Competitive Queue Policies for Differentiated Services

William A. Aiello, AT&T Research
Yishay Mansour, AT&T Research and Tel-Aviv University
S. Rajagopolan, Telcordia Technologies
Adi Rosen, University of Toronto

Abstract—**We consider the setting of a network providing differentiated services. As is often the case in differentiated services, we assume the packets are tagged as either being high or low priority packets. 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 delivers.

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, and in most cases our bounds are tight. We believe that competitive analysis gives important insight to the performance of these simple queuing policies.

I. Introduction

It is widely agreed that future network would be forced to support Quality of Service (QoS) in order to answer user expectation. Today Internet with its single best effort class of service does not provide a satisfactory solution to many users. Many users are willing to pay more, if in return they receive a guaranteed performance from the network.

This vision is by no means new in networking, and has been around for well over a decade [8]. 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 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 seems equally valid. The first solution is to force the incoming traffic to conform to the committed parameters by regulating the traffic at the entrance to the network. The other approach is to label the traffic as "in" and "out" while the data labeled "in" has the desired properties (and therefore it is within the bandwidth allocated) and the data labeled "out" is an excess load (and therefore outside the allocated bandwidth). In the second approach we allow the network element (routers) to handle the two types of traffic differently.

Recently there is wide interest in 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 [4] and Assured service [2]. 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 [3], [6].

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 interpretation might be 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 one we are basically optimizing the total traffic, ignoring the various priorities.

We concentrate on scheduling in a single FIFO buffer. We assume that packets are delivered (emptied) from the buffer one per unit time. During a unit time several packets of each type may arrive. A queue policy decides for each packet, one at a time, whether to insert the packet into the FIFO queue or drop the packet. Once the policy decides to accept a packet, it cannot latter preempt it from the 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 queue delivers. In this model we study a few simple 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, there are arrival sequences for which the benefit of any queue policy will 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 model) 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 and/or verify in many cases (especially for QoS, where reliable data is hard to find).

In this paper we use an approach known as *competitive analysis* [7], which is model independent. As such, the bounds derived cannot be expected to be as tight as those for a specific model. However, no assumptions about the arrival sequence are required to apply the bounds. This approach should be viewed as a compliment to the queuing theory approach in gaining insight on the behavior of queuing policies in a variety of real world situations.

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 input sequence in advance. The competitive ratio is the minimum, over all input 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 in case that an optimal offline algorithm would have low benefit for the same arrival sequence. The online queue policy is only penalized when there is an input 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 one. Our aim is to find queue policies with the largest competitive ratio. (An excellent introduction to competitive analysis is found in [1].)

In this work we chose five simple queue policies and study their competitive ratio for various values of α. For each policy we try to lower and upper bound the competitive ratio as a function of α. We perform a comparison of the results for various regimes of α.

The paper is organized as follows. In Section II we give a formal description of our model. In Section III we outline in detail each of the five policies we consider. Section IV gives a detailed summary of our results. The performance of the various policies is given in Section V, while the impossibility results are detailed in Section VI. In Section VII we outline how to efficiently compute an optimal schedule, given the entire arrival sequence.

II. Model

In this work we concentrate on a single FIFO queue. The FIFO queue can hold B packets, and once a packet enters the queue it can not 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 (although the structure of output stream has no impact on our results). 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.

An input sequence Λ is a sequence of 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). One can think of the send events as synchronized with time, while a large number of packets arriving between two consecutive send events can be viewed as a burst of arrivals.

A queue policy specifies for each incoming packet whether to accept it to the queue, or to reject it. The benefit of a queue policy on a given arrival sequence is the sum of the benefits of the packets it receives. 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 accepted and m is the number of high priority packets accepted.[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 input sequence. The competitive ratio is the minimum over all input sequences of the ratio between the benefit of a given online policy and the benefit of an optimal offline policy. More formally, for an input sequence Λ let the benefit of an online policy π be denoted by $\pi(\Lambda)$, and the optimal benefit be denoted by $opt(\Lambda)$. The competitive ratio of policy π is $\min_{\Lambda}\{\pi(\Lambda)/opt(\Lambda)\}$.

