Competitive Queue Policies for Differentiated Services *

William A. Aiello † Yishay Mansour ‡ S. Rajagopolan § Adi Rosén ¶

Abstract

We consider the setting of a network providing differentiated services. As is often the case in differentiated services, we assume that the packets are tagged as either being a high priority packet or a low priority packet. Outgoing links in the network are serviced by a single FIFO queue.

Our model gives a benefit of $\alpha \geq 1$ to each high priority packet and a benefit of 1 to each low priority packet. A queue policy controls which of the arriving packets are dropped and which enter the queue. Once a packet enters the queue it is eventually sent. The aim of a queue policy is to maximize the sum of the benefits of all the packets it sends.

We analyze and compare different queue policies for this problem using the competitive analysis approach, where the benefit of the online policy is compared to the benefit of an optimal offline policy. We derive both upper and lower bounds for the policies we consider. We believe that competitive analysis gives important insight to the performance of these queuing policies.

1 Introduction

It is widely agreed that future packet networks will likely support Quality of Service (QoS) in order to provide a full array of services. Today, the Internet with its single best effort class of service does not provide a satisfactory solution for many applications. Many users may be willing to pay more for some services if in return they receive some guarantee of performance from the network. This vision is by no means new in networking, and has been considered in the literature for well over a decade [15]. One of the main targets of ATM has been to address this very basic problem, and to give a unified architecture for a diverse set of classes of service.

The basic methodology of QoS is rather simple and widely accepted. The user reaches a contract with the network. This contract is composed of two parts. The first part consists of the commitments of the user to the network about how his traffic will behave (average bandwidth, peak bandwidth, burst size etc.) The second part of the contract consists of the network guarantees to the user (maximum delay, jitter, etc.). The basic premise of the contract is that if the user keeps his commitments about his traffic, the network maintains the performance guarantees.

As in any contract, what should be done when the terms of the contract are not fulfilled is a basic question. The interesting case in our setting is what should be done when the user's traffic does not conform to his commitments. There are two alternative solutions, both seem equally valid. The first solution is to force the incoming traffic to conform to the committed parameters by regulating the traffic

*An early version of this paper appeared in the proceedings of INFOCOM 2000, pp. 431–440.

†AT&T Labs-Research, 180 Park Avenue, Florham Park NJ 07932, U.S.A.

‡School of Computer Science, Tel-Aviv University, Tel-Aviv 69978, Israel.

§Telcordia Technologies, 445 South street, Morristown NJ 07960, U.S.A.

¶Dept. of Computer Science, Technion, Haifa 32000, Israel. Part of this research was done while visiting AT&T Labs-Research.

at the entrance to the network. The other approach is to label the traffic as "in" and "out" so that the data labeled "in" has the desired properties and the data labeled "out" is an excess load. In the second approach we allow the network element (routers) to handle the two types of traffic differently.

Moreover, there is recently a wide interest within the framework of IP in providing differentiated services, in which different users receive different classes of service from the network. The two basic paradigms suggested are Premium service [11] and Assured service [3]. The premium service states that the user quality of service will be (almost) indistinguishable from a dedicated line with the same given capacity. This approach naturally leads to shaping the user's traffic at the entry to the network. The assured service has a vague guarantee in which traffic conforming to the user profile is much less likely to be dropped in the network. This approach naturally leads to having "in" and "out" packets, where "in" packets have a higher priority over "out" packets. There has been considerable amount of research comparing the various approaches to differentiated services. Most of the comparisons have been done by assuming a simple probabilistic model for the packets arrival [10, 13].

We abstract the question as follows. We assume that we have two type of packets. One type is a low priority packet and has a low benefit, say 1. The other type is a high priority packet and has a benefit of $\alpha \geq 1$. One can relate the high priority packets to the "in" packets and the low priority packet to the "out" packets. Another possible interpretation is that the higher priority packets belong to a higher priority class of service which carries a higher benefit. Note that if we make α very large we are, implicitly, giving the high priority packets an absolute preference over low priority packets. If we choose a moderate value of α we have an interesting tradeoff between the two packet types. For α near 1 we are basically optimizing the total traffic, ignoring the various priorities.

We concentrate on scheduling in a single FIFO outgoing queue. We model this as follows. We assume that packets are emptied (sent) from the queue one packet in every time unit. During a time unit several packets of each type may arrive and they have to be routed through the outgoing queue. The arriving packets are considered, one at a time, by a *queue policy* which decides for each incoming packet whether to insert the packet into the FIFO outgoing queue or drop the packet. Once the policy decides to accept a packet, it cannot later preempt it from the outgoing queue, and therefore the packet is eventually transmitted. The aim of the queue policy is to maximize the total benefit of the packets that the outgoing queue sends. In this model we study a number of queuing policies and compare their performance. Note that a queue policy is an "online" algorithm – it must commit to decisions about packets even though subsequent packet arrivals may prove the decisions to be far from optimal.

If the behavior of the arrival sequence is not constrained, then there are arrival sequences for which the benefit of any queue policy might be very low, e.g., the sequence of all low priority packets. Thus a lower bound on the benefit of a queue policy over all arrival sequences will not differentiate between queue policies. To avoid this difficulty one may constrain (or give a model for) the arrival sequences, as classically done in queuing theory, and then analyze the benefit of various queue policies. In many settings similar to that of this paper this approach has provided a great deal of insight and when the arrival model closely approximates that of the application it can provide very tight analysis. However, an accurate arrival model is difficult to obtain or verify in many cases (especially for QoS, where reliable data is hard to find).

In this paper we use the approach of *competitive analysis* [14, 2], which is input-model independent. As such, the bounds derived cannot be expected to be as tight as those for a specific traffic model. However, no assumptions about the arrival sequence are required to apply the bounds.

In competitive analysis applied to this setting, one compares the performance of an online queue policy to an optimal offline policy, which is given the entire event sequence in advance. The competitive ratio is the minimum, over all event sequences, of the ratio of the online benefit to the offline benefit.

In this way, the online queue policy is not penalized for achieving low benefit on an arrival sequence for which even an optimal offline algorithm would achieve low benefit. The online queue policy is only penalized when there is an event sequence on which its benefit is only a fraction of the benefit of the optimal offline algorithm. Clearly an optimal offline will always have a better or equal performance to the online policy so the competitive ratio will always be less than 1. Our aim is to find queue policies with the largest competitive ratio.

In this work we consider five queue policies and study their competitive ratio for various values of α. Our contribution is twofold. First, we provide upper and lower bounds on the competitive ratios of a number of simple queue policies that are in common use, and some of their natural extensions. Second, we design a new policy, give upper and lower bounds on its performance, and show that it outperforms the other policies. In all our analyses we take into account the size of the queue, denoted B, since the size of the queue is sometimes small in practice.

There has been a considerable amount of related work on the subject, much of it following the initial publication of the present work. The case of multiple packet values was studied in [1]. A queue model which includes a preemption operation (and multiple values) was introduced and studied in [9], where it is shown that the greedy policy (which preempts the lowest value packet in the buffer) is at least 1/4 competitive. The bound on the competitive performance of greedy was improved to a tight bound of 1/2 in [5] (in that work additional models for preemptive buffer policies are studied). For the two values case, one can achieved an improved performance by allowing high value packets to preempt many low value packets. In [6] a policy which achieves near optimal value for large values of α is given. In [8] it is shown that for any value of α, in the two value model, there is an 0.78 competitive buffer policy.

All the above results are specific to a single queue. Considering the performance at the switch level, in [4] it was shown that the Longest-Queue-Drop policy is 1/2 competitive. For non-preemptive case the Harmonic policy [7] was shown to be $1/\log M$ competitive, where M is the size of the shared memory.

Organization The paper is organized as follows. In Section 2 we give a formal description of our model. In Section 3 we define each of the five online policies that we consider. We then outline, in Section 4, how to efficiently compute an optimal schedule (i.e., the schedule to which we compare our online policies). Section 5 gives a summary of our results. The performance of the various policies is derived in Section 6, while the impossibility results are given in Section 7.

2 Model

In this paper we concentrate on a single FIFO queue. The FIFO queue can hold B packets, and once a packet enters the queue it cannot be preempted. We make no assumptions about the input stream of packets that arrive at the queue, and assume that the packets leave at a constant rate. The stream of packets that arrive to the queue includes two types of packets: high priority packets, which have a benefit of $\alpha \geq 1$; and low priority packets, which have a benefit of 1. (Throughout the paper we use interchangeably the terms *queue* and *buffer*.)

An event sequence Λ is a sequence of timed events. An event can be either an arrival of a high priority packet, an arrival of a low priority packet or a send event (in which a packet is sent from the queue, assuming it is not empty). It would be convenient to define the times of the send events as synchronized with integral times, namely, for each integral time t we have a send event. Between two consecutive send events (possibly) a large number of packets arrive, and this sequence of arrivals can be viewed as a burst of arrivals. Each packet arrival is assigned a distinct (non-integer) time.

Policies	$\alpha \to \infty$	$\alpha = 2$	$\alpha = 1$
General Impossibility Results	1/2	2/3	1
Greedy	0	1/2	1
Round Robin	1/2	1/2	1/2
Fixed Partition	$[1/4, 0.41]$	$[1/4, 1/2]$	1/2
Flexible Partition	$\sqrt{2} - 1 \approx 0.41$	$[0.5, 0.62]$	1
Dynamic Flexible Partition	1/2	$[0.53, 0.62]$	1

Figure 1: This figure gives the competitive ratios of the various policies we consider in three different regimes of α. The competitive ratios presented are for large values of B, i.e., these are the limits when $B \to \infty$. For each policy in each regime we chose the best parameter x. Therefore, in different regimes the same policy may have a different parameter x. The general impossibility result, given in the last column, upper bounds the competitive ratio of any online policy. In the cases where our lower and upper bounds do not match, we give the interval which contains the competitive ratio.

A queue policy specifies for each incoming packet whether to accept it into the queue, or to reject it. The benefit of a queue policy on a given event sequence is the sum of the benefits of the packets that it sends. Since we assume that there are only two types of packets, the benefit is $k + \alpha m$, where k is the number of low priority packets sent and m is the number of high priority packets sent.[1]

Our results are stated using competitive analysis. We compare the performance of an online queue policy to an optimal offline schedule, which knows in advance the entire event sequence. The competitive ratio is the minimum over all event sequences of the ratio between the benefit of a given online policy and the benefit of an optimal offline policy. More formally, for an event sequence Λ let the benefit of an online policy π be denoted by $\pi(\Lambda)$, and the optimal benefit be denoted by $opt(\Lambda)$. We say that the policy π is c-competitive if for any event sequence Λ, it holds that $c \cdot opt(\Lambda) \leq \pi(\Lambda) + a$, where a is a constant independent of the event sequence Λ.

