An improved algorithm for CIOQ switches [*]

Yossi Azar [†] Yossi Richter [‡]

Abstract

The problem of maximizing the weighted throughput in various switching settings has been intensively studied recently through competitive analysis. To date, the most general model that has been investigated is the standard CIOQ (Combined Input and Output Queued) switch architecture with internal fabric speedup $S \geq 1$. CIOQ switches, that comprise the backbone of packet routing networks, are $N \times N$ switches controlled by a switching policy that incorporates two components: Admission control and scheduling. An admission control strategy is essential to determine the packets stored in the FIFO queues in input and output ports, while the scheduling policy conducts the transfer of packets through the internal fabric, from input ports to output ports. The online problem of maximizing the total weighted throughput of CIOQ switches was recently investigated by Kesselman and Rosén in [15]. They presented two different online algorithms for the general problem that achieve non-constant competitive ratios (linear in either the speedup or the number of distinct values, or logarithmic in the value range). We introduce the first constant-competitive algorithm for the general case of the problem, with arbitrary speedup and packet values. Specifically, our algorithm is 8-competitive, and is also simple and easy to implement.

1 Introduction

Overview: Recently, packet routing networks have become the dominant platform for data transfer. The backbone of such networks is composed of $N \times N$ switches, that accept packets through multiple incoming connections and route them through multiple outgoing connections. As network traffic continuously increases and traffic patterns constantly change, switches routinely have to efficiently cope with overloaded traffic, and are forced to discard packets due to insufficient buffer space, while attempting to forward the more valuable packets to their destinations.

Traditionally, the performance of queuing systems has been studied within the stability analysis framework, either by a probabilistic model for packet injection (queuing theory, see e.g. [9, 17])

[*]A preliminary version of this paper appears in the proceedings of the 12th Annual European Symposium on Algorithms (ESA), 2004, pp. 65–76.

[†]**Corresponding author.** Phone: +972-3-6406354, Fax:+972-3-6409357, azar@tau.ac.il. School of Computer Science, Tel Aviv University, Tel Aviv, 69978, Israel. Research supported in part by the Israeli Ministry of industry and trade and by the Israel Science Foundation.

[‡]yo@tau.ac.il. School of Computer Science, Tel Aviv University, Tel Aviv, 69978, Israel. Research supported in part by the Israeli Ministry of industry and trade.

or an adversarial model (adversarial queuing theory, see e.g. [4, 10]). In stability analysis packets are assumed to be identical, and the goal is to determine queue sizes such that no packet is ever dropped. However, real-world networks do not usually conform with the above assumptions, and it seems inevitable to drop packets in order to maintain efficiency. As a result, the competitive analysis framework, which avoids any assumptions on the input sequence and compares the performance of online algorithms to the optimal solution, has been adopted recently for studying throughput maximization problems. Initially, online algorithms for single-queue switches were studied in various settings [1, 3, 8, 12, 13, 14, 16]. Later on, switches with multiple input queues were investigated [2, 5, 6, 7], as well as CIOQ switches with multiple input and output queues [15].

To date, the most general switching model that has been studied using competitive analysis is CIOQ (Combined Input and Output Queued) switching architecture. A CIOQ switch with speedup $S \geq 1$ is an $N \times N$ switch, with N input ports and N output ports. The internal fabric that connects the input and output FIFO queues is S times faster than the queues. A switching policy for a CIOQ switch consists of two components. First, an admission control policy to determine the packets stored in the bounded-capacity queues. Second, a scheduling strategy to decide which packets are transferred from input ports to output ports through the intermediate fabric at each time step. The goal is to maximize the total value of packets transmitted from the switch. The online problem of maximizing the total throughput of a CIOQ switch was studied by Kesselman and Rosén in [15]. For the special case of unit-value packets (all packets have the same value) they presented a greedy algorithm that is 2-competitive for a speedup of 1 and 3-competitive for any speedup. For the general case of packets with arbitrary values two different algorithms were presented, by using techniques similar to those introduced in [6]. The first algorithm is $4S$-competitive and the second one is $(8\min\{k, 2\lceil\log\alpha\rceil\})$-competitive, where k is the number of distinct packet values and α is the ratio between the largest and smallest values.

Our results: We present the first constant-competitive algorithm for the general case of the problem with arbitrary packet values and any speedup. Specifically, our algorithm is 8-competitive and is also simple and easy to implement. Our analysis includes a new scheme to map discarded packets to transmitted packets with the aid of generating dummy packets.

