Online Algorithms for Packet Scheduling and Buffer Management

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften der RWTH Aachen University zur Erlangung des akademischen Grades eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

M.Sc.

Kamal Al-Bawani

aus Sanaa, Jemen

Berichter: Assistant Professor Dr. Matthias Englert
Privatdozent Dr. Walter Unger
Universitätsprofessor Dr. Peter Rossmanith

Tag der mündlichen Prüfung: 22. April 2016

Zusammenfassung

In dieser Arbeit untersuchen wir aus einer algorithmischen Perspektive das Puffer-Management-Problem in Netzwerk-Switches. In typischen Szenarios in der Datenpaketvermittlung kommen Pakete mit verschiedenen Service-Anforderungen an den Eingangsports des Switches an, und werden in Puffern (Warteschlangen) mit begrenzter Kapazität gespeichert. Danach werden sie über die Switching-Fabric in ihre entsprechenden Ausgangsports übertragen, wo sie anderen Warteschlangen beitreten. Schließlich werden die Pakete aus dem Switch durch seine ausgehenden Verbindungen zu ihren nächsten Zielen im Netzwerk gesendet.

Da die Bandbreite der ein- und ausgehenden Verbindungen und auch die Kapazitäten der Puffer beschränkt sind, kann Pufferüberlauf auftreten. Es ist in diesem Fall unvermeidlich, einige Pakete zu verwerfen. In anderen Switching-Modellen können Pakete, die verzögerungsempfindlich sind, verworfen werden, wenn sie eine bestimmte Deadline in der Warteschlange überschreiten. Wir betrachten mehrere Switching-Modelle mit dem Ziel, den Durchsatz der Switches zu maximieren. Wenn alle Pakete gleich behandelt werden, d.h. in Übereinstimmung mit dem Best-Effort-Konzept des Internet, quantifizieren wir den Durchsatz als die Anzahl der Pakete, die erfolgreich durch den Switch übertragen werden. In Netzwerken mit Quality of Service (QoS) Anforderungen werden Pakete Werte, die ihren Dienststufen entsprechen, zugeordnet. Der Durchsatz ist in diesem Fall gleich dem Gesamtwert der Pakete, die erfolgreich übertragen werden.

Wir präsentieren verschiedene Algorithmen für das Puffer-Management-Problem. In der Buffering-Phase dieser Algorithmen untersuchen wir, ob ein ankommendes Paket angenommen oder abgelehnt werden soll, und ob ein bereits in der Warteschlange enthaltenes Paket verdrängt werden soll, um Platz für weitere wertvollen Pakete zu reservieren. In der Scheduling-Phase beantworten wir Fragen von der Art "welches Paket soll in einem festen Zeitschritt von einem Eingangsport zu einem Ausgangsport übertragen werden?", und "welches Paket soll von einem Ausgangspuffer entlassen werden?".

Eine Eingabe für unsere Algorithmen ist eine endliche Folge von Paketen, die in einer "Online" Art und Weise betrachtet werden. Das heißt, dass die

Pakete nach einander über die Zeit ankommen und eine unwiderrufliche Entscheidung *on the fly* gemacht werden muss, bevor zukünftige Ankünfte bekannt werden. Solche Algorithmen, die unvollständige Informationen über die Zukunft bewältigen müssen und deren Entscheidungen nicht rückgängig gemacht werden können, werden *Online-Algorithmen* genannt.

Es ist bekannt, dass Paketankunftszeiten keine spezifische Ankunftsverteilung einhalten. In der Wirklichkeit haben Pakete die Tendenz, in "Bursts" und nicht in glatten Poisson-like-Verteilungen anzukommen. Aus diesem Grund machen wir keine vorherigen Annahmen über die Ankunftsverteilung der Pakete. Deshalb greifen wir zur Methode der Competitive-Analyse, welche die typische Worst-Case-Analyse ist, die verwendet wird, um die Leistung von Online-Algorithmen zu beurteilen.

In der Competitive-Analyse vergleichen wir die Güte eines Online-Algorithmus mit der Güte eines optimalen Algorithmus, von dem angenommen wird, dass er die gesamte Eingabesequenz im Voraus kennt. Ein Online-Algorithmus heißt *c-competitive*, wenn für jede Eingangssequenz die Güte des optimalen Algorithmus höchstens c mal die Güte des Online-Algorithmus ist. Der Wert c wird auch als *competitive ratio* des Online-Algorithmus genannt. Wir beweisen obere und untere Schranken des competitive ratio von mehreren Online-Algorithmen, und zeigen, dass einige dieser Algorithmen optimal sind.

Abstract

In this work, we study the problem of buffer management in network switches from an algorithmic perspective. In a typical switching scenario, packets with different service demands arrive at the input ports of the switch and are stored in buffers (queues) of limited capacity. Thereafter, they are transferred over the switching fabric to their corresponding output ports where they join other queues. Finally, packets are transmitted out of the switch through its outgoing links to their next destinations in the network.

Due to limitations in the link bandwidth and buffer capacities, buffers may experience events of overflow and thus it becomes inevitable to drop some packets. In other switching models, packets that are sensitive to delay are dropped if they exceed a specific deadline inside the queue. We consider multiple models of switching with the goal of maximizing the throughput of the switch. If all packets are treated equally, i.e., corresponding to the best-effort concept of the Internet, we quantify the throughput as the number of packets that are successfully transmitted through the switch. In networks with Quality of Service (QoS) requirements, packets are assigned values that correspond to their levels of service, and the throughput in this case is equal to the total value of packets that are successfully transmitted.

We present different algorithms for the problem of buffer management. In the buffering phase of these algorithms, we examine whether an arriving packet is accepted or rejected, and whether an already queued packet is preempted (dropped) to save space for more valuable packets. In the scheduling phase, we seek to answer questions of the kind "which packet to transfer from an input port to an output port in a given time step?", and "which packet to transmit from an output buffer?".

An input instance for our algorithms is a finite sequence of packets arriving in an "online" manner, i.e., packets arrive one by one over time and an irrevocable decision has to be made *on the fly* before future arrivals become known. Such algorithms which need to cope with incomplete information about future and cannot undo their decisions are called *online algorithms*.

It is known that packet arrivals do not adhere to any specific arrival distribution. In reality, packets tend to arrive "in bursts" rather than in smooth Poisson-like distributions. Thus, we do not make any prior as-

sumptions about the arrival process of packets. We therefore resort to the framework of competitive analysis which is the typical worst-case analysis used to assess the performance of online algorithms.

In competitive analysis, the benefit of an online algorithm is compared to the benefit of an optimal algorithm which is assumed to know the entire input sequence in advance. An online algorithm is called *c-competitive* if for each input sequence, the benefit of the optimal algorithm is at most c times the benefit of the online algorithm. The value c is also called the *competitive ratio* of the online algorithm. We prove upper and lower bounds on the competitive ratio of several online algorithms, and show that some of these algorithms are optimal.

Acknowledgments

First of all, I am very grateful to Berthold Vöcking for being the extraordinary supervisor and teacher he was. He granted me a great degree of freedom and confidence to pursue the research I found most appealing. He did though help me with many insightful discussions and taught me many interesting things in different aspects of algorithms—and life in general. Berthold passed away in summer 2014 at an early age. His sudden leaving was a great loss for me, on both a scientific level and, maybe more significantly, on a personal level.

With no less gratitude, I would also like to thank Matthias Englert who has been a great mentor and co-author for me, and in times when I was with him at Warwick University, a very generous host also. The many discussions we had on many topics in many places were mostly fruitful and improved my research in numerous ways. I especially appreciate his tremendous support as a first reviewer of my thesis. I would also like to thank my other co-authors, Matthias Westermann, especially for his great hospitality in Dortmund, and Alexander Souza who introduced me to the problem of buffer management.

I am also indebted to Walter Unger and Peter Rossmanith for several casual discussions in Winterberg and also for acting as reviewers of my thesis. I am especially thankful to Walter for being always supportive after the leaving of Berthold. I would also like to thank all my current and former colleagues in the Algorithms and Complexity group for the pleasant and friendly atmosphere they have continued to preserve in our group. Special thanks to Oliver Göbel who kindly guided me through many bureaucratic procedures and proof-read the German abstract of this thesis.

Last, and as I would say it in Arabic "the musk of the closure", I would like to express my endless gratefulness to my family, foremost to my parents, my wife and my kids, for their unconditional love and support. This thesis is dedicated with all humility to them.

Contents

Chapter 1

Introduction

1.1 Overview

The problem of buffer management arises in the core of computer networks—the mesh of switching devices that interconnects the network's end systems. In a network application, computation devices, such as PCs, laptops, smart phones, gaming consoles, etc., that are placed on the terminal nodes of the network are called *hosts*. Hosts communicate with each other by exchanging messages. A message can contain any kind of information. For example, it can just be a standard request to download a website or it can contain some data, such as a text of an Email, an image, or an audio file. Before they are popped into the network, messages are split into smaller pieces, called *data packets*. Each packet travels alone from the source host to the destination host through a path of multiple switching devices. In networking jargon, these switching devices are called *switches* or *routers*, and they are placed on the inner nodes of the network to store and forward packets. Packets are transmitted into a switch through its *ingoing* and are transmitted out of the switch through its *outgoing* links.

