Competitive Queueing Policies for QoS Switches

Nir Andelman* Yishay Mansour† An Zhu‡

Abstract

We consider packet scheduling in a network providing differentiated services, where each packet is assigned a value. We study various queueing models for supporting QoS (Quality of Service). In the *nonpreemptive* model, packets accepted to the queue will be transmitted eventually and cannot be dropped. The *FIFO preemptive* model allows packets accepted to the queue to be preempted (dropped) prior to their departure, while ensuring that transmitted packets are sent in the order of arrival. In the *bounded delay* model, packets must be transmitted before a certain deadline, otherwise it is lost (while transmission ordering is allowed to be arbitrary). In all models the goal of the buffer policy is to maximize the total value of the accepted packets.

Let α be the ratio between the maximal and minimal value. For the non-preemptive model we derive a $\Theta(\log \alpha)$ competitive ratio, both exhibiting a buffer policy and a general lower bound. For the interesting case of two distinct values, we give an $\frac{2\alpha-1}{\alpha}$ competitive buffer policy, which exactly matches the lower bound. We also analyze a RED-like policy and derive its competitive ratio, which is approximately $\frac{2\alpha-0.5}{\alpha}$ for two values and $\Theta(\log \alpha)$ for multiple values. In addition we improve the previous known lower and upper bounds of the Fixed Partition and Flexible Partition policies.

For the FIFO preemptive model, we improve the general lower bound and show a tight bound for the special case of queue size 2. We prove that the bounded delay model with uniform delay 2 is equivalent to a modified FIFO preemptive model with queue size 2. We then give improved upper and lower bounds on the 2-uniform bounded delay model. We also give lower bound for the 2-variable bounded delay model, which matches the previously known upper bound.

1 Introduction

Currently, the Internet infrastructure provides "best effort" service for all traffic streams. The uncertainty of the actual performance is not satisfactory for many applications. The widely foreseen next-generation networks will provide guaranteed services to meet various user demands. This gives rise to the recent interest in the Quality of Service (QoS) feature.

This vision has been around the networking community for more than a decade [15]. For instance, ATM networks serve as an example of a unified architecture that supports a diverse set of service classes. Of late, there has been termendous interest in IP in providing *differentiated services* via QoS guarantees. The basic methodology of QoS is rather intuitive — commit resources to each admitted connection. Thus, the network is capable of providing different users with different classes of service. In particular, a contract between users and service providers ensures that the network maintains the performance guarantees provided the users stick to their commitments about traffic generation.

However, due to a variety of reasons, the incoming traffic patterns may not coincide with that specified in the service contract. A typical example is that the traffic from the user does not conform to the patterns defined in the contract. The difficult situation is when the traffic exceeds the allocated bandwidth at some point. Another equally serious problem is that by guaranteeing the worst-case performance, the QoS network might not be efficient due to its conservative policy, as network traffic tends to be bursty. Recognizing this phenomenon, most modern QoS networks allow some "overbooking," employing the policy popularly known as *statistical multiplexing* [8]. In either case, QoS networks must resolve the unavoidable issue of overloading. This paper analyzes queueing policies under overloading using competitive analysis.

In the past few years the networking community has had an increasing interest in QoS networks [6, 12, 13, 14]. A major new paradigm suggested is that of assured service [5]. This service has a loose guarantee in which traffic conforming to the specified pattern is much less likely to be dropped in the network. This approach leads to two types of packets in the system: those of high priority (conformed traffic) and those of low priority (unconformed traffic).

We can now abstract the problem as follows. We assign a value to each packet: value 1 for the low priority

*School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel. E-mail : andelman@cs.tau.ac.il.

†School of Computer Science, Tel-Aviv University, Tel-Aviv, Israel. E-mail : mansour@cs.tau.ac.il.

‡Department of Computer Science, Stanford University, Stanford, CA 94305. Supported by a GRPW fellowship from Bell Labs, Lucent Technologies. E-mail : anzhu@cs.stanford.edu.

packets, and value $\alpha > 1$ for the high priority packets. This is called the *2-value model*. For differing network requirements, we can adjust α to achieve the desired performance guarantee. Later we will also consider the extension where packets take on arbitrary values in the range $[1, \alpha]$. We assume that the queue can hold a maximum of B packets. Packets can arrive at any time, but are sent out at integer times only. The goal is to maximize the total value of the packets transmitted. In terms of competitive analysis [4], we compare the total value of the packets transmitted by an online algorithm to the total value of the optimal offline algorithm. We say that an online algorithm has a competitive ratio of β, if for any input sequence, the total value transmitted is at least a $\frac{1}{\beta}$ fraction of the offline optimal.

We concentrate on three different queueing policies. The *nonpreemptive policy* transmits all packets admitted into the queue; observe, under this policy, the queue can easily maintain a FIFO order.

The *FIFO preemptive policy* is allowed to drop packets already admitted to the queue. The *bounded delay model*, on the other hand, transmits packets in any order, but each packet must be transmitted before a fixed deadline or else the packet is lost.

The nonpreemptive model for the 2-value model was first proposed by Aiello et al [1], who studied four different queueing policies. Each of the four policies has competitive ratio strictly worse than the known lower bound of $\frac{2\alpha-1}{\alpha}$. We present a practical online algorithm and prove that its competitive ratio is precisely $\frac{2\alpha-1}{\alpha}$, thereby completely solving the problem for this case. We also derive a fairly natural online buffer policy called Selective Barrier Policy, which is a RED-like policy [7], and whose competitive ratio is $\min\{\alpha, 1 + \sqrt{1 - 1/\alpha}\}$. Note that for $\alpha > 1.62$ the second bound dominates, and for large values of α it is approximately $\frac{2\alpha-0.5}{\alpha}$, which is near optimal. Additionally, we improve both lower and upper bounds for the Fixed and Flexible Partition policies studied in [1]. For the general model, where the packet values lie in the range $[1, \alpha]$, we establish matching upper and lower bounds of $\Theta(\log \alpha)$. We derive a $1 + \ln \alpha$ lower bound for any online policy and exhibit two online policies that are at most $e \ln(\alpha)$ competitive.

