

2.2 Service Curve Model

To address this problem, Cruz has proposed a new service model, called service curve (SC) [10, 11], in the context of real-time guaranteed traffic. In this model, each flow is associated with a service curve $S_i(\cdot)$, 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

$$S_i(t_2 - t_1) \leq w_i(t_1, t_2), \tag{2}$$

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 [12]. 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 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.

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

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

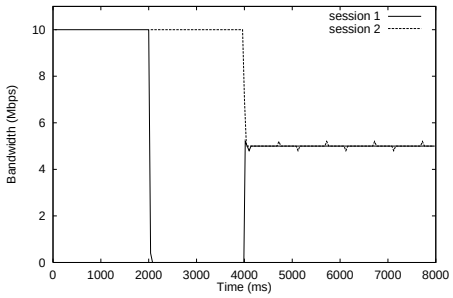


Figure 3: Measured bandwidth of two TCP sessions, which startup 2 seconds apart under SCED.

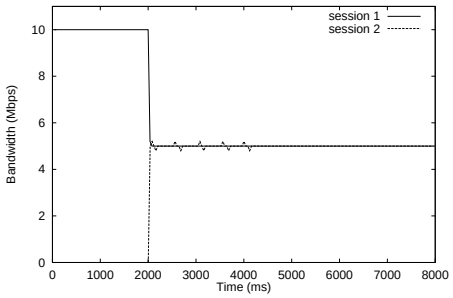


Figure 4: Measured bandwidth of two TCP sessions, which startup 2 seconds apart under FSC.

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. (2) 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 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 [13]. 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. As a result the absolute values of the weights or reservations cannot be arbitrarily set. 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 [1]. 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 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.

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. However, the difference between FSC and PFQ is in the computation of the time stamps. In 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. The details of the virtual time computation and algorithm pseudocode can be found in [14].

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 continuously 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

[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 [1].

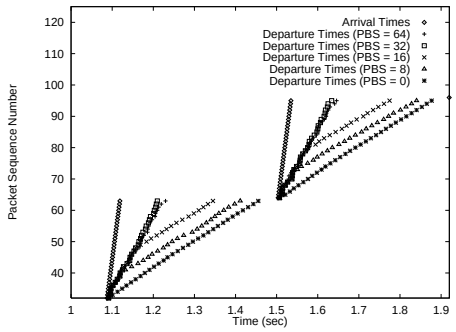


Figure 5: The packet arrival and departure times of a bursty flow for various service curves.

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 5 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, 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

We evaluate the FSC algorithm through extensive simulations. All simulations are performed in ns-2 [15]. We examine the behavior of FSC under a taxonomy of transport protocol and traffic model combinations. For transport protocols, we use both TCP (ns-2's TCP Reno without any protocol hand-shake) and UDP. For traffic models, we use periodic ON-OFF source, exponentially distributed ON-OFF source, pseudo WWW traffic source (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 source. 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 defined 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. 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 [16]. In addition, the size of all packets is 1000 bytes except for Telnet, which uses 64 byte packets. The simulation time is 20 seconds.

In this paper, we present only a subset of our simulation results. Additional simulation results are presented in [14].

3.1 Basic Demonstrations

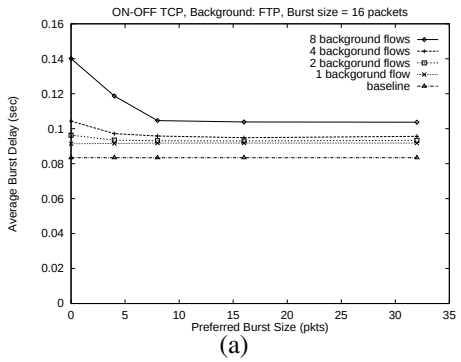
All simulations presented in this section use periodic ON-OFF foreground sources with a period of one second and a peak rate of 4 Mbps. Since the packet size is 1000 bytes, the inter-packet arrival time is 2 ms. All flows within the same simulation have the same burst size and the bursts occur at the beginning of each period. 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.