The path between a source and a destination is not known for the traveling packet before hand. Packets are forwarded over the network in a manner similar to guiding a tourist who is driving through a country without a map and remembers only the address of her final destination. In this case, the tourist is forwarded by local people from a city to another, based on the portion of the address that contains the city name. Once she arrives in the final city, she is forwarded further to the district of the city that appears in another portion of the address. After that, inside that district, she is forwarded to the goal street and then, using the house number in the address, to her final destination. In a similar way, a packet stores in its header, among other information, the IP address of its destination host. When the packet arrives at an intermediate switch, its destination's IP address is looked up in a table, known as *routing table*, that maps IP addresses, or more precisely,

prefixes of IP addresses to outgoing links of the switch. The packet is then transmitted to the next switch over the specified outgoing link and proceeds further towards its final destination.

Let's now have a closer look into packet switches. The process described above of determining the outgoing link of an incoming packet using the routing table is called *routing*. Although routing is a major function of the switch, the topics of this work revolve around another major function, namely, the process of transmitting the packet throughout the switch down to its specified outgoing link. This process is known as *forwarding* or *switching*. In a nutshell, three components of the switch are involved in forwarding: the input ports, the output ports, and the switching fabric interconnecting the input and output ports. As the name implies, *input ports* interface the ingoing links attached to the switch, and *output ports* interface the outgoing links. When a packet arrives, it enters the switch at one of its input ports, based on the packet's source, and its destination's IP address is immediately looked up in the routing table to determine the output port which corresponds to its next destination. The packet is then transferred to the determined output port through the *switching fabric*, and after that it is transmitted from the output port onto the outgoing link which takes it further to the next switch on its path.

While transmitting a packet from an output port is implemented in a straightforward way, moving packets from input to output ports over the switching fabric is implemented in different ways. Without sinking in technical details, we distinguish between two major designs of the switching fabric in modern switches: In the first design, the switching fabric are implemented as a single shared bus (channel) that transfers only one packet at a time. That is, even if several packets demand to be transferred at the same time, each from a different input port to a different output port, only one of them can cross the bus at a time. In the second design, the switching fabric are implemented as multiple buses in a way that parallelizes the transfer of packets. So multiple packets from different input ports that are destined for different output ports can be transferred at the same time. In either design, a single bus can not transfer more than one packet at a time. In practice, the first design is adopted only in small local area and enterprise networks.

From the first moment it enters the switch, a data packet goes through several processes that all enhance the switch's main process of forwarding. As described above, the output port of the packet is first determined in the process of routing. Another process is *fragmentation*. A massive network, such as the Internet, is in fact a network of networks, and some problems of heterogeneity arise at the interfaces between these networks. In particular, each of the links along the path between the packet's source and destination may adhere to different transmission protocols, and each of these protocols may set a different limit on the size of the packet. In this case, if the size of an incoming packet exceeds the defined limit of the switch's protocol, the

packet is fragmented into smaller packets, and specific fields in the headers of these fragments are properly modified so that they can be "glued" together at the destination. In following sections, we will discuss further processes that address other fields of the packet's header, namely, the *service type* and *time-to-live* fields which are most relevant to the topics of this work [1].

The rate of data transfer over a network link is called the *bandwidth* of the link. A link that has a bandwidth of W bps can transmit up to W bits in each second. Even after fragmentation, the size of a packet may exceed the bandwidth of an outgoing link. In this case, the packet is transmitted over multiple time steps (seconds), a process that is controlled by a special mechanism called *store-and-forward*. Specifically, a packet that is transmitted from switch A to switch B over multiple time steps is not forwarded further by B until it is completely received at the input port of B.

Due to limitation in the bandwidth of links, the rate in which packets arrive at the switch, incoming from multiple ingoing links, may exceed the rate of transmission over the outgoing links, and therefore packets may experience some delay inside the switch. To solve this problem, switches are devised with *buffers*, also known as *queues*, to accommodate delayed packets until they are transmitted to their next destination. As we will see in the next section, the architecture of switches varies by the placement of buffers inside the switch, but in most cases, at least one buffer is placed at each output port. This kind of buffers is called *output buffers*. However, buffers in general are of limited capacity and this in turn leads to another problem, namely, the *packet loss*. More specifically, in cases of congestion, a packet that arrives at the switch while the buffer of its corresponding output port is full is either dropped or one of the already queued packets is dropped.

In this work, we deal with the interplay between packet scheduling (transmission) and packet buffering. In *packet scheduling*, we seek to answer questions of the kind "which packet to transfer from an input port to an output port in a given time step?", and "which packet to transmit from an output buffer?". In *packet buffering*, the questions that frequently arise are whether an arriving packet is accepted (queued) or rejected (dropped), and whether an already queued packet is preempted (dropped). In all variants of these problems, our goal is to maximize the switch's throughput, i.e., to ensure that as many packet as possible are transmitted successfully throughout the switch. A more concise definition of the throughput will be given in Section 1.4.

A detailed and up-to-date investigation of packet routing and switching (as implemented in practice) and computer networking in general can be found in the thorough textbook of Kurose and Ross [KR12].

[1] We here assume the widely deployed IP protocol version 4, denoted as IPv4.

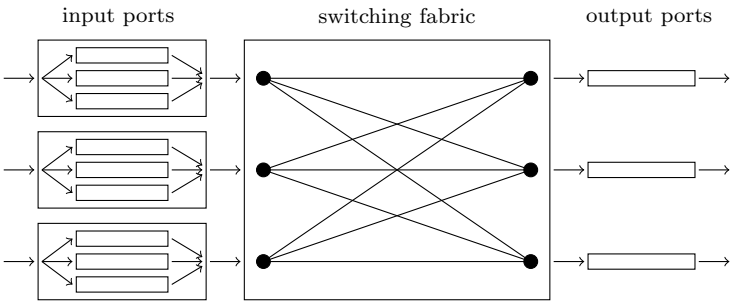

Figure 1.1: CIOQ switch — An example with $N = 3$

1.2 Switch Architecture

The placement of buffers in the switch is essential in the architecture of switches. The following types of switches are the most dominant in the currently used devices, where the location and capacity of buffers depend on the expected traffic load, the relative speed of the switching fabric, and the line speed, i.e., the speed of ingoing and outgoing links. A switch with N input ports and M output ports is called an $N \times M$ switch.

- *Output-queued (OQ) switches.* Buffers in this type of switches are placed only at output ports. When a packet arrives at an input port, it immediately crosses the switching fabric and joins the buffer of its corresponding output port. In some cases, more than one buffer are placed at each output port, one for each class of packets, in order to support different types of service. Typically, the speed of the input links are much higher than the speed of the switching fabric, and packets from different input ports may be destined at the same time for the same output port. In such cases, congestion at input ports may arise and some packets are lost before they are transferred to their output ports. This problem leads to the next architecture.

- *Input-queued (IQ) switches.* In this type of switches, one buffer is placed at each input port. Arriving packets are stored in this buffer until they are transferred through the switching fabric to their output ports. However, another problem, the so-called head-of-line (HOL) blocking, occurs in this design. Consider for example the following scenario. At some time t, a packet p is at the head of the buffer of an input port and is destined for output port i. Assume that packets are transferred in a first-in-first-out (FIFO) order, and that the buffer of output port i is full at time t. Consequently, not only p is blocked at t, but also all packets behind p are blocked, even if they are destined for output ports whose buffers are still not full. This problem is solved by the next switch design.

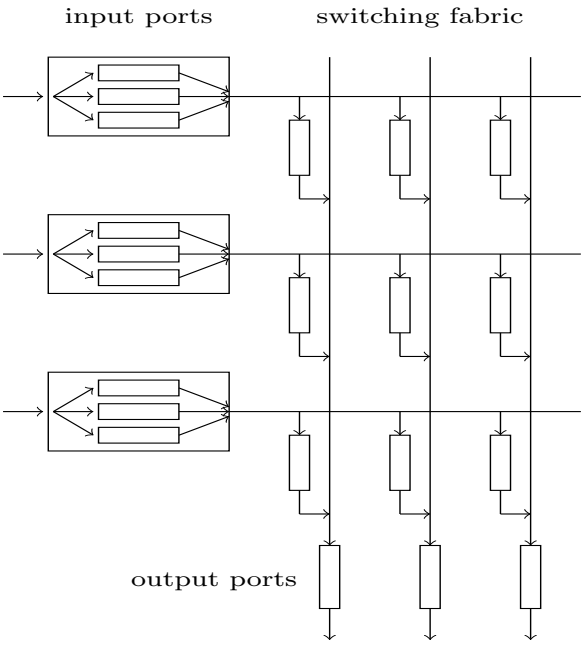

Figure 1.2: Buffered crossbar switch — An example with $N = 3$

- *Combined input and output queued (CIOQ) switches.* Similar to IQ switches, buffers are placed at both input and output ports. However, in each input port, a separate buffer is dedicated for each output port, where all packets that are destined for a specific output port are stored in the buffer that corresponds to that output port. Typically, CIOQ switches are $N \times N$ switches. Thus, N buffers are placed at each input port and one buffer at each output port. The buffers at input ports are called the input queues or sometimes the virtual output queues (VOQ). Figure 1.1 depicts an example of a CIOQ switch.