III. Policies Definition

In this work we choose to concentrate on simple 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 x, and it limits the number of low priority packets 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*. As with the Fixed Partition Policy it has a threshold parameter x and limits the number of low priority packets to be at most xB. However, unlike the Fixed Partition Policy, it always accepts high priority packets. Note that at some points in some arrival sequences the buffer may contain more than $(1-x)B$ high priority packets and will not be able to accept up to xB low priority packets even if there is a burst of arrivals of low priority packets before the next send. Not surprisingly, we will see that the Flexible Partition Policy has a better competitive ratio than the Fixed Partition Policy. [2]

The fourth policy is 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 the ratio is below a threshold parameter x. More formally, let k and m be the number of low 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 V) we accept a low priority packet if after accepting it the number of

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

[2] In Appendix B we show that it is not true that for every input sequence the Flexible Partition Policy has benefit at least that of the Fixed Partition Policy.

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.

The last policy we consider is the *Round-Robin Policy*. The simplest way to think about the Round Robin Policy is by maintaining two queues each of size $B/2$. (Later we show how we translate this to a single FIFO queue policy.) One queue is dedicated for low priority packets while the other for high priority packets. During the send events it 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. [3]

Translating the Round Robin Policy to a single FIFO queue retains its simplicity. We have two variables `num-low` and `num-high`. We accept a low (high) priority packet if `num-low` (`num-high`) is less than $B/2$, and increment the 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. (More precisely, we are simulating Generalized Processor Sharing [5] rather than strict round robin.)

Our translation of the Round Robin Policy to a single FIFO queue can be generalized to a translation of an arbitrary non-preemptive policy. We can simulate any non-preemptive policy using a single FIFO queue and guarantee that the set of packets sent by the original policy and the simulated policy is identical. (However, the order of the packets delivered may be different.) See Appendix A for more details.

IV. Overview of our Results

In this section we give a detailed overview of our results, while we defer the proofs to the following sections. For this overview we will use three different regimes of the value of α. The first is the case of very large α, i.e., $\alpha \to \infty$. The second is of a moderate α, we chose $\alpha = 2$. The third is the case when the high priority and low priority have the same benefit, i.e. $\alpha = 1$. To make the comparison fair we allow each policy to adjust its parameter x to the value of α. The results are summarized in Figure 1.

A. 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.)

In this regime one can see that the Greedy Policy performs very poorly. This is not surprising since the Greedy Policy ignores the value of α, and treats all the packets equally. At the other extreme we have the Round Robin Policy and the Dynamic Flexible Policy. Both policies have an asymptotic competitive ratio of $1/2$. We also show that any online policy can have a competitive ratio of at most $1/2$, and therefore the two policies are optimal in this case.

For the Flexible Partition Policy with $x = \sqrt{2} - 1$ we show that it achieves a competitive ratio of $\sqrt{2} - 1$, which we show to be tight.

For the Fixed Partition Policy with $x = 1/2$ we show a competitive ratio of $1/4$. In this case we are not certain that this is the right competitive ratio even for the parameter $x = 1/2$. The impossibility result in this case is identical to the Flexible Partition Policy, and give a upper bound of $\sqrt{2} - 1$, for any choice of x.

B. Moderate Benefit Value.

We choose to look at the case when the high priority packets are more valuable than the low priority ones, but not overwhelmingly more valuable. One can set the parameter α in many ways, as a test case we chose the setting of $\alpha = 2$. In this case it seems fair that we allow each policy to readjust the parameter x, and 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 we can show for each of our policies an input sequence for which the competitive ratio it has is strictly less than $2/3$, for any setting of the parameter x.)

The best policy in this regime is the Dynamic Flexible Partition Policy, for which we can show that the competitive ratio is strictly better than $1/2$ (using $x = 3/4$). On the other hand we can show an impossibility result that states that for any value of x the competitive ratio is at most 0.62.

The performance of the Flexible Partition Policy does not improves when we consider $\alpha = 2$ and remains $\sqrt{2} - 1$. (It does improve for α less than 2.) The upper bound on the competitive ratio, for the value of $\alpha = 2$, coincides with the one for Dynamic Flexible Partition Policy, and is 0.62.