Since we consider in this paper non-preemptive policies that send the packets in FIFO order, the total benefit of the sent packets up to some time t is at least the total value of the accepted packets until time t minus $B \cdot \alpha$. On the other hand, the total benefit of the packets sent by the optimal policy until time t is at most the total value of the packets this policy accepted until time t. We can therefore compare in our proofs the total value of the packets *accepted* by the two policies until time t, rather than the total value of the packets sent.

3 Definition of Online Policies

In this paper we analyze five queue policies. The most trivial queue policy is the *Greedy Policy*, which accepts a packet if the buffer is currently not full. (As one may suspect, this policy has low performance.)

The second policy is the *Fixed Partition Policy*. This policy has a threshold parameter $0 \leq x \leq 1$, and it limits the number of low priority packets in the buffer to be at most xB and the number of high priority packets to be at most $(1 - x)B$.

The third queue policy is the *Flexible Partition Policy*. This is a natural extension of the Fixed Partition Policy. As with the Fixed Partition Policy it has a threshold parameter $0 \leq x \leq 1$ and limits

[1] There are other important issues which are not addressed in our model such as fairness, blocking and starvation.

the number of low priority packets in the buffer to be at most xB. However, unlike the Fixed Partition Policy, it always accepts high priority packets. The logic is that one cannot lose by accepting a high priority packet. Note that at some point in time the buffer may contain more than $(1-x)B$ high priority packets and will not be able to accept xB low priority packets even if there is a burst of arrivals of low priority packets before the next send event. Not surprisingly, we will see that the Flexible Partition Policy has a better competitive ratio than the Fixed Partition Policy. One might suspect that the Flexible Partition Policy always outperforms the Fixed Partition Policy. However, in Appendix B we show that this is not true. We present an event sequence on which the Fixed Partition Policy performs better than the Flexible Partition Policy.

We then define a fourth policy, with the aim of optimizing the competitive ratio for our problem. We call this policy the *Dynamic Flexible Partition Policy*. This policy computes the ratio of the number of low priority packets to the number of slots that do not contain a high priority packet (i.e., are either empty or contain a low priority packet). It accepts a new low priority packet as long as that ratio is below a threshold parameter $0 \leq x \leq 1$. More formally, let k and m be the number of low priority and high priority packets, respectively, in the buffer. The Dynamic Flexible Partition Policy accepts a low priority packet if $k' \leq x\hat{B}$, where $k' = k+1$ and $\hat{B} = B-m$. Note that for the case of $x = 1/2$ (discussed in Section 6) we accept a low priority packet if after accepting it the number of low priority packets does not exceed the amount of free space. The Dynamic Flexible Partition Policy always accepts a high priority packet, unless the buffer is full.

We further consider the *Round-Robin Policy*, defined as follows. The intuitive way to think about the Round Robin Policy is by maintaining two queues each of size $B/2$. One queue is dedicated to low priority packets while the other queue is dedicated to high priority packets. During the send events the policy alternates between sending a low priority packet and sending a high priority packet. In case one of the buffers is empty, it sends from the non-empty buffer [2]. Based on the above intuitive idea we define the following policy for a single FIFO queue (we are in fact simulating Generalized Processor Sharing [12] rather than strict round robin.) We maintain two variables `num-low` and `num-high`. We accept a low (resp. high) priority packet if `num-low` (resp. `num-high`) is at most $(B/2) - 1$, and increment the corresponding counter by 1. Each time a packet is sent both `num-low` and `num-high` are decremented by 1/2, assuming they are both non-zero. In case one of them is zero the other is decremented by 1. (We note that decrementing by 1 the counter corresponding to the packet being sent will not result in the same protocol, or the same behaviour on any event sequence. We define the protocol as above so as to simulate GPS).

In all of the above policies we assume that the values of B and x are such that $\lfloor xB \rfloor \geq 1$.

We note that in general we can simulate an arbitrary non-preemptive policy on a single FIFO queue. In the appendix we show how to simulate any non-preemptive policy using a single FIFO queue, in a way that the set of packets accepted by the original policy is identical to the set of packets accepted by simulated policy. (However, the order in which the packets are sent may be different.) See Appendix A.

4 An Optimal offline schedule

Before describing our results for the online policies, we give a description of OPTIMAL, an optimal offline policy. Given an event sequence Λ, OPTIMAL works in two phases. In the first phase it finds a schedule that accepts only high priority packets from Λ. (In this phase OPTIMAL accepts a high

[2] One can generalize the Round Robin Policy using the threshold parameter x. However, we have an upper bound of the minimum of x and $1-x$, which is maximized at $x = 1/2$, and we have a lower bound of (roughly) 1/2 for the case of $x = 1/2$. We therefore concentrate on the case of $x = 1/2$.

priority packet if when the packet arrives the buffer is not full.) Note that any greedy policy that accepts only high priority packets accepts the same number of packets. This guarantees that the number of high priority packets accepted by OPTIMAL is maximized.

The second phase is to augment the schedule by adding low priority packets. OPTIMAL considers the low priority packets in the order they arrive, and accepts a low priority packet if it does not interfere with the current schedule. Namely, if adding the low priority packet does not force a later high priority (previously accepted) packet to be rejected.

Theorem 4.1 *For any event sequence* Λ, $OPTIMAL(\Lambda)$ *generates the maximum benefit schedule.*

Proof. Given a schedule s of Λ we first modify s such that it accepts the same high priority packets as the ones accepted by $OPTIMAL(\Lambda)$. We show that this transformation does not decrease the benefit of the schedule.

Consider the first high priority packet that is accepted by $OPTIMAL(\Lambda)$ and rejected in s. Let us modify s and accept this high priority packet. If the schedule remains valid, then we have increased the total benefit by α. Otherwise there is some later packet that is accepted in s and now has to be rejected. Let s' be the new schedule. The schedule s' has at least the same benefit as s and has a larger prefix in which it agrees with $OPTIMAL(\Lambda)$ on the scheduling of the high priority packets. Continuing with s' eventually we get a schedule s^* that agrees with $OPTIMAL(\Lambda)$ on the schedule of the high priority packets and has an equal or larger benefit than s.

Let s^* be a schedule which accepts and rejects high priority packets in the same way $OPTIMAL(\Lambda)$ does. We show that we can modify in s^* the acceptance of the low priority packets such that the schedule would be identical to that of $OPTIMAL(\Lambda)$. Each time s^* and $OPTIMAL(\Lambda)$ differ it has to be on a low priority packet. Consider the first packet p on which they differ. It has to be the case that $OPTIMAL(\Lambda)$ accepts p and s^* rejected p. (Since s^* is identical to $OPTIMAL(\Lambda)$ up to this point, and if $OPTIMAL(\Lambda)$ rejects a low priority packet, it means that the packet cannot be scheduled with the remaining high priority packets.) We modify s^* by accepting p. This implies that when some later packet p' arrives, the buffer is full, although s^* is scheduled to accept p'. It has to be the case that p' is a low priority packet, otherwise $OPTIMAL(\Lambda)$ would have rejected p. We modify s^* by accepting p and rejecting p'. We continue this process until the schedules are identical. ■

5 Overview of Our Results

In this section we give an overview of our results. Some of our upper and lower bounds in the next sections are a function of the benefit ratio α. For this overview we use three different regimes of the value of α, and compare the performance of the different policies under each one of the regimes. The first regime is the case of very large α, i.e., $\alpha \to \infty$. The second regime is of a moderate α, i.e., $\alpha = 2$. The third regime is the case when the high priority and low priority packets have the same benefit, i.e., $\alpha = 1$. To make the comparison fair we allow each policy to adjust its parameter x to be the optimal value for the given α. The competitive ratios that we consider in this overview are for the case when B is large, i.e., the values discussed are the limits when $B \to \infty$. This overview is summarized in Figure 1.

5.1 Large Benefit Value

Here we are interested in the case when the value of α is very large, and examine the competitive ratios in the limit where $\alpha \to \infty$. This case can be viewed as saying that the benefit of a single high

priority packet is much greater than the benefit of many low priority packets. (However we need to be competitive also when the low priority packets completely dominate.)

For this regime our general impossibility result shows that any online policy can have a competitive ratio of at most 1/2. In this regime the Greedy Policy performs as poorly as possible. Its asymptotic competitive ratio is 0. This is not surprising since the Greedy Policy ignores the value of α, and treats all the packets equally. At the other extreme the Round Robin Policy has an optimal asymptotic competitive ratio of 1/2.

For the Fixed Partition Policy with $x = 1/2$ we show a competitive ratio of at least 1/4, but we do not have matching impossibility results even for the fixed parameter $x = 1/2$. Our impossibility result for this queue policy gives an upper bound of $\sqrt{2} - 1 \approx .414$ for any choice of x.

For the Flexible Partition Policy with $x = \sqrt{2} - 1$ we show that it achieves a competitive ratio of $\sqrt{2} - 1$. Our upper bound for this policy coincides with the upper bound for the Fixed Partition Policy and thus the bound is tight.

Like the Round Robin Policy for this regime, the Dynamic Flexible Policy has an asymptotic competitive ratio of 1/2 and is therefore optimal in this case.

5.2 Moderate Benefit Value

We now consider the case where the high priority packets are more valuable than the low priority ones, but not overwhelmingly more valuable. The parameter α can be set in many ways but for the purposes of this comparison we chose the setting of $\alpha = 2$. As before, we allow each policy to adjust the parameter x in order to optimize its performance.

In this setting the general impossibility result states that no online policy can have a competitive ratio better than 2/3. Unfortunately none of our policies achieves this bound. In fact one can show for each of our policies an event sequence for which the competitive ratio it has is strictly less than 2/3, for any setting of the parameter x.

In this regime the Greedy Policy achieves a competitive ratio of 1/2, which implies that in this case a competitive ratio of 1/2 should not be considered impressive.