- *Buffered crossbar switches.* Closely related to CIOQ switches, this type of switch architecture is obtained by adding queues at the crosspoints of the switching fabric. More specifically, for every input port i and every output port j, an additional queue is placed at the crosspoint (i, j) of the switching fabric. A packet that is intended to move from i to j is first stored in the corresponding virtual output queue at i, then transferred to the new queue at the crosspoint (i, j), and finally moved to the output queue of j. Although more space requirements are involved in this kind of switches, practice has shown that the adoption of crossbar queues significantly decreases the scheduling overhead of CIOQ switches. Figure 1.2 depicts an example of a buffered crossbar switch.

In this work, we consider all these types of switches. We obtain new results for buffer management in the OQ, CIOQ and buffered crossbar switches. Notice that CIOQ switches are a generalization of IQ switches, i.e., an $N \times 1$ CIOQ switch is also an IQ switch.

1.3 Switching with QoS Guarantees

In modern IP networks, the concept of Quality of Service (QoS) is supported to provide preferential treatment for different classes of end-users or network applications. According to a Service Level Agreement (SLA) with the service provider, a packet that belongs to a customer's predefined data stream is guaranteed a certain level of service, i.e., a fixed value of bit rate, delay, or jitter. QoS guarantees are essential for many kinds of data streams that require fixed bit rate or are latency sensitive, e.g., real-time streaming applications such as voice over IP (VoIP), online games and IP-television (IPTV).

In the widely deployed IP protocol version 4, denoted as IPv4, the two fields of *service type* and *time-to-live* in the packet's header are reserved for the implementation of the QoS concept. In a heterogeneous network like the Internet, classification of packets may occur at the switch level; for example, to realize a scheme of differentiated services designed for a private network. In this case, the service type field in the packet's header is rewritten upon the arrival of the the packet with a value corresponding to its Class of Service (CoS).

Additionally, in applications where packets are sensitive to delay, the other field of time-to-live is decreased by one each time the packet arrives at a new switch, and the packet is eventually discarded (lost) if this value reaches 0 at some point on its rout. Later on in this chapter, we will call this model of switching the *bounded-delay* model, and instead of the time-to-live value, we assign to each packet a *deadline* after which the packet is not relevant any more and thus is discarded.

1.4 Problem Definition: Framework and Objectives

As mentioned above, we consider the problem of buffer management as it arises in the different types of switches discussed in Section 1.2. We therefore consider different models of packet switching that will be discussed in depth in Section 1.6. For the moment, we introduce the theoretical framework of these models and discuss their common aspects and objectives.

First, we consider a switch with multiple input and output ports. Buffers (queues) are placed at the input and output ports in a manner that is specified by each switching model. Buffers are bounded in capacity. They may have the same capacity or each buffer may be of a different capacity.

We discretize time into steps of unit length. A time step may for example correspond to one second in practice. In each time step, an arbitrary number of packets arrive at the input ports of the switch, while at most one packet is transmitted from any output port.

All packets are assumed to have the same size. In fact, this assumption complies to a reasonable extent with the practice in real switches. As mentioned in Section 1.1, a process of fragmentation takes place in switches in order to split large packets into smaller packets of nearly equal size.

We consider two cases with regard to the packet size: either all packets have size of 1, or they all have size of $P > 1$, and P is integer. The first choice abstracts the case in practice where the switch's bandwidth allows to transmit the packet as a whole in one time step, and the second abstracts the case where each packet is transmitted over $P > 1$ time steps. Assume that all ingoing and outgoing links have the same bandwidth of W bps. Thus, we assume that the size of a packet is either W or a multiple of W.

An input instance for all problems considered in this work is a finite sequence of packets. Each packet is characterized by the following attributes: the input port at which it arrives, the output port from which it leaves the switch, its time of arrival, and a value that corresponds to its service class or priority. Notice that the packet's input and output ports are part of the input, i.e., the algorithm does not choose the input and output queues of the packets.

We consider two cases with regard to the packet value: either all packets have the same value of 1, or each packet has an arbitrary value. We call these cases the unit-value case and the general-value case, respectively. In the unit-value case, all packets are treated equally, which corresponds to the *best-effort* concept that is dominant in the core network of the Internet.

Packets arrive in an online manner, i.e., one by one over time and a decision has to be made *on the fly* before future arrivals become known. More specifically, an algorithm for such problems must *immediately* decide whether to accept or reject an arriving packet. Also, at transmission times, it must decide which packet to send based only on its past schedule and the current configurations of the buffers. Furthermore, all decisions made in this manner are irrevocable. For example, the algorithm cannot accept a packet that was rejected in a previous time step or re-queue a packet that was transmitted before. Such algorithms which need to cope with incomplete information about future and cannot undo their decisions are called *online algorithms*. This is in contrast to the typical (*offline*) algorithms which first receive the entire input at once, process the input back and forth, and then output a final solution. Online algorithms are discussed in details in Section 1.5.

An algorithm for the problem of buffer management consists of two phases. In the buffering phase, the algorithm decides for each arriving packet whether to admit it to the designated buffer or reject it. In the scheduling

phase, the algorithm decides which packet to transmit (either completely or partially) from each output buffer in each time step. The algorithm may also drop a packet that is already queued in order to make room for a new one. Dropping a packet in the latter case is called *preemption*.

For any input sequence, rejected and preempted packets are lost, and the total value of the transmitted packets defines the throughput of the switch. Clearly, if all packets have value 1, the throughput is equal the number of packets that are successfully transmitted. All problems that we study in this work are optimization problems. We aim by each algorithm for these problems to maximize the switch's throughput.

In the bounded-delay model, which will be studied in Chapter 6, another attribute of packets is involved. Each packet is assigned an integral value corresponding to its deadline. If a packet is not transmitted before its deadline, it is lost. In this case, the goal is to maximize the total value of packets that are transmitted before their deadlines. Furthermore, we relax the constraint on buffers in this model and assume that their capacity is unlimited. However, we may assume in this case that the life span of any packet, i.e., the difference between its arrival time and its deadline, is bounded by some value s. Thus, the number of packets pending in the buffer at any time is bounded by s since any more packets beyond this bound would expire in the next s time steps.

Throughout this work, an input sequence will be considered also as an *event sequence*. An event sequence consists of arrival and send events, where an arrival event corresponds to the arrival of a new packet and a send event corresponds to the transmission of an enqueued packet. Finally, we denote the time preceding the first arrival event as time 0, and we assume that the queues of any algorithm are all empty at time 0.

1.5 Online Algorithms and Competitive Analysis

Online problems are common in real life. They arise from our lack of information on future events. A classical example of online problems is the skiing problem: A person in a ski resort has to decide whether to rent skis for \$10 per day or rather buy a new pair for \$100. This becomes an online problem if it is not known in advance when to quit skiing. For example, due to a wavering weather, it is not known for how long this facility is open in the current season. So how to cope with this uncertainty about the weather? If the person buys at an early point of time, it can happen that a storm blows out in a few days and the optimal choice would have been in this case to rent for those days. However, if she rents everyday, the weather can remain perfect for many days and the optimal choice would have been in this case to buy at the first day. So what is the optimal choice in this game?

Formally, the input of an online problem is revealed over time as a se-

quence of requests $\sigma = \sigma(t_0), \sigma(t_1), \ldots$, where $\sigma(t_i)$ is the input request at time step t_i. An algorithm that must respond to input $\sigma(t_i)$ at time step t_i, i.e., without knowing the future input requests, and cannot change its responses to previous requests, is called *online*. Online problems are typically optimization problems. The goal is either to maximize profits or minimize costs. From now on, we will assume a maximization problem. We denote the benefit of an online algorithm ALG from an input sequence σ by $\text{ALG}(\sigma)$.

Due to incomplete information about future, the online algorithm is not expected to compute the optimal benefit. For many online problems, it suffices to compute approximated solutions that are near optimal. Although time and space requirements are essential in the evaluation of algorithms, the performance of an online algorithm is not primarily measured by these two aspects, but rather by the ratio of its benefit from an input sequence to the maximum benefit that can be ever made from the same input sequence. Thus, for the purpose of comparison, one fixes an *offline* (sometimes hypothesized) algorithm that knows the entire input in advance and computes the optimal benefit. We call this algorithm OPT and denote its benefit from a sequence σ by $\text{OPT}(\sigma)$.

This method of analyzing online algorithms by comparing ALG to OPT is called the *competitive analysis*. Competitive analysis is a worst-case analysis, i.e., the ratio $\text{OPT}(\sigma)/\text{ALG}(\sigma)$ is considered for any input sequence σ. More specifically, an online algorithm ALG is called *c-competitive* if there is a constant α such that for any input sequence σ,

$$\text{OPT}(\sigma) \leq c \cdot \text{ALG}(\sigma) + \alpha\,.$$

We call c in this case the *competitive ratio* of ALG. Notice that α can be a function of problem parameters, e.g., the queue capacity, but it must be independent of the input sequence. Allowing this additive constant α helps us smooth the analysis, so that the competitive ratio does not depend on any initial configurations whose effect becomes asymptotically negligible, i.e., as the input sequence becomes longer and longer. If $\alpha = 0$, we say that ALG is *strictly c-competitive*. In fact, almost all the online algorithms that we study in this work are strictly competitive.