The FIFO preemptive policy has been studied extensively. The 2-value model was considered by Kesselman and Mansour [9], who provided approximately optimal performance for large values of α and B. We concentrate on the general model, for which previous lower and upper bounds were developed by Kesselman et al [8]; in particular, the lower bound is 1.281, and the upper bound is $\frac{2\alpha}{\alpha+1}$ for a natural greedy algorithm. The greedy algorithm accepts a packet if possible, otherwise preempts the lowest value packet. Intuitively, it would seem better to sometimes reject low value packets. We confirm this intuition by establishing tight upper and lower bounds of $\frac{5+\sqrt{13}}{6} \approx 1.434$ for the case where $B = 2$[1]. As a byproduct of our techniques, we improve the general lower bound for this problem to $\sqrt{2} \approx 1.414$.

Our techniques for the FIFO preemptive model are quite general and apply to the bounded-delay model as well. In particular, we show that a modified FIFO preemptive model with queue size 2 is equivalent to the 2-uniform bounded-delay model, where each arriving packet must be sent out within the next 2 time units. We establish upper and lower bounds of 1.414 and 1.366, respectively. This is an improvement upon the previous bounds of 1.434 and 1.25, respectively, due to Kesselman et al [8]. For the model where some of the arriving packets must be sent out within one time unit (the 2-variable bounded delay), we establish a lower bound of 1.618. This is an improvement of the previous lower bound of 1.414 and matches the upper bound due to Kesselman et al [8].

The rest of the paper is organized as follows, Section 2 gives a tight analysis for nonpreemptive queueing policies, Section 3 deals with FIFO preemptive queueing policies. Finally, Section 4 presents our results for the bounded-delay model.

2 Nonpreemptive Queueing policy

We first consider the 2-value nonpreemptive model. Consider a switch buffer (queue) with enough memory to hold B packets. Between time $(i-1, i)$ $(i \in \mathbb{Z}^+)$, a set of packets arrive. The queueing policy (online algorithm) has to decide whether or not to accept a packet into the queue when it arrives. Packets that are accepted to the queue stay in the queue, and get transmitted at a rate of 1 packet per time unit. A low priority packet has benefit 1, while a high priority packet has benefit $\alpha > 1$. The aim of the queueing policy is to maximize the sum of the benefits of all packets that get transmitted.

Aiello et al [1] showed that for a particular value of α, there is a general lower bound of $\frac{2\alpha-1}{\alpha}$ for any online algorithm (deterministic or randomized). Here we present the Ratio Partition (RP) policy, which builds on early policies given in [1], and show that it's competitive ratio is indeed $\frac{2\alpha-1}{\alpha}$, matching the lower bound.

We use OPT to denote the optimal algorithm, and RP to denote the Ratio Partition algorithm. And for

[1]For $B = 1$, an online algorithm can simply send the most valuable packet at each time, which is easily seen to be optimal.

convenience, we use H.P. to denote "high priority," and L.P. to denote "low priority." RP sets the threshold parameter r to be $\frac{\alpha}{\alpha-1}$. Whenever a H.P. packet is arrived, it is accepted as long as the queue is not full (there is free slot). The accepted H.P. packet is then matched to $\frac{\alpha}{\alpha-1}$ unmatched L.P. packets currently in the queue, starting with the earliest. When a L.P. packet arrives, RP only accepts it (as an unmatched L.P. packet) if including that packet, the total number of unmatched L.P. packet vs. the total number of free slots is at most $r = \frac{\alpha}{\alpha-1}$.[2]

Before analyzing RP, we first bring more structure to OPT without changing its optimality.

LEMMA 2.1. *We can rearrange the packets admitted to OPT's queue at any time, without changing the total values of the packets transmitted by OPT.*

LEMMA 2.2. *If OPT is idle at time t, then any online algorithm must also be idle at time t.*

LEMMA 2.3. *We can modify OPT so that it accepts all the H.P. packets that RP accepts, without changing its optimal value.*

LEMMA 2.4. *If we restrict the sequence to only consist of the packets that either OPT or RP accepts, neither value of OPT nor that of RP will change.*

We let S denote this restricted sequence and our analysis will only concentrate on OPT and RP's behavior on S. In particular, from lemma 2.3, we know that the H.P. packets in S are exactly the ones that OPT accepted. We further restrict OPT's behavior as follows.

LEMMA 2.5. *If at time t, RP accepts a L.P. packet p but OPT rejects, we may instead let OPT accept p, provided OPT accepts some L.P. packets later on before OPT's queue is full.*

Proof. We look into the future t', where the queue for OPT is full for the first time since t. If prior to that, OPT also accepted some other L.P. packet p', we make OPT accept p instead and reject p' as p' arrives. Since the queue was never full before t', this exchange will guarantee that the modified OPT's queue is never overflown till t', when it contains the same packets as the original one. Obviously, this does not change the optimal solution. Note that if before t', OPT only accepts H.P. packets, accepting p will interfere with the H.P. packets, then OPT must reject p. $\square$

[2]Notice that inevitably, we will encounter some roundoff errors due to the inability to split some queue space, but this is minor, and not considered here.

We now are ready for the analysis of RP. We imagine the sequence of packets accepted by an algorithm is placed on a one-way infinite tape T from left to right. The tape's index start at 0, we use $T[i]$ to denote the tape position with index i. Packets arrive between time $(t, t+1)$ can only be placed at or after $T[t]$, and at or before position $T[t+B-1]$. A packet is pushed to as far left as possible. Thus $T[t]$ denotes the packet transmitted at the end of time t (or simply $t+1$). The total value of the packets transmitted is the total value of the packets being placed on the tape. Our analysis breaks down the tape into pieces, and compares the tape content between two consecutive break down points. We use $T^{OPT}[i, j]$ ($T^{RP}[i, j]$) to denote the total value of packets from $T[i]$ to $T[j]$ on OPT (RP)'s tape.