The competitive ratio of the Round Robin Policy remains 1/2 as in the case of $\alpha \to \infty$. This is due to the fact that if packets of only a single type arrive, the competitive ratio is at most 1/2. Even if we consider an extension of the Round Robin Policy to have a parameter x, the fact that the policy uses a hard partition implies that the competitive ratio is at most 1/2.

For the same reason, the Fixed Partition Policy cannot have a better upper bound than 1/2. As for a lower bound, the best competitive ratio we can prove is 1/4 which is not better than the one for the case when $\alpha \to \infty$.

The performance of the Flexible Partition Policy improves to 1/2 (with $x = 1$) when we consider $\alpha = 2$. The upper bound on the competitive ratio for the value of $\alpha = 2$ is $\sqrt{1.25} - 0.5 \approx 0.62$.

The best policy in this regime is the Dynamic Flexible Partition Policy for which we can show that the competitive ratio of approximately 15/28 which is strictly better than 1/2 (using $x = 3/4$). On the other hand the upper bound coincides with the upper bound for the Flexible Partition Policy at approximately 0.62.

5.3 Low Benefit Value

In this regime we consider the case when the benefit of the high priority packets and the low priority packets is identical, i.e. $\alpha = 1$. In this case the aim of the queue policy reduces to simply maximizing the total number of packets that the policy accepts.

Not surprisingly, the Greedy Policy is optimal in this case, since it is by definition maximizing the number of packets accepted, and ignores the value of α.

The policies that have a fixed partition between the allocation of the low priority packets and the high priority packets do not perform well in this case. Both the Round Robin Policy and the Fixed Partition Policy have a competitive ratio of only 1/2. This is due to the fact that when an overwhelming majority of the arriving packets are from a single type, these policies are forced to drop packets when the preallocated space in the buffer for packets of that type is full. On the other hand, the optimal policy will only drop packets when the entire buffer is full.

Both the Flexible Partition Policy and the Dynamic Flexible Partition Policy also achieve optimal competitive ratio, simply by setting $x = 1$. For the setting $x = 1$ both policies simply reduce to the Greedy Policy.

5.4 Conclusions

Our results seem to indicate that the policies which allow a flexible use of the buffer space achieve a good competitive ratio in all the regimes of α. On the other hand, policies which perform a preallocation of the buffer space perform poorly when the difference in benefits is not significant.

The best policy, of the five we consider here, is the Dynamic Flexible Partition Policy. This policy achieves the best competitive ratio in each one of the three regimes. It has a good mix of high preference to high priority packet, with the ability to accept large bursts of low priority packets. By adjusting the threshold parameter we can tune the behavior of the Dynamic Flexible Partition Policy.

The Flexible Partition Policy, which has similarities to the Dynamic Flexible Partition Policy, achieves similar performance, but its performance is lower in each of the three regimes. This might suggest that we may attribute the difference in performance to the difference in the policies.

The Fixed Partition Policy has consistently lower competitive ratio than the Flexible Partition Policy and the Dynamic Flexible Partition Policy. This is not surprising, since the buffer partition forces it to reject high priority packets even when there is space in the buffer.

The Round Robin Policy has a competitive ratio of 1/2 in all three regimes. While this is optimal for the regime of large α, this is a poor competitive ratio when α approaches 1. As discussed before, the poor competitive ratio for small α is due to the rigidity of the buffer partition.

Finally, the Greedy Policy should be viewed as a minimal performance measure. As expected, for large α it has a poor competitive ratio, and for small α it behaves well since in this regime maximizing the throughput is a reasonable strategy.

6 Analysis of Queue Policies

In this section we give lower bounds on the benefit of the policies compared to an optimal offline policy. We start this section with a few lemmas that are useful for our proofs.

6.1 Basic Proofs

We start by defining and analyzing a number of policies that will be helpful as tools in the analysis. In particular some of these policies are allowed to accept and to send fractions of packets. The sequence of arriving packets always consists however of *complete* packets. We emphasize that these are used only as building blocks in our proofs.

In the sequel we use the following notations. For a given event sequence Λ, and for a given policy P, we denote by $P^{\mathcal{P}}(\Lambda)$ the number of packets accepted by P when presented with the sequence Λ.

Analogously we denote by $P^{\mathcal{L}}(\Lambda)$ (resp. $P^{\mathcal{H}}(\Lambda)$) the number of low priority packets (resp. high priority packets) accepted by P out of Λ. Note that $P^{\mathcal{P}}(\Lambda) = P^{\mathcal{H}}(\Lambda) + P^{\mathcal{L}}(\Lambda)$. When a policy is allowed to accept fractions of packets, these notations denote the amount of packets accepted (i.e. they can denote non integral values). This will be clear from the context.

We first define the GREEDY(B,1) policy. This policy uses a buffer of size B, where B is an integer; it *greedily* accepts any packet presented to it (i.e., as long as it has at most B packets in the buffer after accepting the packet). For each send event it sends one packet. We then define more generally the policy GREEDY(B,x), for any $0 \leq x \leq 1$ such that $\lfloor xB \rfloor \geq 1$. This policy uses a buffer of size xB. It greedily accepts any packet presented to it (i.e., as long as the occupied space in the buffer does not exceed xB after the packet is accepted). For each send event, it sends an x fraction of a packet. Finally, we define the FRAC_GREEDY(B,x) policy. This policy uses a buffer of size xB. However it is allowed to accept also fractions of packets. That is, we consider the packets as a continuous rather than a discrete entity. The FRAC_GREEDY(B,x) policy fills its buffer greedily as long as its size does not exceed xB. For each send event it sends an x fraction of a packet. Since we do not consider in the following only complete packets, we will sometimes talk in terms of "occupied space" in the buffer of a policy, and the "amount of packets" accepted by a policy.

In this subsection we prove the following two lemmas that will be useful in our proofs.

Lemma 6.1 *For any event sequence Λ, $GREEDY(B,1)^{\mathcal{P}}(\Lambda) \leq \frac{1}{x}(1 + \frac{1}{\lfloor xB \rfloor}) \cdot GREEDY(B,x)^{\mathcal{P}}(\Lambda)$.*

The next lemma compares an arbitrary policy P to GREEDY(B,1), assuming that P always accepts a packet when the buffer holds less than $\lfloor xB \rfloor$ packets, but can either accept or reject packets when the buffer holds at least $\lfloor xB \rfloor$ packets.

Lemma 6.2 *Let P be an arbitrary policy that sends a complete packet in each send event, accepts only complete packets, and never rejects a packet unless it already has $\lfloor xB \rfloor$ packets in the buffer, for some x such that $\lfloor xB \rfloor \geq 1$. Then, for any event sequence Λ, $GREEDY(B,1)^{\mathcal{P}}(\Lambda) \leq \frac{1}{x}(1 + \frac{1}{\lfloor xB \rfloor}) \cdot P^{\mathcal{P}}(\Lambda)$.*

We now proceed to prove these lemmas. We start with the following lemma which gives the analogue claim of Lemma 6.1 for FRAC_GREEDY(B,x).

Lemma 6.3 *For any event sequence Λ, $GREEDY(B,1)^{\mathcal{P}}(\Lambda) \leq \frac{1}{x} \cdot FRAC\_GREEDY(B,x)^{\mathcal{P}}(\Lambda)$.*

Proof. To prove the lemma we start with the following claim.

Claim 6.4 *At any time the size of the occupied space in the buffer of GREEDY(B,1) is at most the size of the occupied space in the buffer of FRAC_GREEDY(B,x) times $1/x$.*

Proof. We prove the claim by induction on the events. The basis of the induction is before any event occurs. Both buffers are empty and the claim trivially holds. Assume the claim holds for the first $k-1$ events, and consider the k-the event.

Assume the k-the event is a send event. If the buffer of GREEDY(B,1) is empty after the event, then the claim trivially holds. Otherwise, the buffer of GREEDY(B,1) must have been nonempty before the send event. Since GREEDY(B,1) always holds an integer number of packets in its buffer, before event k there are at least two packets in its buffer. It follows that in the send event GREEDY(B,1) sends one packet. The FRAC_GREEDY(B,x) policy always sends at most an x fraction of a packet. By the induction hypothesis the claim holds before the send event, and therefore it holds also after the send event.

Assume the k-the event is an arrival of a packet. If FRAC_GREEDY(B,x) accepts the entire packet, clearly the claim still holds. Otherwise, it has to be the case that after this event FRAC_GREEDY(B,x) has a full buffer, namely the occupied space in its buffer is of size xB. Since at any time GREEDY(B,1) has at most B packets in the buffer the claim follows.

■

From Claim 6.4 we have that whenever the buffer of GREEDY(B,1) contains at least one packet, then the occupied space in the buffer of FRAC_GREEDY(B,x) is at least x. Therefore we know that whenever GREEDY(B,1) sends a packet, FRAC_GREEDY(B,x) sends a fraction x of a packet. Therefore the amount of packets sent by GREEDY(B,1) is at most the amount of packets sent by FRAC_GREEDY(B,x) times $1/x$.

The lemma follows since for any policy and any time the amount of packets accepted by that time, equals the amount of packets sent by that time, plus the amount of packets in the buffer at that time.

■

We now compare the number of packets accepted by GREEDY(B,x) to the amount of packets accepted by FRAC_GREEDY(B,x). We will use the following notations. For any event τ let G^τ be the size of the occupied space in the buffer of GREEDY(B,x) after event τ. Similarly, let F^τ be the size of the occupied space in the buffer of FRAC_GREEDY(B,x) after event τ. Sometimes we index G^t and F^t where t is the time of an event τ, this should be interpreted as G^τ and F^τ. We now prove the following lemma.

Lemma 6.5 *For any event sequence* Λ, $FRAC\_GREEDY(B,x)^{\mathcal{P}}(\Lambda) \leq (1+\frac{1}{\lfloor xB \rfloor})\cdot GREEDY(B,x)^{\mathcal{P}}(\Lambda)$.

Proof. We start with the following simple claim.

Claim 6.6 *For any event* τ, $F^\tau \geq G^\tau$.

Proof. We prove the claim by induction on the events. The base of the induction is before any event occurs, when both buffers are empty, and therefore the claim holds.

Assume the claim holds for the first $k-1$ events. Let τ be the k-the event and τ' the $k-1$-st event. Assume τ is a send event. If $G^\tau = 0$ then the claim clearly holds. Otherwise we have that $G^{\tau'} > x$, and by the induction hypothesis also $F^{\tau'} > x$. It follows that $F^\tau = F^{\tau'} - x$ and that $G^\tau = G^{\tau'} - x$. Using the induction hypothesis for τ' the claim follows.