The competitive analysis is also called adversarial analysis because one may imagine the generation of inputs as a game between the online algorithm and an omniscient adversary who can read the online player's mind—it knows how the online algorithm works. The adversary's task is then to generate an input piece in each time step and the online algorithm must immediately respond to this input. Due to its omniscience, the adversary generates inputs in a malicious way so as to minimize the benefit of the online player and in the same time maximize its own benefit. For more insight into online algorithms and competitive analysis, we refer the reader to the textbook of Borodin and El-Yaniv [BEY98].

Employing the competitive analysis in the buffer management problem is well-motivated from a practical point of view. It is well known that network traffics do not adhere to any specific arrival distribution (see, e.g., [PF95, VB00]). In reality, packets tend to arrive in bursts rather than in smooth Poisson-like streams. Moreover, it has been shown that network traffics, e.g., Ethernet traffics, follow a self-similar behavior that is difficult to model [LTWW94]. To this end, stochastic models have been recently undermined in networking analysis, and tools such as the competitive analysis started to gain more interest.

In the next section, we present an example online problem to illustrate the main concepts and techniques of competitive analysis that are used throughout this work. We address in this example the buffer management problem in a simple switching model.

1.5.1 Example Problem: Throughput Maximization in IQ Switches

This problem corresponds to an IQ switch of N input ports and one output port. We assume that only one packet can be transferred over the switching fabric in each time step. Hence, packets are sent directly to outside the switch without queuing at the output port. Furthermore, the size of each packet is assumed to be 1. Thus, a packet can be sent (transmitted) completely in one time step.

More formally, the system consists of N queues, all have the same capacity B, i.e., each queue can store up to B packets at a time. We denote the i-th queue as Q_i. Moreover, all packets have value 1. In each time step, a non-empty queue is chosen and exactly one packet is sent from this queue. The goal is to maximize the number of sent packets.

Notice that since all packets have the same value, preemption does not help in this model. Also, it does not help to reject an arriving packet while its corresponding queue is yet not full. We therefore consider the following greedy online algorithm.

> GREEDY**:** Accept any arriving packet as long as its corresponding queue is not full.
>
> In each time step, choose any arbitrary non-empty queue and send the packet at its head.

The rest of this section is organized as follows. First, we show that GREEDY has a competitive ratio of 3. The main purpose of presenting this result is to illustrate the analysis technique of *mapping schemes* which is probably the most used tool in the analysis of online switching algorithms. Then, we show how to improve the competitive ratio of GREEDY to 2 using another technique, which we call the technique of *modifying the optimal*

algorithm, and which we extensively use in Chapters 4 and 5. After that, we show a lower bound on the competitive ratio of GREEDY and a general lower bound of $2 - 1/N$ on the competitive ratio of any deterministic online algorithm. Finally, we use a third technique, called the *potential-function technique*, to further improve the competitive ratio of GREEDY. Specifically, we show that GREEDY is indeed $(2 - 1/N)$-competitive and thus it is an optimal algorithm for this problem. Although they may be proved here differently, all these results on the IQ switches, except Theorem 1.7, are already known [AS06, AR05].

The Technique of Mapping Schemes

Before we start, we slightly abuse our notations so that OPT and GREEDY also denote the set of packets sent by OPT and GREEDY, respectively. Thus, OPT $\setminus$ GREEDY denotes the set of packets that are accepted by OPT but rejected by GREEDY.

As stated before, to prove that GREEDY is c-competitive, we have to show that $|\text{OPT}| \leq c\,|\text{GREEDY}|$ holds for any input sequence. One way to do that is to define a function from OPT $\setminus$ GREEDY to GREEDY with the following two properties: (i) each packet from OPT $\setminus$ GREEDY is mapped to a packet from GREEDY, and (ii) for each packet $p \in$ GREEDY, the total number of packets from OPT $\setminus$ GREEDY that are mapped to p is at most $c-1$. Obviously, having such a function means that the number of packets that are sent only by OPT is at most $c - 1$ times the number of packets sent by GREEDY. Thus, the number of all packets sent by OPT, including packets sent by GREEDY, is at most c times the number of packets sent by GREEDY.

We now define a mapping scheme for any input sequence of GREEDY that satisfies the above two conditions with $c = 3$. We adopt the same mapping scheme that is given by Azar and Richter [AR04] for a related switching model. We denote by $Q_i^*(t)$ and $Q_i(t)$ the set of packets in the i-th queues of OPT and GREEDY, respectively, at time t. Moreover, we will call a packet of GREEDY that is mapped by a packet of OPT a *marked packet*.

Mapping Scheme. For each time t, starting from $t = 0$,

- t **is an arrival event.** Let a packet p arrive at queue Q_i at t. If p is accepted by OPT and rejected by GREEDY, let q be the first unmarked packet in $Q_i(t)$, i.e., the packet closest to the queue's head to which none of OPT's packets has been mapped yet. Map p to q. We say in this case that q is marked.
- t **is a send event.** Let OPT and GREEDY send from the i-th and the j-th queue, respectively. If $Q_i(t)$ contains marked packets, let p_i be the last marked packet in $Q_i(t)$, i.e., the

packet closest to the queue's tail to which a packet of OPT is mapped. Let p^* be the packet of OPT that is mapped to p_i. Moreover, let p_j be the packet sent by GREEDY from Q_j at t. Re-map p^* to p_j, i.e., in terms of marks, remove the mark of p_i and mark p_j.

First, we notice that packets are marked in a FIFO order, i.e., in the order of their arrivals. Hence the following remark.

Remark 1.1. *At any time t, if* GREEDY *sends an unmarked packet from a queue Q_i at t, then $Q_i(t)$ contains no marked packets.*

Let $M_i(t)$ denote the set of marked packets that are in queue Q_i at time t. The next lemma shows that the number of marked packets in $Q_i(t)$ does not exceed the number of packets in $Q_i^*(t)$.

Lemma 1.2. *For any queue Q_i and any time t, $|M_i(t)| \leq |Q_i^*(t)|$.*

Proof. We use an induction argument over time steps. Let the induction basis be at time 0, i.e., before the input sequence begins. Clearly, all queues are empty at that time and thus the claim holds. Now assume the claim holds at any time before t. We want to show it also holds at t. Notice that $M_i(t)$ and $Q_i^*(t)$ change only in arrival and send events. Thus, t is either of these two events.

Assume first that t is a send event. The only critical scenario in this case is when OPT sends a packet from Q_i^* at t and GREEDY does not send any marked packet from Q_i. If GREEDY does not send any packet at all, then its queues must be all empty at t. Also, if it sends an unmarked packet from Q_i at t, then, by Remark 1.1, this queue is empty of marked packets. Clearly, the claim holds in either case. A third case in this scenario is that GREEDY sends a packet from a different queue Q_j at t while $Q_i(t)$ contains marked packets. By the mapping scheme, GREEDY will in this case remove the mark of the last marked packet in $Q_i(t)$ and mark the packet sent from Q_j. therefore, the number of marked packets in $Q_i(t)$ decreases also by 1 and thus the claim continues to hold.

Now assume that t is an arrival event. The only critical case in this scenario is when a packet is marked in $Q_i(t)$ while OPT does not accept any new packet. However, marking a new packet in an arrival event occurs only when OPT accepts the arriving packet. Hence, this case cannot happen and thus the claim holds. □

Lemma 1.2 is crucial to show the feasibility of the mapping scheme. Now, we show the 3-competitive ratio of GREEDY based on the mapping scheme.

Theorem 1.3. *The competitive ratio of* GREEDY *is at most 3.*

Proof. We want to show that (i) each packet of OPT $\setminus$ GREEDY is mapped to a packet from GREEDY, and (ii) each packet of GREEDY has at most 2 marks.

For each packet $p \in$ OPT $\setminus$ GREEDY, since p is not accepted by GREEDY, the queue of GREEDY must be full at the arrival time of p. Thus, if p arrives at queue Q_i, $|Q_i(t)| = B > |Q_i^*(t)|$. Hence, by Lemma 1.2, the number of marked packets in Q_i must be strictly less than B, and thus at least one packet in $Q_i(t)$ is not marked and therefore p is mapped to this packet. In a send event, p might be remapped to another packet, but it is never unmapped.

To show (ii), notice that each packet $p \in$ GREEDY is marked for the first time in an arrive event, then it may be marked for a second time when it is sent. After that, p cannot be marked any more. Thus, p can have at most 2 marks. □

Modifying the Optimal Algorithm

Another analysis technique is to modify OPT in a manner that does not decrease its benefit in order to ensure that the queues of both OPT and GREEDY maintain a specific configuration at any given time. In particular, we make the queues of OPT and GREEDY identical at any time by allowing OPT to send more than one packet in some time steps, and by generating extra packets exclusively for it. Since the queues of the two algorithms are identical, GREEDY sends exactly one packet in each time step in which OPT sends. Thus, we show the competitive ratio for any send event, and this is equal to the maximum number of packets that OPT sends in one time step.

Theorem 1.4. *The competitive ratio of* GREEDY *is at most* 2.

Proof. Before we start, we make an assumptions about OPT. Since all packets are of value 1, we can assume without loss of generality that OPT accepts each arriving packet as long as its corresponding queue is not full. Thus, OPT and GREEDY adopt the same policy in arrival events.