LEMMA 2.6. *At any time we have that $l \times (\alpha - 1) \leq f \times \alpha$, where l is the number of unmatched L.P. packets in the queue and f is the number of free slots in the queue.*

Proof. When a L.P. packet is accepted, by the definition, the number of unmatched L.P. packet is not more than $\frac{\alpha}{\alpha-1}$ times the total number of the free space, so the bound trivially holds. When a H.P. packet is accepted, by definition, the free slot decrease by 1, but the number of unmatched L.P. packets is decreased by $\frac{\alpha}{\alpha-1}$, or there is no unmatched L.P. packet at the end, which we are again happy. Notice that a H.P. packet hp is always matched to some L.P. packets that arrives before hp. So when a packet departs, the bound still holds. $\square$

COROLLARY 2.1. *When a queue is full in RP, all the packets are matched in the queue.*

THEOREM 2.1. *We can break down the tape at some selected m positions $bp_1, bp_2, \ldots, bp_m$, such that $\frac{T^{OPT}[bp_i, bp_{i+1}-1]}{T^{RP}[bp_i, bp_{i+1}-1]} \leq C$, where $C = \frac{2\alpha-1}{\alpha}$.*

Proof. We will use induction, assume now we are starting from some position bp_i and there are exactly the same packets stored currently in position bp_i and afterwards, plus all the unmatched L.P. packets in the current queue is at or after the breaking point.[3] The breaking point could be somewhere in the middle of the current queue, and the end of queue marker is at $bp_i + s$, where $0 \leq s \leq B - 1$. Note that this is true for $bp_1 = 0$, and we will make sure such condition is met as we do the induction.

We examine the packets admitted to the queue by both OPT and RP. Assume at time $t'' > bp_i$, OPT and

[3]This is very important to make the whole analysis work and not double charging some L.P. packets.

RP make different decisions for the first time. There are only four general situations:

1. OPT accepted a H.P. packets, while RP rejected the packet. This cannot happen, since the queue content prior to t'' is the same for OPT and RP. RP employs greedy strategy for H.P packets, and so if OPT accepts, RP accepts as well.
2. RP accepts a H.P. packet, while OPT rejects. This cannot happen due to lemma 2.3.
3. OPT rejects a L.P. packet, while RP accepts.
4. RP rejects a L.P. packet, while OPT accepts.

We will deal with the last two situation separately and show that in either case, we can find a breaking point and maintain the hypothesis invariance.□

Let's consider situation 3. Assume everything is the same for OPT and RP from bp_i on up to time t''. At time t'', OPT rejects a L.P. packet *sp*, while RP accepts. Then by Lemma 2.5, there is a time $t' \geq t''$, such that OPT only accepts H.P. packets between time t'' and t', and at t', OPT's queue is full[4]. We then set $bp_{i+1} = t' + B$ at time t', and we compare $T^{RP}[bp_i, bp_{i+1} - 1]$ with $T^{OPT}[bp_i, bp_{i+1} - 1]$.

LEMMA 2.7. *Both OPT's and RP's queue is full at time t'.*

Proof. At time t'', RP used up more queue spaces than OPT by accepting some additional L.P. packets. RP is greedy in terms of H.P. packets, so unless RP's queue is full, it will always accept the same number of H.P. packets OPT accepts. So RP's queue will always have more or equal packets compared to OPT's until time t'. This implies that at time t', RP's queue is full. □

Note at time t', the end of the queue for OPT and RP is at $t' + B - 1$, exactly the position of $bp_{i+1} - 1$. So it makes sense to set $bp_{i+1} = t' + B$, the same free slot for OPT and RP at time $t' + 1$. Plus all L.P. packets are matched up to $T[t' + B - 1]$ in the current queue by Corollary 2.1. By default, all unmatched L.P. packet will arrive at/after the new breaking point.

We have that up to time t'', RP and OPT accepted exactly the same packets. We now compare the rest of the packets RP and OPT accepted, till $T[t' + B - 1]$. There must be equal number of such packets, since the packets occupy $T[t'' + k]$ to $T[t' + B - 1]$, where $t'' + k$ is the position of *sp* in the tape. And during this time, neither queue is idle. Ideally, we want to claim that the L.P. packets accepted by RP during this period are matched eventually. This will immediately imply the bound, since that means at least $\frac{\alpha-1}{2\alpha-1}$ fraction of the the packets are H.P. packets. However, this is not true in general. To fix this, we give a virtual marking of H.P. packets to L.P. packets as follows: for the H.P. packets that arrive after position $t'' + k$, it will mark up to $\frac{\alpha}{\alpha-1}$ unmarked L.P. packets starting only from position $t'' + k$ on if there is any. And we treat all the packets before *sp* as being marked already. So this scheme differs from our real matching scheme in that now we pair H.P. packets together with L.P. packets starting from *sp* and afterwards.

LEMMA 2.8. *From time t'' on, at any time we have that $l \times (\alpha - 1) \leq f \times \alpha$, where l is the number of unmarked L.P. packets in the queue and f is the number of free slots in the queue.*

This in particular means, when the RP's queue is full, all the L.P. packets are marked. This lemma allows us to prove that all the L.P. packets after *sp* are marked, which imply the following.

LEMMA 2.9. *At least $\frac{\alpha-1}{2\alpha-1}$ fraction of the the packets between $T^{RP}[t'' + k]$ and $T^{RP}[t']$ are H.P. packets.*

The above lemma implies $\frac{T^{OPT}[bp_i, bp_{i+1}-1]}{T^{RP}[bp_i, bp_{i+1}-1]} \leq \frac{2\alpha-1}{\alpha}$. Now let's deal with the last situation 4. Assume at some time t'', OPT accepted a L.P. packet, while RP rejected it. Then we look at the first time t' when RP's queue contains the same number of packets as OPT's does. We set $bp_{i+1} = \lfloor t' \rfloor$ at time t', i.e., we only compare packets transmitted up to but not including $T[\ \lfloor t' \rfloor\ ]$.

LEMMA 2.10. *We can rearrange OPT's queue content from time t'' on so that the prefix of its queue matches the entire RP's queue content. To be more specific, at any time,*

1. *The number of packets in OPT's queue is always more than the number of packets in RP's queue before time t'.*
2. *The H.P. packets in OPT's queue match in position to those H.P. packets in RP's queue. In particular, that means the number of H.P. packets in either queue is the same.*

That means RP's H.P. packet is synchronized with that of OPT's all the time. So the total number of H.P. packets for OPT and RP is even. We just have to bound the loss of L.P. packets for RP. We use the unmatched L.P. packets to account such loss.

We introduce the following virtual counting scheme. Every L.P. packet contributes another $\frac{\alpha-1}{\alpha}$ virtual L.P.