The Fixed Partition Policy cannot have a better upper bound than $1/2$. This is due to the fact that the partition of the buffer is fixed. If packets of only a single type arrive, the competitive ratio is at most $1/2$. The same difficulty occurs for the Round Robin Policy. Even if we consider an extension of the Round Robin Policy to have a parameter x, the fact that it induces a hard partition implies that the competitive ratio is at most $1/2$. For this reason the competitive ratio of the Round Robin Policy, remains $1/2$. For the Fixed Partition Policy our proofs do not improve the competitive ratio in the case $\alpha = 2$, and the best competitive ratio we can prove is $1/4$.

It is worth noting that in this regime the Greedy Policy also achieves a competitive ratio of $1/2$, which implies that in this case a competitive ratio of $1/2$ should not be considered impressive.

C. Low Benefit Value.

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

Not surprisingly, the Greedy Policy is optimal in this case,

[3] One can generalize this the Round Robin Policy using the threshold parameter x. However, as we will see, the competitive ratio is at most the minimum of x and $1 - x$ which is maximized at $x = 1/2$. Moreover, we show that for $x = 1/2$, a competitive ratio of $1/2$ is achievable.

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

Fig. 1. This figure gives the competitive ratios of the various policies we consider in three different regimes of α. 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.

since it is by definition maximizing the number of packets accepted, and ignores the value of α.

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.

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 in the case when an overwhelming majority of the arriving packets are from a single type, they are forced to drop packets when the preallocated space in the buffer for packets of that type is full, whereas the optimal policy will only drop packets when the entire buffer is full.

D. 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 try to perform a preallocation of the buffer space, have a problem in the case that the difference in prices is not significant.

The best policy, of the five we consider here, is 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 is very similar to the Dynamic Flexible Partition Policy, achieves similar performance, but its performance is lower in each of the three regimes. This might suggests 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. 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.

V. Policies Analysis

In this section we lower bound the benefit of the policies compared to an optimal offline policy.

A. Greedy Policy

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

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

Proof: Clearly the Greedy Policy maximizes the total number of packets accepted. This implies that the benefit of the policy is most a factor of α from optimal. ■

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 goes to infinity.

B. Fixed Partition Policy

In this subsection we analyze the Fixed Partition Policy and derive a bound on its competitive ratio. Notice that in this policy there is limited interaction between the low and high priority packets. Therefore, at the beginning, we analyze the competitive ratio for each setting (high or low priority) separately, and then combine the results.

At a high level the analysis is done as follows. We consider a Fixed Partition Policy with $x = 1/2$. We analyze separately its performance on the high and low priority packets. Consider the low value packets. Clearly the number of low priority packets is bounded by the number of low priority packets accepted by a greedy policy that accepts only low priority packets, uses a buffer of size B, and (possibly) sends at each step a packet. We introduce an intermediate model, which is a greedy policy that accepts only low packets, but it is limited to a buffer of size $B/2$ and send half a packet at each step. We show that the latter policy sends at least half the packets of the greedy. The greedy policy with buffer $B/2$ has already the same allocation of buffer space to the low priority packets as the Fixed Partition Policy. However, it has the benefit that it empties the buffer at a constant rate (of half) while the Fixed Partition guarantees only an average rate (of half). We show that this difference can decrease the number of packets by at most a half. Combining the two parts we get that the Fixed Partition Policy sends at least a $1/2 * 1/2 = 1/4$ of the packets that the greedy policy send. This establishes the competitive ratio.

The analysis is done first on a single type of packet, and therefore the only interesting parameter is the number of packets accepted. The analysis is done in two phases. First we compare the number of packets sent by a greedy policy that uses a buffer of size B and sends a complete packet every time unit (called Greedy(B,1) policy) to a policy that uses a buffer of size xB that sends an x fraction of a packet every time unit (called Greedy(B,x) policy). (The two policies are used only as part of the analysis and are not proposed as actual policies.) The next lemma compares the number of packets accepted by Greedy(B,1) and Greedy(B,x).

Lemma 2: At any time t we have that the number of packets in the buffer of the Greedy(B,1) policy is at most the number of packets in the buffer of the Greedy(B,x) policy times $1/x$.

Proof: The proof is by induction on time. First let us examine the send events. If the Greedy(B,1) buffer is not empty then by induction the Greedy(B,x) buffer is not empty as well. A send event decreases the former buffer by one and decreases the latter by x, thus maintaining the invariant. If the Greedy(B,1) buffer is empty the invariant is maintained trivially.