Now, we show by induction that OPT sends in each time step at most twice what GREEDY sends. Fix a time step t that starts with identical queues in both OPT and GREEDY. By assumption, OPT accepts in t as many packets as GREEDY does. Thus, the queues of the two algorithms remain identical at the end of t.

Consider now the send event of t. Let OPT and GREEDY send from the i-th and the j-th queues, respectively. If $i \neq j$, we modify OPT as follows. We send a packet from Q_j^* and insert a new packet into Q_i^*. Obviously, modifying OPT in this way can only increase its benefit. Furthermore, the queues of OPT and GREEDY remain identical after the send event, and OPT sends at most twice what GREEDY sends. □

Lower Bounds

We gave first, by Theorem 1.3, an upper bound of 3 on the competitive ratio of GREEDY. It then turned out, by Theorem 1.4, that GREEDY has a competitive ratio of 2. So, we may ask ourselves in this position whether we can improve the competitive ratio any further. In other words, how far are we from the "right" competitive ratio of GREEDY? A more general question in this respect is whether another online algorithm can achieve a better competitive ratio than GREEDY's. Or, what is the *best* competitive ratio of this online problem? In this section, we answer questions of this kind by showing two lower bounds, one on the competitive ratio of GREEDY and the other on the competitive ratio of any deterministic online algorithm. Both lower bounds are of $2 - 1/N$.

The first lower bound on the competitive ratio of GREEDY shows that our last analysis of this algorithm is almost tight, and the second lower bound shows that GREEDY is a good candidate for an *optimal* algorithm for the problem of throughput maximization in IQ switches. In the next section, we give a more involved analysis of GREEDY, using the technique of potential functions, and show its competitive ratio is indeed $2 - 1/N$. Hence, given the two lower bounds of $2-1/N$, the competitive ratio of GREEDY cannot be improved any more, and no other online algorithm has a competitive ratio better than GREEDY. We say in this case that GREEDY is an optimal online algorithm.

Theorem 1.5. *The competitive ratio of* GREEDY *is at least* $2 - 1/N$.

Proof. Recall the definition of the competitive ratio. An online algorithm ALG is called *c-competitive* if the inequality $\text{OPT}(\sigma)/\text{ALG}(\sigma) \leq c$ for any input σ. Thus, to show that the competitive ratio of GREEDY cannot be better than $2 - 1/N$, it is sufficient to show that $\text{OPT}(\sigma)/\text{GREEDY}(\sigma) \geq 2 - 1/N$ for a single input sequence σ.

Now, we take the place of an adversary who knows how GREEDY works, and generate a *worst-case* sequence for GREEDY. In the first time step, B packets arrive at each queue. Then, for the next $(N-1)B$ time steps, a single packet arrives at queue Q_N in each time step. Clearly, GREEDY accepts all packets arriving in the first time step. Since GREEDY sends from an arbitrary non-empty queue in each time step, we assume that it sends from the non-empty queue with the smallest index. Thus, in steps $2, \ldots, (N-1)B + 1$, queue Q_N remains full and therefore all the $(N-1)B$ packets arriving in these steps are rejected by GREEDY. Hence, the benefit of GREEDY equals $N \cdot B$.

On the other hand, the optimal algorithm, which works offline, sends only from queue Q_N during the first $(N-1)B$ time steps, and thus can accept all packets of the sequence. Hence, the competitive ratio of GREEDY

in this case cannot be better than

$$\frac{N \cdot B + (N-1)B}{N \cdot B} = 2 - \frac{1}{N}.$$

□

Next, we show a general lower bound of $2-1/N$ on the competitive ratio of any deterministic online algorithm.

Theorem 1.6. *For any value of B, no deterministic online algorithm has a competitive ratio better than $2 - 1/N$.*

Proof. To show a general bound of $2-1/N$, i.e., for any online algorithm, we fix an online algorithm ALG and show that it cannot achieve a competitive ratio better than $2 - 1/N$. Thus, we, playing the role of the adversary, generate a worst-case sequence σ and show that $\text{OPT}(\sigma)/\text{ALG}(\sigma) \geq 2-1/N$. Obviously, we do not know how algorithm ALG works, but it suffices to compute an upper bound on its benefit, i.e., the maximum benefit that an online algorithm can make from this sequence. Since a competitive ratio for this problem holds for any queue capacity B, we may further assume a fixed value $B = 1$ for the capacity of any queue.

We now generate the worst-case sequence as follows. For each time step $1 \leq i \leq N$, we define the set of arriving packets in i as a set of queue indices $S_i \subseteq \{1, \ldots, N\}$ in the following way. First, let $S_1 = \{1, \ldots, N\}$. This means that a set of N packets, each destined for a different queue, arrives in the first time step. Clearly, since all packets have the same value, accepting these packets can only increase the benefit of ALG. Thus, we assume that ALG accepts all packets arriving in the first time step. Furthermore, let s_1 be the index of the queue from which ALG sends in the first time step. Since ALG is deterministic, this value is well-defined given S_1. Now, define $S_2 = S_1 \setminus \{s_1\}$, and again let s_2 be the index of the queue served by ALG in time step 2 provided that S_2 arrives in this time step. Similarly, define $S_3 = S_2 \setminus \{s_2\}$, and in general $S_{t+1} = S_t \setminus \{s_t\}$ for $t = 1, \ldots, N-1$. Clearly, $S_N \subset \cdots \subset S_1 = \{1, \ldots, N\}$.

Observe now that packets arriving in time steps $2, \ldots, N$ all correspond to full queues in ALG, and thus ALG must reject all of these packets. Hence, the benefit of ALG does not exceed N, i.e., the number of packets arriving in the first time step.

Now, we compute the benefit of the optimal algorithm. OPT accepts all packets corresponding to S_1 in the first time step, and sends a packet from the queue corresponding to s_2 in the send event of the first time step. Since $s_2 \in S_2$, OPT accepts the packet corresponding to s_2 in the second time step and rejects all other packets. Similarly, in the send event of the second step, OPT sends a packet from the queue corresponding to s_3. In general, in the send event of time step t, OPT sends a packet from the queue

corresponding to s_{t+1} for $t = 1, \ldots, N$; and, for $t = 2, \ldots, N$, it accepts the packet corresponding to s_t in time step t and rejects all other packets. Thus, OPT sends $N + (N-1)$ packets in total and therefore the competitive ratio of ALG cannot be better than

$$\frac{N + (N-1)}{N} = 2 - \frac{1}{N}.$$

□

The Potential-Function Technique

For many online algorithms, it is not possible to show the competitive ratio by comparing the benefits of the online and optimal algorithms over every single time step. For example, in our problem of throughput maximization, there are time steps where the optimal algorithm sends a packet while the online algorithm does not send any packet at all. Also, it may happen that the optimal algorithm accepts an arriving packet while the online algorithm rejects it. Thus, over such single time steps, the local benefit of the online algorithm is 0 and thus it is not competitive at all.

Therefore, in cases where the optimal benefit in a single time step exceeds the online benefit by a factor greater than the desired competitive ratio, one seeks to compare the optimal benefit in that time step with rather the "amortized" benefit of the online algorithm. The amortized benefit of a time step t can be computed as the online benefit of t plus another amount that has been accumulated over previous time steps where the online benefit exceeded the desired amount. To illustrate the idea of amortized benefits, let $\text{OPT}(t)$ and $\text{ON}(t)$ denote the optimal and online benefits in a time step t. Thus, our initial goal was to show that for any time step t, $c \cdot \text{ON}(t) \geq \text{OPT}(t)$, where c is the competitive ratio we want to show. The idea now is to save amounts of the online benefit in those time steps where $c \cdot \text{ON}(t)$ exceeds $\text{OPT}(t)$ and use these saved amounts in future time steps where $c \cdot \text{ON}(t)$ falls short of $\text{OPT}(t)$. We call the *savings account* in which we maintain those amounts *the potential function*.

More Formally, a potential function Φ is typically a mapping from the set of all possible configurations [2] of the optimal-online game to the set of real numbers. As in savings accounts, we maintain a potential function Φ such that $\Phi(T) = \Phi(0)$, where 0 and T correspond to the start and end configurations of the optimal-online game. Our goal now is to show that $c \cdot \text{ON}(t) + \Delta\Phi(t) \geq \text{OPT}(t)$ holds at each time t, where $\Delta\Phi(t) = \Phi(t) - \Phi(t-1)$ is the change in the potential value between steps $t-1$ and t. Thus, summing over all time steps and given that $\Phi(T) = \Phi(0)$, it follows that $c \cdot \text{ON}(\sigma) \geq \text{OPT}(\sigma)$, and thus ON is shown to be c-competitive [3].

[2] The term of *configuration* will become clear in the proof of the next theorem.

[3] In fact, it suffices to define a potential function which satisfies $\Phi(T) \leq \Phi(0)$.

Potential functions are an original tool of the theory of online algorithms and have been used since early formulations of competitive analysis [ST85]. In fact, the way we describe them above is only the standard style to show how they are used in general. Although the main principle remains the same, the style of potential functions we use in our problems of throughput maximization is different. In this section, we show how such potential functions are used to prove that GREEDY is optimal.