Other related results: The online problem of throughput maximization in switches has been explored extensively during recent years. Initially, single-queue switches were investigated, for both arbitrary packet sequences, and 2-value sequences. The preemptive model, in which packets stored in the queue can be discarded, was studied in [8, 12, 13, 14, 16]. The non-preemptive model, in which a packet can be discarded only upon arrival, was initially studied by Aiello *et al.* [1], followed by Andelman *et al.* [3] who showed tight bounds. The results for a single queue were generalized for switches with an arbitrary number of input queues in [6], by a general reduction from the multi-queue model to the single-queue model. Specifically, a 4-competitive algorithm is presented in [6] for the weighted multi-queue switch problem. An improved 3-competitive algorithm was later shown in [7]. The multi-queue switch model has been also investigated for the special case of unit-value packets, which corresponds to IP networks. Albers and Schmidt [2] introduced a deterministic 1.89-competitive algorithm for the unit-value problem, followed by Azar and Litichevskey [5] that showed a 1.58-competitive algorithm for switches with large buffers. A randomized 1.58-competitive algorithm was previously presented in [6]. An alternative model to the multiple input queues model is the shared memory switch, in which memory is shared among all queues. Hahne *et al.* [11] studied

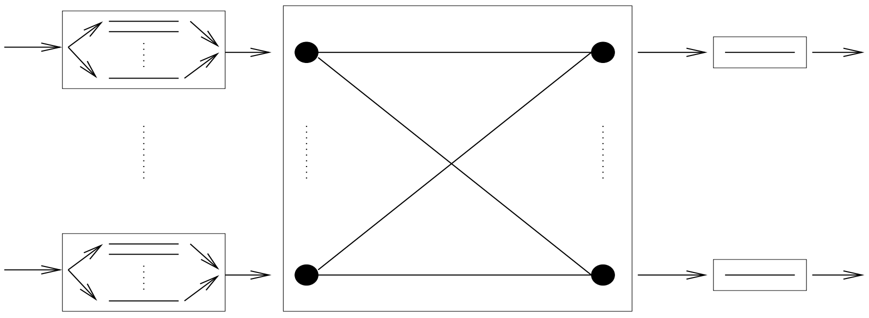

Figure 1: CIOQ switch - an example

buffer management policies in this model while focusing on deriving upper and lower bounds for the natural Longest Queue Drop policy.

2 Problem definition and notations

In the online CIOQ (Combined Input and Output Queued) switch problem, originally introduced in [15], we are given an $N \times N$ switch with N input ports and N output ports (see figure 1). Input port i ($i = 1, \ldots, N$) contains N virtual output FIFO queues (referred to as *input queues* henceforth), denoted $\{VO_{ij}\}_{j=1}^{N}$, with bounded capacities, where queue VO_{ij} is used to buffer the packets arriving at input port i and destined to output port j. Output port j ($j = 1, \ldots, N$) contains a bounded capacity FIFO queue (referred to as *output queue*), denoted O_j, to buffer the packets waiting to be transmitted through output port j. For each queue Q in the system, we denote by $B(Q)$ the capacity of the queue, and by $Q(t)$ the ordered set of packets stored in the queue at time step t. We denote by $h(Q(t))$ and $\min(Q(t))$ the packet at the head of queue Q, and the minimal value among packets in the queue, respectively.

Time proceeds in discrete time steps. We divide each time step into three phases. The first phase is the *arrival phase*, during which packets arrive online at the input ports. All packets have equal size, and each packet is labelled with a value, an input port, through which it enters the switch, and a destination output port, through which is has to leave the switch. We denote the finite sequence of online arriving packets by σ. For every packet $p \in \sigma$ we denote by $v(p)$, $in(p)$ and $out(p)$ its value, input port and output port, respectively. Online arriving packets can be enqueued in the input queues that correspond to their destination ports, without exceeding their capacities. Remaining packets must be discarded. We allow preemption, i.e. packets previously stored in the queues may be discarded in order to free space for newly arrived packets. The second phase at each time step is the *scheduling phase* during which packets are transferred from the input queues to the output queues. For a switch with speedup $S \geq 1$ at most S packets can be removed from each input port and at most S packets can be enqueued into the queue of each output port. This is done in S consecutive rounds, during each we compute a matching between input and output ports. We denote round s ($s = 1, \ldots, S$) at time step t by t_s. During the scheduling phase the capacity of the output queues may not be exceeded, and again preemption is allowed. The third phase at each time step is the *transmission phase*. During this phase a packet may be transmitted from the head of each output queue.

A switching algorithm $\mathcal{A}$ for the CIOQ switch problem is composed of two components, not necessarily independent. The first component is admission control; Algorithm $\mathcal{A}$ has to exercise an admission control strategy in all queues (input and output), that decides which packets are admitted into the queues and which packets are preempted. The second component is a scheduling strategy; During scheduling round t_s algorithm $\mathcal{A}$ first decides the set of eligible packets for transfer $E^{\mathcal{A}}(t_s) \subseteq \{h(VO_{ij}(t_s)) : (i,j) \in [N]^2\}$, i.e. a subset of the packets currently located at the head of the input queues. The set $E^{\mathcal{A}}(t_s)$ can be translated into the bipartite weighted graph $G^{\mathcal{A}}(t_s) = ([N],[N],E)$ such that $E = \{(in(p), out(p)) : p \in E^{\mathcal{A}}(t_s)\}$, where the weights on the edges correspond to the packet values, namely $w(e) = v(p)$ for $e = (in(p), out(p)) \in E$. Throughout the paper we refer to the edges in E and their corresponding packets interchangeably. Algorithm $\mathcal{A}$ then constructs a matching $M^{\mathcal{A}}(t_s) \subseteq G^{\mathcal{A}}(t_s)$ that corresponds of the set of packets that are transferred from input queues to output queues at round t_s. Given a finite sequence σ we denote by $\mathcal{A}(\sigma)$ the set of packets algorithm $\mathcal{A}$ transmitted from the switch. The goal is to maximize the total value of transmitted packets, denoted $V(\mathcal{A}(\sigma))$, i.e. maximize $V(\mathcal{A}(\sigma)) = \sum_{p \in \mathcal{A}(\sigma)} v(p)$. An online algorithm $\mathcal{A}$ is said to be *c-competitive* if and only if for every packet sequence σ, $V(\mathsf{Opt}(\sigma)) \leq c \cdot V(\mathcal{A}(\sigma))$ holds, where Opt denotes the optimal off-line algorithm for the problem.