Now consider the arrival of a packet. By induction, if the buffer of Greedy(B,1) is full then so is the buffer of Greedy(B,x). Thus, if Greedy(B,1) rejects a packet, so does Greedy(B,x). The invariant is trivially maintained in this case. If Greedy(B,1) accepts there are two cases. In the case that Greedy(B,x) rejects it does so because it is full and thus the invariant is true trivially. In the case that Greedy(B,x) also accepts the invariant is maintained since $1 \leq 1/x$. ■

What we can deduce from the above lemma is that whenever the buffer of the Greedy(B,1) policy is non-empty then the buffer of the Greedy(B,x) policy is not empty. This implies the following corollary.

Corollary 3: The number of packets sent by the Greedy(B,1) policy is at most the number of packets sent by the Greedy(B,x) policy times $1/x$.

We are not done yet, since the in the Fixed Partition Policy only one type of packet is sent each time unit. Define the G-LOW(B,x) Policy (G-HIGH(B,x) Policy) to be a policy on sequences of high and low priority packets that accepts only low (high) priority packets and does so according to the Greedy(B,x) Policy on those low (high) priority packets.

Let the FP-LOW(B,1/2) Policy (FP-HIGH(B,1/2) Policy) have a single queue of size $B/2$. As with G-LOW(B,1/2) Policy (G-HIGH(B,1/2) Policy), it receives high and low priority packets but accepts only low (high) priority packets. It accepts and sends exactly the same low (high) priority packets (and at the same time) as the Fixed Partition Policy with $x = 1/2$.

Lemma 4: The number of packets sent by the G-LOW(B,1/2) Policy and the G-HIGH(B,1/2) Policy is bounded by two times the number of packets sent by the FP-LOW(B,1/2) Policy and FP-HIGH(B,1/2) Policy, respectively..

Proof: We first add an intermediate policy, FP-LOW'(B,1/2). This policy behaves the same as the FP-LOW(B,1/2) policy, except that whenever it has a full buffer it enters a *block state* for B time units. In the block state it rejects any incoming packet and sends $1/2$ a packet each time step. Note that at the end of the block state we are assured that the buffer of FP-LOW'(B,1/2) is empty. Clearly, the benefit of the FP-LOW'(B,1/2) policy is no more then the benefit of the FP-LOW(B,1/2) policy.

Consider the following marking algorithm for the FP-LOW'(B,1/2). Each time the policy enters the block state it marks all the packets in the buffer. Since when the policy exits the interval of the block state the buffer is empty, it is clear that each packet is marked at most once.

It is easy to see by induction that the number of packets in the G-LOW(B,1/2) buffer is at least the number of packets in the FP-LOW'(B,1/2) buffer. Therefore, at the beginning of a blocking period for FP-LOW'(B,1/2), the G-LOW(B,1/2) buffer is full. It follows that the number of packets accepted by G-LOW(B,1/2) in the next B time units is at most $B/2$.

During each interval of block state we mark B/2 packets in the FP-LOW'(B,1/2) buffer, and each packet is marked at most once. This implies that the number of packets accepted by the G-LOW(B,1/2) policy, during all the intervals of block state is bounded by the number of marked packets. The number of marked packets is bounded by the total number of packets accepted by FP-LOW'(B,1/2).

The total number of packets that G-LOW(B,1/2) accepts is the sum of the number of packets it accepts in the intervals of block state, and the number it accepts outside the block states. The number of packets it accepts in blocked states is at most the number of packet FP-LOW'(B,1/2) accepts. The number of packets it accepts outside the intervals of blocked states is at most the number of packets FP-LOW'(B,1/2) accepts. Therefore the number of packets G-LOW(B,1/2) accepts is at most twice the number of packets sent by FP-LOW(B,1/2).

The arguement for G-HIGH(B,1/2) and FP-HIGH(B,1/2) is similar. ■

Combining the two results we derive a bound on the competitive ratio.

Theorem 5: The competitive ratio of the Fixed Partition Policy with $x = 1/2$ is at least $1/4$.

Proof: Consider first the (sub)sequence of low priority packets. We compare the G-LOW(B,1) policy, the optimal offline policy, to the G-LOW(B,1/2) policy. Then we compare the G-LOW(B,1/2) policy to the FP-LOW(B,1/2) policy. The competitive ratio is at least the product of the two ratios.