3.1.1 Impact of Preferred Burst Size (PBS)

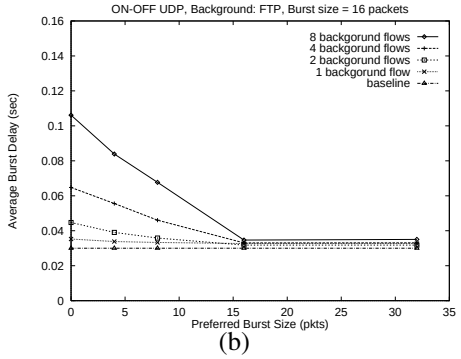
In this section we study the impact of the preferred burst size (PBS) and the number of background flows on the behavior of FSC. We consider 16 flows sharing a congested link. The number of persistent background flows varies from 1 to 8. Figure 6 and 7 plot the average burst delay as a function of PBS in four different scenarios using all 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 6 and 7, 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 (according the the first slope $m1$ of their service curves). Note that the data points for PBS equals zero are the corresponding performance points of PFQ. 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

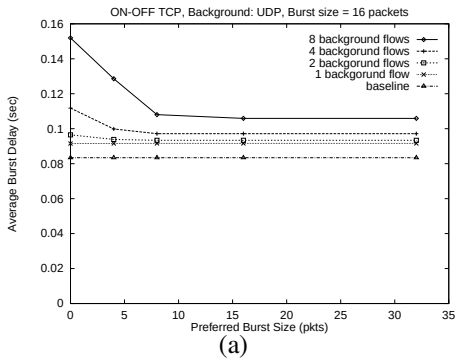


(a)

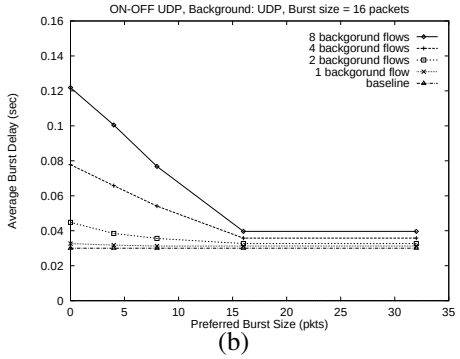


(b)

Figure 6: The average burst delay vs PBS with FTP background traffic.



(a)



(b)

Figure 7: The average burst delay vs PBS with UDP background traffic.

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 3.6.

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. The more background flows, the higher the relative priority of the bursty 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 increases, there are more "opportunities" to shift the delay from the bursty 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 than when it uses TCP. 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. In our case, 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 (Figure 8). Again, the average burst delay decreases as PBS increases. When the ON-OFF traffic is TCP (Figure 8(a)), the decrease in the average burst delay is more significant for larger bursts because when the burst size is small, the burst delay is dominated by the round-trip-time as the sender waits for acknowledgements. 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 (Figure 8(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 9, in both cases the average burst delay decreases as the BPR increases. This is expected since increasing BPR results in an increase of

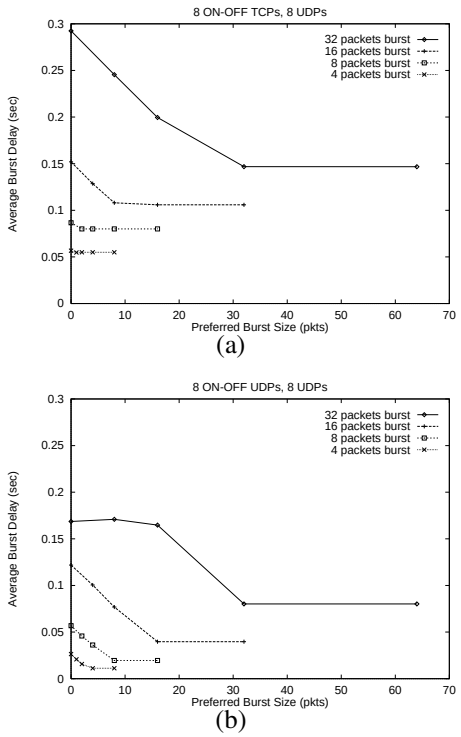


Figure 8: The average burst delay of eight ON-OFF TCP/UDP flows as a function of PBS for various burst sizes.

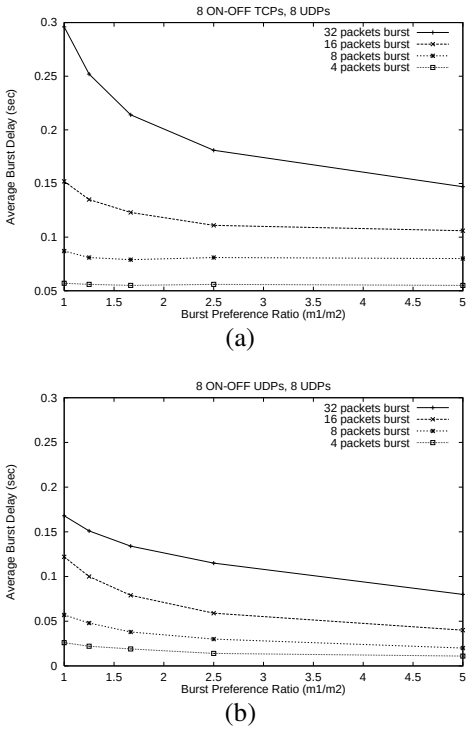


Figure 9: The average burst delay vs BPR for eight flows when the ON-OFF traffic is (a) TCP or (b) UDP.

the relative priority of the bursty traffic. Also, similar to the previous experiment (Figure 8), FSC is more effective for larger burst sizes, especially when the ON-OFF traffic is TCP. Again, notice that the data points for BPR$= 1$ are the corresponding performance points of PFQ under the same scenarios. The advantage of FSC over PFQ can be seen clearly.