3 A constant-competitive algorithm

In this section we define and analyze our algorithm for the CIOQ switch problem. We begin with a definition of a parameterized preemptive admission control policy GR_β for a single queue (figure 2). This policy is a generalization of the ordinary greedy policy from [12], that is obtained by setting $\beta = 1$. The latter will be denoted by GR. We then present our parameterized algorithm $\mathsf{SG}(\beta)$ (abbreviation for **Semi-Greedy(β)**) for the problem (figure 3). For simplicity of notation, we drop the parameter β for the remainder of this section, and use the abbreviated notation SG for the algorithm. The value of the parameter β will be determined later.

Algorithm GR_β [Single-Queue]

Enqueue a new packet p if:

- $|Q(t)| < B(Q)$ (the queue is not full).
- Or $v(p) > \beta \cdot \min(Q(t))$. In this case the smallest packet is discarded and p is enqueued.

Figure 2: Algorithm GR_β.

Algorithm SG exercises the greedy preemptive admission control policy GR on all input queues, and the modified admission control policy GR_β on all output queues. At each scheduling round SG considers only those packets that would be admitted to their destined output queue if transferred, namely packets whose destined output queue is either not full, or they have a value greater than β times the currently smallest packet in this queue. Algorithm SG then computes a maximum weighted matching in the corresponding bipartite graph. Note that according to SG operation, each packet that is transferred to an output queue is always accepted, and may preempt another packet. We now proceed to prove the competitive ratio of SG.

```
Algorithm SG(β) [CIOQ switch]

  1. Admission control:
     (a) Input queues: Use algorithm GR.
     (b) Output queues: Use algorithm $\mathsf{GR}_\beta$.
  2. Scheduling: at scheduling round $t_s$ do:
     (a) Set $E^{\mathsf{SG}}(t_s) = \{h(VO_{ij}(t_s)) : |O_j(t_s)| < B(O_j) \text{ or } v(h(VO_{ij}(t_s))) > \beta \cdot \min(O_j(t_s))\}$
     (b) Compute a maximum weighted matching $M^{\mathsf{SG}}(t_s) \subseteq G^{\mathsf{SG}}(t_s)$.
```

Figure 3: Algorithm $\mathsf{SG}(\beta)$.

Theorem 3.1 *Algorithm* SG *achieves constant competitive ratio. Specifically, for an optimal choice of the parameter β, algorithm* SG *is 8-competitive.*

Proof: The outline of the proof is as follows. Given an input sequence σ, we wish to construct a global weight function $w_\sigma : \mathsf{SG}(\sigma) \rightarrow \mathbb{R}^+$, that maps each packet transmitted by SG from the switch to a real value. In order to achieve a $(c+1)$-competitive ratio it is sufficient to prove the following:

1. $\sum_{p \in \mathsf{Opt}(\sigma) \setminus \mathsf{SG}(\sigma)} v(p) \leq \sum_{p \in \mathsf{SG}(\sigma)} w_\sigma(p)$.
2. $w_\sigma(p) \leq c \cdot v(p)$, for each packet $p \in \mathsf{SG}(\sigma)$, and for some global constant c.

Note that by the first condition the total weight that is mapped to packets which were transmitted by SG covers the total value lost by SG compared with Opt. By the second condition this is done by charging each packet with a weight that exceeds its own by no more than a constant factor. In the following we show how to construct w_σ.

The main obstacle of the proof is to handle the different transmission timing at input queues of SG compared with Opt, as well as the different packet sequences seen at output queues. The proof consists of several parts. In the first part we prove a main lemma concerning the operation of a single queue when transmission timing is the same. In the second part we apply this lemma to the input queues, after adjusting the transmission timing by creating dummy packets. Output queues are handled in the third part. Our construction is then concluded in the forth part, when we show how the dummy packets, which were added for the analysis, can be absorbed in the system.

Step 1: The basic mapping scheme (for a single queue in the system). We begin by introducing some additional notations. Consider the single-queue admission control policy GR_β, and let $\mathcal{A}$ denote any single-queue admission control policy that is exercised by Opt. Assume that GR_β and $\mathcal{A}$ do not operate under the same conditions, specifically, let σ_1 and σ_2 be the input packet sequences given to $\mathcal{A}$ and GR_β, and let T_1 and T_2 denote the time steps at which $\mathcal{A}$ and GR_β, respectively, transmit from the queue. We denote by $\sigma^{\mathcal{A}-\mathsf{GR}_\beta} = \{p \in \sigma_1 \cap \sigma_2 : p \in \mathcal{A}(\sigma_1) \setminus \mathsf{GR}_\beta(\sigma_2)\}$ the set of packets in the intersection of the sequences that were transmitted by $\mathcal{A}$ and not by GR_β. The following online-constructed local mapping will serve as the basic building block in the construction of our global weight function w_σ.