By Corollary 3 the G-LOW(B,1/2) policy accepts at least a half of the low priority packets that an optimal offline accepts. By Lemma 4 the FP-LOW(B,1/2) policy accepts at least a half of the low priority packets of the G-LOW(B,1/2) policy. Therefore the number of low priority packets accepted by the Fixed Priority Policy is at least $1/4$ of the low priority packets accepted by an optimal offline policy.

Similarly, the number of high priority packets accepted by the Fixed Priority Policy is at least $1/4$ of the high priority packets accepted by an optimal offline policy. Therefore, the competitive ratio of the Fixed Priority Policy is at least $1/4$. ■

C. Round Robin Policy

Using the basic lemma shown for the Fixed Partition Policy, we show that the competitive ratio of the Round-Robin Policy is at least $1/2$.

Theorem 6: The competitive ratio of the Round-Robin Policy is at least $1/2$.

Proof: As in the analysis of the Fixed Partition Policy, we first consider the (sub)sequence of low priority packets. Observer that the number of low priority packets accepted by the Round Robin Policy is at least the number of packets accepted by the G-LOW(B,1/2) policy. This in turn is at least $1/2$ the number of packets accepted by G-LOW(B,1), as per Corollary 3. Likewise, the number of high priority packets accepted by the Round Robin Policy is at least $1/2$ the number of high priority packets accepts by G-HIGH(B,1). Therefore the competitive ratio of the Round-Robin Policy is at least $1/2$. ■

D. Dynamic Flexible Partition Policy

In this section we study the competitive ratio of the Dynamic Flexible Partition Policy. 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 $1/2$.

Our proof uses a matching between the high and 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 matched packets that the Dynamic Flexible Partition Policy accepts. In addition, the total number of packets accepted by an optimal offline is at most twice the total number of packets accepted by the Dynamic Flexible Partition Policy. Those two facts enables us to derive a competitive ratio of a half.

We define a *matching* as follows: when a high priority packet arrives it is matched to the lowest unmatched low priority packet in the buffer (if such a packet exists).

The following lemma relates the number of unmatched packets to the number of free slots in the buffer for the Dynamic Flexible Partition Policy.

Lemma 7: In the Dynamic Flexible Partition Policy at any time we have that $l \leq f$, where l is the number of unmatched low priority packets in the queue and f is the number of free slots.

Proof: By definition of the Dynamic Flexible Partition Policy with $x = 1/2$, we accept a low priority packet if after we accept it, we have more free slots than low priority packets in the queue. Therefore the claim holds when we accept a low priority packet.

When we accept a high priority packet we have one less free space, but also one less unmatched low priority packets 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 or low priority) the number of free slots increases by 1, and the number of unmatched low priority packets can not increase (it can either decrease by 1 or stay unchanged). ■

An immediate corollary of the above lemma is the following.

Corollary 8: In the Dynamic Flexible Partition Policy when the buffer is full all the low priority packets are matched.

We call a packet *good* if it is: (1) a high priority packet that was accepted, and (2) a matched low priority packet.

Given the online buffer, we define the *good prefix* as the number of consecutive good packets from the start of the buffer.

In order to bound the number of high priority packets the offline accepts, we consider the G-HIGH(B,1) policy, that accepts only high priority packets. Clearly, for any input sequence, the number of high priority packets G-HIGH(B,1) accepts bounds the number of high priority packets any other policy can accept. The next lemma compares the number of packets G-HIGH(B,1) accepts to that of our online policies.

Lemma 9: In the Dynamic Flexible Partition Policy, for any input sequence Λ, at any time the number of packets in the good prefix is greater or equal to the number of packets in G-HIGH(B,1).

Proof: The proof is by induction on time. 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 8 all the packets in the buffer are matched, and hence the good prefix is the entire buffer, i.e., 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 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, which is increased by one. 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.

During a send event, if the G-HIGH(B,1) buffer is empty the inductive claim holds. Otherwise, we can use the inductive claim and argue that the good prefix is also non-zero. In this case both are decreased by one. ■

Corollary 10: In the Dynamic Flexible Partition Policy for any input sequence Λ, the number of packets G-HIGH(B,1) sends is bounded by the number of good packets.

