Online Packet Scheduling for CIOQ and Buffered Crossbar Switches[*]

Kamal Al-Bawani
Department of Computer Science
RWTH Aachen University
Germany
kbawani@cs.rwth-aachen.de

Matthias Englert
DIMAP and Department of Computer Science
University of Warwick
UK
englert@dcs.warwick.ac.uk

Matthias Westermann
Department of Computer Science
TU Dortmund
Germany
matthias.westermann@cs.tu-dortmund.de

ABSTRACT

We consider the problem of online packet scheduling in Combined Input and Output Queued (CIOQ) and buffered crossbar switches. In the widely used CIOQ switches, packet buffers (queues) are placed at both input and output ports. An $N \times N$ CIOQ switch has N input ports and N output ports, where each input port is equipped with N queues, each of which corresponds to an output port, and each output port is equipped with only one queue. In each time step, arbitrarily many packets may arrive at each input port, and only one packet can be transmitted from each output port. Packets are transferred from the queues of input ports to the queues of output ports through the internal fabric. Buffered crossbar switches follow a similar design, but are equipped with additional buffers in their internal fabric. In either model, our goal is to maximize the number or, in case the packets have weights, the total weight of transmitted packets.

Our main objective is to devise online algorithms that are both competitive and efficient. We improve the previously known results for both switch models, both for unweighted and weighted packets.

For unweighted packets, Kesselman and Rosén (J. Algorithms '06) give an online algorithm that is 3-competitive for CIOQ switches. We give a faster, more practical algorithm achieving the same competitive ratio. In the buffered crossbar model we also show 3-competitiveness, improving the previously known ratio of 4.

For weighted packets, we give 5.83- and 14.83-competitive algorithms with an elegant analysis for CIOQ and buffered crossbar switches, respectively. This improves upon the previously known ratios of 6 and 16.24.

[*]The second and third author's work was supported by ERC Grant Agreement No. 307696.

SPAA '16, July 11 - 13, 2016, Pacific Grove, CA, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ISBN 978-1-4503-4210-0/16/07...$15.00

DOI: http://dx.doi.org/10.1145/2935764.2935792

Keywords

Online algorithms; competitive analysis; scheduling; buffer management; CIOQ switches; buffered crossbar switches

1. INTRODUCTION

In the widely used *Combined Input and Output Queued (CIOQ) switches*, packet buffers (queues) are placed at both input and output ports. An $N \times N$ CIOQ switch has N input ports and N output ports. Each input port is equipped with N queues, each of which corresponds to an output port, and each output port is equipped with only one queue. The switching fabric of the switch connects the input ports with the output ports and are used to transfer packets from the queues of input ports to the queues of output ports. Figure 1 depicts an example of a CIOQ switch.

When a packet arrives at a CIOQ switch, it is first tagged with the following information: the value that represents its class of service, i.e., its priority, the input port through which it enters the switch, and the output port through which it has to leave the switch. Packets proceed inside the switch in the following way. They are first stored in the queues of the input ports, such that each packet is stored in the queue that corresponds to its output port. After that, they are transferred from input to output ports through the switching fabric, and reside in the queues of the output ports until they are eventually sent out of the switch. However, queues inside the switch are of limited capacities and there may be bursts of packets arriving which exceed the capacities. Thus, queues may overflow. Typically, packets are transferred through the switching fabric with a rate that is S times the rate of transmission, i.e., they are transferred through the switching fabric over S cycles of speed in each time step. We call S the speedup of the switch. It is worth noting here that we consider non-FIFO queues, i.e., packets can be stored in and released from queues in any arbitrary order.

Closely related to CIOQ switches, another type of switch architecture, the so-called *buffered crossbar switches*, is obtained by adding further queues at the crosspoints of the switching fabric. More specifically, for every queue at the input ports, an additional queue is placed at the switching fabric and dedicated to accommodate packets that are transferred from the input queue before they later on are transferred further to the corresponding output port. The number of those crossbar queues is proportional to the number of crosspoints, i.e., N^2, but it has been shown that the adoption of crossbar queues significantly decreases the

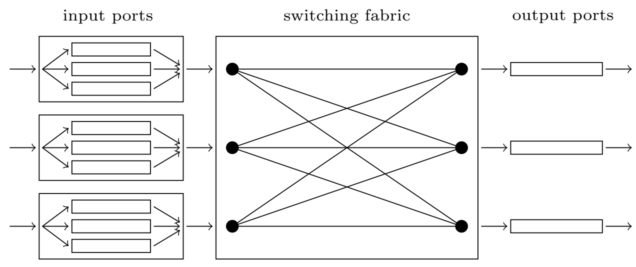

Figure 1: CIOQ switch — An example with $N = 3$

scheduling overhead of CIOQ switches. Figure 2 depicts an example of a buffered crossbar switch.

Packet scheduling in both CIOQ and buffered crossbar switches has been extensively studied in the networking literature (see, e.g., [9, 10]). The design and analysis of scheduling algorithms in that line of research is mostly based on prior assumptions about the traffic distribution, e.g., Poisson-like distributions. However, it has been shown that Internet traffics do not necessarily adhere to such particular distributions (see, e.g., [26, 28]). We do not make any prior assumptions about the arrival behavior of packets, and instead resort to the framework of competitive analysis [27], which is the typical worst-case analysis used to assess the performance of *online* algorithms, i.e., algorithms whose input is revealed piece by piece over time, and the decision they make in each time step is irrevocable.

In competitive analysis, the benefit, in our case the switch's throughput, of an online algorithm is compared to the benefit of an optimal algorithm OPT which is assumed to know the entire input sequence in advance. An online algorithm ONL is called *c-competitive* if, for each input sequence σ, the benefit of OPT over σ is at most c times the benefit of ONL over σ. The value c is also called the *competitive ratio* of ONL.

1.1 Our Contribution

Our objective in the CIOQ model is twofold: to devise online algorithms that are both competitive and efficient. All online algorithms known for this problem are based on computing a maximum matching in each scheduling cycle, and thus are far from being efficient for real-world switches. We present new algorithms that are significantly more efficient and yet achieve the best competitive ratios known for this problem.

In each scheduling cycle, a bipartite graph is induced from the current configuration of the input and output queues, where the vertices of the left-hand side correspond to the input ports, and the vertices of the right-hand side correspond to the output ports. An edge (i, j) indicates that a packet can be transferred from the i-th input port to the j-th output port. Clearly, a matching in this graph corresponds to an admissible schedule for the current scheduling cycle.

We present two online algorithms in this model: Greedy Matching (GM) for the unit-value case, i.e., where all packets have the same value, and Preemptive Greedy (PG) for the general-value case. Both algorithms are based on greedy maximal matching computations, i.e., we construct a matching incrementally by adding edges, one by one, until no more edges can be added. This is much more efficient than computing the maximum matchings that have been used in previous works. Moreover, computing maximal matchings complies more with the current practice in distributed systems where packet scheduling has to perform in real time.

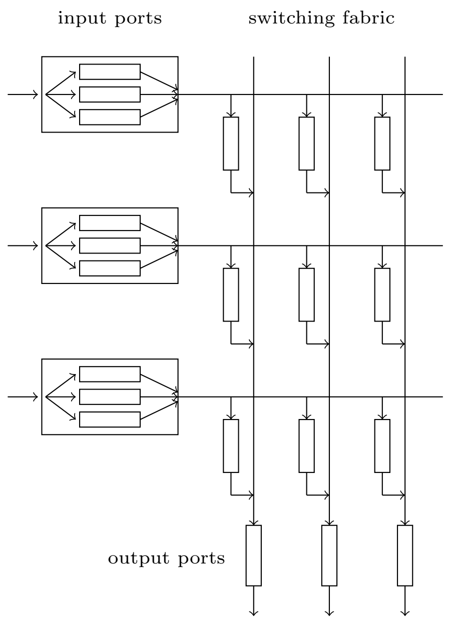

Figure 2: Buffered crossbar switch — An example with $N = 3$

With respect to competitiveness, we show in Section 2.1 that GM is 3-competitive for any speedup, and thus it achieves the best competitive ratio known for this problem [22]. In Section 2.2, we show that PG has a competitive ratio of $3 + 2\sqrt{2} \approx 5.828$ for any speedup, which improves upon the previously known competitive ratio of 6 [23].