Assume event τ is an arrival of a packet. If FRAC_GREEDY(B,x) accepts the complete packet then the claim continues to hold. Otherwise by the greediness of FRAC_GREEDY(B,x) we have that its buffer is full, i.e., $F^\tau = xB$. By definition, $G^\tau \leq xB$ and the claim holds. ■

To continue the proof of the lemma we divide Λ into *phases*. The first phase starts with the first event. A phase ends after a send event which results in the buffer of GREEDY(B,x) being empty. The next event is the first event of the following phase. Since send events occur at integral times, a phase is a time interval of the form $(t,T]$, where t and T are integral times. We remark that a phase may consist of a single time step (i.e., $(t,t+1]$), if the buffer becomes empty, and no packet arrives in the following time step. We start with the following simple claim.

Claim 6.7 *If GREEDY(B,x) rejects a packet during a phase, then it accepted during the phase at least* $\lfloor xB \rfloor$ *packets.*

Proof. If GREEDY(B,x) rejects a packet, then the occupied space in its buffer at this time is of size *strictly more* than $xB-1$. The buffer of GREEDY(B,x) is empty when the phase starts, therefore it

accepted during the phase strictly more than $xB - 1$ packets. Since GREEDY(B,x) accepts an integer number of packets, it accepted during the phase at least $\lfloor xB \rfloor$ packets. ■

Claim 6.8 *Consider a phase $(t_0, T]$, and let t, $t_0 < t \leq T$ be any time in the phase. Let g^t be the number of packets accepted by GREEDY(B,x) in time interval $(t_0, t]$ and f^t be the amount of packets accepted by FRAC_GREEDY(B,x) in time interval $(t_0, t]$. Then, if $G^t > 0$ then*

$$f^t - g^t \leq (F^t - F^{t_0}) - G^t \leq 1.$$

A more intuitive interpretation of the first inequality of the claim is the following. For any strict prefix $(t_0, t]$ of the phase, the amount of packets sent by GREEDY(B,x), i.e., $g^t - G^t$, bounds the number of packets sent by FRAC_GREEDY(B,x), i.e. $f^t - (F^t - F^{t_0})$. Note that $F^t - F^{t_0}$ is the increase in size of the occupied space of the buffer of FRAC_GREEDY(B,x) since the start of the phase, and $G^t - G^{t_0} = G^t$ is the increase in the size of the occupied space of the buffer of GREEDY(B,x) since the start of the phase. The second inequality bounds the difference in the changes by at most one.

Proof. We prove the claim by induction on the events. Let G^k be the size of the occupied space in the buffer of GREEDY(B,x) after the k-the event and let F^k be the size of the occupied space in the buffer of FRAC_GREEDY(B,x) after event k in the phase. The base of the induction is the first event. If the first event is a send event then $G^1 = 0$ and the phase ends. Since no packet arrives at the phase the claim clearly holds. If the first event is an event of the arrival of a packet, then $G^1 = g^1 = 1$ and $f^1 = F^1 - F^{t_0}$ and the claim holds.

Assume the claim holds for the first $k - 1$ events. If the k-the event is a send event, after which $G^k = 0$, then the claim holds (since the claim requires that $G^t > 0$), and the phase ends. Otherwise assume the k-the event is a send event after which $G^k > 0$. This implies that $G^{k-1} > x$. Using Claim 6.6 we have that $F^{k-1} > x$. Therefore $G^k = G^{k-1} - x$ and $F^k = F^{k-1} - x$. Therefore

$$f^k - g^k = f^{k-1} - g^{k-1} \leq (F^{k-1} - F^{t_0}) - G^{k-1} = (F^k + x - F^{t_0}) - (G^k + x) = (F^k - F^{t_0}) - G^k \leq 1 \ .$$

Now assume that the k-the event is an arrival of a packet. Assume that FRAC_GREEDY(B,x) accepts a $z \in [0, 1]$ fraction of a packet and GREEDY(B,x) accepts $y \in \{0, 1\}$ packet.

$$f^k - g^k = f^{k-1} - g^{k-1} + z - y \leq (F^{k-1} - F^{t_0}) - G^{k-1} + z - y = (F^k - F^{t_0}) - G^k \ .$$

Now we need to show that $d_k = (F^k - F^{t_0}) - G^k \leq 1$. From the above inequality it follows that this is equivalent to showing that $d_k = d_{k-1} + z - y \leq 1$, using the induction hypothesis that says that $d_{k-1} \leq 1$. To prove that observe that if $y = 1$ then $d_k \leq d_{k-1} \leq 1$. If $y = 0$ (i.e., GREEDY(B,x) rejects the packet), then $G^k = G^{k-1} > xB - 1$. Since $F^k \leq xB$ it follows that $d_k \leq 1$. ■

To conclude the proof of the lemma we consider each phase separately, and the amount of packets accepted by GREEDY(B,x) and FRAC_GREEDY(B,x) in each phase. For each phase we claim the following.

Claim 6.9 *Consider a phase $(t_0, T]$. Let g be the number packets accepted by GREEDY(B,x) in that phase, and let f be the number of packets accepted by FRAC_GREEDY(B,x) in that phase. Then $f \leq (1 + \frac{1}{\lfloor xB \rfloor}) \cdot g$.*

Proof. If GREEDY(B,x) does not reject any packet during the phase, then $g \geq f$. Otherwise by Claim 6.7 we have that $g \geq \lfloor xB \rfloor$. Now consider the last event of an arrival of a packet in the phase and let t be the time of this event. The amount of packets accepted by any of the two policies clearly does not

grow after time t. Therefore (using the notations of Claim 6.8) we have that $g = g^t$ and $f = f^t$, and in particular $g^t = g \geq \lfloor xB \rfloor$. By Claim 6.8 $f^t - g^t \leq 1$ and by Claim 6.7 we have that $1 \leq g^t / \lfloor xB \rfloor$. Combining the two we have,

$$f = f^t \leq g^t + 1 \leq g^t + \frac{g^t}{\lfloor xB \rfloor} = g^t \cdot (1 + \frac{1}{\lfloor xB \rfloor}) = g(1 + \frac{1}{\lfloor xB \rfloor}) \ ,$$

which completes the proof. ■

By summing over all phases we conclude the proof of Lemma 6.5. ■

Proof of Lemma 6.1 By Lemma 6.3 we have that $GREEDY(B,1)^{\mathcal{P}}(\Lambda) \leq \frac{1}{x} \cdot FRAC\_GREEDY(B,x)^{\mathcal{P}}(\Lambda)$. By Lemma 6.5 $FRAC\_GREEDY(B,x)^{\mathcal{P}}(\Lambda) \leq (1 + \frac{1}{\lfloor xB \rfloor}) \cdot GREEDY(B,x)^{\mathcal{P}}(\Lambda)$. The lemma follows. □

We can now give the proof of the second main lemma of this subsection.

Proof of Lemma 6.2 We first compare the number of packets accepted by P to the number of packets accepted by an optimal policy that uses a buffer of size $\lfloor xB \rfloor$ (and sends a complete packet in each send event), i.e., $GREEDY(\lfloor xB \rfloor, 1)$. We show now that the number of packets accepted by P is at least the number of packets accepted by $GREEDY(\lfloor xB \rfloor, 1)$.

By induction on the events we claim that after any event the number of packets in the buffer of P is at least the number of packets in the buffer of $GREEDY(\lfloor xB \rfloor, 1)$. The claim clearly holds at the beginning when both buffers are empty. For an inductive step we consider the two types of events. For a send event, if $GREEDY(\lfloor xB \rfloor, 1)$ sends a packet clearly the relation still holds (and P also sends a packet). If $GREEDY(\lfloor xB \rfloor, 1)$ does not send a packet, then its buffer is empty, and the claim clearly holds. For a packet arrival event, if P accepts the packet then the claim clearly continues to hold. Otherwise, if P rejects the packet, it must be that its buffer contains at least $\lfloor xB \rfloor$ packets. Since the buffer of $GREEDY(\lfloor xB \rfloor, 1)$ is bounded by $\lfloor xB \rfloor$, the claim clearly holds.

From the above it follows that when the buffer of the $GREEDY(\lfloor xB \rfloor, 1)$ policy is nonempty so is the buffer of P. Therefore the number of packets sent by P by any time t is at least the number of packets sent by the $GREEDY(\lfloor xB \rfloor, 1)$ by time t.

The number of packets a policy accepts in at interval of time $(0, t]$ equals the number of packets it sends plus the number of packets it has in the buffer, assuming it starts with an empty buffer. Since the number of packets in the buffer of P at time t is at least the number of packets in the buffer of $GREEDY(\lfloor xB \rfloor, 1)$ at time t, and during the time interval $(0, t]$ policy P sends at least the same number of packets as $GREEDY(\lfloor xB \rfloor, 1)$, it follows that P accepts at least the same number of packets as the $GREEDY(\lfloor xB \rfloor, 1)$ policy accepts during any time interval.

Next we show that $GREEDY(\lfloor xB \rfloor, 1)$ accepts at least the same number of packets accepted by the $GREEDY(B, x)$ policy. Let E_k be the number of packets accepted by $GREEDY(\lfloor xB \rfloor, 1)$ minus the number of packets accepted by $GREEDY(B, x)$ after the k-the event. We claim that after event k the amount of packets in the buffer of $GREEDY(\lfloor xB \rfloor, 1)$ is at most the amount of packets in $GREEDY(B, x)$ plus E_k, and that $E_k \geq 0$. We prove this by induction on the events. Initially, both buffers are empty, and $E_k = 0$, so the claim holds.

Assume that the claim holds up to event $k - 1$. If event k is a send event, then $E_k = E_{k-1} \geq 0$. Also, if the buffer of $GREEDY(\lfloor xB \rfloor, 1)$ is empty after the send event, then the claim clearly holds. Otherwise, the size of the buffer of $GREEDY(\lfloor xB \rfloor, 1)$ is reduced by 1 and the size of the buffer of $GREEDY(B, x)$ is reduced by at most x, and the inequality holds.