Proof: Each time G-HIGH(B,1) sends a packet the Dynamic Flexible Partition Policy sends a good packet. ■

The next claim bounds the total number of packets accepted by our online policies.

Claim 11: For any input sequence Λ, the number of packets (high or low) any schedule sends is at most twice the number of packets sent by the Dynamic Flexible Partition Policy.

Proof: The Dynamic Flexible Partition Policy rejects a low priority packet only if the buffer is at least half full. ■

Theorem 12: The competitive ratio of Dynamic Flexible Partition Policy is at least $1/2$.

Proof: Consider a fixed sequence of packet arrivals. 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.

We show that $k_1 + \alpha m_1 \leq 2[k_2 + \alpha m_2]$, which establishes the theorem. By Claim 11,

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

Let g_2 be the number of good packets in the Dynamic Flexible Partition Policy. The matching guarantees that at least half of the good packets are high priority packets. Using Corollary 10 we have,

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

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

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

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. (Proof omitted.)

Theorem 13: The Dynamic Flexible Partition Policy, with $x = 3/4$, has a competitive ratio of at least $15/28$ for $\alpha = 2$.

E. Flexible Partition Policy

In this section we study the competitive ratio of the Flexible Partition Policy, and show that its competitive ratio is at least $\sqrt{2} - 1 \approx 0.41$. The proof techniques is similar in spirit to that of the Dynamic Flexible Partition Policy.

Our proof uses an "accounting" technique to relate the high and low priority packets. Each high priority packet "gives" each low priority packet, which is at some time in the buffer with it, a weight of $1/((1-x)B)$. Each low priority packet has a weight which is the sum of the weights it receives from high priority packet, i.e. the number of high priority packets that at some point of time are in the buffer with it times $1/((1-x)B)$. We call a low priority packet matched if its weight is at least 1. First we bound the number of matched low priority packets.

Lemma 14: In Flexible Partition Policy, for any input sequence, the number of matched low priority packet is at most $2x/(1-x)$ times the number of high priority packets accepted.

Proof: We show that for each accepted high priority packet the sum of the weights it gives the low priority packets is at most $2x/(1-x)$. When a high priority packet h is accepted we have in the buffer at most xB low priority packets. When h is sent, we have in the buffer at most xB (different) low priority packets.

Note that any low priority packet that at some time is in the buffer with h, has either to be in the buffer when h is accepted or when h is sent. Therefore the number of low priority packets that are in the buffer with h is at most $2xB$. Since each gets a weight of $1/((1-x)B)$, the total weight h gives is at most $2x/(1-x)$.

Since any matched low priority packet has a weight of 1, the number of matched low priority packets is at most $2x/(1-x)$ times the number of high priority packets. ■

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 15: In the Flexible Partition Policy when the buffer is full then each low priority packet in the buffer is matched, i.e. has weight at least 1.

Proof: When the buffer is full there are at least $(1-x)B$ high priority packets in the buffer. Each low priority packet in the buffer gets a weight of $1/((1-x)B)$ from each high priority packet, and therefore has a weight of at least 1. ■

We call a packet *good* if it is: (1) a high priority packet that was accepted, and (2) a matched low priority packet.

In order to bound the number of high priority packets the offline accepts, we consider the G-HIGH(B,1) policy, that accepts only high priority packets.

Lemma 16: For any input sequence Λ, the number of packets G-HIGH(B,1) sends is bounded by the number of good packets Flexible Partition Policy sends.

Proof: After running the Flexible Partition Policy we can mark all the low priority packets which are matched, and thus fix the set of good packets.

Consider a schedule that would maximize the number of good packets accepted. The Flexible Partition Policy accepts greedily all the good packets (by Lemma 15 each time it rejects a good packet all the packets in the buffer are good), therefore it maximizes the number of good packets accepted. This implies that the number of good packets sent by Flexible Partition Policy is greater or equal to the number of good packet sent by any other policy, including G-HIGH(B,1) (which is restricted to high priority packets only). ■

The next claim bounds the total number of packets accepted by the Flexible Partition Policy.

Lemma 17: For any input sequence Λ, the number of packets (high or low) any schedule sends is at most $1/x$ times the number the packets sent by Flexible Partition Policy.

Proof: When a low priority packet is rejected there are at least xB packets in the buffer. ■

Theorem 18: The competitive ratio of Flexible Partition Policy is at least $\sqrt{2} - 1$.