To obtain these results in an elegant way, we manipulate the queues of an optimal offline algorithm such that certain invariants in relation to our online algorithms are maintained. The techniques we use in the analysis of GM and PG also allow us to achieve improved upper bounds in the related model of buffered crossbar switches. For the unit-value case of this model, Kesselman et al. [20] present a greedy algorithm, which we call Crossbar Greedy Unit (CGU), with a competitive ratio of 4 for any speedup. We improve on this result and show that CGU is indeed 3-competitive. For the general-value case, they give an algorithm that is 16.24-competitive for any speedup. We present a slightly different algorithm, Crossbar Preemptive Greedy (CPG), and show that it achieves a competitive ratio of $12 + 2\sqrt{2} \approx 14.828$ for any speedup.

A similar analysis technique has been successfully used by Jeż et al. [18] for other packet scheduling related problems. However, in that work the buffer is manipulated in such a way that the optimal algorithm and the online algorithm always have an identical buffer content. In our proofs, we maintain different invariants.

1.2 Related Work

For the general-value case of CIOQ switches with FIFO queues, i.e., packets are stored and released in order of their arrival, Kesselman and Rosén [22] give two algorithms with competitive ratios of $4 \cdot S$ and $8 \cdot \min\{k, 2\log\alpha\}$, where k is the number of distinct packet values and α is the ratio between the largest and the smallest packet value. The latter result was improved by Azar and Richter [7] where they give an algorithm with a competitive ratio of 8 for any speedup. Kesselman et al. [21] show that this algorithm is 7.47-competitive. For the buffered crossbar model with FIFO queues, Kesselman et al. [19] give a 19.95-competitive algorithm for any speedup.

The model of input queued switches (IQ) consists of m queues of the same capacity B and one output port. It is worth noticing that CIOQ switches generalize this model since if the speedup is 1 and only one input port is in use, CIOQ and IQ switches become equivalent. In the unit-value case of the IQ model, Azar and Richter [6] provide a lower bound of $2-1/m$ on the competitive ratio of any deterministic algorithm. Clearly, this lower bound carries over to the CIOQ model. Further results on the IQ model can be found in [3, 4, 5, 8, 17].

The problem of packet scheduling (also known as buffer management) has also been studied under several other models. For example, the multi-queue model with shared memory [1, 14, 15], the multi-queue model with class segregation [2], the single-queue (FIFO) model [12], and the bounded delay model, where packets have deadlines besides their values [11, 24]. Comprehensive and up-to-date surveys on this problem and its variants can be found in [13, 16, 25].

1.3 Models and Notations

We consider a CIOQ switch with N input ports and N output ports. Each input port has N queues and each output port has one queue. We call the queues at the input ports the *input queues* and those at the output ports the *output queues*. An input queue that is placed at input port i ($i = 1, \dots, N$) and corresponds to output port j ($j = 1, \dots, N$) is denoted by Q_{ij}. An output queue that is placed at output port j ($j = 1, \dots, N$) is denoted by Q_j. For any input or output queue Q, the capacity of Q is denoted by $B(Q)$, and $Q(t)$ denotes the set of packets that reside in Q at time t. All queues in the switch are non-FIFO, i.e., packets may be stored in and released from queues in any arbitrary order.

An input instance of this problem is a sequence of packets arriving at the switch in an online manner, i.e., packets that arrive at time t are not known before t. All packets have the same size. For each packet p in the input sequence, $v(p)$, $\text{arr}(p)$, $\text{in}(p)$, and $\text{out}(p)$ denote p's value, arrival time, input port, and output port, respectively, where $\text{in}(p)$ and $\text{out}(p)$ take on values between 1 and N.

Time is discretized into steps of unit length, and each of these time steps is divided into three phases; namely, arrival, scheduling and transmission phases. In the *arrival phase*, arbitrarily many packets may arrive at the switch. An arriving packet p is either *accepted* and thus inserted in queue Q_{ij}, where $i = \text{in}(p)$ and $j = \text{out}(p)$, or it is *rejected*, i.e., discarded.

In the *scheduling phase*, a set of packets that are stored in input queues are transferred to their corresponding output queues through the switching fabric of the switch. These transfers take place in internal time cycles which we call the *scheduling cycles*. We say that a switch has a speedup S when it is capable of performing S scheduling cycles within a single time step. we denote the s-th cycle of time step t by $t[s]$, for $s = 1, \dots, S$. In any scheduling cycle, a matching between input and output ports is computed, such that at most one packet is released from each input port and at most one packet is admitted to each output port. More specifically, when a packet p is transferred from queue Q_{ij} in scheduling cycle $t[s]$, it is forwarded through the switching fabric to queue Q_j, and no packet except p is released from input port i or forwarded to output port j in $t[s]$.

Finally, in the *transmission phase*, at most one packet is sent out from each output queue, i.e., transmitted to its next destination on the network.

Preemption is allowed, i.e., a packet that was previously inserted into a queue can be *preempted*, i.e., discarded, before it is sent. Therefore, a packet may be lost in one of two occasions: rejection upon its arrival, or preemption after getting stored in a queue.

The benefit made by an online algorithm ONL on an input sequence σ is denoted by $\text{ONL}(\sigma)$, and is defined as the total value of packets that ONL sends from the output queues. We aim at maximizing this benefit. An algorithm that knows the entire input beforehand and makes the maximum benefit on any sequence is denoted as OPT. An online algorithm ONL is *c-competitive* if $\text{OPT}(\sigma) \le c \cdot \text{ONL}(\sigma)$ for any input sequence σ.

Finally, the time that precedes the first arrival of the sequence is denoted as time 0. We assume that the queues of any algorithm are all empty at time 0.

Buffered crossbar switches are obtained by adding further queues at the crosspoints of the switching fabric. A crossbar queue that is placed at the crosspoint of input port i ($i = 1, \dots, N$) and output port j ($j = 1, \dots, N$) is denoted by C_{ij}. Again, all queues in the switch are non-FIFO, i.e., packets may be stored in and released from queues in any arbitrary order.

All other notations and conventions of the CIOQ model hold also for the buffered crossbar model. However, each cycle of the scheduling phase in the buffered crossbar model is divided into two subphases: the *input subphase* and the *output subphase*. In the input subphase, packets can be transferred from any input queue Q_{ij} to its corresponding crossbar queue C_{ij}, such that at most one packet is transferred from each input port i. In the output subphase, packets can be transferred from any crossbar queue C_{ij} to its corresponding output queue Q_j, such that at most one packet is transferred to each output port j.

2. CIOQ SWITCHES

2.1 Unit-Value Case

In this case, all packets have the same value 1. Thus, our goal is to maximize the number of transmitted packets. In the following, we present the Greedy Matching algorithm (GM).

Arrival phase: For every arriving packet p with $\text{in}(p) = i$ and $\text{out}(p) = j$, accept p if Q_{ij} is not full; otherwise, reject p.