[4]Without loss of generality, we extend t' till just before the next transmission, i.e., t' is integral.

packet to RP's queue, while it enters the queue. In particular, we have that if the L.P. packet is currently in the queue, its virtual part is also in the current queue (we allow the virtual packets to overflow the queue). The virtual packet is always shifted to the end of the real packets in RP's queue. RP always transmits a real packet as normal, but when it's idle, we let RP transmit a virtual packet if there is any.

LEMMA 2.11. *At any time, the total number of packets in RP's queue, including the virtual packets, is always no less than the total number of packets in OPT's queue until t'.*

Since we have H.P. packets all lined up, that means that including the virtual L.P. packets generated by packets between $T[bp_i]$ and $T[bp_{i+1}-1]$, RP transmitted as many as OPT did during that period. So the total number of L.P. packet RP transmitted is at least $\frac{\alpha}{2\alpha-1}$ of the total number of L.P. packet OPT transmitted. Hence the ratio $\frac{T^{OPT}[bp_i,bp_{i+1}-1]}{T^{RP}[bp_i,bp_{i+1}-1]} \leq \frac{2\alpha-1}{\alpha}$.

At time t' we know that by rearranging OPT's queue, the two queues looks exactly the same. By setting breaking point $bp_{i+1} = t'$, our next step will pick up from time t' with RP and OPT have the same entire queue content. We thus obtain the following theorem.

THEOREM 2.2. *RP is $\frac{2\alpha-1}{\alpha}$-competitive.*

2.1 Improved Bounds for Policies

In this section we improve the both upper and lower bounds from [1] for the Fixed and Flexible Partition Policies.

2.1.1 Fixed Partition Policy

The Fixed Partition Policy partitions the queue between the low and high packet values values, using a variable $0 \leq x \leq 1$ such that xB is integer. The Fixed Partition Policy accepts a L.P. (H.P.) packet if including this packet, there are at most xB L.P. packets ($(1-x)B$ H.P. packets) in the queue.

We prove the following lower and upper bounds for the competitive ratio of the Fixed Partition Policy (proofs omitted):

THEOREM 2.3. *The competitive ratio of the Fixed Partition Policy ρ_{FP} is at most:*

$$\rho_{FP} \leq \min\left\{\sqrt{2}+1,\ 3-\frac{3}{2\alpha+1}\right\}$$

THEOREM 2.4. *The competitive ratio of the Fixed Partition Policy ρ_{FP} is at least:*

$$\rho_{FP} \geq \max\left\{2,\ 1-\frac{1}{2\alpha}+\sqrt{2-\frac{1}{\alpha}+\frac{1}{4\alpha^2}}\right\}$$

We derive the lower bound by comparing the performance of FPP to OPT among the following scenarios:

1. B L.P. packets arrive.
2. B H.P. packets arrive.
3. B L.P. packets followed by B H.P. packets arrive. After xB time units xB H.P. packets arrive.

2.1.2 Flexible Partition Policy

Similarly to the Fixed Partition Policy, the Flexible Partition Policy partitions the queue between the low and high value packets, using a variable $0 \leq x \leq 1$ such that xB is integer. The difference is that in the Flexible Partition Policy we allow H.P. packets enter the queue allocation of L.P. packets, since this will intuitively only improve the total benefit. Specifically, it accepts a L.P. packet if including this packet, the number of L.P. packets in the queue is at most xB, and accepts a H.P. packet if the queue is not full.

We prove the following lower and upper bounds for the competitive ratio of the Flexible Partition Policy (proofs omitted):

THEOREM 2.5. *The competitive ratio of the Flexible Partition Policy ρ_{FPP} is at most:*

$$\rho_{FPP} \leq \min\left\{\alpha, \sqrt{2}+1\right\}$$

THEOREM 2.6. *The competitive ratio of the Flexible Partition Policy ρ_{FPP} is at least:*

$$\rho_{FPP} \geq \min\left\{\alpha,\ \sqrt{2-\frac{2}{\alpha}+\frac{1}{4\alpha^2}}\ +1-\frac{1}{2\alpha}\right\}$$

We derive the lower bound by comparing the performance of FPP to OPT between the following two scenarios:

1. B L.P. packets arrive.
2. B L.P. packets followed by B H.P. packets arrive. After xB time units xB low packets followed by xB high packets arrive, and after $(1-x)B$ additional time units, $(1-x)B$ low packets arrive.

2.2 Generalizations of the Nonpreemptive Queue Model

We now generalize the 2-value model to allow packets take on arbitrary values in $[1, \alpha]$. We prove $\Theta(\log \alpha)$ upper and lower bounds for this model.

2.2.1 Lower Bound

THEOREM 2.7. *There exists a set $\mathcal{V}$ of n values such that any online scheduling policy has a competitive ratio of at least* $\frac{n\sqrt[n-1]{\alpha}-(n-1)}{\sqrt[n-1]{\alpha}}$

Proof. Let $\mathcal{V} = \{1, 1+\epsilon, (1+\epsilon)^2, \ldots, (1+\epsilon)^{n-1} = \alpha\}$, where $\epsilon = \sqrt[n-1]{\alpha} - 1$. Consider the following n scenarios, defined recursively:
In the first scenario, B packets of the lowest value (i.e., 1) arrive. The online policy accepts $x_1 B$, thus reaching a benefit of $V_1 = x_1 B$. Since the optimal policy may accept all B packets, we have a competitive ratio of $\rho_1 = x_1$.
In the k-th scenario, the packets from scenario $k-1$ arrive, followed by B packets of value $(1+\epsilon)^{k-1}$. Initially, the online policy accepts the same packets as in scenario $k-1$, and adds $x_k B$ additional packets of value $(1+\epsilon)^{k-1}$, reaching a benefit of $V_k = V_{k-1} + x_k(1+\epsilon)^{k-1}$. An optimal offline schedule would have accepted all of the last B packets, therefore the competitive ratio is $\rho_k = \frac{V_{k-1}+x_k(1+\epsilon)^{k-1}}{(1+\epsilon)^{k-1}} = \frac{\rho_{k-1}}{1+\epsilon} + x_k$.