Theorem 1.7. *The competitive ratio of* GREEDY *is at most* $2 - 1/N$.

Proof. First, we assume that no packets arrive after the first time step in which all queues of GREEDY become empty. To justify this assumption, we show a partition of the input sequence into phases such that each phase satisfies the following two conditions: (i) the queues of GREEDY and OPT are all empty at the start and end of the phase, and (ii) no packets arrive in the phase after the first time step in which all queues of GREEDY become empty. Hence, each phase can be seen as a separate input sequence satisfying the assumption, and thus it suffices in this case to show the competitive ratio on any arbitrary phase.

To show how to create these phases, we consider the creation of the first phase. Let t be the first time in which all queues of GREEDY become empty. We postpone the packets arriving after t until OPT's queues become empty as well, say at time t'. Thus, OPT and GREEDY are both empty at time t'. Since the queues of GREEDY are already empty at t, changing the input sequence in this way can increase the benefit of OPT only. Let t' define the end of the first phase, and the next arrival time after t' define the start of the second phase. Clearly, we can continue in the same way to partition the remaining of the input sequence.

Before we proceed, we define a new notation. Let $\delta_j(t)$ (resp. $\delta_j^*(t)$) denote the total number of packets sent by GREEDY (resp. OPT) from Q_j (resp. Q_j^*) between times 0 and t, inclusive. Thus, if no further packets arrive at Q_j after t, then $\delta_j(t) + Q_j(t)$ and $\delta_j^*(t) + Q_j^*(t)$ respectively define the benefits of GREEDY and OPT from the j-th queue.

We now define a potential function $\Phi : \mathbb{Z} \mapsto \mathbb{Z}$ as follows.

$$\Phi(t) = (2 - \frac{1}{N}) \sum_{j=1}^{N} (\delta_j(t) + Q_j(t)) - \sum_{j=1}^{N} \left(\delta_j^*(t) + Q_j^*(t)\right)$$

Clearly, it suffices to prove that $\Phi(t) \geq 0$ for any $t \geq 0$. We conduct an induction proof over time. As nothing arrives before the sequence starts, $\Phi(0) = 0$ trivially holds. Now, assume the claim holds at any time before t. We want to show that it also holds at t. Notice that $\Phi(t)$ changes only in arrival and send events. Thus, t is either of these two events.

First, assume that t is a send event. Since sending a packet from a queue Q_j increases $\delta_j(\cdot)$ by 1 and decreases $Q_j(\cdot)$ by 1, the potential does not change and thus $\Phi(t) = \Phi(t-1) \geq 0$.

Assume now that t is an arrival event. Notice first that in arrival events, $\delta_j(t) = \delta_j(t-1)$ and $\delta_j^*(t) = \delta_j^*(t-1)$. If both algorithms accept the arriving packet or both reject it, the potential increases by $(1 - 1/N)$ or it does not change at all. If only GREEDY accepts the arriving packet, the potential increases by $(2 - 1/N)$. In any of these cases, $\Phi(t) \geq \Phi(t-1) \geq 0$.

Now, consider the critical case where the arriving packet is accepted by OPT and rejected by GREEDY. Let Q_k be the queue at which the packet is rejected. We re-write the potential function in the following way.

$$\Phi(t) = \sum_{j=1}^{N} Q_j(t) - \frac{1}{N}\sum_{j=1}^{N} Q_j^*(t) + \sum_{j=1}^{N}\delta_j(t) - \sum_{j=1}^{N}\delta_j^*(t)$$
$$+ (1 - \frac{1}{N})\sum_{j=1}^{N}\delta_j(t) - (1 - \frac{1}{N})\sum_{j=1}^{N}\left(Q_j^*(t) - Q_j(t)\right) \ .$$

Clearly, $Q_k(t) = B$, and thus $\sum_{j=1}^{N} Q_j(t) \geq B$. Moreover,

$$\frac{1}{N}\sum_{j=1}^{N} Q_j^*(t) \leq \frac{1}{N} \cdot N \cdot B = B \ .$$

Thus,

$$\sum_{j=1}^{N} Q_j(t) - \frac{1}{N}\sum_{j=1}^{N} Q_j^*(t) \geq 0 \ . \tag{1.5.1}$$

Furthermore, by assumption, the queues of GREEDY have not been empty before t, and thus GREEDY has sent a packet in each time step before t. Hence,

$$\sum_{j=1}^{N}\delta_j(t) - \sum_{j=1}^{N}\delta_j^*(t) \geq 0 \ . \tag{1.5.2}$$

We want to show now that $\delta_j(t) + Q_j(t) \geq Q_j^*(t)$, for any $1 \leq j \leq N$. Let $A_j(t)$ denote the number of packets that arrive at Q_j between times 0 and t, inclusive. We distinguish between two cases: either (i) Q_j overflows before t, and thus $\delta_j(t) + Q_j(t) \geq B \geq Q_j^*(t)$; or (ii) Q_j does not overflow before t, and thus $\delta_j(t) + Q_j(t) = A_j(t) \geq Q_j^*(t)$. Hence, $\delta_j(t) \geq Q_j^*(t) - Q_j(t)$ in either case, and therefore

$$(1 - \frac{1}{N})\sum_{j=1}^{N}\delta_j(t) - (1 - \frac{1}{N})\sum_{j=1}^{N}\left(Q_j^*(t) - Q_j(t)\right) \geq 0 \ . \tag{1.5.3}$$

Finally, we conclude from Inequalities 1.5.1 - 1.5.3 that $\Phi(t) \geq 0$. □

1.6 Problem Definition: Models and Results

The problem of buffer management has been studied in many different models in the context of online algorithms. These models do not vary only in the switching architecture they adopt, but also in several other aspects, such as whether buffers are of FIFO or an arbitrary regime, whether packet values are arbitrary or just the same, or whether packets are delay-sensitive or not.

In this work, we study this problem in several models, and each model in different varieties. We present new results, some of which are tight, and give more insight into the "competitiveness" of these problems. In this section, we define our models in details and present the results we obtained in each one.

1.6.1 The FIFO Model

In the FIFO model, the switch consists only of one buffer. This can be seen as an N $\times$ 1 OQ switch, i.e., with only one output port and packets arrive directly at its single output queue. Due to the FIFO property, the sequence of sent (transmitted) packets has to be a subsequence of the arriving packets. Packets in this model are assumed to have size 1, so a packet can be sent completely in one time step. Furthermore, the queue capacity is a fixed parameter B, i.e., the queue cannot store more than B packets at a time.

Clearly, throughput maximization would be trivial in this model if all packets have the same value. Therefore, we assume that each packet p has an arbitrary value, denoted by $v(p)$. A buffer management algorithm can reject arriving packets and also preempt (drop) packets that were previously inserted into the queue. The goal is to maximize the total value of sent packets.

Our Results

We study a special class of online algorithms, called *comparison-based algorithms*. A comparison-based algorithm makes its decisions based solely on the relative order between packet values with no regard to the actual values. This kind of algorithms proves to be robust in the realm of QoS switches. As we saw in Section 1.3, a packet value stands only for the packet's service level, and thus it does not have any intrinsic meaning in itself. However, it has been observed that even slight changes to the packet values can result in substantial changes in the outcome of current buffer management algorithms, even though the relative order of the corresponding service levels is preserved. Hence, comparison-based algorithms emerge as robust buffer management policies since their behavior is independent of how the service levels are implemented in practice.

We aim first to give some insight into the competitiveness of comparison-based algorithms. Our main result is a lower bound of $1 + 1/\sqrt{2} \approx 1.707$ on the competitive ratio of any deterministic comparison-based algorithm. This significantly improves upon a lower bound of 1.419 that is known for general deterministic algorithms. We then give a comparison-based algorithm, CPG, that matches our lower bound in the case of *monotonic sequences*, i.e., packets arrive in a non-decreasing order according to their values. We further study the competitive ratio of CPG for general sequences and show that its competitive ratio in this case is at least 1.829.

Our results on the FIFO model are presented in Chapter 2.

1.6.2 The Model of Priority Queues

In this model, the switch is equipped with multiple queues of limited capacities, where each queue stores packets of one value only. More specifically, let $V = \{v_1 < \cdots < v_m\}$ be a set of m non-negative packet values, and $Q = \{Q_1, \ldots, Q_m\}$ be a set of m queues. Arriving packets are stored in the m queues, such that a packet of value v_i is stored in Q_i. All packets have size 1, and all queues have the same capacity B. In each time step, a non-empty queue is chosen and exactly one packet is sent from this queue. Again, the goal is to maximize the total value of sent packets.

This model can be seen as an OQ switch of only one output port, and at this output port, m queues are placed, where m is the number of service classes that are defined by the network administrator. In fact, in real-world switches, e.g. Cisco switches [cis06], packets are buffered exactly in this style and this is referred to by *priority queuing* in the terminology of packet switching.

Our Results

We analyze a natural greedy algorithm, GREEDY, which sends in each time step a packet with the greatest value. We show that GREEDY is $(1 + r)$-competitive, where $r = \max_{1 \le i \le m-1}\{v_i/v_{i+1}\}$. Thus, the competitive ratio of GREEDY is strictly below 2, and it tends to 1 as r tends to 0. For example, choosing the packet values to be powers of 2 makes GREEDY 3/2-competitive.