Lemma 3.2 *Let $\mathcal{A}$ be any admission control policy for a single queue exercised by* Opt. *Let σ_1, σ_2 be input packet sequences, and T_1, T_2 be transmission times that correspond to $\mathcal{A}$ and GR_β (for $\beta \geq 1$), respectively. If $T_1 \subseteq T_2$ then there exists a mapping $\mathsf{M_L} : \sigma^{\mathcal{A}-\mathsf{GR}_\beta} \to \mathsf{GR}_\beta(\sigma_2)$ such that the following properties hold:*

1. *$v(p) \leq \beta \cdot v(\mathsf{M_L}(p))$, for every packet $p \in \sigma^{\mathcal{A}-\mathsf{GR}_\beta}$.*
2. *Every packet in $\mathsf{GR}_\beta(\sigma_2) \backslash \mathcal{A}(\sigma_1)$ is mapped to at most twice.*
3. *Every packet in $\mathsf{GR}_\beta(\sigma_2) \cap \mathcal{A}(\sigma_1)$ is mapped to at most once.*

Proof: We construct the mapping $\mathsf{M_L}$ on-line following the operation of the admission control policies. We assume w.l.o.g that $\mathcal{A}$ does not preempt packets. Note that an optimal solution can always mark the packets that will eventually be transmitted out of the queue and enqueue only them, therefore rendering preemption redundant. For the remainder of the proof we denote the contents of the queue according to GR operation (respectively $\mathcal{A}$ operation) by Q_{GR} (respectively by $Q_{\mathcal{A}}$). We further denote by $\mathsf{M_L}^{-1}(p)$ the packets that are mapped to packet p (note that $|\mathsf{M_L}^{-1}(p)| \leq 2$ for every packet p). Given a packet $p \in Q_{\mathsf{GR}}(t)$ we denote by *potential*(p) the maximum number of packets that can be mapped to p according to the properties of the lemma, namely *potential*$(p) = 1$ for $p \in \mathcal{A}(\sigma_1)$, otherwise *potential*$(p) = 2$. We further denote by *available*$_t(p)$ the number of available mappings to p at time t, namely *potential*(p) minus the number of packets already mapped to p until time t. A packet $p \in Q_{\mathsf{GR}}(t)$ is called *fully-mapped* if no additional packet can be mapped to it, i.e. if *available*$_t(p) = 0$. The definition of the mapping $\mathsf{M_L}$ appears in figure 4.

Mapping $\mathsf{M_L} : \sigma^{\mathcal{A}-\mathsf{GR}} \to \mathsf{GR}(\sigma_2)$

1. Let Packet p arrive at time t; If p is discarded by GR but accepted by $\mathcal{A}$, map it to the packet closest to the head in queue Q_{GR} that is not fully-mapped.
2. Let $q \in Q_{\mathsf{GR}}(t)$ be a packet that is preempted during time t. Update $\mathsf{M_L}$ as follows:
   (a) If $q \in \mathcal{A}(\sigma_1)$ map it to the packet closest to the head that is not fully-mapped.
   (b) Do the same with respect to $\mathsf{M_L}^{-1}(q)$ (if exists).

Figure 4: Local mapping $\mathsf{M_L}$.

For simplicity of notation, we prove the lemma for GR, i.e. GR_1. The lemma will then follow directly for the general case, using the same construction, since for GR_β the ratio between a discarded packet and any packet in the queue is at most β. We begin by observing that the mapping $\mathsf{M_L}$ indeed adheres to the required properties of the lemma.

Claim 3.3 *The mapping $\mathsf{M_L}$ meets properties 1–3 of Lemma 3.2*

Proof: By induction on the time steps. First, observe that a packet is mapped to only if it is not fully-mapped, therefore properties 2–3 are satisfied. Second, by the definition of GR, when a packet p is discarded from the queue all other packets residing in the queue have higher values. By the induction hypothesis, this is also true with respect to $\mathsf{M_L}^{-1}(p)$. Therefore, all mappings concur with property 1. This completes the proof. ■

We now turn to characterize the packets in queue Q_{GR} whenever a mapped packet resides in the queue.

Claim 3.4 *Let $p \in Q_{\mathsf{GR}}(t)$ be a mapped packet in queue Q_{GR} at time step t. Then the following holds:*

1. *$h(Q_{\mathsf{GR}}(t))$ (the packet at the head) is mapped.*
2. *For every packet $q \in {\mathsf{M_L}}^{-1}(p)$ we have: $v(q) \leq v(h(Q_{\mathsf{GR}}(t)))$*

Proof: If packet p is at the head of queue Q_{GR} at time t then we are clearly done by Claim 3.3. Otherwise, let $t' \leq t$ be any time step at which packet p was mapped to. By the definition of $\mathsf{M_L}$ all the packets closer to the head than p were already fully-mapped at time t'. Furthermore, all those packets have higher values than the value of the packet (or packets) mapped to p at that time, according to the definition of GR. Since FIFO order is obtained, these properties continue to hold at time t, and the claim follows. ■

To complete the proof of Lemma 3.2 we need to prove that the definition of $\mathsf{M_L}$ is valid, i.e. that indeed whenever a packet is discarded, queue Q_{GR} contains packets that are not fully-mapped.