Our lower bound would be the maximal value of $\rho = \min\{\rho_1, \rho_2, \ldots, \rho_n\}$ under the restrictions $x_i \geq 0$ for $1 \leq i \leq n$, and $\Sigma_{i=1}^{n} x_i \leq 1$. The maximum is achieved when $\rho_1 = \rho_2 = \ldots = \rho_n$ and $\Sigma_{i=1}^{n} x_i = 1$. The solution to the set of the equations is $x_2 = x_3 = \ldots = x_n = \frac{\epsilon}{1+\epsilon} x_1$. Since $\alpha = (1+\epsilon)^{n-1}$, i.e. $\epsilon = \sqrt[n-1]{\alpha} - 1$, and the maximum is reached when:

$$\begin{aligned} x_i &= \frac{1}{(n-1)+\frac{1+\epsilon}{\epsilon}} = \frac{\epsilon}{n\epsilon+1} \\ &= \frac{\sqrt[n-1]{\alpha}-1}{n\sqrt[n-1]{\alpha}-(n-1)} \quad (i = 2, 3, \ldots, n) \\ x_1 &= \frac{\sqrt[n-1]{\alpha}}{n\sqrt[n-1]{\alpha}-(n-1)} \end{aligned}$$

Since $\frac{1}{\rho} \leq x_1$ the theorem follows. □

Next we derive the dependance on the number if distinct values.

THEOREM 2.8. *For any fixed n and any $\delta > 0$ there exists a series of values $V = \{\alpha_j\}_{j=0}^{n-1}$ such that the competitive ratio of any online policy A is at least $n - \delta$.*

Proof. By Theorem 2.7 we have:

$$\rho(A) \geq \frac{n\sqrt[n-1]{\alpha}-(n-1)}{\sqrt[n-1]{\alpha}} = n - \frac{n-1}{\sqrt[n-1]{\alpha}}.$$

For $\alpha \geq (\frac{n-1}{\delta})^{n-1}$ the theorem follows. □

The following theorem generalizes the lower bound to the case of infinite number of possible values.

THEOREM 2.9. *For arbitrary packet values between 1 and α, any online scheduling policy has a competitive ratio of at least* $(1 + \ln\alpha)$

Proof. For contradiction, assume that there exists a policy A, such that $\rho(A) \leq 1 + (\ln\alpha)(1-\epsilon)$. Let $n = 1 + \frac{1}{\epsilon}\ln\alpha$, hence $\sqrt[n-1]{\alpha} = e^{\epsilon}$. By Theorem 2.7 we have:

$$\begin{aligned} \rho &\geq \frac{n\sqrt[n-1]{\alpha}-(n-1)}{\sqrt[n-1]{\alpha}} = \frac{(1+\frac{1}{\epsilon}\ln\alpha)e^{\epsilon} - \frac{1}{\epsilon}\ln\alpha}{e^{\epsilon}} \\ &= 1 + \frac{\frac{1}{\epsilon}(\ln\alpha)(e^{\epsilon}-1)}{e^{\epsilon}} = 1 + (\ln\alpha)\frac{1}{\epsilon}(e^{\epsilon}-1)e^{-\epsilon} \end{aligned}$$

Since for any $\epsilon > 0$ we have $e^{\epsilon} > 1+\epsilon$ and $e^{-\epsilon} > 1-\epsilon$, therefore,

$$\rho(A) > (\ln\alpha)\epsilon\frac{1}{\epsilon}(1-\epsilon) = 1 + (\ln\alpha)(1-\epsilon),$$

which contradicts our assumption. □

2.2.2 Upper Bound

We analyze the Round Robin Policy that equally divides the buffer into n partitions of size $\frac{B}{n}$, where n is the number of different packet values. Each partition is assigned a different value, and only packets of this value are accepted into this partition, in a greedy manner. The partitions take turns in sending packets. If a partition's turn to send a packet arrives, but it is empty, its turn passes to the next partition.

We can simulate the Round Robin method using a single queue as follows. We virtually keep track of the current state of the Round Robin method, use the virtual state to decide whether or not to accept the currently incoming packets. Since the Round Robin method transmits 1 packet per time step in total, the real queue size coincides with that of the virtual queue size, and both queues accept the same set of packets.

THEOREM 2.10. *The competitive ratio of the Round Robin policy is n.*

For large values of n the competitive ratio of the Round Robin policy is unbounded, however, the competitive ratio is independent of α. The performance of the Round Robin policy can be improved for large values of n by partitioning the buffer into k parts of size $\frac{B}{k}$, each part accepts only packets with values in a certain range. This implies that we split the interval $[1, \alpha]$ into k sub-intervals $[1 = \alpha_0, \alpha_1)[\alpha_1, \alpha_2)\ldots[\alpha_{k-1}, \alpha_k = \alpha]$. Each sub-interval accepts in a greedy manner packets with values from this sub-interval. As in the original policy, partitions take turns in sending packets. Exponential-Intervals Round Robin using k partitions sets $\alpha_j = \alpha^{j/k}$.

LEMMA 2.12. *The competitive ratio of the Exponential-Interval Round Robin policy, using k partitions is $k\alpha^{1/k}$, and for any Round Robin Policy using k partitions, the competitive ratio is at least $k\alpha^{1/k}$.*

We can derive the following corollary for the case of infinite number of values.

COROLLARY 2.2. *The competitive ratio of the Exponential-Interval Round Robin with $k = \lceil \ln \alpha \rceil$ partitions is $e\lceil \ln \alpha \rceil$.*

The Round-Robin Policy's actions might be counter intuitive, since it equally prefers packets of all types and may reject packets of one value in order to save place for packets of a lower value. Intuitively, we would prefer a policy that discriminates between packets of different values, preferring those of higher value. We introduce the Selective Barrier (SB) Policy (which can be viewed as a deterministic variant of RED (Random Early Detection)[7]) and preserves the same upper bound that was reached by the Round Robin Policy.

The Selective Barrier Policy uses a function $\mathcal{F}(x) : [1, \alpha] \mapsto [0, 1]$, to accept packets with value v iff the queue contains less than $\mathcal{F}(v)B$ packets (of any value). Naturally, $\mathcal{F}(x)$ should be monotone and $\mathcal{F}(\alpha) = 1$.

LEMMA 2.13. *The Selective Barrier Policy with n values and $\mathcal{F}(\alpha_i) = \frac{i}{n}$ has a competitive ratio of at most n.*

Next we derive a bound for an unbounded number of values.