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 10.

In the first scenario (Figure 10(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 (Figure 10(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.

3.3 Exponential ON-OFF Sources

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

Figure 12(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 traf-

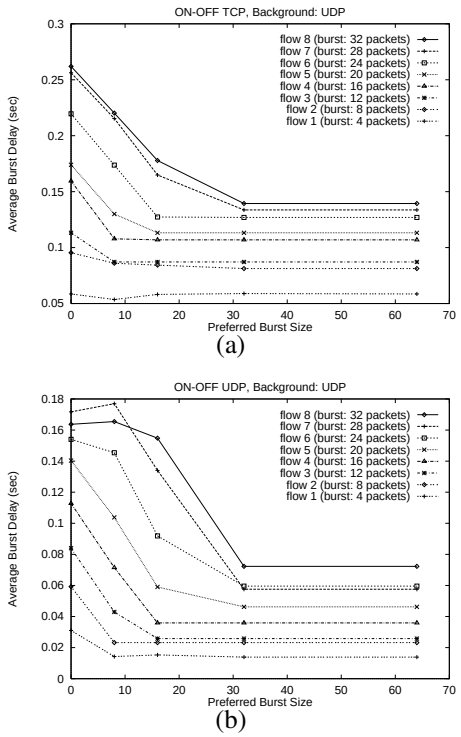


Figure 10: The average burst delay vs PBS for eight flows, with burst sizes between 4 and 32 packets, when the ON-OFF traffic is (a) TCP or (b) UDP.

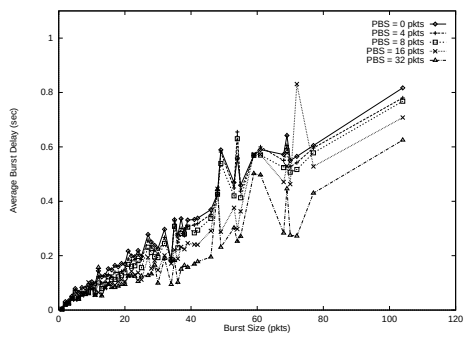


Figure 11: The average burst delay vs burst size for eight ON-OFF TCP flows, which have burst sizes exponentially distributed with a mean of 16 packets.

fic. 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 12(b) shows the distribution of the frame delay for the video traffic. As expected, the tail of the distribution decreases as PBS increases.

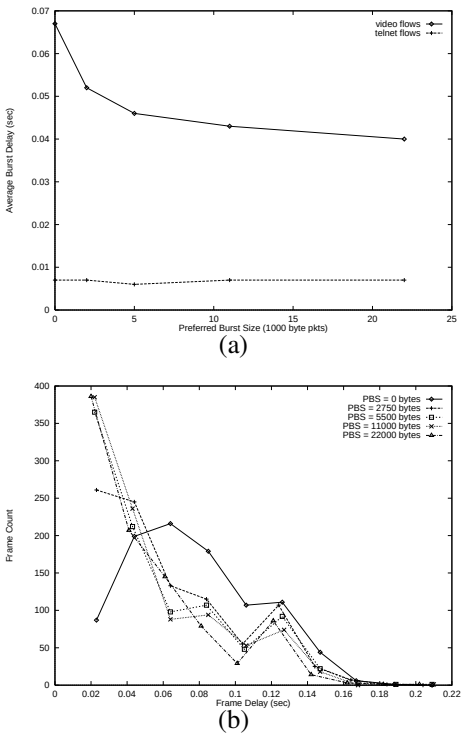


Figure 12: (a) The average frame delay for MPEG-1 video traffic and the average packet delay for Telnet traffic versus PBS. (b) The frame delay distribution for the video traffic for different PBS values.

3.5 Impact on Background Traffic

We have shown that FSC is effective in reducing the average burst delay of bursty traffic. 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.

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). What is somewhat surprising is that the TCP throughput is essentially unchanged under PFQ. This seemingly contradictory result is simple to

	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 worst case flow arrival scenario.

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 (this is 0.01 Mbps larger because there are less than 6 backlogged flows during the very beginning of the simulation). 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.

At the same time, FSC is 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.

3.6 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 return 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 adversely affecting background traffic.