Furthermore, we show a lower bound of $2 - v_m/\sum_{i=1}^{m} v_i$ on the competitiveness of any deterministic online algorithm. In fact, one can easily show that GREEDY is asymptotically optimal, even with a competitive ratio near 2. Notice that when r tends to 1, the differences between packet values will shrink to 0 and thus the lower bound will tend to $2 - 1/m$, which is asymptotically 2.

Our results on this model are presented in Chapter 3. In subsequent, independent works, some of these results were improved. Namely, Itoh and Yoshimoto [IY15] show that GREEDY is $(1 + r)$-competitive also in the case

where each queue has a different capacity. Kawahara et al. [KKM15] conduct a tight analysis on the competitive ratio of GREEDY in the case where $v_1 = 1$ and $v_m = \alpha \geq 1$, and show it is exactly $2 - \min_{1 \leq i \leq m-1}\{v_{i+1} / \sum_{i=1}^{i+1} v_i\}$. They also give a lower bound of $1 + (\alpha + \alpha^2 + \alpha^3)/(1 + 4\alpha + 3\alpha^2 + 4\alpha^3 + \alpha^4)$.

1.6.3 The CIOQ Model

We consider in this model a CIOQ switch as it is described in Section 1.2, i.e., 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 ($i = 1, \ldots, N$) and corresponds to output port j ($j = 1, \ldots, N$) is denoted by Q_{ij}. An output queue that is placed at output port j ($j = 1, \ldots, N$) is denoted by Q_j. For any input or output queue Q, $Q(t)$ 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. Furthermore, all queues have limited capacity. We denote by $B(Q)$ the capacity of a queue Q. All packets have size 1. For each packet p in the input sequence, $v(p)$, $\text{in}(p)$, and $\text{out}(p)$ denote p's value, input port, and output port, respectively, where $\text{in}(p)$ and $\text{out}(p)$ take on values between 1 and N.

Each time step is divided into three phases: arrival, scheduling and transmission phases. The arrival and transmission of packets are defined as usual. 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. 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 $s = 1, \ldots, S$. 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 $t[s]$.

We study two variants of this model, the unit-value and the general-value variants. As the names imply, packets have all value 1 in the first variant, and arbitrary values in the second variant. In the latter case, packets that are already queued can be preempted before they are sent.

Our Results

Our main objective in this model is to devise online algorithms that are both competitive and efficient. All online algorithms known for this prob-

lem are based on computing 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.

All algorithms that we present in this chapter 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. Such matchings can be built in time linear to the number of edges in the graph. Clearly, this is substantially more efficient than computing a maximum matching, which is known to take $O(N^{5/2})$ time [HK73]. Moreover, computing maximal matchings complies more with the current practice in distributed systems where packet scheduling has to perform in real time.

With respect to competitiveness, we show that our algorithm for the unit-value case is 3-competitive for any speedup, and thus it achieves the best competitive ratio known for this problem [KR06]. In the general-value case, we improve on a previous algorithm that is 6-competitive [KR08] and show that our algorithm has a competitive ratio of $3 + 2\sqrt{2} \approx 5.828$ for any speedup.

Our results on the CIOQ model are presented in Chapter 4.

1.6.4 The Buffered Crossbar Model

The model of buffered crossbar switches is obtained from the CIOQ model by adding further queues at the crosspoints of the switching fabric. Crossbar queues are also non-FIFO. A crossbar queue that is placed at the crosspoint of input port i ($i = 1, \ldots, N$) and output port j ($j = 1, \ldots, N$) is denoted by C_{ij}.

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.

We also study two variants of this model, the unit-value and the general-value variants, and packets can be preempted in the general-value variant.

Our Results

For the unit-value case, Kesselman et al. [KKS12a] give 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 variant of this algorithm and show that it achieves a competitive ratio of $12 + 2\sqrt{2} \approx 14.828$ for any speedup.

These results on the CIOQ model are presented in Chapter 6.

1.6.5 The Bounded-Delay Model

With this model, we go back to switches of one buffer. As mentioned in Section 1.4, packets with deadlines are stored in a single queue. The queue has unlimited capacity, so all arriving packets are stored in the queue and the packet which is not sent before its deadline is dropped (lost). Each packet has an arbitrary value. Furthermore, each packet p has a size $\rho(p) \geq 1$. Thus, to send p completely, the packet must be transmitted over $\rho(p)$ time steps.

For ease of exposition, we call $\rho(p)$ the processing time of p. We also call a packet that is completely transmitted a *completed* packet. Our objective in this case is to maximize the total value of packets that are completed before their deadlines.

For each packet p, we denote by $r(p)$, $d(p)$, and $v(p)$ the arrival time, deadline and value of p, respectively. Since time is discrete in our models, processing times, arrival times and deadlines are integers, while a packet's value can be any positive number. Clearly, for any packet p, it must hold that $d(p) - r(p) \geq \rho(p)$. Moreover, in any time step, only one packet can be processed. Preemption in this model has a boarder meaning. Specifically, a packet that is under process can be preempted, i.e., interrupted, to process another packet. However, a preempted packet can be later on processed again and its processing is resumed from the last point at which it was interrupted. Therefore, we say that a packet p is completed if p is processed over a number of integral time intervals $[a_1, b_1], \ldots, [a_l, b_l]$, such that $r(p) \leq a_1$, $b_l \leq d(p)$, and $\sum_{k=1}^{l} b_k - a_k = \rho(p)$.

Let $s_p(t)$ denote the remaining processing time of a packet p at time t. We say that a packet p is pending at time t if $s_p(t) > 0$ and $t + s_p(t) \leq d(p)$, i.e., p is still not completed at t and if it is immediately processed at t without preemption, it can be completed before its deadline. Since the queue's capacity is unlimited, unlimited number of pending packets can be maintained at any time.

Our Results

This problem has its roots in the area of job scheduling. It is known that without any restriction on deadlines and processing times, no deterministic online algorithm can achieve a bounded competitive ratio, even if all packets have value of 1 [BHS94]. Therefore, only results for special cases have been

pursued.

Woeginger [Woe94], also in the context of job scheduling, shows that no deterministic algorithm can be better than 4-competitive if packets have equal processing times and tight deadlines, i.e., for any packet p, $d(p) = r(p) + \rho(p)$. He also provides an algorithm with a matching competitive ratio.

We first consider a special case of deadlines, the so-called *agreeable* or *order-respective* deadlines, i.e., for any two packets p and q, $r(p) \leq r(q) \Leftrightarrow d(p) \leq d(q)$. We show that no deterministic algorithm has a bounded competitive ratio in this case. Then, we consider the more restricted case where all processing times are equal and give a natural greedy algorithm, which we call SG, and show that it achieves a competitive ratio of 4. Clearly, Woeginger's model is a special case of agreeable deadlines. Therefore, his lower bound of 4 carries over to our model and thus SG is optimal.

Our result on the CIOQ model is presented in Chapter 6.

1.7 Related Work

The problem of buffer management has gained a substantial interest in the theory community since the beginning of the last decade. This line of research, in the context of online algorithms and competitive analysis, was initiated by the two seminal papers of Aiello et al. [AMRR00] and Mansour et al. [MPSL00] on the FIFO model. (These two papers appeared after that in journal versions as [AMRR05] and [MPSL04].)

We next survey the previous results on the models that are most related to ours. For a comprehensive and relatively up-to-date survey of competitive buffer management, we refer the reader to the survey of Goldwasser [Gol10].

The FIFO Model

Aiello et al. [AMRR05] consider a more restricted variant of the FIFO model where preemption is not allowed and packet values are either 1 or $\alpha > 1$. They present a lower bound of $2 - 1/\alpha$ on the competitive ratio of any deterministic or randomized algorithm. This lower bound is matched by Andelman et al. [AMZ03, Zhu04] with a deterministic online algorithm.

In a more general case of the non-preemptive model, packets have arbitrary values between 1 and $\alpha > 1$. Andelman et al. [AMZ03, Zhu04] give a lower bound of $1 + \ln(\alpha)$ on the competitive ratio of deterministic algorithms, and Zhu [Zhu04] shows a lower bound of $(\log \alpha + 2)/2$ for randomized algorithms. Finally, Andelman and Mansour [AM03] give a deterministic algorithm with a competitive ratio $2 + \ln(\alpha) + O(\ln^2(\alpha)/B)$, which matches the lower bound of $1 + \ln(\alpha)$ up to a constant additive factor.

In the preemptive model with arbitrary packet values, which is considered in our work, Mansour et al. [MPSL04] show that a natural greedy

algorithm, which is called GREEDY and is presented in Chapter 2, is 4-competitive. Kesselman et al. [KLM$^+$04] show that the exact competitive ratio of GREEDY is $2 - 1/B$. Kesselman et al. [KMvS05] give the state-of-the-art algorithm PG, and prove that PG is 1.983-competitive. Additionally, they give a lower bound of $(1 + \sqrt{5})/2 \approx 1.618$ on the competitive ratio of PG and a lower bound of 1.419 on the competitive ratio of any deterministic algorithm. Bansal et al. [BFK$^+$04] slightly modify PG and show that the modified algorithm is 1.75-competitive. Finally, Englert and Westermann [EW09] show that PG is in fact 1.732-competitive and give a lower bound of $1+(1/\sqrt{2}) \approx 1.707$ on its competitive ratio. We will discuss GREEDY and PG further in Chapter 2, but we notice here that PG is not a comparison-based algorithm.