THEOREM 2.11. *For any $k \geq 1$, the competitive ratio of the Selective Barrier Policy is at most $k\alpha^{1/k}$.*

By Theorem 2.11, we reach the best bound for the competitive ratio when $k \in \{\lfloor \ln \alpha \rfloor, \lceil \ln \alpha \rceil\}$, and is approximately $e \ln \alpha$.

2.3 Selective Barrier: 2 values For the two value case, we prove tight bounds for the competitive ratio of the Selective Barrier Policy.

THEOREM 2.12. *The competitive ratio of the Selective Barrier Policy with two values is:*

$$\min\{\alpha, 1 + \sqrt{1 - 1/\alpha}\}$$

For $\alpha > \phi \approx 1.62$, the competitive ratio is $1 + \sqrt{1 - 1/\alpha}$, which is approximately $\frac{2\alpha - 0.5}{\alpha}$, i.e. very near to the best possible competitive ratio.

We denote the value of $F(1)$ by x, i.e. SB accepts high value packets greedily and low value packets as long as the queue contains less than xB packets. When $\alpha \leq \phi$, we simply set $x = 1$ and greedily accept all the packets until the queue is full, hence we have a trivial bound of α.

For the case of $\alpha > \phi$, we derive the lower bound by comparing the performance of SB to OPT between the following two scenarios:

1. B L.P. packets arrive.
2. B L.P. packets followed by B H.P. packets arrive. After $(1 - x)B$ time units, $(1 - x)B$ L.P. packets arrive.

To prove the upper bound, we generate OPT by changing SB's schedule, and bound the difference between the schedules. OPT is generated by two steps:

1. `SWAP`: For every L.P. packet SB accepted, if rejecting it allows to accept latter a H.P. packet, then swap the actions.
2. `ADD_LOW`: For every L.P. packet SB rejected, accept if it doesn't interfere with future H.P. packets.

LEMMA 2.14. *Applying* `SWAP` *and* `ADD_LOW` *to SB generates an optimal schedule.*

We use the following notations in order to refer to sets of packets: L_M denotes the L.P. packets replaced in the `SWAP` step, and H_M denotes the extra H.P. packets OPT accepts. L denotes the L.P. packets added to OPT in the `ADD_LOW` step. BOT denotes L.P. and H.P. packets that entered the queue of SB before it was xB full, while TOP denotes the H.P. packets accepted by SB when its queue already contained at least xB packets.

We define time intervals starting and ending at the events of the queue of SB being either empty or full. Then, we divide the packets into two sets, according to the interval they arrived in. Afterwards, we analyze each type of packets separately, according to this division.

DEFINITION 2.1. *Let $T_i = [t_{i-1}, t_i]$ be successive time intervals, where t_i are the time points where the queue of SB just became either empty or full. Let G_i be the packets that arrived during the interval T_i. Let G_E be the union of sets G_i such that the queue of SB is empty both at t_{i-1} and at t_i. Let G_F be the union of sets G_i such that the queue of SB is full at either t_{i-1} or t_i.*

Since the queue never gets full at G_E intervals, we need to consider only `ADD_LOW` when analyzing G_E. In each interval, OPT can accept at most $(1 - x)B$ L.P. packets due to the `ADD_LOW` step, because the queue of SB is already at least xB full when such packets arrive.

On the other hand, if such packets arrived, we have a lower bound of xB on the number of packets that SB accepted before the queue was xB full (these are either low value packets, or high value packets that would still be accepted even if their value was low).

LEMMA 2.15. $x|L \cap G_E| \leq (1-x)|(BOT \setminus L_M) \cap G_E|$

To analyze G_F, we bound the number of additional high packets OPT accepts by the number of high packets the SB accepts. If there were swaps, then the queue was full at the end of the interval, meaning at least $(1-x)B$ high value packets arrived after the queue was at least xB full. On the other hand, at most xB swaps were done in this interval, since the swapped low value packets arrive only when the queue is less that xB full.

LEMMA 2.16. $(1-x)|L_M \cap G_F| = (1-x)|H_M \cap G_F| \leq x|TOP \cap G_F|$

If the interval contains packets from ADD_LOW, then there were at most $(1-x)B$ packets. In this case, the endpoint where the queue was full must be the beginning of the interval, which means that at least $(1-x)B$ high value packets were in the end of the queue.

LEMMA 2.17. $|L \cap G_F| \leq |TOP \cap G_F|$.

By choosing $x = 1 + \sqrt{1 - \frac{1}{\alpha}}$, we combine the previous lemmas to prove the upper bound of SB separately G_E and G_F, thus deriving Theorem 2.12.

3 Preemptive Queueing policy

Preemptive queueing policy gives online algorithms the ability to preempt (drop) packets already accepted to the queue, hence better performance ratios can be derived. To maintain the FIFO order, packets are always added to the end of the queue, and transmitted from the beginning of the queue. In addition, a packet can be dropped from the middle of the queue and the packets behind it shift forwards.

We concentrate on the model where packets can take on values in the range $[1, \alpha]$. We first consider the case $B = 2$. We show the $\frac{5+\sqrt{13}}{6} \approx 1.434$ upper and lower bounds.

3.1 Lower Bound Construction

We start with some intuition via simple constructions that give loose bounds.

The basic intuition behind the lower bound is of the following: suppose the online algorithm currently have two packets in the queue, with values p and q. Suppose $p < q$ and p has to be transmitted before q. Without knowing future arrivals, the online algorithm faces a dilemma: either to transmit p and buffer q, which puts the online algorithm in danger of discarding q later on; or discard p and transmit q, which puts the online algorithm in danger of being idle the next time unit. Thus the adversary will try to balance these two situations in order to maximize the lower bound.

We introduce the following notation for packet arrival sequence: Packet values are listed in their arrival order, with packets arrived within the same time unit grouped into parentheses.[5]

Consider the following sequence: $(1, \alpha > 1)$, if the online algorithm transmits α, then the sequence ends. OPT transmits both 1 and α in two time units. The ratio is then $\frac{1+\alpha}{\alpha}$. Otherwise the online algorithm can transmit 1 and put α in the queue, then the whole sequence becomes $(1, \alpha)(\alpha, \alpha)$. OPT transmits all three α-valued packets, while the online algorithm has to discard one such packet during the second time unit. So the ratio becomes $\frac{3\alpha}{2\alpha+1}$. Balancing the two ratios, we have $\alpha = \frac{3+\sqrt{13}}{2}$, and the bound is approximately 1.303.