Claim 3.5 *The definition of the mapping $\mathsf{M_L}$ is valid, namely, at each time step the queue Q_{GR} contains unmapped packets as assumed in the definition of $\mathsf{M_L}$.*

Proof: We define a potential function $\Phi(t) = \sum_{p \in Q_{\mathsf{GR}}(t)} available_t(p)$, that counts the number of available mappings at every time step. We prove the claim by induction on the changes that occur in the system, i.e. packet arrivals and transmissions. Specifically, the correctness of the claim follows from the invariant inequality $\Phi(t) + |Q_{\mathcal{A}}(t)| \geq |Q_{\mathsf{GR}}(t)|$ that is shown to hold at each time step by induction. For the initial state the inequality clearly holds. We next denote by Δ_Φ, $\Delta_{\mathcal{A}}$ and Δ_{GR} the changes that occur in the values of Φ, $|Q_{\mathcal{A}}(t)|$, $|Q_{\mathsf{GR}}(t)|$, respectively. We examine all possible changes in the system, and show that the inequality continues to hold, either by proving it directly or by showing that $\Delta_\Phi + \Delta_{\mathcal{A}} \geq \Delta_{\mathsf{GR}}$.

1. **Packet $p \in \sigma_1 \cap \sigma_2$ arrives**: We distinguish three possible cases:
   (a) p **is accepted by GR while no packet is preempted**: Clearly, $\Delta_\Phi \geq \Delta_{\mathsf{GR}}$.
   (b) p **is rejected by GR**: If p is also rejected by $\mathcal{A}$ nothing changes. Otherwise, queue $Q_{\mathcal{A}}$ was not full before the arrival of p as opposed to queue Q_{GR}. Hence by the induction hypothesis before the packet arrival $\Phi > 0$. After we map a packet to p, $\Delta_\Phi + \Delta_{\mathcal{A}} = 0$ and the inequality holds.
   (c) p **is accepted by GR while a packet is preempted**: Clearly, $\Delta_{\mathsf{GR}} = 0$ since a packet is discarded only when queue Q_{GR} is full. Note that in either case (p is accepted or rejected by $\mathcal{A}$) $\Delta_\Phi + \Delta_{\mathcal{A}} \geq 0$ since the left-hand side of the inequality first increases by two and then decreases by at most two (case 2 in $\mathsf{M_L}$ definition). Therefore, the inequality continues to hold.
2. **Packet $p \in \sigma_2 \backslash \sigma_1$ arrives**: $\Delta_\Phi \geq \Delta_{\mathsf{GR}}$ whether p is accepted by GR or not.

3. **Packet** $p \in \sigma_1 \backslash \sigma_2$ **arrives**: $\Delta_{\mathcal{A}} \geq 0$, while other values remain unchanged.

4. **Transmission step** $t \in T_1 \cap T_2$: If queue Q_{GR} does not hold mapped packets, the inequality trivially holds, since in this case $\Phi(t) \geq |Q_{\mathsf{GR}}(t)|$. Otherwise, by Claim 3.4 the packet at the head of the queue is a mapped packet. If the packet at the head of the queue is a fully-mapped packet then after the transmission takes place $\Delta_{\mathcal{A}} = \Delta_{\mathsf{GR}} = -1$ while $\Delta_{\Phi} = 0$ and the inequality holds; Otherwise, by the definition of $\mathsf{M_L}$, there are no additional mapped packets in queue Q_{GR} and we are back to the previous case.

5. **Transmission step** $t \in T_2 \backslash T_1$: If the packet at the head of queue Q_{GR} is fully-mapped then after the transmission takes place $\Delta_{\mathsf{GR}} = -1$ while $\Delta_{\Phi} = \Delta_{\mathcal{A}} = 0$ and the inequality holds. Otherwise, after the transmission there are no mapped packets in the queue and, again, by the same considerations given in the previous case, the inequality goes on.

■

This completes the proof of Lemma 3.2. ■

Corollary 3.6 *Let* $\mathsf{M_L} : \sigma^{\mathcal{A}-\mathsf{GR}_\beta} \to \mathsf{GR}_\beta(\sigma_2)$ *be the local mapping defined in Lemma 3.2. Then we can construct a weight function* $w : \mathsf{GR}_\beta(\sigma_2) \to \mathbb{R}^+$ *such that the following properties hold:*

1. $\sum_{p \in \sigma^{\mathcal{A}-\mathsf{GR}_\beta}} v(p) \leq \sum_{p \in \mathsf{GR}_\beta(\sigma_2)} w(p)$.

2. $w(p) \leq 2\beta \cdot v(p)$, *for every packet* $p \in \mathsf{GR}_\beta(\sigma_2) \backslash \mathcal{A}(\sigma_1)$.

3. $w(p) \leq \beta \cdot v(p)$, *for every packet* $p \in \mathsf{GR}_\beta(\sigma_2) \cap \mathcal{A}(\sigma_1)$.