Scheduling phase: In every scheduling cycle $t[s]$, a bipartite graph $G_{t[s]} = (U, V, E)$ is induced from the current configuration of the switch, where $U = \{u_1, \dots, u_N\}$, $V = \{v_1, \dots, v_N\}$ and an edge $(u_i, v_j) \in E$ if and only if the input queue Q_{ij} is not empty and the output queue Q_j is not full at $t[s]$.

A greedy matching $M_{t[s]}$ is then computed on $G_{t[s]}$ in the following way: Start with an empty matching and iterate over all edges of E. Add an edge e to the current matching if e does not violate the matching property.

After $M_{t[s]}$ is computed, for each edge $(u_i, v_j) \in M_{t[s]}$, the head packet of Q_{ij} is transferred to Q_j.

Transmission phase: For every non-empty output queue Q_j, send the packet at the head of Q_j.

The next theorem shows that GM is 3-competitive for any speedup.

THEOREM 1. *The competitive ratio of* GM *is at most 3 for any speedup.*

From now on, we fix an input sequence σ, and, for any input or output queue Q, we reserve the notation Q for the online algorithm and use Q^* to denote the corresponding queue of the offline algorithm OPT.

First, without loss of generality, we assume that OPT is greedy in transmission events, i.e, it sends a packet from an output queue as long as its queue is not empty. Obviously, as OPT knows in advance which packets it is going to send, holding packets back in output queues, rather than sending them as early as possible, cannot improve its benefit.

Now, we modify OPT in a way that does not decrease its benefit of σ. Specifically, at the end of each scheduling cycle $t[s]$, i.e., immediately after OPT performs its scheduling policy, we apply the following two modifications on the configuration of OPT in the given order:

MODIFICATION 2.1.1. *Suppose that* GM *transfers a packet from Q_{ij} and* OPT *does not transfer any packet from Q^*_{ij} in $t[s]$. If Q^*_{ij} is not empty in $t[s]$, we release a packet p from Q^*_{ij} and send it directly out of the switch, i.e., through an imaginary channel. In this case, p is called a* privileged packet of Type 1 *and it contributes to the benefit of the optimal algorithm.*

MODIFICATION 2.1.2. *Suppose that* OPT *transfers a packet p to Q^*_j and* GM *does not transfer any packet to Q_j in $t[s]$. If Q_j is not full in $t[s]$, we send p directly out of the switch. In this case, we call p a* privileged packet of Type 2 *and it contributes to the benefit of the optimal algorithm.*

Clearly, these modifications do not decrease the benefit of the optimal algorithm. They can only make it stronger by allowing it to send packets directly from input ports to outside the switch without being enqueued in output ports. The input and output queues will respectively become shorter in this case and thus the optimal algorithm may accept more new packets.

Before we continue, we introduce further notations. We call packets that OPT schedules through the normal channels, i.e., they are not privileged, *normal* packets. We use S^* and P^* to denote the sets of OPT's normal and privileged packets, respectively. Clearly, the benefit of OPT is given by $|P^*| + |S^*|$. We also use S to denote the set of packets sent by GM. Thus, we want to show that $|P^*| + |S^*| \le 3\,|S|$.

We now show how to derive the competitive ratio of 3. First, we show in Lemma 1 how Modifications 2.1.1 and 2.1.2 are used to preserve the following invariant: At any time, each queue in GM is no shorter than its counterpart in OPT. Therefore, for any time step t and output port j, if OPT sends a packet from j in t, GM must also send a packet from j in t. Hence, $|S^*| \le |S|$. After that, we show by Lemma 3 that $|P^*| \le 2\,|S|$. Thus, the proof of Theorem 1 follows directly from these two lemmas.

LEMMA 1. *For any $i, j \in \{1, \ldots, N\}$ and any time t, the following inequalities hold:*

I1. $|Q^*_{ij}(t)| \le |Q_{ij}(t)|$

I2. $|Q^*_j(t)| \le |Q_j(t)|$

PROOF. Inequalities I1 and I2 can be shown by a simple induction over time. Let the induction base be at time 0, i.e., before the sequence starts. All queues are empty at this time and thus I1 and I2 hold. Assume now that they hold for any time up to time $t-1$. We next show that they hold for t as well.

Clearly, queues change in arrival, scheduling and transmission events only. So, we assume that t is the time immediately after an event τ that is either an arrival, scheduling or transmission event.

Assume τ is an arrival event. Clearly, output queues do not change in arrival events and thus I2 holds for this case. For I1, the only critical case is when the arriving packet is rejected by GM and accepted by OPT. However, the input queue of GM must be full in this case and thus I1 still holds.

Now, let τ be a scheduling event. Here, the only critical case for I1 is when GM transfers a packet from Q_{ij} while OPT does not transfer anything from Q^*_{ij}. However, either Q^*_{ij} is empty in this case or it cannot happen due to Modification 2.1.1. For I2, the only critical case is when OPT inserts a packet into Q^*_j while GM does not insert anything into Q_j. However, either Q_j is full in this case or it cannot happen due to Modification 2.1.2.

Finally, assume τ is a transmission event. Clearly, the input queues do not change in transmission events and thus I1 holds for this case. For I2, the only critical case is when GM sends a packet from Q_j while OPT does not send anything from Q^*_j. However, since we assume that OPT is greedy at sending, its output queue must be empty in this case and thus I2 still holds. □

The following lemma shows that if Modification 2.1.2 takes place, GM must transfer a packet from the same input port.

LEMMA 2. *Suppose that, in $t[s]$,* OPT *transfers a packet p from Q^*_{ij} to Q^*_j and* GM *does not transfer any packet to Q_j. If Q_j is not full in $t[s]$, then* GM *transfers a packet p' from $Q_{ij'}$ in $t[s]$, where $j' \ne j$.*

PROOF. Recall the bipartite graph $G_{t[s]}$ and the corresponding matching $M_{t[s]}$ which are induced from the configuration of GM right before performing the scheduling cycle $t[s]$.

Assume that Q_j is not full in $t[s]$. By Inequality I1 of Lemma 1, since OPT transfers p from Q^*_{ij} in $t[s]$, GM must have at least one packet in Q_{ij}. Therefore, an edge (u_i, v_j) must be in E. Nevertheless, since GM does not transfer any packet to Q_j, (u_i, v_j) is not in $M_{t[s]}$. Since $M_{t[s]}$ is a maximal matching, there must exist an edge $(u_i, v_{j'})$, for $j' \ne j$, such that $(u_i, v_{j'}) \in M_{t[s]}$. Hence, a packet p' is transferred from $Q_{ij'}$ in $t[s]$. □

Lemma 3. *The following inequality holds:*

$$|P^*| \le 2\,|S| \ .$$

Proof. We carry out the following mapping scheme from P^* to S in each scheduling cycle $t[s]$.

1. Let p be a privileged packet of Type 1 that is sent by OPT from Q^*_{ij} in $t[s]$. By Modification 2.1.1, GM transfers a packet p' from Q_{ij} in $t[s]$. Map p to p'.

2. Let p be a privileged packet of Type 2 that is sent by OPT from Q^*_{ij}. By Lemma 2, GM transfers a packet p' from $Q_{ij'}$ in $t[s]$, where $j' \neq j$. Map p to p'.

Clearly, this mapping scheme is feasible, i.e., each packet $p \in P^*$ is mapped to a packet $q \in S$. Furthermore, at most two privileged packets can be mapped to each packet $q \in S$. To see that, let q be a packet transferred by GM from Q_{ij} in a scheduling cycle $t[s]$. Clearly, q can get mapped only in $t[s]$, provided that OPT sends privileged packets in this time. By Modifications 2.1.1 and 2.1.2, OPT can send at most 2 privileged packets from input port i in $t[s]$: one of Type 1 if OPT's queue of Q^*_{ij} is not empty, and one of Type 2 if it transfers a packet from another queue $Q^*_{ij'}$. Thus, these two privileged packets are mapped to q. $\square$

2.2 General-Value Case

For the case of arbitrary packet values, we present the Preemptive Greedy algorithm (PG) that is a variant of a 6-competitive algorithm given by Kesselman and Rosén [23]. We show next that PG has a competitive ratio of $3 + 2\sqrt{2} \approx 5.828$ for any speedup.