Proof: Consider a fixed sequence of packet arrivals. Let k_1 and m_1 be the number of low and high priority packets an optimal offline accepts, respectively. Let k_3 and m_3 be the number of low and high priority packets the Flexible Partition Policy accepts, respectively.

By Lemma 17,

$$k_1 + m_1 \leq (1/x)(k_3 + m_3). \tag{3}$$

Let g_3 be the number of good packets in the Flexible Partition Policy. By Lemma 16 we have that $m_1 \leq g_3$. By Lemma 14 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,

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

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

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

By setting $x = \sqrt{2} - 1$ we have that the competitive ratio is at least $1/A = \sqrt{2} - 1$. ■

VI. 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 19: The competitive ratio of the Greedy Policy is at most $1/\alpha$.

Proof: Consider the input sequence that includes B low priority packets followed by B high priority packets, in the same unit time. 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 and high priority packets.

Theorem 20: For any value of α and any parameter x, the Round Robin Policy and the Fixed Partition Policy have competitive ratios at most $1/2$.

Proof: Consider the Fixed Partition Policy on the input sequence of exactly B low priority packets arriving simultaneously. The Fixed Partition Policy accepts 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 input 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 analysis for the Round Robin Policy is identical. ■

The proofs of the other impossibility results are slightly more complicated; however, the general structure is similar. 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 which sequence it is. This early decision of the online is the source of its sub-optimal performance. On the other hand, an optimal offline simply considers each sequence separately.

Theorem 21: Any online policy has a competitive ratio of at most $1/2 + 1/(4\alpha - 2)$.

Proof: We consider two scenarios. In the first scenario B low priority packets arrive. The online policy accepts a fraction z of them, while an optimal offline policy accepts all of them.

The second scenario starts like the first scenario with B low priority packets. Again, the online policy accepts a 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 arrive at the same unit time. The online policy fills the buffer with high priority packets, i.e. accepts $(1-z)B$ high priority packets. An optimal offline policy accepts all the high priority packets.

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

$$\min\{\frac{zB}{B}, \frac{zB + \alpha(1-z)B}{\alpha B}\} \leq \frac{\alpha}{2\alpha - 1} = \frac{1}{2} + \frac{1}{4\alpha - 2},$$

which bounds the performance for 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.

Lemma 22: The Flexible Partition Policy with a parameter x has a competitive ratio of at most $\min\{x, \frac{(1-x)\alpha + 2x}{\alpha(1+x)}\}$.

Proof: Again, we consider two scenarios. In the first scenario B low priority packets arrive. By definition of the Flexible Partition Policy it accepts a fraction x of them. An optimal offline policy accepts all of them, and therefore the competitive ratio of the Flexible Partition Policy is at most x.

The second scenario starts like the first scenario with B low priority packets. They are immediately followed by B high priority packets, in the same unit time. After xB time units (i.e., xB sends) another xB low priority packets arrive followed by xB high priority packets, in the same unit time.

The Flexible Partition Policy accepts xB of the first low priority packets, then accepts $(1-x)B$ high priority packets, and finally accepts xB low priority packets. An optimal offline policy accepts $(1+x)B$ high priority packets. The competitive ratio in this case is $(\alpha(1-x) + 2x)/(\alpha(1+x))$.

The lemma follows since the competitive ratio is the minimum over all input sequences. ■

Optimizing x for each value of α we have the following bound.

Corollary 23: The competitive ratio of the Flexible Partition Policy is at most $\sqrt{2 - (2/\alpha) + (1/\alpha)^2} - (1 - (1/\alpha))$.

Note that the above proof and resulting bound applies also to the Fixed Partition Policy. This yield the corollary below.

Corollary 24: The competitive ratio of the Fixed Partition Policy is at most $\min\{1/2, \sqrt{2 - (2/\alpha) + (1/\alpha)^2} - (1 - (1/\alpha))\}$.

Using a similar technique we can derive an impossibility result for the Dynamic Flexible Partition Policy.

Lemma 25: The Dynamic Flexible Partition Policy with a parameter x has a competitive ratio of at most $\min\{x, (\alpha(1-x) + x)/(\alpha + x(1-x))\}$.