If the k-the event is a packet arrival, then the simple cases is either when both policies accept the packet or when both policies reject the packet, keeping $E_k = E_{k-1} \geq 0$. If $GREEDY(\lfloor xB \rfloor, 1)$ accepts the packet and $GREEDY(B, x)$ rejects the packet, then $E_k = E_{k-1} + 1$ and the claim holds. If

$GREEDY(\lfloor xB \rfloor, 1)$ rejects the packet and $GREEDY(B, x)$ accepts the packet then $E_k = E_{k-1} - 1$, and the first part of the claim holds. In the last case we need to verify that $E_k \geq 0$. We prove this by contradiction. Assume that $E_k = -1$ and therefore $E_{k-1} = 0$. Since $GREEDY(\lfloor xB \rfloor)$ rejects the packet we know that it has exactly $\lfloor xB \rfloor$ packets in the buffer. Since $E_{k-1} = 0$, by the inductive claim the buffer of $GREEDY(B, x)$ has at least $\lfloor xB \rfloor$ packets. Therefore, $GREEDY(B, x)$ cannot accept the packet that arrives. A contradiction.

From the above inductive claim, it follows that $E_k \geq 0$ at all times, namely, that $GREEDY(\lfloor xB \rfloor, 1)$ accepts at least as many packets as $GREEDY(B, x)$.

We have shown that P accepts at least the number of packets that $GREEDY(\lfloor xB \rfloor, 1)$ accepts, and that $GREEDY(\lfloor xB \rfloor, 1)$ accepts at least the number of packets that $GREEDY(B, x)$ accepts. Therefore P accepts at least the number of packets that $GREEDY(B, x)$ accepts. Lemma 6.1 derives the relation between $GREEDY(B, 1)$ and $GREEDY(B, x)$ which completes the proof. □

6.2 The Greedy Policy

We start by analyzing the simplest policy, the Greedy Policy, and show that its competitive ratio is at least $1/\alpha$.

Theorem 6.10 *The competitive ratio of the Greedy Policy is at least* $1/\alpha$.

Proof. Clearly, the Greedy Policy maximizes the total number of packets accepted. Thus, the number of packets accepted by the optimal policy is at most the number of packets accepted by the Greedy Policy. Since for every i, the benefit of the $i$the packet accepted by the optimal policy is at most α times the benefit of the $i$the packet accepted by the Greedy Policy, the theorem follows. ■

A competitive ratio of $1/\alpha$ is not appealing since as α increases the competitive ratio decreases to zero. The other policies that we consider will guarantee a competitive ratio bounded away from zero as α tends to infinity.

6.3 The Round Robin Policy

We now prove a bound on the competitive ratio of the Round Robin Policy. We consider the case of a Round Robin Policy that uses for each one of the two types of packets a buffer of size $B/2$. As discussed above we do not generalize to the case of two buffers of sizes xB and $(1-x)B$ since the best possible competitive ratio is achieved for $x = 1/2$.

Theorem 6.11 *The competitive ratio of the Round-Robin Policy is at least* $\frac{1}{2(1+\frac{1}{\lfloor B/2 \rfloor})} \geq 1/2 - 1/(B+1)$.

Proof. We consider the (sub)sequence of low priority packets and the (sub)sequence of high priority packets. Observer that the number of low priority packets (resp. high priority packets) accepted by the Round Robin Policy is at least the number of packets accepted by the GREEDY(B,1/2) policy, if this policy is presented only with the low priority packets (resp. high priority packets).

The maximum number of packets that can be accepted out of the (sub)sequence of low priority packets (resp. high priority packets) is at most $2(1 + \frac{1}{\lfloor B/2 \rfloor})$ times the number of low priority packets (resp. high priority packets) accepted by GREEDY(B,1/2), by Lemma 6.1. The Lemma then follows. ■

6.4 The Fixed Partition Policy

In this subsection we analyze the Fixed Partition Policy (denoted FPP) and derive a bound on its competitive ratio. In this policy there is some interaction between the low priority and the high priority packets due to the fact that they share a single buffer. That is, the policy does not behave exactly like two separate queues, one for the high priority packets and one for the low priority packets, each queue with half the size. To see that consider the following scenario. In the first time step $B/2$ low priority packets arrive followed by $B/2$ high priority packets. The Fixed Partition Policy accepts them all so that the low priority packets precede the high priority packets in the FIFO output queue. For $B/4$ time steps no packets arrive and then $B/8$ high priority packets arrive. The Fixed Partition Policy must discard all of them since no high priority packets have yet been sent from the FIFO queue. In spite of this interaction, we will first analyze the competitive ratio for each setting (high priority or low priority) separately, and then combine the results.

We prove the following theorem.

Theorem 6.12 *The competitive ratio of the Fixed Partition Policy with $x = 1/2$ is at least $\frac{1}{4+\frac{1}{\lfloor B/2 \rfloor}}$.*

Proof. We analyze the performance of the Fixed Partition policy by separately considering the low priority packets and the high priority packets. We show that the number of low priority packets (resp. high priority packets) accepted by the Fixed Partition Policy (with $x = 1/2$) is at least $\frac{1}{4+\frac{1}{\lfloor B/2 \rfloor}}$ the number of low priority packets (resp. high priority packets) accepted by the optimal policy.

We now consider the low priority packets. The main technical claim is given in the following lemma.

Lemma 6.13 *For any event sequence Λ, $OPT^{\mathcal{L}}(\Lambda) \leq (4 + \frac{1}{\lfloor B/2 \rfloor}) \cdot FPP^{\mathcal{L}}(\Lambda)$.*

Proof. To prove the lemma we bound the difference between the number of low priority packets accepted by the optimal algorithm and the number of those packets accepted by the Fixed Partition Policy. That is, we show that $OPT^{\mathcal{L}}(\Lambda) - FPP^{\mathcal{L}}(\Lambda) \leq (3 + \frac{1}{\lfloor B/2 \rfloor}) \cdot FPP^{\mathcal{L}}(\Lambda)$. To give this bound we need only consider packets that are not accepted by the Fixed Partition Policy. If, before an event of an arrival of a packet the buffer of the Fixed Partition Policy includes less than $\lfloor B/2 \rfloor$ low priority packets then we know that the arriving packet is accepted by the Fixed Partition Policy. We therefore have to consider only arrivals of packets when the buffer of the Fixed Partition Policy contains $\lfloor B/2 \rfloor$ low priority packets. We define *overload periods*. The first overload period starts immediately after the first event after which the buffer of the Fixed Partition Policy contains $\lfloor B/2 \rfloor$ low priority packets and ends when B send events have occurred. Any subsequent overload period starts the first time, after the end of the previous overload interval, when the buffer contains $\lfloor B/2 \rfloor$ low priority packets. This implies that an overload period is an interval of the form $(t, \lfloor t + B \rfloor]$.

We now give the following claim.

Claim 6.14 *For any overload period $\mathcal{T}$, the difference between the number of low priority packets accepted by the optimal policy, and the number of low priority packets accepted by the Fixed Partition Policy is at most $B + \lfloor B/2 \rfloor$.*

Proof. Let t_0 be the time when the overload period started. The overload period ends at time $\lfloor t_0 + B \rfloor$. Let t_1 be the time of the last send event in the overload period, before which the buffer of the Fixed Partition Policy contains $\lfloor B/2 \rfloor$ low priority packets. Clearly the Fixed Partition Policy accepts all the low priority packets that arrive during the overload period after time t_1.

We now consider two cases. The first is $t_1 \leq \lfloor t_0 \rfloor + \lfloor B/2 \rfloor + 1$, and the second is $t_1 > \lfloor t_0 \rfloor + \lfloor B/2 \rfloor + 1$.

If $t_1 \leq \lfloor t_0 \rfloor + \lfloor B/2 \rfloor + 1$ then the total number of packets that the optimal policy accepts during the period $(t_0, t_1]$ is at most $B + \lfloor B/2 \rfloor$. This is because it can accept at most the size of its buffer (B) plus the number of send events during the time interval, after which there are events of arrivals of packets ($\lfloor B/2 \rfloor$).

If $t_1 > \lfloor t_0 \rfloor + \lfloor B/2 \rfloor + 1$ then we claim that the Fixed Partition Policy accepts at least $w = t_1 - \lfloor t_0 \rfloor - \lfloor B/2 \rfloor - 1$ low priority packets in $(t_0, t_1]$. To see that first observe that out of the $t_1 - \lfloor t_0 \rfloor - 1$ send events in (t_0, t_1), at least w are used by the Fixed Partition Policy to send low priority packets. This follows from the fact that when the overload period starts the buffer of the Fixed Partition Policy contains at most $\lfloor B/2 \rfloor$ *high* priority packets and the Fixed Partition Policy maintains a FIFO buffer. Now, since before the send event of time t_1 the buffer of the Fixed Partition Policy contains $\lfloor B/2 \rfloor$ low priority packets, we have that it accepted at least w such packets. On the other hand the optimal policy can accept in $(t_0, t_1]$ at most $B + (t_1 - \lfloor t_0 \rfloor - 1)$ low priority packets. The number of low priority packets accepted by the optimal policy minus the number of low priority packets accepted by the Fixed Partition Policy is therefore at most $B + (t_1 - \lfloor t_0 \rfloor - 1) - w = B + (t_1 - \lfloor t_0 \rfloor - 1) - (t_1 - \lfloor t_0 \rfloor - \lfloor B/2 \rfloor - 1) = B + \lfloor B/2 \rfloor$. ■

Now observe that when an overload period starts we have $\lfloor B/2 \rfloor$ low priority packets in the buffer of the Fixed Partition Policy. Further observe that when the overload period ends all these packets have been sent from the buffer (this is because any packet is send within B send events from its arrival into he buffer). Therefore we can "charge" the (at most) $B + \lfloor B/2 \rfloor$ "excess" packets of the optimal policy of a certain overload period to the $\lfloor B/2 \rfloor$ low priority packets present in the buffer when that overload period starts, and we will never charge twice to the same packet. If follows that the number of "excess" packets that the optimal policy accepts is at most $\frac{B + \lfloor B/2 \rfloor}{\lfloor B/2 \rfloor} \leq 3 + \frac{1}{\lfloor B/2 \rfloor}$ times the number of low priority packets that the Fixed Partition Policy accepts. ■

An analogue proof gives the following lemma.

Lemma 6.15 *For any event sequence* Λ $OPT^{\mathcal{H}}(\Lambda) \leq (4 + \frac{1}{\lfloor B/2 \rfloor}) \cdot FPP^{\mathcal{H}}(\Lambda)$.

Theorem 6.12 now immediately follows from the above two lemmas.