Before we describe PG formally, we introduce further notations. Let $g_{ij}(t)$ denote the packet with the greatest value in Q_{ij} at time t, and $l_{ij}(t)$ (resp. $l_j(t)$) denote the packet with the least value in Q_{ij} (resp. Q_j) at time t. Additionally, let $\beta \ge 1$ be a parameter of the algorithm that will be determined later.

Arrival phase: If a packet p arrives at time t with $\text{in}(p) = i$ and $\text{out}(p) = j$, accept p if

$$|Q_{ij}(t)| < B(Q_{ij}) \bigvee v(l_{ij}(t)) < v(p) \ ;$$

otherwise, reject p. If p is accepted while $|Q_{ij}(t)| = B(Q_{ij})$, then $l_{ij}(t)$ is preempted.

Scheduling phase: In every scheduling cycle $t[s]$, a weighted bipartite graph $G_{t[s]} = (U, V, E, w)$ is induced from the current configuration of the switch, where $U = \{u_1, \ldots, u_N\}$, $V = \{v_1, \ldots, v_N\}$, an edge $(u_i, v_j) \in E$ if and only if

$$|Q_{ij}(t[s])| > 0 \bigwedge \Big(|Q_j(t[s])| < B(Q_j) \bigvee v(g_{ij}(t[s])) > \beta\, v(l_j(t[s])) \Big) \ ,$$

and the weight of (u_i, v_j) is given by $w(u_i, v_j) = v(g_{ij}(t[s]))$.

A greedy matching $M_{t[s]}$ is then computed on $G_{t[s]}$ in the following way: Start with an empty matching and iterate over all edges of E in a descending order of their weights. Add an edge e to the current matching if e does not violate the matching property.

After $M_{t[s]}$ is computed, for each edge $(u_i, v_j) \in M_{t[s]}$, the packet $g_{ij}(t[s])$ is transferred to Q_j. If $g_{ij}(t[s])$ is transferred while $|Q_j(t[s])| = B(Q_j)$, then $l_j(t[s])$ is preempted.

Transmission phase: For every non-empty output queue Q_j, send the packet with the greatest value in Q_j.

As described above, unlike the algorithm given in [23], PG computes a maximal weighted matching in each scheduling cycle rather than a maximum weighted matching.

Theorem 2. *For $\beta = \sqrt{2} + 1$, the competitive ratio of* PG *is at most $3 + 2\sqrt{2} \approx 5.828$ for any speedup.*

First, we fix an input sequence σ. Without loss of generality, we make the following assumptions about OPT:

A1. OPT is greedy in scheduling and transmission events, i.e, when it transfers or sends a packet p from an input or output queue, it chooses p as the one with the greatest value in the queue.

A2. OPT is work-conserving at output ports, i.e., it sends a packet from every non-empty output queue in each transmission event.

Obviously, as OPT knows in advance which packets it is going to send, it does not matter for OPT in which order these packets are released from queues or when they are transmitted from output queues. Now, based on the greediness of both PG and OPT, we make another harmless assumption:

A3. In all input and output queues, PG and OPT store packets in the order of their values, where the packet with the greatest value is at the queue's head and the one with the least value is at the queue's tail.

Similarly to the unit-value case, we modify OPT without decreasing its benefit. Specifically, at the end of each scheduling cycle $t[s]$, i.e., immediately after OPT performs its scheduling policy, we apply the following modifications on the configurations of OPT:

Modification 2.2.1. *Suppose that* PG *transfers a packet from Q_{ij} and* OPT *does not transfer any packet from Q^*_{ij} in $t[s]$. If Q^*_{ij} is not empty in $t[s]$, we release the head packet p of Q^*_{ij}, i.e., the packet with the greatest value in Q^*_{ij}, and send it directly out of the switch. In this case, we call p a* privileged packet of Type 1 *and it contributes to the benefit of the optimal algorithm.*

Modification 2.2.2. *If* OPT *transfers a packet p to Q^*_j and* PG *transfers a packet q to Q_j in $t[s]$ with $v(q) < v(p)$, we send p directly out of the switch. In this case, we call p a* privileged packet of Type 2 *and it contributes to the benefit of the optimal algorithm.*

Modification 2.2.3. *Suppose that* OPT *transfers a packet p to Q^*_j and* PG *does not transfer any packet to Q_j in $t[s]$. If Q_j is not full in $t[s]$ or $v(p) > \beta\, v(l_j(t[s]))$, we send p directly out of the switch. In this case, we call p a* privileged packet of Type 3 *and it contributes to the benefit of the optimal algorithm.*

Note that Modifications 2.2.2 and 2.2.3 are closely related and dealing with them separately is only for ease of exposition.

Let $\delta_{ij}(k,t)$ (resp. $\delta_j(k,t)$) denote the packet at position k in Q_{ij} (resp. Q_j) at time t, where position 1 corresponds to the head of the queue. Let $\delta^*_{ij}(k,t)$ and $\delta^*_j(k,t)$ be the corresponding notations for OPT. The following lemma shows that each packet in an OPT's input queue is aligned to a packet of the same or greater value in the corresponding input queue of PG, and each packet p in an OPT's output queue is aligned to a packet q in the corresponding output queue of PG, where $v(p) \le \beta v(q)$.

LEMMA 4. *For any $i, j \in \{1, \ldots, N\}$ and any time t, the following inequalities hold:*

I1. $v(\delta^*_{ij}(k,t)) \le v(\delta_{ij}(k,t))$, for $k = 1, \ldots, |Q^*_{ij}(t)|$

I2. $v(\delta^*_j(k,t)) \le \beta\, v(\delta_j(k,t))$, for $k = 1, \ldots, |Q^*_j(t)|$

PROOF. Inequalities I1 and I2 can be shown by a simple induction over time. Let the induction base be at time 0, i.e., before the sequence starts. All queues are empty at this time and thus I1 and I2 hold. Assume now that they hold for any time up to time $t-1$. We next show that they hold for time t as well.

Clearly, input queues change only in arrival, scheduling or transmission events. So, we assume that t is the time immediately after an event τ which is either an arrival, scheduling or transmission event. In the following, we will argue only for $I2$. The argument for $I1$ is analogous, and we will put the main differences between [ ] at the respective positions.

Before we start, we say that a packet $p \in Q^*_j(t)$ is in a legal alignment, if p is aligned in time t to a packet $q \in Q_j(t)$ with $v(p) \le \beta v(q)$. Clearly, it suffices to show that any packet $p \in Q^*_j(t)$ is in a legal alignment. We distinguish between two cases:

Case I2.1 $p \in Q^*_j(t-1)$. Thus, by induction, p is aligned in $t-1$ to a packet $q \in Q_j(t-1)$ with $v(p) \le \beta v(q)$ [resp. $v(p) \le v(q)$]. We need to show in this case that p either remains in the same alignment in t or it changes to another legal alignment. Assumption A3 implies that any packet p from $t-1$ either remains in its position in time t, moves one step ahead (if a packet, that is in front of p, is sent from the queue) or moves one step back (if a new packet is inserted in front of p).