Now we may consider to improve the bound, via extending the sequence to $(1, \alpha_1)(\alpha_1, \alpha_2)(\alpha_2, \alpha_2)$. If at time 1 online transmits α_1, then the sequence stops and no new packets arrive, with the ratio being $\frac{1+\alpha_1}{\alpha_1}$. Otherwise, let's consider time 2. If online transmits α_2, then the sequence again stops, with the ratio being $\frac{\alpha_1+\alpha_1+\alpha_2}{1+\alpha_2}$. Otherwise, then at time 4, the ratio becomes $\frac{\alpha_1+\alpha_2+\alpha_2+\alpha_2}{1+\alpha_1+\alpha_2+\alpha_2}$. We then have $\alpha_1 = \frac{3+\sqrt{5}}{2}$, $\alpha_2 = \frac{9+4\sqrt{5}}{2}$, and the bound is approximately 1.382.

Thus if we continue the trend, we can improve the bound further. Consider the following general sequence $(\alpha_0 = 1, \alpha_1)(\alpha_1, \alpha_2) \ldots (\alpha_{k-1}, \alpha_k)(\alpha_k, \alpha_k)$. We then again have to balance the following equations, derived from various scenarios similar to the two previous constructions:

(3.1)
$$\frac{\alpha_0 + \alpha_1}{\alpha_1} = \frac{\alpha_1 + \alpha_1 + \alpha_2}{\alpha_0 + \alpha_2} = \frac{\alpha_1 + \alpha_2 + \alpha_2 + \alpha_3}{\alpha_0 + \alpha_1 + \alpha_3} = \ldots$$
$$= \frac{\alpha_1 + \alpha_2 + \ldots + \alpha_{k-1} + \alpha_{k-1} + \alpha_k}{\alpha_0 + \alpha_1 + \ldots + \alpha_{k-2} + \alpha_k}$$
$$= \frac{\alpha_1 + \alpha_2 + \ldots + \alpha_{k-1} + \alpha_k + \alpha_k + \alpha_k}{\alpha_0 + \alpha_1 + \ldots + \alpha_{k-2} + \alpha_{k-1} + \alpha_k + \alpha_k}$$

Solving the above equations, we have $\alpha_i = \frac{1}{3}(\frac{\sqrt{13}+5}{2})^i + \frac{2}{3}(\frac{\sqrt{13}-1}{2})^i$ and as $k \to \infty$, the final ratio

[5] For instance, $(a, b)(c, d)$ means packets valued a and b arrived during the first time unit, with a in front of b. Similar situation applies to packets valued c and d, which arrive during the second time unit.

```
RATIO —
For the arrival of a new packet p at time t.
IF p < V_1^t and p < V_2^t
    DROP p
ELSE
    DROP the smaller of B_1^t and B_2^t
    ACCEPT p into the queue, so now B_2^t = p
    IF V_2^t/V_1^t ≥ r
        DROP B_1^t, so now B_1^t = p
```

Figure 1: Pseudo-code for the RATIO Algorithm

$\rightarrow \frac{\sqrt{13}+5}{6} \approx 1.434$.

Theorem 3.1. *No deterministic online algorithm can achieve a competitive ratio better than* $\frac{\sqrt{13}+5}{6} - \epsilon$, *for any constant* $\epsilon \geq 0$.

3.2 The *RATIO* Algorithm

The α_i series provides much insight on designing the matching online algorithm. The determining critical ratio is $\frac{\sqrt{13}+5}{2}$, as $\frac{\alpha_i}{\alpha_{i-1}}$ converges to this value. The other ratio $\frac{\sqrt{13}-1}{2}$ in the expression is only used by the adversary to make sure the early equations' values are large enough to match the convergent ratio. The online algorithm thus has much less work to do and is strikingly simple. Let r denote the constant $\frac{\sqrt{13}+5}{2}$. We use B_1^t to denote the first packet in the queue at time t, and B_2^t the second packet. Similarly V_1^t and V_2^t refer to the values of the packets. The *RATIO* algorithm looks at all packets that arrived between time $(t, t+1)$, and the two most valuable packets will be considered. If the second packet has value more than r times that of the first packet, the first packet will be dropped and only the second one will be transmitted. Otherwise the first packet will be transmitted, while the second is held in the queue . Figure 1 shows the Pseudo-code.

We describe the analysis at a high level, which is motivated by the lower bound construction. There are two situations where we could claim *RATIO* transmitted enough packets:

1. When *RATIO* transmitted the second higher valued packet and discarded the first packet.
2. When *RATIO* finished transmitting both packets that was once in the queue together.

The difficult situation is that *RATIO* always transmits the first smaller valued packet and drops the second packet later. *OPT* instead transmits the higher valued packet, which maybe r times more valuable. Thus, the analysis has to take care of this escalating effect. Indeed, one should expect to deal with long expressions similar to the equations in the lower bound construction. This makes the analysis quite involved. Detailed proofs are deferred to the full version of the paper.

Theorem 3.2. *The RATIO algorithm is* $\frac{5+\sqrt{13}}{6}$-*competitive.*

3.3 Lower Bound for the General Model

We now briefly discuss the generalization of our lower bound construction to arbitrary queue sizes. In particular, for a queue of size B, consider the following general sequence (where $Z < B$ is some constant to be determined later):

$$\underbrace{(1,1,\ldots,1,\alpha_1)}_{B}\underbrace{(\alpha_1)(\alpha_1)\ldots(\alpha_1)}_{Z-1}\underbrace{(\alpha_1,\alpha_1,\ldots,\alpha_1,\alpha_2)}_{B}$$
$$\underbrace{(\alpha_2)(\alpha_2)\ldots(\alpha_2)}_{Z-1}\underbrace{(\alpha_2,\alpha_2,\ldots,\alpha_2,\alpha_3)}_{B}\ldots$$
$$\ldots\underbrace{(\alpha_{k-1},\alpha_{k-1},\ldots,\alpha_{k-1},\alpha_k)}_{B}\underbrace{(\alpha_k)(\alpha_k)\ldots(\alpha_k)}_{Z}$$
$$\underbrace{(\alpha_k,\alpha_k,\ldots,\alpha_k)}_{B}\underbrace{(\,)(\,)\ldots(\,)}_{B}$$

At times 1 through Z, if the online algorithm decides to transmit any one of the α_1 packets, then the adversary will stop the sequence after time Z. The online algorithm transmits no more than $(Z-1)+Z\alpha_1$, while OPT transmits all of the packets. Thus the ratio is then no less than $\frac{B-1+Z\alpha_1}{Z-1+Z\alpha_1}$. The adversary then start from the beginning of the sequence again.