In order to answer these questions, we use the flow length data from AT&T Labs' recent Internet traffic analysis [17]. Figure 13 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

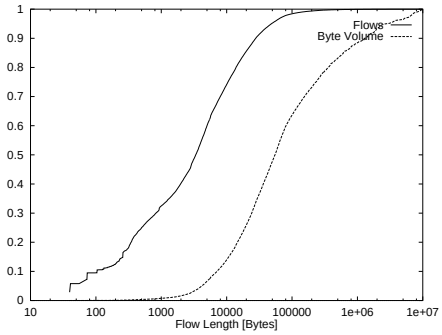


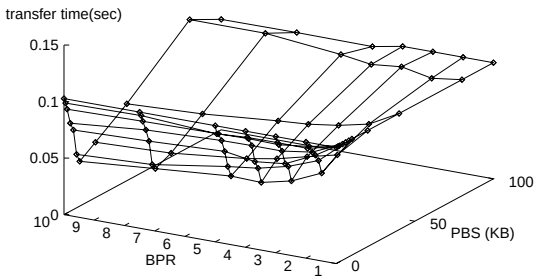
Figure 13: Cumulative probability distribution of flow lengths and their portion of the total byte volume.

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.

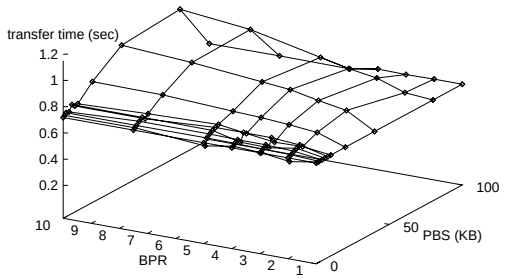
To determine how to configure FSC's parameters for WWW traffic, we generate a synthetic workload of FTP traffic, whose flow lengths are chosen to model this distribution. We divide the flows into 10 groups, each representing 10 % of the flows, and compute the average flow length within each group. Based 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 10 Mbps link with a 2 ms latency while setting the maximum segment size of the TCP sources to 576 bytes. Figure 14 plots the average transfer time experienced by flows in groups 8 and 10, as we vary the BPR from 1 to 10 and the PBS from 0 KB to 100 KB.

Note that all points with PBS = 0 and/or BPR = 1 correspond to PFQ. Groups 0 through 3 are 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 14 (b) shows that Group 10 in fact sees a noticeable 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 [18]. 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. 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)



(b)

Figure 14: (a) Transfer time for 70 to 80th percentile flows. (b) Transfer time for 90 to 100th percentile flows.

4 Summary

In this paper we study the Fair Service Curve (FSC) algorithm in the context of supporting best-effort service. We show that FSC can significantly improve the delay of bursty traffic compared to PFQ without negatively affecting the throughput of long-lived traffic. 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.

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. Intuitively, more background traffic provides more "opportunities" to shift the delay from the bursty traffic towards the background traffic.

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 noticeable impact on the background traffic.

Compared to PFQ, the added complexity to implement FSC is minimal. We believe FSC is a powerful solution that offers better support for best effort traffic in today's Internet.

5 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] 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, Aug. 1997.

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

[3] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control - the single node case," *ACM/IEEE Transactions on Networking*, vol. 1, no. 3, pp. 344–357, June 1993.

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

[5] S.J. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proceedings of IEEE INFOCOM'94*, Toronto, CA, Apr. 1994, pp. 636–646.

[6] 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*, Palo Alto, CA, Aug. 1996, pp. 157–168.

[7] 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.

[8] 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*, vol. 17, no. 6, pp. 1145–1158, June 1999.

[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] R.L. Cruz, "Service burstiness and dynamic burstiness measures: A framework," *Journal of High Speed Networks*, vol. 1, no. 2, pp. 105–127, 1992.

[11] R.L. Cruz, "Quality of service guarantees in virtual circuit switched network," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 6, pp. 1048–1056, Aug. 1995.

[12] 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*, Sept. 1995, pp. 512–520.

[13] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switching networks," in *Proceedings of ACM SIGCOMM'90*, Philadelphia, PA, Sept. 1990, pp. 19–29.

[14] T.S.E. Ng, D.C. Stephens, I. Stoica, and H. Zhang, "Supporting Best-Effort Traffic with Fair Service Curve," Tech. Rep. CMU-CS-99-169, Carnegie Mellon University, Nov. 1999.

[15] "Ucb/lbnl/vint network simulator - ns (version 2)," .

[16] 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, Mar. 1998.

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

[18] J. Davin and A. Heybey, "A simulation study of fair queueing and policy enforcement," *Computer Communication Review*, vol. 20, no. 5, pp. 23–29, Oct. 1990.