Proof: Directly from Lemma 3.2 by defining $w(p) = \sum_{q \in \mathsf{M_L}^{-1}(p)} v(q)$. ■

Step 2: Handling lost packets in input queues. Note that algorithm SG can lose packets by one of two ways. Packets can be discarded at the input queues, and packets stored in the output queues can be preempted in favor of higher value packets. In the following we show how to handle both incidents. Ideally, we would like to apply the weight function construction from Corollary 3.6 to each input queue separately to account for preempted packets. However, the condition specified in Lemma 3.2, that is needed for the construction of the local mapping $\mathsf{M_L}$, is not met; We have no guarantee that $T_1 \subseteq T_2$, namely that SG transmits from each input queue whenever Opt does. As we shall see this obstacle can be overcome with some extra cost.

To fix the problem described in the previous paragraph we first rectify the definition of $\mathsf{M_L}$. Looking closer into case 4 in the proof of Claim 3.5, it should be obvious that in order to maintain the invariant inequality defined in the proof of Claim 3.5, it is sufficient to remove a single mapping from one of the mapped packets in the queue whenever Opt transmits a packet from the queue while SG does not *and* the queue contains mapped packets. With that in mind, we handle such instances by releasing $\mathsf{M_L}$ mappings and moving them to a set of *dummy packets* (see figure 5). As a result, our $\mathsf{M_L}$ mapping for the input queues is now valid (although not all lost packets are accounted for). We define the corresponding weight function according to Corollary 3.6, denoted by

$w^1_\sigma(\cdot)$, in order to cover the total value lost by SG compared to Opt at the input queues, excluding the value of the dummy packets. However, we just shifted the problem further. For the analysis to work, we need to prove that the value of the dummy packets can be absorbed in the system, i.e. it can be assigned to real packets. This will be done shortly.

Creating dummy packets

1. For scheduling round t_s define:

$$\begin{aligned} S(t_s) &= \{(i,j) : VO_{ij} \text{ contains a mapped packet}\} \\ S_1(t_s) &= \{(i,j) \in S(t_s) : (i,j) \in M^{\mathsf{Opt}}(t_s) \wedge (i,j) \in G^{\mathsf{SG}}(t_s) \backslash M^{\mathsf{SG}}(t_s)\} \\ S_2(t_s) &= \{(i,j) \in S(t_s) : (i,j) \in M^{\mathsf{Opt}}(t_s) \wedge (i,j) \notin G^{\mathsf{SG}}(t_s)\} \end{aligned}$$

2. For each $(i,j) \in S_1(t_s) \cup S_2(t_s)$ do:
   (a) Let p be the mapped packet closest to the tail in input queue VO_{ij} at time t_s, and let $q \in \mathsf{M_L}^{-1}(p)$.
   (b) Create dummy packet $d_{ij}(t_s)$ such that $v(d_{ij}(t_s)) = v(q)$.
   (c) Set $\mathsf{M_L}(q) = \emptyset$.

Figure 5: Creating dummy packets.

Step 3: Handling lost packets in output queues. We define the modifications in the weight function $w^1_\sigma(\cdot)$, in case of packet preemption at the output queues. Let packet p preempt packet q at an output queue. We modify $w^1_\sigma(p) = w^1_\sigma(p) + w^1_\sigma(q) + v(q)$, in order to account for the value of packet q and the weight already assigned to it. We therefore cover all preempted packets at output queues.

Step 4: Covering dummy packets. In order to complete the definition of the weight function, we should handle the dummy packet created at step 3. The first stage is to account for all dummy packets $d_{ij}(t_s) \in S_1(t_s)$. From now on these dummy packets will be referred to as *dummy packets of the first kind*, while dummy packets created due to $S_2(t_s)$ will be referred to as *dummy packets of the second kind*. Recall that $S_1(t_s) \subseteq G^{\mathsf{SG}}(t_s)$, and that $S_1(t_s)$ forms a matching whose value is lower than the value of the maximum weighted matching $M^{\mathsf{SG}}(t_s)$. We modify the weight function $w^1_\sigma(\cdot)$ as follows. Let $p \in M^{\mathsf{SG}}(t_s)$ be a packet scheduled by SG at time step t_s. We modify $w^1_\sigma(p) = w^1_\sigma(p) + v(p)$. We thus absorbed the value of dummy packets of the first kind. Therefore, we may continue the analysis while ignoring these dummy packets.

We are now left only with the dummy packets of the second kind, i.e. packets of the form $d_{ij}(t_s) \in S_2(t_s)$. Denote by σ_j the sequence of packets scheduled by SG to output queue j throughout the operation of the algorithm. Further denote by $\sigma_j^{dummy} = \cup_{t_s:(i,j)\in S_2(t_s)} d_{ij}(t_s)$ all the dummy packets of the second kind that are destined to output queue O_j. Now recall that for every scheduling round t_s, $S_2(t_s) \cap G^{\mathsf{SG}}(t_s) = \emptyset$. By the definition of algorithm SG this means that $v(h(VO_{ij}(t_s))) \leq \beta \cdot \min(O_j(t_s))$. By Claim 3.4 it follows that $v(d_{ij}(t_s)) \leq \beta \cdot \min(O_j(t_s))$. Therefore, dummy packet $d_{ij}(t_s)$ would have been rejected from queue O_j had it been sent to it.