Assume now that p remains in its position in t but q moves. Note that neither q nor any packet in front of it can be released from the queue in time t; otherwise, by Assumption A2 [resp. Modification 2.2.1], some packet would be also released from Q^*_j, which makes p move one step ahead. Thus, q can only move back in t. In this case, however, the packet q' that is directly in front of q is aligned with p. Since $v(q) \le v(q')$, p is again in a legal alignment.

Next, assume that p moves one step ahead in t. In this case, p either remains in a legal alignment with q (in case q moves ahead as well) or it aligns with a packet that is in front of q in $t-1$ and thus makes again a legal alignment.

Finally, assume that p moves one step back in t. Thus, a packet p' must be inserted in front of p, implying that $v(p) \le v(p')$. Note that the insertion of p' happens only in one of two cases: (i) if a packet r with $v(r) \ge v(p')$ is inserted into Q_j (by Modifications 2.2.2), or (ii) if Q_j is full in t and $v(p') \le \beta v(l_j(t))$ (by Modifications 2.2.3). Let k denote the position of the alignment (p,q) in time $t-1$. In case (i), either (1) r is inserted in a position $k' \le k$, and thus p will be aligned again with q in t, or (2) r is inserted in a position $k' > k$, and thus p will be aligned with some packet q' in t. Clearly, the second case implies that $v(r) \le v(q')$. Since $v(p) \le v(p') \le v(r)$, then $v(p) \le v(q')$. Hence, p is in a legal alignment in either case.

In case (ii), since Q_j is full in t, p must be aligned with some packet q' in t. Clearly, $v(l_j(t)) \le v(q')$. Moreover, since $v(p') \le \beta v(l_j(t))$, $v(p) \le v(p') \le \beta v(q')$. Thus, p makes a legal alignment with q'. [The respective cases for I1 are: case (i) p' is also inserted into Q_{ij}, thus $r = p'$ in the above argument, and case (ii) Q_{ij} is full in t and $v(l_j(t)) \ge v(p')$.]

Case I2.2 $p \notin Q^*_j(t-1)$. Thus, p is a new packet that is inserted in the queue in time t. Again, note that the insertion of p into Q^*_j happens only in one of two cases: (i) if a packet r with $v(r) \ge v(p)$ is inserted into Q_j (by Modification 2.2.2), or (ii) if Q_j is full in t and $v(p) \le \beta v(l_j(t))$ (by Modifications 2.2.3). In case (ii), since Q_j is full in t, p must be aligned with a packet q in t. Since $v(p) \le \beta v(l_j(t))$, $v(p) \le \beta v(q)$. Thus, p makes a legal alignment with q.

Now, consider case (i). Let k denote the position at which p is inserted. If $k = 1$, p is aligned with the most valuable packet in Q_j in t. Since r is in Q_j in time t, p must be aligned with a packet of value at least $v(r) \ge v(p)$. Now suppose $k > 1$. Let p' be the packet that is directly in front of p in t. Clearly, $p' \in Q^*_j(t-1)$ and $v(p) \le v(p')$. Furthermore, let q' be the packet aligned with p' in time $t-1$. Thus, $v(p) \le v(p') \le \beta v(q')$. Additionally, let q be the packet at position k in Q_j in time $t-1$ (assume $q = \emptyset$, if this is an empty position in Q_j).

Note that (1) r is inserted in position k, and thus p will be aligned with r in t, (2) r is inserted in a position $k' < k$, and thus p will be aligned with q' in t, or (3) r is inserted in a position $k' > k$, and thus p will be aligned with q in t. Clearly, the last case implies that $q \ne \emptyset$ and that $v(q) \ge v(r) \ge v(p)$. Therefore, we have $v(p) \le v(r)$ in the first case, $v(p) \le \beta v(q')$ in the second, and $v(p) \le v(q)$ in the third. Hence, p is in a legal alignment in any case.

[The respective cases for I1 are: case (i) p is also inserted into Q_{ij}, thus $r = p$ in the above argument, and case (ii) Q_{ij} is full in t and $v(l_j(t)) \ge v(p)$.] ☐

Similarly to the analysis of the unit-value case, granting OPT with privileged packets must be done carefully, so that the total value of privileged packets remains within a certain factor of the total value of packets that PG sends. Obviously, each privileged packet of Type 1 can be paired with a packet that PG transfers from the same input queue. In the following two lemmas, we show that such a pairing is feasible for privileged packets of Type 2 and 3 as well. Of course, as packets of PG may be preempted after being transferred to output queues, some pairs can be destructed. However, we will show in Lemma 7 how to fix this problem.

LEMMA 5. *If* OPT *transfers a packet p from Q^*_{ij} to Q^*_j and* PG *transfers a packet q to Q_j in $t[s]$ with $v(q) < v(p)$, then* PG *transfers a packet p' from $Q_{ij'}$ in $t[s]$ with $j' \ne j$ and $v(p') \ge v(p)$.*

PROOF. Recall the bipartite graph $G_{t[s]}$ and the corresponding matching $M_{t[s]}$ which are induced from the configuration of PG right before performing the scheduling cycle $t[s]$.

By Inequality I1 of Lemma 4, since OPT transfers p from Q^*_{ij} in $t[s]$, PG must have at the head of Q_{ij} a packet r with $v(r) \geq v(p)$. Obviously, $v(r) > v(q)$ and thus $q \neq r$. As a result, q must be transferred from an input queue $Q_{i'j}$ with $i' \neq i$. Moreover, since q is inserted in Q_j, the edge $(u_{i'}, v_j) \in E$, and either $|Q_j(t[s])| < B(Q_j)$ or $v(q) > \beta v(l_j(t[s]))$. Thus, it holds also for r that either $|Q_j(t[s])| < B(Q_j)$ or $v(r) > \beta v(l_j(t[s]))$. Hence, the edge $(u_i, v_j) \in E$ as well, and clearly $w(u_i, v_j) \geq w(u_{i'}, v_j)$. This implies that (u_i, v_j) is considered before $(u_{i'}, v_j)$ during the computation of $M_{t[s]}$. However, since (u_i, v_j) is not in the matching, the node u_i must have been matched before considering (u_i, v_j), and thus there exists an edge $(u_i, v_{j'})$, for $j' \neq j$, that is inserted in the matching before considering (u_i, v_j). As a result, a packet p' is transferred from $Q_{ij'}$, and it must hold that $w(u_i, v_{j'}) \geq w(u_i, v_j)$. Hence, $v(p') \geq v(r) \geq v(p)$. □

The proof of the following lemma is analogous to that of Lemma 5.

LEMMA 6. *Suppose that, in $t[s]$, OPT transfers a packet p from Q^*_{ij} to Q^*_j and PG does not transfer any packet to Q_j. If Q_j is not full in $t[s]$ or $v(p) > \beta v(l_j(t[s]))$, then PG transfers a packet p' from $Q_{ij'}$ in $t[s]$ with $j' \neq j$ and $v(p') \geq v(p)$.*

Now, recall Inequality I2 of Lemma 4. It implies that if OPT sends a packet of value v from some output queue at some time, PG must send a packet of at least v/β from the same output queue at the same time. Let S (resp. S^*) denote the set of all packets that PG (resp. OPT) sends from output queues. Thus,

$$\sum_{p \in S^*} v(p) \leq \beta \sum_{p \in S} v(p) \ .$$

Moreover, let P^* denote the set of all privileged packets, of all types, that OPT sends directly out of the switch. The next lemma shows that

$$\sum_{p \in P^*} v(p) \leq \frac{2\beta}{\beta - 1} \sum_{p \in S} v(p) \ .$$

Thus, we can conclude the competitive ratio of PG as follows

$$\begin{aligned} \text{OPT}(\sigma) &= \sum_{p \in S^*} v(p) + \sum_{p \in P^*} v(p) \\ &\leq \beta \sum_{p \in S} v(p) + \frac{2\beta}{\beta - 1} \sum_{p \in S} v(p) \\ &= \left(\beta + \frac{2\beta}{\beta - 1}\right) \text{PG}(\sigma) \ . \end{aligned}$$

Finally, it is easy to verify that the optimal value for β is $\sqrt{2} + 1$, resulting in a competitive ratio of $3 + 2\sqrt{2} \approx 5.828$.