■

6.5 The Dynamic Flexible Partition Policy

In this section we study the competitive ratio of the Dynamic Flexible Partition Policy (denoted $DFPP$), and in the next section we study the competitive ratio of the Flexible Partition Policy (denoted $FLPP$). We restrict our attention to the case where the low priority packets can occupy at most half of the buffer, i.e., the case that the parameter x is set to $1/2$, and derive a competitive ratio of $\frac{1}{2(1+\frac{1}{\lfloor B/2 \rfloor})} \geq 1/2 - 1/(B+1)$.

Our proof uses a matching between the high priority and the low priority packets that the Dynamic Flexible Partition Policy accepts. Later we bound the number of high priority packets that an optimal offline accepts by the number of high priority packets plus the low priority matched packets of the Dynamic Flexible Partition Policy. In addition, the total number of packets accepted by an optimal offline policy is at most (roughly) twice the total number of packets accepted by the Dynamic Flexible Partition Policy. Those two facts enables us to derive a competitive ratio of $\frac{1}{2(1+\frac{1}{\lfloor B/2 \rfloor})} \geq 1/2 - 1/(B+1)$.

We define a *matching* as follows: when a high priority packet arrives it is matched to the unmatched low priority packet closest to the head of the queue (if such a packet exists).

The following lemma gives a relation between the number of unmatched packets to the number of free slots in the buffer for the Dynamic Flexible Partition Policy.

Lemma 6.16 *In the Dynamic Flexible Partition Policy at any time we have that $\ell \leq f$, where ℓ is the number of unmatched low priority packets in the queue and f is the number of free slots.*

Proof. We prove the lemma by induction on events. Initially we have $\ell = 0$ and $f = B$ and clearly the claim holds. Assume the claim holds up to event $k-1$ and consider the k-the event. Assume the k-the event is an arrival of a packet. If we reject it, clearly the claim holds, therefore assume we accept it. When we accept a packet in the queue it can be either a low priority packet or a high priority packet.

For a low priority packet, by the definition of the Dynamic Flexible Partition Policy with $x = 1/2$, we accept a low priority packet only if after we accept it, the number of low priority packets in the queue is no more than the number of free slots in the queue. Therefore the claim holds after we accept a low priority packet.

When we accept a high priority packet we have one less free slot, but also one less unmatched low priority packet assuming there are unmatched low priority packets in the queue. If there are no unmatched low priority packets in the queue the lemma holds trivially.

When we send a packet (either high priority or low priority) the number of free slots increases by 1, and the number of unmatched low priority packets cannot increase (it can either decrease by 1 or stay unchanged). ■

An immediate corollary of the above lemma is the following.

Corollary 6.17 *In the Dynamic Flexible Partition Policy when the buffer is full all the low priority packets in the buffer are matched.*

We call a packet *good* at a given time if it is: (1) a high priority packet that was accepted, or (2) a low priority packet that is matched. Observe that once a packet becomes good it stays good.

In the following we bound the number of high priority packets that the optimal offline policy accepts, by the number of packets that become good in the Dynamic Flexible Partition Policy. To this end we consider the G-HIGH(B,1) policy, which rejects any low priority packet, and accepts any high priority packets, as long as it has less than B packets in its buffer. Clearly, for any event sequence, the number of high priority packets G-HIGH(B,1) accepts bounds from above the number of high priority packets any other policy can accept.

Given the online buffer, we define the *good prefix* as the sequence of consecutive good packets from the head of the buffer. We proceed with the following lemma.

Lemma 6.18 *In the Dynamic Flexible Partition Policy, for any event sequence Λ, at any time the number of packets in the good prefix is at least the number of packets in the queue of G-HIGH(B,1).*

Proof. The proof is by induction on the events. Initially both buffers are empty and the claim holds trivially. When a high priority packet arrives, G-HIGH(B,1) can either accept it (increasing the number of packets in the buffer by one) or reject it (if the buffer is full). The Dynamic Flexible Partition Policy has the property that if it rejects a high priority packet, then the buffer is full. By Corollary 6.17 all the packets in the buffer are matched, and hence the good prefix is the entire buffer, i.e., of size B. Therefore if the high priority packet is rejected the inductive claim holds.

If the high priority packet is accepted and there is some unmatched low priority packet in the buffer, then we add a matching, which increases the size of the good prefix by at least one. If the high priority packet is accepted and there is no unmatched low priority packet in the buffer, then the new packet is added to the good prefix, whose size is increased by 1. We showed that in all the cases the inductive claim is maintained after an arrival of a high priority packet.

An arrival of a low priority packet does not change the buffer of G-HIGH(B,1) or the good prefix of the Dynamic Flexible Partition Policy buffer.

Now consider a send event. If before the send event the buffer of G-HIGH(B,1) is empty then the inductive claim clearly holds also after the event. Otherwise, we can use the inductive hypothesis to know that before the event the good prefix was at least as large as the buffer of G-HIGH(B,1). Since both policies send one packet, the inductive claim holds also after the event. ■

Lemma 6.19 *For any event sequence Λ, the number of packets G-HIGH(B,1) accepts is at most the number of packets that become good in the Dynamic Flexible Partition Policy.*

Proof. From Lemma 6.18 it follows that for every send event in which the Dynamic Flexible Partition either sends a packet which is not good or does not send a packet at all, then the G-HIGH(B,1) policy does not send a packet. Therefore, the number of packets sent by the G-HIGH(B,1) policy is at most the number of good packets sent by the Dynamic Flexible Partition policy.

The number of packets accepted by the G-HIGH(B,1) policy is the number of packets sent by the G-HIGH(B,1) policy plus the number of packets in the buffer of the G-HIGH(B,1) policy. As argued above the former is at most the number of good packets sent by the Dynamic Flexible Partition policy. By Lemma 6.18, the latter is at most the number of good packets in the buffer of the Dynamic Flexible Partition policy. ■

The next claim shows that the number of accepted packets (regardless of whether being high priority packets or low priority packets) is not too low compared to the number of packet accepted by the optimal policy.

Claim 6.20 *For any event sequence Λ, $OPT^{\mathcal{P}}(\Lambda) \leq 2(1 + \frac{1}{\lfloor B/2 \rfloor}) \cdot DFPP^{\mathcal{P}}(\Lambda)$.*

Proof. The Dynamic Flexible Partition Policy with $x = 1/2$ never rejects a high priority packet if it has less than B packets in the buffer, and never rejects a low priority packet if it has less than $\lfloor B/2 \rfloor$ packets (of any type) in the buffer. The present claim then follows from Lemma 6.2 with $x = 1/2$. ■

Theorem 6.21 *The competitive ratio of the Dynamic Flexible Partition Policy is at least $\frac{1}{2(1+\frac{1}{\lfloor B/2 \rfloor})} \geq 1/2 - 1/(B+1)$.*

Proof. Consider a fixed sequence of events. Let k_1 and m_1 be the number of low and high priority packets an optimal offline accepts, respectively. Let k_2 and m_2 be the number of low and high priority packets the Dynamic Flexible Partition Policy accepts, respectively.

Let $\epsilon = \frac{2}{\lfloor B/2 \rfloor}$. We show that $k_1 + \alpha m_1 \leq (2+\epsilon)[k_2 + \alpha m_2]$, which establishes the theorem. By Claim 6.20,

$$k_1 + m_1 \leq (2+\epsilon)(k_2 + m_2). \tag{1}$$

Let g_2 be the number of packets that become good packets in the Dynamic Flexible Partition Policy. The matching guarantees that at least half of the good packets are high priority packets. Using Lemma 6.19 we have,

$$m_1 \leq g_2 \leq 2m_2. \tag{2}$$

By multiplying (2) by $\alpha - 1$ and adding to (1), we have,

$$\alpha m_1 + k_1 \leq (2+\epsilon)[\alpha m_2 + k_2]$$

which completes the proof. ■

For the special case of $\alpha = 2$ we can derive a better competitive ratio, by using a different value for x.

Theorem 6.22 *The Dynamic Flexible Partition Policy, with $x = 3/4$, has a competitive ratio of at least $\frac{1}{\frac{28}{15}(1+\frac{1}{\lfloor 3B/4 \rfloor})}$ for $\alpha = 2$.*

Proof. We use the same notations as in the proof of the previous theorem. First, we use the value of $x = 3/4$ and the same arguments as those in the proof of Claim 6.20 to bound from below the total number of packets accepted by the Dynamic Flexible Partition Policy. Let $\epsilon = \frac{4}{3\lfloor 3B/4 \rfloor}$, then

$$m_1 + k_1 \leq (4/3 + \epsilon)[m_2 + k_2] \tag{3}$$

We now modify the down matching to match a high packet to 3 low priority packets. This implies that,

$$m_1 \leq g_2 \leq 4m_2. \tag{4}$$

Finally, the number of packets accepted by the Dynamic Flexible Partition Policy is at least the number of high priority packets accepted by the optimal policy.

$$m_1 \leq m_2 + k_2 \tag{5}$$

We add (3) to 7/15 times (4) and 8/15 time (5). We get,

$$2m_1 + k_1 \leq (\frac{28}{15} + \epsilon)[2m_2 + k_2]$$

■

6.6 The Flexible Partition Policy

In this section we study the competitive ratio of the Flexible Partition Policy (denoted $FLPP$), and show that its competitive ratio is at least $\frac{\sqrt{2}-1}{1+\frac{1}{\lfloor(\sqrt{2}-1)B\rfloor}}$. This is achieved when the parameter x is set to $\sqrt{2}-1$.

The proof technique is similar in spirit to the proof given for the Dynamic Flexible Partition Policy, except that the notion of "matching" low priority packets to high priority packets is somewhat more involved, as described below. Each high priority packet "gives" some "weight" to some low priority packets with whom it shares the buffer at some point in time. The total weight that each high priority packet distributes is at most $2x/(1-x)$. The total weight that each low priority packet receives is at most 1. We call a low priority packet *matched* if its weight is 1. Whenever a new packet is added to the buffer the following procedure of weight assignment is performed: (1) If the new packet is a high priority packet, then the low priority packets in the buffer are scanned from the packet closest to the head of the queue to the packet closest to the tail of the queue. Each unmatched low priority packet is assigned weight so as to complete its weight towards 1, until the full $2x/(1-x)$ weight of the high priority packet is used. If some of the weight available to the high priority packet is not used, then this weight is left as credit. (2) If the new packet is a low priority packet, and some high priority packets have some credit, then this credit is assigned to the new packet (up to weight 1).