We can now apply the local mapping construction $\mathsf{M_L}$ from Lemma 3.2 to queue O_j with arrival sequence $\sigma_j \cup \sigma_j^{dummy}$ in order to map all dummy packets in σ_j^{dummy} to real packets in the output queue O_j. Looking closer into the definition of the local mapping $\mathsf{M_L}$ (case 2), this can be done while mapping each real packet in the output queue at most once. Using Corollary 3.6 we define the corresponding weight function, denoted $w_\sigma^2(\cdot)$. Since we have now accounted for all dummy packets of the second kind, we may consider them as absorbed in the system and ignore them.

Defining the global weight function - recap and conclusion. We now define our global weight function by $w_\sigma(p) = w_\sigma^1(p) + w_\sigma^2(p)$, for every packet $p \in \mathsf{SG}(\sigma)$. Figure 6 summarizes the construction of the weight functions we used.

Weight function $w_\sigma : \mathsf{SG}(\sigma) \to \mathbb{R}^+$

1. Apply mapping scheme $\mathsf{M_L}$ to each input queue. Add dummy packets to ensure proper definition of $\mathsf{M_L}$. Denote the corresponding weight function $w_\sigma^1(\cdot)$.
2. Whenever packet p preempts packet q at an output queue, modify $w_\sigma^1(p) = w_\sigma^1(p) + w_\sigma^1(q) + v(q)$.
3. Map the value of dummy packets of the first kind created at time t_s to real packets scheduled at that time. Modify $w_\sigma^1(\cdot)$ accordingly.
4. Use mapping scheme $\mathsf{M_L}$ to map dummy packets of the second kind to real packets in output queues. Denote the corresponding weight function $w_\sigma^2(\cdot)$.
5. Define $w_\sigma(p) = w_\sigma^1(p) + w_\sigma^2(p)$.

Figure 6: Construction of the global weight function w_σ - summary.

So far we have shown how to account for the total value of lost packets through the use of the weight function. To complete the proof of Theorem 3.1, it remains to bound the weight assigned to each packet.

Claim 3.7 *By setting $\beta = 3$, the following holds:*

1. $V(\mathsf{Opt}(\sigma) \backslash \mathsf{SG}(\sigma)) \leq \sum_{p \in \mathsf{SG}(\sigma)} w_\sigma(p)$.
2. $w_\sigma(p) \leq 8 \cdot v(p)$, *for each packet* $p \in \mathsf{SG}(\sigma) \backslash \mathsf{Opt}(\sigma)$.
3. $w_\sigma(p) \leq 7 \cdot v(p)$, *for each packet* $p \in \mathsf{SG}(\sigma) \cap \mathsf{Opt}(\sigma)$.

Proof: The first part follows directly from the construction of the weight function. For the second part, consider any packet $p \in \mathsf{SG}(\sigma) \backslash \mathsf{Opt}(\sigma)$. Clearly, according to our construction, $w_\sigma^1(p) \leq 3 \cdot v(p)$ before packet p reaches its destined output queue (a weight of at most $2v(p)$ can be assigned to packet p while it resides in the input queue, and an additional weight of $v(p)$ to account for dummy packets of the first kind when it is scheduled to be transferred to the output queue). If p preempts a packet q once it arrives at the output queue, then $w_\sigma^1(p)$ increases by $w_\sigma^1(q) + v(q)$ (note that in this case $v(p) > \beta \cdot v(q)$). Denote $c = \max_{q \in \mathsf{SG}(\sigma) \backslash \mathsf{Opt}(\sigma)} w_\sigma^1(q)/v(q)$. In order to bound c we require that $3 + \frac{c+1}{\beta} \leq c$. In addition, while packet p resides in the output queue, $w_\sigma^2(p) \leq \beta \cdot v(p)$, since it can be assigned with the value of a single dummy packet of the second kind, which is bounded by $\beta \cdot v(p)$. Therefore, $w_\sigma(p) = w_\sigma^1(p) + w_\sigma^2(p) \leq (c+\beta)v(p)$, for every packet $p \in \mathsf{SG}(\sigma) \backslash \mathsf{Opt}(\sigma)$. We arrive at the following minimization problem:

$$\begin{aligned} \min \ & c + \beta \\ s.t. \ & 3 + \frac{c+1}{\beta} \leq c. \end{aligned} \qquad (*)$$

Optimizing over β, we conclude that the optimum is obtained for $\beta = 3$, $c = 5$, and $w_\sigma(p) \leq 8 \cdot v(p)$.