LEMMA 7. *The following inequality holds:*

$$\sum_{p \in P^*} v(p) \leq \frac{2\beta}{\beta - 1} \sum_{p \in S} v(p) \ .$$

PROOF. We consider the following mapping scheme:

1. Let p be a privileged packet of Type 1 that is sent by OPT from Q^*_{ij} in scheduling cycle $t[s]$. By Modification 2.2.1, PG transfers a packet p' from Q_{ij} in $t[s]$, and by Inequality I1 of Lemma 4, $v(p) \leq v(p')$. Map p to p'.
2. Let p be a privileged packet of Type 2 that is sent by OPT from Q^*_{ij} in scheduling cycle $t[s]$. By Lemma 5, PG transfers a packet p' from $Q^*_{ij'}$ in $t[s]$ with $j' \neq j$ and $v(p) \leq v(p')$. Map p to p'.
3. Let p be a privileged packet of Type 3 that is sent by OPT from Q^*_{ij} in scheduling cycle $t[s]$. By Lemma 6, PG transfers a packet p' from $Q^*_{ij'}$ in $t[s]$ with $j' \neq j$ and $v(p) \leq v(p')$. Map p to p'.
4. Let q be a packet that is preempted from an output queue Q_j by another packet p'. For each privileged packet p that is mapped to q, re-map p to p'.

As shown above, this mapping scheme is feasible, i.e., each packet $p \in P^*$ is mapped to a packet $p' \in S$. Now, it remains to show that the total value of privileged packets that are mapped to each packet $p' \in S$ is at most $\frac{2\beta}{\beta-1} v(p')$.

For any packet $p' \in S$, p' can get mapped in two events: when it is scheduled and when it preempts a packet from an output queue.

Assume that p' is scheduled from $Q_{ij'}$ to $Q_{j'}$ during scheduling cycle $t[s]$. Now, assume that OPT transfers a packet from Q^*_{ij} to Q^*_j during $t[s]$. Clearly, we can only send one privileged packet p_1 of Type 1 from $Q^*_{ij'}$ in $t[s]$ (in case $j \neq j'$). Furthermore, we can only send from Q^*_{ij} either a privileged packet p_2 of Type 2 (in case PG transfers a packet q to Q_j with $v(q) < v(p_2)$), or a privileged packet p_3 of Type 3 (in case PG does not transfer any packet to Q_j). Hence, at most two privileged packets may be sent during $t[s]$ from each input port i. Since privileged packets are mapped only to packets that are transferred by PG from the same input port during the same scheduling cycle, at most two packets from $\{p_1, p_2, p_3\}$ can be mapped to p'. Furthermore, as shown in the mapping scheme above, the value of any of these privileged packets is at most the value of p'. Thus, the total value of privileged packets that are mapped to p' when it is scheduled is at most $2\,v(p')$.

Assume now that p' is the m-th packet in a chain of packets $q_0, \ldots, q_m$ in which packet q_n preempts packet q_{n-1}, for $1 \leq n \leq m$. Let $x(q_n)$ denote the total value of privileged packets that are mapped to a packet q_n after it preempts q_{n-1}. Thus, the total value of privileged packets that are mapped to p' is given by $x(q_m)$. Note that q_0 does not preempt any packet and thus the total value of privileged packets that are mapped to q_0 is at most $2\,v(q_0)$. Thus, $x(q_m)$ can be given by the following recursion:

$$\begin{aligned} &x(q_0) \leq 2\,v(q_0) \ \text{ and} \\ &x(q_n) \leq 2\,v(q_n) + x(q_{n-1}) \ , \text{ for } 0 < n \leq m \ . \end{aligned}$$

Solving this recursion, we obtain that

$$x(q_m) \leq 2 \sum_{n=0}^{m} v(q_n) \ .$$

Note also that $v(q_{n-1}) \leq v(q_n)/\beta$, for $1 \leq n \leq m$. Hence, we can rewrite $x(q_m)$ as follows:

$$\begin{aligned} x(q_m) &\leq 2v(q_m) \sum_{n=0}^{m} (1/\beta)^n \\ &\leq \frac{2\beta}{\beta - 1} v(q_m) \ . \end{aligned}$$

□

3. BUFFERED CROSSBAR SWITCHES

3.1 Unit-Value Case

For the case where all packets have value 1, Kesselman et al. [20] considered the following algorithm, which we call Crossbar Greedy Unit (CGU). The arrival and transmission phases of CGU are the same as those of GM (Section 2.1). In the scheduling phase, CGU works as follows.

> **Scheduling phase:** We divide every scheduling cycle $t[s]$ into two subphases:
>
> - **Input subphase**: For each input port i, choose an arbitrary input queue Q_{ij} which satisfies
>
> $$|Q_{ij}(t[s])| > 0 \bigwedge |C_{ij}(t[s])| < B(C_{ij}) \ ,$$
>
> and transfer its head packet.
>
> - **Output subphase**: For each output queue Q_j, choose an arbitrary crossbar queue C_{ij} which satisfies
>
> $$|Q_j(t[s])| < B(Q_j) \bigwedge |C_{ij}(t[s])| > 0 \ ,$$
>
> and transfer its head packet.

The next theorem shows that CGU is 3-competitive for any speedup.

THEOREM 3. *The competitive ratio of* CGU *is at most 3 for any speedup.*

First, we fix an input sequence σ. Again, we modify OPT in a way that does not decrease its benefit over σ. Specifically, at the end of each scheduling cycle $t[s]$, i.e., immediately after OPT performs its scheduling policy, we apply the following modifications on the configuration of OPT in the given order:

> MODIFICATION 3.1.1. *Suppose that* CGU *transfers a packet from Q_{ij} and* OPT *does not transfer any packet from $Q^*_{ij} \neq \emptyset$ in $t[s]$. We transfer a packet p from Q^*_{ij} in $t[s]$. If C^*_{ij} is not full in $t[s]$, p is transferred to C^*_{ij}. Otherwise, p is sent directly out of the switch. In either case, p is called a* privileged packet *and it contributes to the benefit of the optimal algorithm.*

> MODIFICATION 3.1.2. *Suppose that* CGU *transfers a packet to C_{ij} and* OPT *does not transfer any packet to C^*_{ij} in $t[s]$. If C^*_{ij} is not full in $t[s]$ and no privileged packet is transferred to C^*_{ij} by Modification 3.1.1 in $t[s]$ (possibly because* CGU *transfers from Q_{ij} while Q^*_{ij} is empty), we generate a new packet and insert it into C^*_{ij}. Such a new packet is called an* extra packet of Type 1 *and it contributes to the benefit of the optimal algorithm.*

> MODIFICATION 3.1.3. *Suppose that* OPT *transfers a packet from C^*_{ij} and* CGU *does not transfer any packet from C_{ij} in $t[s]$. If C_{ij} is not empty in $t[s]$, we generate a new packet and insert it into C^*_{ij}. Such a new packet is called an* extra packet of Type 2 *and it contributes to the benefit of the optimal algorithm.*

Note that extra packets are not used in the analysis of the algorithms presented in Section 2 for the CIOQ model. Next, we show how the above modifications are used to show a set of invariants that is different from the invariants shown in Section 2.1.