The following lemma is immediate from the description above.

Lemma 6.23 *In the Flexible Partition Policy, for any event sequence Λ, the number of matched low priority packets is at most the number of accepted high priority packets times $2x/(1-x)$.*

The next lemma shows that when the buffer is full in the Flexible Partition Policy, all the low priority packets in the buffer are matched.

Lemma 6.24 *In the Flexible Partition Policy when the buffer is full then each low priority packet in the buffer is matched, i.e., has weight 1.*

Proof. Consider a time t when the buffer is full. First note that when the buffer is full there are at least $B - \lfloor xB \rfloor$ high priority packets in the buffer. Consider the high priority packet closest to the head of the queue. When it arrived to the buffer there were at most $\lfloor xB \rfloor$ low priority packets in the buffer. All low priority packets that arrived later are still in the buffer at time t. Next observe that if there is in the buffer a low priority packet which is not matched, then no high priority packet has any weight credit at this time. This means that the (at least) $B - \lfloor xB \rfloor$ high priority packets in the buffer have distributed all of the (at least) $\frac{2x}{1-x}(B - \lfloor xB \rfloor)$ weight that they could distribute. This weight was distributed to packets that are in the buffer at t and to packets that left the buffer before t. But the number of packets that received (some of) this weight and left the buffer before t is at most $\lfloor xB \rfloor$. Since each packet receives weight of at most 1, the total weight assigned to the low priority packets in the buffer at t is at least $\frac{2x}{1-x}(B - \lfloor xB \rfloor) - \lfloor xB \rfloor \geq \lfloor xB \rfloor$. Since there are at most $\lfloor xB \rfloor$ such packets, they are all matched. ■

We call a packet *good* at a given time if it is: (1) a high priority packet that was accepted, or (2) a low priority packet that is matched. Observe that once a packet becomes good it stays good. Given the Flexible Partition Policy buffer, we define the *good prefix* as the sequence of consecutive good packets from the head of the buffer towards the tail of the buffer.

In order to bound the number of high priority packets the offline policy accepts, we consider the G-HIGH(B,1) policy, defined in the previous section. The next lemma compares the number of packets G-HIGH(B,1) accepts to that of our online policy.

Lemma 6.25 *In the Flexible Partition Policy with parameter $x \geq 1/3$ for any event sequence Λ, at any time the number of packets in the good prefix is at least the number of packets in the buffer of G-HIGH(B,1).*

Proof. We first note that for $x \geq 1/3$, $2x/(1-x) \geq 1$. The proof then proceeds by induction on the events in a way identical to the proof of Lemma 6.18.

■

The next claim gives a bound on the total number of packets accepted by the Flexible Partition Policy.

Lemma 6.26 *For any event sequence Λ, $OPT^{\mathcal{P}}(\Lambda) \leq \frac{1}{x}(1 + \frac{1}{\lfloor xB \rfloor}) \cdot FLPP^{\mathcal{P}}(\Lambda)$.*

Proof. The Flexible Partition Policy rejects a high priority packet only if it has already B packets in its buffer. It rejects a low priority packet only if it has already $\lfloor xB \rfloor$ low priority packets in its buffer. The present lemma then follows from Lemma 6.2. ■

Theorem 6.27 *The competitive ratio of the Flexible Partition Policy is at least $\frac{\sqrt{2}-1}{1+\frac{1}{\lfloor(\sqrt{2}-1)B\rfloor}}$.*

Proof. Consider a fixed sequence of events, and consider any $x \geq 1/3$. Let k_1 and m_1 be the number of low priority and high priority packets an optimal offline accepts, respectively. Let k_3 and m_3 be the number of low priority and high priority packets the Flexible Partition Policy accepts, respectively. Let $\epsilon = \frac{1}{x\lfloor xB \rfloor}$.

By Lemma 6.26,

$$k_1 + m_1 \leq (\frac{1}{x} + \epsilon)(k_3 + m_3). \tag{6}$$

Let g_3 be the number of packets that become good in the Flexible Partition Policy. By Lemma 6.25 we have that $m_1 \leq g_3$. By Lemma 6.23 we bound g_3, hence,

$$m_1 \leq g_3 \leq (\frac{2x}{1-x} + 1)m_3 = \frac{1+x}{1-x}m_3.$$

Therefore,

$$\alpha m_1 + k_1 \leq (\frac{1+x}{1-x}(\alpha - 1) + \frac{1}{x} + \epsilon)m_3 + (\frac{1}{x} + \epsilon)k_3.$$

For $A = \max\{1/x + \epsilon, \frac{1+x}{1-x} \cdot \frac{\alpha-1}{\alpha} + \frac{1}{\alpha x} + \epsilon/\alpha\}$, we have that,

$$\alpha m_1 + k_1 \leq A[\alpha m_3 + k_3].$$

By setting $x = \sqrt{2} - 1$ we have that A is at most $\frac{1}{\sqrt{2}-1}(1 + \frac{1}{\lfloor(\sqrt{2}-1)B\rfloor})$, and the competitive ratio is at least $\frac{\sqrt{2}-1}{1+\frac{1}{\lfloor(\sqrt{2}-1)B\rfloor}}$. (Note that $x \geq 1/3$, as assumed initially.) ■

7 Impossibility Results

In this section we derive impossibility results both for the specific policies we study and a general impossibility result that bounds the performance of any online policy. We start with the simple case of the Greedy Policy.

Theorem 7.1 *The competitive ratio of the Greedy Policy is at most* $1/\alpha$.

Proof. Consider the event sequence that includes B low priority packets followed by B high priority packets, before any send event is given. The Greedy Policy accepts the B low priority packets and has a benefit of B, while an optimal offline policy would accept the B high priority packets, and has a benefit of αB. Therefore the competitive ratio of the Greedy Policy is at most $1/\alpha$. ■

Now we turn to the Round Robin Policy and the Fixed Partition Policy. In both cases the lower bound is based on the fact that the policy fixes in advance a partition of the buffer between low priority and high priority packets.

Theorem 7.2 *For any value of* α *and any parameter* x*, the Round Robin Policy has competitive ratio at most* $1/2$.

Proof. Consider the Round Robin Policy on the event sequence of exactly B low priority packets arriving before any send event is given. The Round Robin Policy accepts at most xB packets and the optimal policy accepts B so that the ratio of benefits for this sequence is x. Likewise the ratio of benefits for the event sequence of exactly B high priority packets is $1 - x$. Therefore, the competitive ratio is at most $\min\{x, 1 - x\} \leq 1/2$. ■

The proof of the following theorem is identical.

Theorem 7.3 *For any value of α and any parameter x, the Fixed Partition Policy has competitive ratio at most $1/2$.*

The proofs of the other impossibility results are slightly more complicated, but they all share a similar structure. In these proofs, we exhibit two specific sequences of packets arrivals, whose prefix is identical. The online policy makes the same decisions for both sequences in the common prefix, since it has no knowledge of the future. This early decision of the online policy is the source of its sub-optimal performance. On the other hand, an optimal offline policy simply considers each sequence separately.

Theorem 7.4 *Any online policy has a competitive ratio of at most $\alpha/(2\alpha-1) = 1/2 + 1/(4\alpha-2)$.*

Proof. We fix an online policy and consider two scenarios. In the first scenario B low priority packets arrive. The online policy accepts some fraction $0 \leq z \leq 1$ of them, while an optimal offline policy accepts all of them.

The second scenario starts like the first scenario with B low priority packets. The online policy accepts the same fraction z of them (since it behaves the same as in the first scenario). Following the low priority packets, a set of B high priority packets arrives before any send event is given. The online policy (at most) fills the buffer with high priority packets, i.e. accepts at most $(1-z)B$ high priority packets. An optimal offline policy accepts all the high priority packets.

The benefit of the online policy in the first scenario is zB while the benefit of an optimal offline is B. The benefit of the online policy in the second scenario is (at most) $zB + \alpha(1-z)B$ while the benefit of an optimal offline is αB. The competitive ratio of the online policy is therefore at most

$$\min\{\frac{zB}{B}, \frac{zB+\alpha(1-z)B}{\alpha B}\} = \min\{z, \frac{z}{\alpha} + (1-z)\} \ .$$

The maximum over all z of the above expression is at most

$$\frac{\alpha}{2\alpha-1} = \frac{1}{2} + \frac{1}{4\alpha-2} \ ,$$

which bounds the competitive ratio of any online policy. ■

Note that the general bound above implies that no online policy can guarantee a competitive ratio of $1/2+\epsilon$, for sufficiently large α, i.e., $\alpha > (1/2) + 1/(4\epsilon)$.

In the following, using similar techniques, we derive tighter impossibility results for some of our online policies.

Theorem 7.5 *The competitive ratio of the Fixed Partition Policy is at most* $\min\{1/2, \sqrt{2-(1/\alpha)+(1/2\alpha)^2} - (1-(1/2\alpha))\}$.

Proof. For any value of α we give the following proof. Let x be the the parameter according to which the Fixed Partition Policy works. Let y be the largest value $y \leq x$ such that yB is an integer, i.e., $y = \lfloor xB \rfloor / B$. Consider now two sequences of packets. One sequence is a sequence of B low priority packets. Since the optimal policy would take all of these packets the competitive ratio cannot be more than y, as the Fixed partition Policy would take yB packets. The second sequence is as follows: First B low priority packets are given and then B high priority packets. Then, after yB send events, yB high priority packets are given. The Fixed Partition Policy will accept yB of the B low priority packets, and $(1-y)B$ of the first B high priority packets. The optimal policy accepts all of the first B high priority packets, and then another yB high priority packets. The benefit of the optimal policy is $(1+y)B\alpha$,

while the benefit of the Fixed Priority Policy is $yB + (1-y)B\alpha$. The competitive ratio is therefore bounded from above by $(y + (1-y)\alpha)/((1+y)\alpha))$.

For a given α, the competitive ratio of the Fixed Partition Policy with parameter x is therefore bounded from above by $\min\{y, \frac{y+(1-y)\alpha}{(1+y)\alpha}\}$, for y as defined above. For a given α we choose x such that the above expression is maximized. The maximum of the above expression cannot be bigger than its value when y is set such that $y = \frac{y+(1-y)\alpha}{(1+y)\alpha}$. This occurs for $y = \sqrt{2-(1/\alpha)+(1/2\alpha)^2} - (1-(1/2\alpha))$, and the competitive ratio is bounded from above by $y = \sqrt{2-(1/\alpha)+(1/2\alpha)^2} - (1-(1/2\alpha))$.