Otherwise, the online algorithm then at the best transmitted Z packets of value 1 during times 1 through Z. OPT instead transmitted Z packets of value α_1 during the same period, while discarding the earlier packets with value 1. The adversary then make $B-1$ packets of value α_1 and another packet of value α_2 to arrive one by one after time Z, and then repeat the strategy as during times 1 through Z. If such situation continues, the adversary will end the sequence at α_k.

To maximize the lower bound, we need not only to balance the ratios in all cases, but also to pick a value Z that maximizes the terms. We have the following final result[6]: $Z = \lfloor \frac{B}{2} \rfloor$, $\alpha_i = \frac{1}{1+B}(\frac{2B+1+\sqrt{2B^2+2B+1}}{B})^i + \frac{B}{1+B}(\frac{-1+\sqrt{2B^2+2B+1}}{B})^i$, and the final bound is $\frac{5+\sqrt{2B^2+2B+1}}{B+4} > \frac{5+\sqrt{(\sqrt{2}B+1/\sqrt{2})^2}}{B+4} > \sqrt{2}$.

Theorem 3.3. *With* $Z = \lfloor \frac{B}{2} \rfloor$, *the lower bound ratio approaches* $\sqrt{2}$, *for queue size* B.

[6]These expressions are for even B. We omit the expressions for odd B for simplicity.

4 Bounded-delay Queueing Policy

We first discuss the connection of the FIFO preemptive model to the uniform bounded-delay model. In particular, we consider the FIFO preemptive model proposed in [8]. The model to be discussed in this section differs from our previous definition in that the online algorithm is allowed to reorder the packets that arrive at the same time unit. The δ-uniform bounded delay model requires every packet to be transmitted within δ time units after its arrival, or otherwise it is lost. The packets can be transmitted out of order.

Clearly, any FIFO preemptive online algorithm with queue size B delays each packet by at most $\delta = B$ time unit, hence also an online algorithm for the bounded delay model. However, the converse is not always true. We show here that the converse is true for $B = 2$.[7]

THEOREM 4.1. *For $B = \delta = 2$, the optimal 2-uniform bounded delay online algorithm needs a buffer with size no more than 2 and can serve packets in FIFO order.*

THEOREM 4.2. *For $B = 2$ or equivalently $\delta = 2$, the competitive ratio is between 1.366 and 1.414.*

The 2-variable model allows two types of packets: the ones that must be sent the next time unit (with delay 1), and the ones that can be delayed for 1 extra time unit (with delay 2). The bad situation for the online algorithm would be a smaller valued packet has an early deadline against a higher valued packet. The idea here is again to repeat the bad situation many time units and prove some convergence result. Consider the following general sequence (the number in brackets denotes the allowed delay for that packet):

$(1[1], \alpha_1[2])(\alpha_1[1], \alpha_2[2]) \dots (\alpha_{k-1}[1], \alpha_k[2])(\alpha_k[1])$.

The calculation uses the same techniques and is omitted here. We thus obtain the following result, which matches the previous deterministic upper bound.

THEOREM 4.3. *For the 2-variable bounded delay model, no deterministic online algorithm can achieve a competitive ratio better than $\frac{\sqrt{5}+1}{2} - \epsilon$, for any constant $\epsilon > 0$.*

References

[1] W.A. Aiello, Y. Mansour, S. Rajagopolan, and A. Rosen. Competitive Queue Policies for Differentiated Services. *Proceedings of the IEEE INFOCOM*, 2000, pages 431–440.

[2] The ATM Forum Technical Committee. Traffic management specification version 4.0. *Available from www.atmforum.com*, Apr. 1996.

[3] D. Black, S. Blake, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. *Internet RFC 2475*, December 1998.

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

[5] D. Clark and J. Wroclawski. An approach to service allocation in the internet. *Internet draft, available from diffserv.lcs.mit.edu*, 1997.

[6] C. Dovrolis, D. Stiliadis, and P. Ramanathan. Proportional Differentiated Services: Delay Differentiation and Packet Scheduling. *Proceedings of ACM SIGCOMM*, 1999, pages 109–120.

[7] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(1993):397–413.

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

[9] A. Kesselman and Y. Mansour. Loss-Bounded Analysis for Differentiated Services. *Proceedings of the 12th Annual SIAM-ACM Symposium on Discrete Algorithms*, 2001, pages 591–600.

[10] M. May, J.-C. Bolot, A. Jean-Marie, and C. Diot. Simple performance models of differentiated services for the internet. Proceedings of IEEE INFOCOM, 1999, pages 1385–1394.

[11] K. Nichols, V. Jacobson, and L. Zhang. A Two-bit Differentiated Services Architecture for the Internet. *Internet Draft, available from citeseer.nj.nec.com/251975.html*, July 1999.

[12] T. Nandagopal, N. Venkitaraman, R. Sivakumar, and V. Bharghavan. Relative Delay Differentiation and Delay Class Adaptation in Core-Stateless Networks. *Proceedings of IEEE INFOCOM*, 2000, pages 421–430.

[13] N. Semret, R.R.-F. Liao, A.T. Campbell, and A.A. Lazar. Peering and Provisioning of Differentiated Internet Services. *Proceedings of IEEE INFOCOM*, 2000, pages 414–420.

[14] I. Stoica and H. Zhang. Providing Guaranteed Services without Per Flow Management. *Proceedings of SIGCOMM*, 1999, pages 81–94.

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

[7] We comment that in general, the two problems are not equivalent. In fact, we believe that for $\delta \geq 3$, the best online algorithm for δ-uniform bounded delay model will serve the packets out of order.