LEMMA 8. *For any time t and any $i, j \in \{1, \dots, N\}$, the following inequalities hold:*

I1. $|Q_{ij}(t)| \geq |Q^*_{ij}(t)|$

I2. $|C^*_{ij}(t)| \geq |C_{ij}(t)|$

PROOF. We show Inequalities I1 and I2 by a simple induction over time. Let the induction base be at time 0, i.e., before the sequence starts. All queues are empty at this time and all inequalities hold. Assume now that they hold for any time up to time $t-1$. We next show that they hold for t as well.

Clearly, input and crossbar queues change only in arrival and scheduling events. So, we assume that t is the time immediately after an event τ which is either an arrival or a scheduling event.

Assume τ is an arrival event. Clearly, crossbar queues do not change in arrival events and thus I2 holds for this case. For I1, the only critical case is when the arriving packet is rejected by CGU and accepted by OPT. However, the input queue of CGU must be full in this case and thus I1 still holds.

Now, let τ be a scheduling event. Here, the only critical case for I1 is when CGU transfers a packet from Q_{ij} while OPT does not transfer any packet from Q^*_{ij}. However, either Q^*_{ij} is empty in this case or it cannot happen due to Modification 3.1.1. For I2, the first critical case is when CGU inserts a packet into C_{ij} while OPT does not insert any packet into C^*_{ij}. However, either C^*_{ij} is full in this case or it cannot happen due to Modification 3.1.2. The second critical case for I2 is when OPT transfers a packet from C^*_{ij} while CGU does not transfer any packet from C_{ij}. However, either C_{ij} is empty in this case or the size of C^*_{ij} does not decrease due to Modification 3.1.3. □

In the following, we use $S^*_{t[s]}$ to denote the set of OPT's *normal* packets in the input subphase of $t[s]$. These are packets that OPT schedules through the normal channels, i.e., they are not privileged, and are part of the original input sequence, i.e., they are not extra. On the other hand, we use $S_{t[s]}$ to denote the set of packets that CGU schedules in the input subphase of cycle $t[s]$, i.e., from input queues to crossbar queues.

LEMMA 9. *For any scheduling cycle $t[s]$, $|S^*_{t[s]}| \leq |S_{t[s]}|$.*

PROOF. We want to show that in the input subphase of any scheduling cycle $t[s]$, if OPT transfers a normal packet from an input port i, CGU also transfers a packet from i.

Assume that OPT transfers a normal packet p from Q^*_{ij} (to C^*_{ij}) in $t[s]$. Thus, by I1 and I2 of Lemma 8, Q_{ij} is not empty and C_{ij} is not full in $t[s]$ (Note that OPT would not schedule a packet to a full crossbar queue, as all packets are of the same value). Hence, CGU transfers a packet from either Q_{ij} or another $Q_{ij'}$ in $t[s]$. □

Let $P^*_{t[s]}$ denote the set of OPT's privileged and extra packets (of either type) that occur in scheduling cycle $t[s]$.

We consider the following mapping scheme from $P^*_{t[s]}$ to $S_{t[s]}$. For packets that are inserted in CGU's output queues, we use the notion of a *marked* packet. Initially, all packets are unmarked.

1. Let p be a privileged packet that is transferred by OPT from Q^*_{ij} in scheduling cycle $t[s]$. By Modification 3.1.1, CGU transfers a packet q from Q_{ij} in $t[s]$. Map p to q.

2. Let p be an extra packet of Type 1 that is inserted into C^*_{ij} in the input subphase of scheduling cycle $t[s]$. By Modification 3.1.2, CGU transfers a packet q into C_{ij} in $t[s]$. Map p to q.

3. Let p be an extra packet of Type 2 that is inserted into C^*_{ij} in the output subphase of scheduling cycle $t[s]$. By Modification 3.1.3, C_{ij} is not empty in $t[s]$, and OPT transfers a packet p' to Q^*_j. Thus, Q^*_j is not full right before $t[s]$. Now, let q be the first unmarked packet in Q_j, i.e., the nearest to the queue's head. Map p to q and then mark q. Note that q can be the packet that CGU may insert into Q_j in $t[s]$.

Next, we show that the mapping scheme is feasible, i.e., each packet $p \in P^*_{t[s]}$ is mapped to a packet $q \in S_{t[s]}$. Clearly, Steps 1 and 2 are feasible. We show now that Step 3 is feasible as well. Let $M_j(t)$ denote the set of marked packets in Q_j at time t, for any $1 \leq j \leq N$. We first show the following lemma.

LEMMA 10. *At any time t, $|M_j(t)| \leq |Q^*_j(t)|$.*

PROOF. We show the claim by induction over the scheduling and transmission events. Clearly, $M_j(t)$ and $Q^*_j(t)$ change only in these events.

Assume first that t is a transmission event. The only critical scenario in this event is that OPT sends a packet from Q^*_j while CGU does not send a marked packet. If that happens, then either CGU sends an unmarked packet or it does not send any packet at all. The first case cannot happen while $|M_j(t)| > 0$ since marked packets are always at the front of the queue. The second case is safe because it implies that Q_j is empty and thus $|M_j(t)| = 0$.

Now, assume that t is a scheduling event. The only critical scenario in this event is that a packet q is marked in Q_j while OPT does not insert any packet into Q^*_j. However, according to Step 3 of the mapping scheme, marking q implies that OPT transfers a packet p' from C^*_{ij} to Q^*_j. Thus, this scenario cannot happen in scheduling events. □

Now, to show that Step 3 of the mapping scheme is feasible, we need to show that at least one packet is unmarked in Q_j in the scheduling cycle $t[s]$. For the sake of contradiction, assume that all packets in Q_j are marked or Q_j is empty in $t[s]$. The first thing that follows from this assumption is that CGU does not insert any packet into Q_j in $t[s]$ (because otherwise the inserted packet would be initially unmarked).

Recall from Step 3 that C_{ij} is not empty in $t[s]$. Thus, since no packet is inserted into Q_j, Q_j must be full in $t[s]$. Hence, since all packets are marked by assumption, $|M_j(t)| = B(Q_j)$, where t is the time right before $t[s]$. Thus, by Lemma 10, $|Q^*_j(t)| = B(Q^*_j)$ as well. However, this contradicts with the fact that OPT inserts p' into Q^*_j in $t[s]$. Hence, at least one packet is unmarked in Q_j in the scheduling cycle $t[s]$, and thus Step 3 is feasible.

LEMMA 11. *For any scheduling cycle $t[s]$, $|P^*_{t[s]}| \leq 2\,|S_{t[s]}|$.*

PROOF. As shown above, the mapping scheme is feasible for each scheduling cycle $t[s]$. So, it remains to show that at most two packets from $P^*_{t[s]}$ are mapped to any packet $q \in S_{t[s]}$.

Consider a packet $q \in S_{t[s]}$. Let Q_{ij} be the input queue from which q is transferred in the scheduling cycle $t[s]$. Obviously, q may get mapped by: (i) a privileged packet p that is transferred from Q^*_{ij} in $t[s]$, (ii) an extra packet p' of Type 1 that is inserted into C^*_{ij} in the input subphase of $t[s]$, and (iii) an extra packet p'' of Type 2 that is inserted into $C^*_{i'j}$ in the output subphase of $t[s]$, with $i \neq i'$. Therefore, at most three packets can be mapped to q during its entire lifespan.