Together with Theorem 7.3, we obtain that the competitive ratio of the Fixed Partition Policy is bounded from above by $\min\{1/2, \sqrt{2-(1/\alpha)+(1/2\alpha)^2} - (1-(1/2\alpha))\}$. ■

We now turn to the Flexible Partition Policy.

Theorem 7.6 *The competitive ratio of the Flexible Partition Policy is at most* $\sqrt{2-(2/\alpha)+(1/\alpha)^2} - (1-(1/\alpha))$.

Proof. We give the following proof for any α. Let x be the the parameter according to which the Flexible Partition Policy works. Let y be the largest value $y \leq x$ such that yB is an integer, i.e., $y = \lfloor xB \rfloor / B$. We consider two scenarios. In the first scenario B low priority packets arrive. By the definition of the Flexible Partition Policy it accepts yB of these packets. An optimal offline policy accepts all of them, and therefore the competitive ratio of the Flexible Partition Policy is at most y. The second scenario starts like the first scenario with B low priority packets. They are immediately followed by B high priority packets, with no send event in between. After yB send events another yB low priority packets arrive followed by yB high priority packets. The Flexible Partition Policy accepts yB of the first low priority packets, then accepts $(1-y)B$ high priority packets, and finally accepts yB low priority packets. An optimal offline policy accepts $(1+y)B$ high priority packets. The competitive ratio is therefore at most $(\alpha(1-y)+2y)/(\alpha(1+y))$.

For a given α, the competitive ratio of the Flexible Partition Policy with parameter x is therefore bounded from above by $\min\{y, \frac{(1-y)\alpha+2y}{\alpha(1+y)}\}$, for y as defined above. For a given α we choose x such that the above expression is maximized. The maximum of the above expression cannot be bigger than its value when y is set such that $y = \frac{(1-y)\alpha+2y}{\alpha(1+y)}$. This occurs for $y = \sqrt{2-(2/\alpha)+(1/\alpha)^2} - (1-(1/\alpha))$ and the competitive ratio is bounded from above by $\sqrt{2-(2/\alpha)+(1/\alpha)^2} - (1-(1/\alpha))$. ■

We can also derive an impossibility result for the Dynamic Flexible Partition Policy.

Theorem 7.7 *The Dynamic Flexible Partition Policy with parameter x has a competitive ratio of at most* $\min\{y, \frac{(1-y)\alpha+y+\frac{1}{B}}{\alpha+y(1-y)}\}$, *where* $y = \lfloor xB \rfloor / B$.

Proof. We consider two scenarios. In the first scenario B low priority packets arrive. By the definition of the Dynamic Flexible Partition Policy it accepts yB of them. An optimal offline policy accepts all of them, and therefore the competitive ratio of the Dynamic Flexible Partition Policy is at most y.

The second scenario starts like the first scenario with B low priority packets. They are immediately followed by B high priority packets, without any send event in between. After $\lceil y(1-y)B \rceil$ send events another $\lceil y(1-y)B \rceil$ low priority packet arrive. The Dynamic Flexible Partition Policy accepts yB of the first low priority packets, then accepts $(1-y)B$ high priority packets, and finally accepts at most one of the remaining low priority packets. To see that the Dynamic Flexible Partition Policy accepts at most one the $\lceil y(1-y)B \rceil$ low priority packets observe that when they start arriving, the number of low priority packets in the buffer is $yB - \lceil y(1-y)B \rceil > yB - (y(1-y)B + 1) = y^2B - 1$. Therefore after accepting one packet there are more than y^2B low priority packets in the buffer, while the space not

occupied by high priority packets is of size yB. Since $x < y + 1/B$, no additional low priority packet can be accepted. An optimal offline policy accepts B high priority packets and $\lceil y(1-y)B \rceil$ low priority packets. Therefore the competitive ratio is at most $(\alpha(1-y) + y + \frac{1}{B})/(\alpha + y(1-y))$. ■

Acknowledgments We thank Nir Andelman for pointing out to us an error in an earlier version of this paper.

References

[1] N. Andelman, Y. Mansour, and A. Zhu, Competitive Queueing Policies for QoS switches. In *proc. of the 14th ACM-SIAM Symposimum on Discrete algorithms*, January 2003, pp. 761–770.

[2] A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998.

[3] D. Clark and J. Wroclawski. An approach to service allocation in the internet. internet draft, draft-clark-diff-svc-alloc-00.txt, 1997. (www-rp.lip6.fr/pub/documentaire/ietf_QoS/draft/draft-clark-diff-svc-alloc-00.txt)

[4] E. L. Hahne, A. Kesselman, Y. Mansour, Competitive Buffer Management for Shared-Memory Switches, In *Proc. of the 13th ACM Symposium on Parallel Algorithms and Architectures*, 2001, pp. 53–58.

[5] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, M. Sviridenko, Buffer Overflow Management in QoS Switches, In *Proc. of the 33rd ACM Symposium on Theory of Computing*, 2001, pp. 520–529.

[6] A. Kesselman, Y. Mansour, Loss Bounded Analysis for Differentiated Services, In *Proc. of 12th ACM-SIAM Symposimum on Discrete algorithms*, January 2001, pp. 591–600.

[7] A. Kesselman and Y. Mansour, Harmonic Buffer Management Policy for Shared Memory Switches, In *Proc. of INFOCOM 2002.*

[8] Z. Lotker, B. Patt-Shamir, Nearly Optimal FIFO Buffer Management for DiffServ, In *proc. of the 21st ACM Symposium on Principles of Distributed Computing* 2002, pp. 691–700.

[9] Y. Mansour, B. Patt-Shamir, O. Lapid, Optimal Smoothing Schedules for Real-time Streams, In Proc. of the 19th ACM Symposium on Principles of Distributed Computing, 2000, pp. 21-29.

[10] M. May, Jean-Chrysostome, A. Jean-Marie, and C. Diot. Simple performance models of differentiated services for the internet. In *proc. of INFOCOM 1998*, pp. 1385-1394.

[11] K. Nichols, V.Jacobson, and L. Zhang. A two-bit diffrentiated services architecture for the internet. Internet draft, RFC 2638, *IETF*, 1997. (www.apps.ieft.org/rfc/rfc2638.html)

[12] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services network: The single node case. *IEEE/ACM trans. on networking*, 3(1):344–357, 1993.

[13] S. Sahu, D. Towsley, and J. Kurose. A quantitive study of differentiated services for the internet. manuscript, 1999.

[14] D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.

[15] J. S. Turner. New directions in communications. *IEEE Communications Magazine*, 24(10), pp. 8–15, 1986.

A Simulating an Arbitrary Policy using a FIFO queue

In this section we show how using a single FIFO buffer we can simulate any non-preemptive queue policy in order to achieve the same benefit. The simulation policy will accept exactly the same packets as the simulated one, but the order in which the packets are sent may differ. However observe that this may introduce at most an additive difference of at most $B \cdot \alpha$ in the sum of benefits of the packets sent by the two policies. Other than being non-preemptive and using the same total buffer space B, the simulated queue policy can be arbitrary. It may use more than a single queue, and each queue it uses is not restricted to be FIFO.

The idea is rather simple and similar to Generalized Processor Sharing [12]. We simulate the given policy Γ on the event sequence as follows. We maintain at each time the state that the policy Γ would have been in, given the event sequence. This is done by running in the background the simulated policy, performing its actions both for packet arrivals and send events. This allows us to deduce, for each packet that arrives, whether the policy Γ would have accepted or rejected the incoming packet. Our simulation policy performs the same action (i.e., either accept or reject) that the policy Γ would have done. For a send event, the simulating policy sends the first packet in our queue (maintaining the FIFO property of the buffer) but, as described above, we also update the state of Γ, according to which packet Γ sends.

The main observation is that we can see by induction on the events, that at any time the simulating policy has the same number of packets in the buffer as the policy Γ. Therefore it can always perform the acceptance (or rejection) of arriving packets exactly as Γ. The two policies may however have a different set of packet in the buffer at a given time.

B Fixed Partition Policy vs. Flexible Partition Policy

Although one might be tempted to assume that the Flexible Partition Policy is always better than the Fixed Partition Policy, the following claim provides a counted example.

Claim B.1 *There exists event sequences Λ_1 and Λ_2 such that on Λ_1 the Flexible Partition Policy has a larger benefit than the Fixed Partition Policy and on Λ_2 the Fixed Partition Policy has a larger benefit than the Flexible Partition Policy. (Using $x = 1/2$ in both.)*

Proof. Let Λ_1 be the sequence of B high priority packets arriving before any send event is given. The Flexible Partition Policy accepts all of them while the Fixed Partition Policy accepts only $B/2$ of them.

The surprising part is Λ_2. The event sequence Λ_2 starts with B high priority packets followed by $B/2$ low priority packets; after $B/2$ send events $B/2$ low priority packets arrive followed by $B/2$ high priority packets, and after another $B/2$ send events a final set of $B/2$ low priority packets arrive.

The Flexible Partition Policy accepts at the beginning B high priority packets, rejecting the $B/2$ low priority packets. It then sends $B/2$ high priority packets, which implies that it has in the buffer

$B/2$ high priority packets. At this time new packets arrive. It accepts $B/2$ low priority packets and rejects the remaining $B/2$ high priority packets. During the next $B/2$ time steps it sends $B/2$ high priority packets, and hence it has in its buffer $B/2$ low priority packets. Therefore the last set of low priority packets is completely rejected. Overall, the Flexible Partition Policy accepts B high priority packets and $B/2$ low priority packets.

The Fixed Partition Policy accepts at the beginning $B/2$ high priority packets, and the $B/2$ low priority packets (rejecting $B/2$ of the high priority packets). It then sends $B/2$ high priority packets, which implies that it has in the buffer $B/2$ low priority packets. At this time new packets arrive. It rejects $B/2$ low priority packets and accepts the $B/2$ high priority packets. During the next $B/2$ time steps it sends $B/2$ low priority packets, and hence it has in its buffer $B/2$ high priority packets. Therefore, the last set of low priority packets is accepted. Overall, the Fixed Partition Policy accepts B high priority packets and B low priority packets. ■