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

The second scenario starts like the first scenario with B low priority packets. They are immediately followed by B high priority packets, in the same unit time. After $x(1-x)B$ time units another $x(1-x)B$ low priority packet arrive.

The Dynamic Flexible Partition Policy accepts xB of the first low priority packets, then accepts $(1-x)B$ high priority packets, and finally rejects all the remaining low priority packets (since the ratio between the number of low priority packets to the number of low priority packets plus empty slots is x). An optimal offline policy accepts B high priority packets and $x(1-x)B$ low priority packets. Therefore the competitive ratio is at most $(\alpha(1-x) + x)/(\alpha + x(1-x))$. ■

VII. Optimal offline schedule

In this section we describe OPTIMAL, an optimal offline policy. Given an input sequence Λ, OPTIMAL works in two phases. In the first phase it find a schedule that includes only high priority packets of Λ. (In this phase OPTIMAL accepts a high priority packet if when it arrives the buffer is not full.) Note that any greedy policy that accepts only high priority packets delivers the same number of packets. This guarantees that the number of high priority packets delivered 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 packet to be rejected.

Theorem 26: For any input 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 in $OPTIMAL(\Lambda)$. 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 latter packet that is accepted in s and now has to be rejected. Let s' be the new schedule. The schedule s' 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' eventual 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 schedule 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^* in schedule 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. ∎

References

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

[2] D. Clark and J. Wroclawski. An approach to service allocation in the internet. In *internet draft*, 1997.

[3] M. May, Jean-Chrysostome, A. Jean-Marie, and C. Diot. Simple performance models of differentiated services for the internet. In *INFOCOMM*, 1998.

[4] K. Nichols, V.Jacobson, and L. Zhang. A two-bit diffrentiated services architechture for the internet. In *internet draft*, 1997.

[5] 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.

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

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

[8] J. S. Turner. New directions in communications. *IEEE Communications Magazine*, 10, 1986.

Appendix

I. Simulating 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. Other than being non-preemptive, the 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 [5]. We simulate the given policy Γ on the input sequence. We compute the state that the policy Γ would have been in, given the input sequence. This allows us to deduce whether the policy Γ would have accepted or rejected each incoming packet. Our simulation policy performs the same action that the policy Γ would have done.

When there is a send event the simulating policy send the first packet in our queue (maintaining the FIFO property of the buffer), but in addition we update the state of Γ, as if Γ sent the packet it schedules to send.

The main observation is that at each point of time the simulating policy has the same number of packets in the buffer as the policy Γ. The difference is that the two policies might have a different set of packet currently in the buffer. However this is not a real issue, since the policy Γ is simulated, and the actions (rejecting or accepting packets) are taken according to Γ's state. The crux of the argument is that for the benefit the order in which the packet depart the buffer is not important. The only important fact is that once a packet has been accepted by the policy Γ it is eventually be sent (i.e. there is no preemption). Therefore if the simulating policy can accepts the same set of packets, its benefit is identical to Γ.

II. Fixed versus Flexible Partition Policy

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

Claim 27: There exists input sequences Λ_1 and Λ_2 such that on Λ_1 the Flexible Partition Policy has a larger benefit than the Fix Partition Policy and on the input sequence Λ_2 the Fix 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 simultaneously. The Flexible partition accepts all of them while the Fix partition accepts only $B/2$ of them.

The surprising part is Λ_2. The input sequence Λ_2 starts with B high priority packets followed by $B/2$ low priority packets, after $B/2$ time units $B/2$ low priority packets arrive followed by $B/2$ high priority packets, and after another $B/2$ time units 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 units it send $B/2$ high priority packets, and hence its buffer includes $B/2$ low priority packets. Therefore the last set of low priority packets are rejected. Overall the Flexible Partition Policy accepts B high priority packets and $B/2$ low priority packets.

The Fix 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 units it send $B/2$ low priority packets, and hence its buffer includes $B/2$ high priority packets. Therefore the last set of low priority packets is accepted. Overall the Fix Partition Policy accepts B high priority packets and B low priority packets. $\blacksquare$

It is worth stressing here that the competative ratio does not provide us information about individual input sequences. While one policy may have a better competitive ratio than another policy, it does *not* imply it has a better performance on each input sequence. It does not even imply it has a better performance on most input sequences. (However it does imply that it has better performance on some input sequence which can be viewed as "hard".)