Assume now, for the sake of contradiction, that both p and p' are mapped to q. According to Modification 3.1.2, since an extra packet p' is inserted into C^*_{ij}, C^*_{ij} cannot be full in $t[s]$. However, by Modification 3.1.1, p must be transferred to C^*_{ij} in this case and not directly to outside the switch. Hence, by Modification 3.1.2, no extra packet is generated in $t[s]$ in this case and thus p' does not exist, leading to a contradiction. □

Now, as CGU does not preempt packets, each packet which CGU schedules in an input subphase must be eventually sent, and thus it contributes to the benefit of CGU. Hence, $\text{CGU}(\sigma) = \sum_{t[s]} |S_{t[s]}|$. Furthermore, note that $\text{OPT}(\sigma) = \sum_{t[s]} |S^*_{t[s]}| + |P^*_{t[s]}|$. Therefore, the proof of Theorem 3 follows immediately from Lemma 9 and 11.

3.2 General-Value Case

For the case of arbitrary packet values, we present the Crossbar Preemptive Greedy algorithm (CPG) that is a variant of a 16.24-competitive algorithm given by Kesselman et al. [20].

Recall the notations $g_{ij}(t)$, $l_{ij}(t)$, and $l_j(t)$ that we used with algorithm PG (Section 2.2). Let $gc_{ij}(t)$ and $lc_{ij}(t)$ be the corresponding notations for crossbar queue C_{ij}, i.e., the packet with the greatest value and the packet with the least value, respectively, in C_{ij} at time t. Additionally, let $\beta \geq 1$ and $\alpha \geq 1$ be two parameters of the algorithm that will be determined later. If $\beta = \alpha$, our algorithm will be the same as the algorithm given in [20]. However, we show that to minimize the competitive ratio for this algorithm, these two parameters must take on different values.

The arrival and transmission phases of CPG are the same as those of PG. In the scheduling phase, CPG works as follows.

Scheduling phase: We divide every scheduling cycle $t[s]$ into two subphases:

- **Input subphase**: For each input port i, let J be defined as follows:

$$J = \Big\{ j : |Q_{ij}(t[s])| > 0 \bigwedge \Big(|C_{ij}(t[s])| < B(C_{ij}) \bigvee v(g_{ij}(t[s])) > \beta\, v(lc_{ij}(t[s])) \Big) \Big\} \ .$$

  If $J \neq \emptyset$, choose Q_{ij} such that for all $j' \in J$,

$$j \in J \bigwedge v(g_{ij}(t[s])) \geq v(g_{ij'}(t[s])) \ .$$

  Transfer $g_{ij}(t[s])$ to C_{ij}. If $|C_{ij}(t[s])| = B(C_{ij})$, preempt $lc_{ij}(t[s])$ first.

- **Output subphase**: For each output queue Q_j, choose a crossbar queue C_{ij} such that for all $i' \neq i$,

$$|C_{ij}(t[s])| > 0 \bigwedge v(gc_{ij}(t[s])) \geq v(gc_{i'j}(t[s])) \ .$$

If the following condition is satisfied

$$|Q_j(t[s])| < B(Q_j) \bigvee v(gc_{ij}(t[s])) > \alpha\, v(l_j(t[s])),$$

transfer $gc_{ij}(t[s])$ to Q_j. If $|Q_j(t[s])| = B(Q_j)$, preempt $l_j(t[s])$ first.

Note that all ties in CPG are broken arbitrarily.

THEOREM 4. *For $\beta = 2\sqrt{2} - 1$ and $\alpha = 2\sqrt{2}$, the competitive ratio of* CPG *is at most $12 + 2\sqrt{2} \approx 14.828$ for any speedup.*

The proof of Theorem 4 is omitted due to space limitations.

4. REFERENCES

[1] W. Aiello, A. Kesselman, and Y. Mansour. Competitive buffer management for shared-memory switches. *ACM Transactions on Algorithms*, 5(1):Article 3, 2008.

[2] K. Al-Bawani and A. Souza. Buffer overflow management with class segregation. *Information Processing Letters*, 113(4):145–150, 2013.

[3] S. Albers and M. Schmidt. On the performance of greedy algorithms in packet buffering. *SIAM Journal on Computing*, 35(2):278–304, 2006.

[4] Y. Azar and A. Litichevskey. Maximizing throughput in multi-queue switches. *Algorithmica*, 45(1):69–90, 2006.

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

[6] Y. Azar and Y. Richter. Management of multi-queue switches in QoS networks. *Algorithmica*, 43:81–96, 2005.

[7] Y. Azar and Y. Richter. An improved algorithm for CIOQ switches. *ACM Transactions on Algorithms*, 2(2):282–295, 2006.

[8] M. Bienkowski and A. Madry. Geometric aspects of online packet buffering: An optimal randomized algorithm for two buffers. In *Proc. of the 8th Latin American Symp. on Theoretical Informatics (LATIN)*, pages 252–263, 2008.

[9] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar. Matching output queueing with a combined input output queued switch. *IEEE Journal on Selected Areas in Communications*, 17:1030–1039, 1999.

[10] S.-T. Chuang, S. Iyer, and N. McKeown. Practical algorithms for performance guarantees in buffered crossbars. In *Proc. of the 24th IEEE Conf. on Computer Communications (INFOCOM)*, pages 981–991, 2005.

[11] M. Englert and M. Westermann. Considering suppressed packets improves buffer management in QoS switches. In *Proc. of the 18th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 209–218, 2007.

[12] M. Englert and M. Westermann. Lower and upper bounds on FIFO buffer management in QoS switches. *Algorithmica*, 53(4):523–548, 2009.

[13] L. Epstein and R. van Stee. Buffer management problems. *SIGACT News*, 35(3):58–66, 2004.

[14] P. T. Eugster, A. Kesselman, K. Kogan, S. I. Nikolenko, and A. Sirotkin. Essential traffic parameters for shared memory switch performance. In *Proc. of the 22nd International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 61–75, 2015.

[15] P. T. Eugster, K. Kogan, S. I. Nikolenko, and A. Sirotkin. Shared memory buffer management for heterogeneous packet processing. In *Proc. of the 34th IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 471–480, 2014.

[16] M. H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41:100–128, 2010.

[17] T. Itoh and N. Takahashi. Competitive analysis of multi-queue preemptive QoS algorithms for general priorities. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E89-A(5):1186–1197, 2006.

[18] Ł. Jeż, F. Li, J. Sethuraman, and C. Stein. Online scheduling of packets with agreeable deadlines. *ACM Transactions on Algorithms*, 9(1):Article 5, 2012.

[19] A. Kesselman, K. Kogan, and M. Segal. Packet mode and QoS algorithms for buffered crossbar switches with FIFO queuing. *Distributed Computing*, 23(3):163–175, 2010.

[20] A. Kesselman, K. Kogan, and M. Segal. Best effort and priority queuing policies for buffered crossbar switches. *Chicago Journal of Theoretical Computer Science*, 2012(5):1–14, 2012.

[21] A. Kesselman, K. Kogan, and M. Segal. Improved competitive performance bounds for CIOQ switches. *Algorithmica*, 63(1–2):411–424, 2012.

[22] A. Kesselman and A. Rosén. Scheduling policies for CIOQ switches. *Journal of Algorithms*, 60(1):60–83, 2006.

[23] A. Kesselman and A. Rosén. Controlling CIOQ switches with priority queuing and in multistage interconnection networks. *Journal of Interconnection Networks*, 9(1–2):53–72, 2008.

[24] F. Li, J. Sethuraman, and C. Stein. Better online buffer management. In *Proc. of the 18th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 199–208, 2007.

[25] S. I. Nikolenko and K. Kogan. Single and multiple buffer processing. *Encyclopedia of Algorithms*, pages 1–9, 2014.

[26] V. Paxson and S. Floyd. Wide-area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[27] D. Sleator and R. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[28] A. Veres and M. Boda. The chaotic nature of TCP congestion control. In *Proc. of the 19th IEEE Conf. on Computer Communications (INFOCOM)*, pages 1715–1723, 2000.