For the third part, note that for a packet $p \in \mathsf{SG}(\sigma) \cap \mathsf{Opt}(\sigma)$, $w_\sigma^1(p) \leq v(p)$ while it resides in the input queue (rather than $2v(p)$ in the previous case), therefore $w_\sigma(p) \leq 7 \cdot v(p)$, using the same analysis. ■

We conclude that:

$$\begin{aligned} V(\mathsf{Opt}(\sigma)) &= V(\mathsf{Opt}(\sigma) \cap \mathsf{SG}(\sigma)) + V(\mathsf{Opt}(\sigma) \backslash (\mathsf{SG}(\sigma))) \\ &\leq V(\mathsf{Opt}(\sigma) \cap \mathsf{SG}(\sigma)) + \sum_{p \in \mathsf{SG}(\sigma)} w_\sigma(p) \\ &\leq V(\mathsf{Opt}(\sigma) \cap \mathsf{SG}(\sigma)) \\ &\quad + 7 \cdot V(\mathsf{Opt}(\sigma) \cap \mathsf{SG}(\sigma)) + 8 \cdot V(\mathsf{SG}(\sigma) \backslash \mathsf{Opt}(\sigma)) \\ &= 8 \cdot V(\mathsf{SG}(\sigma)). \end{aligned}$$

This completes the proof of Theorem 3.1. ■

4 Concluding remarks

- Recall that our proposed algorithm, SG, computes a maximum weighted matching in a bipartite graph during each scheduling round. In many real-world systems, especially distributed ones, computing a maximal weighted matching, by greedily adding the largest possible edge at each step, is substantially faster. We note that if we use a maximal weighted matching in each scheduling round in SG, our analysis will almost remain the same. Specifically, our constructed weight function $w_\sigma^1(\cdot)$ will slightly change as follows. During each scheduling round, a weight of at most $2v(p)$ can be charged to a packet p (rather than $v(p)$ in the original construction) to account for dummy packets of the first kind. As a result, the constraint (*) in the minimization problem in the proof of Claim 3.7 will be changed to $4 + \frac{c+1}{\beta} \leq c$. We still obtain a constant-competitive algorithm with a slightly worse ratio of 9.48.
- Another variation of the model we consider is the less-restrictive model of CIOQ switches with priority queues, rather than FIFO queues. Note that the value of the optimal solution remains the same in this relaxed model. Therefore, a constant-competitive upper bound for the priority queuing model can be directly derived from our result, by an algorithm that simulates ours, and can only outperform it.

References

[1] W. A. Aiello, Y. Mansour, S. Rajagopolan, and A. Rosén. Competitive queue policies for differentiated services. In *Proceedings of the IEEE INFOCOM 2000*, pages 431–440.

[2] S. Albers and M. Schmidt. On the performance of greedy algorithms in packet buffering. In *Proc. 36th ACM Symp. on Theory of Computing*, pages 35–44, 2004.

[3] N. Andelman, Y. Mansour, and A. Zhu. Competitive queueing policies for QoS switches. In *Proc. 14th ACM-SIAM Symp. on Discrete Algorithms*, pages 761–770, 2003.

[4] M. Andrews, B. Awerbuch, A. Fernández, J. Kleinberg, T. Leighton, and Z. Liu. Universal stability results for greedy contention-resolution protocols. In *Proc. 37th IEEE Symp. on Found. of Comp. Science*, pages 380–389, 1996.

[5] Y. Azar and A. Litichevskey. Maximizing throughput in multi-queue switches. In *Proc. 12th Annual European Symposium on Algorithms*, pages 53–64, 2004.

[6] Y. Azar and Y. Richter. Management of multi-queue switches in QoS networks. In *Proc. 35th ACM Symp. on Theory of Computing*, pages 82–89, 2003.

[7] Y. Azar and Y. Richter. The zero-one principle for switching networks. In *Proc. 36th ACM Symp. on Theory of Computing*, pages 64–71, 2004.

[8] N. Bansal, L. Fleischer, T. Kimbrel, M. Mahdian, B. Schieber, and M. Sviridenko. Further improvements in competitive guarantees for QoS buffering. In *Proc. 31st International Colloquium on Automata, Languages, and Programming*, pages 196–207, 2004.

[9] A. Birman, H. R. Gail, S. L. Hantler, Z. Rosberg, and M. Sidi. An optimal service policy for buffer systems. *Journal of the Association Computing Machinery (JACM)*, 42(3):641–657, 1995.

[10] A. Borodin, J. Kleinberg, P. Raghavan, M. Sudan, and D. Williamson. Adversarial queuing theory. In *Proc. 28th ACM Symp. on Theory of Computing*, pages 376–385, 1996.

[11] E. L. Hahne, A. Kesselman, and Y. Mansour. Competitive buffer management for shared-memory switches. In *Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 53–58, 2001.

[12] A. Kesselman, Z. Lotker, Y. Mansour, B. Patt-Shamir, B. Schieber, and M. Sviridenko. Buffer overflow management in QoS switches. In *Proc. 33rd ACM Symp. on Theory of Computing*, pages 520–529, 2001.

[13] A. Kesselman and Y. Mansour. Loss-bounded analysis for differentiated services. In *Proc. 12th ACM-SIAM Symp. on Discrete Algorithms*, pages 591–600, 2001.

[14] A. Kesselman, Y. Mansour, and R. van Stee. Improved competitive guarantees for QoS buffering. In *Proc. 11th Annual European Symposium on Algorithms*, pages 361–372, 2003.

[15] A. Kesselman and A. Rosén. Scheduling policies for CIOQ switches. In *Proceedings of the 15th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 353–362, 2003.

[16] Z. Lotker and B. Patt-Shamir. Nearly optimal fifo buffer management for diffserv. In *Proc. 21st ACM Symp. on Principles of Distrib. Computing*, pages 134–143, 2002.

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