The only randomized algorithm in this model is given by Andelman [And05]. He presents a comparison-based algorithm which flips a coin between two deterministic algorithms, one of them is GREEDY. He shows that this algorithm is 1.75-competitive.

In the case where packets take on only two values, 1 and $\alpha > 1$, Lotker and Patt-Shamir [LPS03] give a preemptive algorithm with a competitive ratio of 1.304. Kesselman et al. [KLM$^+$04] show a lower bound of 1.282 on the competitive ratio of any deterministic algorithm, and Englert and Westermann [EW09] give an algorithm that matches this lower bound. Finally, Andelman [And05] presents a randomized algorithm with a competitive ratio of 5/4 and shows a lower bound of 1.197 on the competitive ratio of any randomized algorithm.

The Model of Priority Queues

In [AB10], we consider a special case of this model with two packet values, 1 and $\alpha > 1$. We show that a greedy algorithm, which is called GREEDY and is presented in Chapter 3, achieves a competitive ratio of $(\alpha + 2)/(\alpha + 1)$. Thus, given the lower bound that we obtain in this work, GREEDY turns out to be optimal in this special case.

The IQ Model

We present this model as an example problem in Section 1.5.1. In fact, if all packets have the same value, this model is a special case of our model of priority queues. Moreover, it is also a special case of the CIOQ model, because with a speedup of 1 and only one output port, the CIOQ model is equivalent to an IQ model.

In the unit-value case of the IQ model, Azar and Richter [AR05] provide the aforementioned lower bound of $2 - 1/N$ on the competitive ratio of any deterministic algorithm. They also show that any work-conserving policy for this model is 2-competitive, i.e., an algorithm that sends a packet in each

time step as long as its queues are not empty. Furthermore, for arbitrary B, they give a lower bound of $(1.366 - \Theta(1/N))$. Albers and Schmidt [AS06] improve these results and give a policy called SEMI-GREEDY that is 1.889-competitive, for any B with $N \gg B$, and 1.857-competitive, for $B = 2$. They also give a lower bound of $e/(e-1) \approx 1.582$ on the competitive ratio of any deterministic algorithm, if $N \gg B$. Azar and Litichevskey [AL06] match this lower bound for large B.

If randomization is allowed, Azar and Richter [AR05] give an $e/(e-1)$-competitive algorithm, for $B > \log N$, and a lower bound of $1.46 - \Theta(1/N)$, for $B = 1$. Albers and Schmidt [AS06] improve the lower bound to 1.466, for any B and large N, and 1.231, for $N = 2$. Bienkowski and Madry [BM08] present a policy for $N = 2$ and any B that achieves the optimal competitive ratio of 1.231.

For the general-value case of the IQ model with N FIFO queues, Azar and Richter [AR05] give a preemptive algorithm with a competitive ratio of $(4 + 2\ln(\alpha))$ for arbitrary packet values in the interval $[1, \alpha]$, and a 2.6-competitive algorithm for the special case of two packet values 1 and $\alpha > 1$. Azar and Richter [AR04] improve that by a comparison-based 3-competitive algorithm, called TLH. Itoh and Takahashi [IT06] improve the analysis of TLH to a competitive ratio of $3 - 1/\alpha$.

The CIOQ Model

For the unit-value case of the CIOQ model, Kesselman and Rosén [KR06] present a greedy algorithm that is 3-competitive for any speedup. For the general-value case, they give an algorithm that is 6-competitive for any speedup [KR08]. As mentioned in Section 1.6.3, we improve upon these results in this work.

In the general-value case, Kesselman and Rosén [KR06] consider also a CIOQ model with FIFO queues. They 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 values. Azar and Richter [AR06] improve The latter result and give an algorithm with a competitive ratio of 8 for any speedup. Kesselman et al. [KKS12b] show that this algorithm is indeed 7.47-competitive.

The Buffered Crossbar Model

For the unit-value case of the buffered crossbar model, Kesselman et al. [KKS12a] present a greedy algorithm with a competitive ratio of 4 for any speedup. For the general-value case, they give an algorithm that is 16.24-competitive for any speedup. As mentioned in Section 1.6.4, these results are improved in this work.

For the buffered crossbar model with FIFO queues, Kesselman et al. [KKS10] give a 19.95-competitive algorithm for any speedup.

The Bounded-Delay Model

The variant of this model where all packets have size 1 has received a considerable interest, probably because it also represents a scheduling problem of unit-length jobs on one machine. We cover here only a small part of this extensive area of research.

The best lower bound known in this case is $\phi \approx 1.618$ [Haj01, AMZ03, CF03]. It is derived using a sequence of packets which can reside in the queue for at most 2 time steps. In the case of arbitrary deadlines, Englert and Westermann [EW07] give a deterministic algorithm with a competitive ratio of 1.828, which is the best competitive ratio known. Thus, it is still an open problem whether there exists an optimal algorithm in the case of arbitrary deadlines.

The case of agreeable deadlines has been also studied in this model, i.e., with packet sizes of 1. An algorithm that is given by Jeż et al. [JLSS12] has a competitive ratio of 1.618. Since the above lower bound of 1.618 applies also for agreeable deadlines, this algorithm is optimal.

In the case of randomized algorithms, Bienkowski et al. [BCJ08] give a lower bound of 1.33 on the competitive ratio of any randomized algorithm. This lower bound is matched only in the case of agreeable deadlines [JLSS12]. For general deadlines, the best randomized algorithm known has a competitive ratio of 1.582 [CCF$^+$06].

In our variant of arbitrary sizes (processing times), all related results are obtained in the context of job scheduling. We therefore adopt the same terminology of that area and use "job" in place of "packet". As mentioned above, without any restriction on deadlines and processing times, no deterministic online algorithm can achieve a bounded competitive ratio, even if all jobs have value of 1 [BHS94]. Therefore, only results for special cases have been pursued.

Woeginger [Woe94] considers the case where all jobs have tight deadlines, i.e., an arriving job is either processed immediately without preemption or it is lost. This problem is also known as *interval scheduling*. In the case of equal processing times, he shows that no deterministic algorithm can be better than 4-competitive. He also provides an algorithm with a matching competitive ratio.

In the model of equal processing times and arbitrary deadlines, Dürr et al. [DJT12] give a deterministic algorithm that is 5-competitive. This result is improved by Kim [Kim11] to a 4.24-competitive algorithm. The lower bound of 4 applies also in this case. Thus, it is still an open problem whether there exists an optimal algorithm for the model of arbitrary deadlines.

Finally, Fung et al. [FPZ14] and Chrobak et al. [CJST07] study randomization under several variants of this model, and Epstein et al. [EJSvS16] study the problem of interval scheduling on m related machines.

1.8 Bibliographical Notes

Most of the results presented in this work have previously been published as joint works. Specifically, Chapter 3 is based on a joint work with Alexander Souza [ABS13], and Chapter 2 is based on a joint work with Matthias Englert and Matthias Westermann [ABEW16]. Chapter 4 and 5 are based on another joint work with Matthias Englert and Matthias Westermann [ABEWar]. The results of Chapter 6 are based on recent work and have not been published yet.

Chapter 2

FIFO Buffer Management

Despite its compact and simple formulation, the FIFO model has raised the most challenging and intriguing questions in the problem of buffer management—most of which are still open. For instance, although the non-preemptive case has been settled along time ago, we still do not know how preemption can be used to obtain optimal online algorithms. Moreover, randomization, in both the preemptive and non-preemptive cases, is still unfrequented. In this chapter, we address a further question of how a special class of online algorithms, the so-called comparison-based algorithms, perform in this model. More specifically, we examine whether these algorithms can achieve a competitive ratio better than 2.

We call an algorithm comparison-based if it makes its decisions based solely on the relative order between values with no regard to the actual values. In other words, changing the packet values does not change the algorithm's behavior as long as the relative order between the values is not changed.

This property renders comparison-based algorithms a robust tool in the realm of QoS switches. Recall from Section 1.3 that a packet value stands only for the packet's service level, and thus it does not have any intrinsic meaning in itself. However, without the comparison-based property, slight changes to the packet values would result in substantial changes in the outcome of the buffer management policy, even though the relative order of the corresponding service levels is preserved. Hence, it is sufficiently interesting to devise online buffer management algorithms that perform independently of how the service levels are implemented in practice.

The following algorithm, which we call GREEDY, is an example of this kind of algorithms.

> When a packet p arrives, accept p if the queue is not full. Otherwise, let q be the packet with the smallest value in the queue. If $v(p) \geq v(q)$, drop q and accept p; otherwise, reject p. In send events, send the packet at the head of the queue.

Clearly, GREEDY keeps the most valuable B packets in each time step. Kesselman et al. [KLM+04] show that GREEDY is 2-competitive. Since the introduction of GREEDY, it has been an open problem to show whether a comparison-based algorithm exists with a competitive ratio below 2.

To improve the competitive ratio of GREEDY, Kesselman et al. [KMvS05] extend its preemption rule in the following way: Upon the arrival of a packet p, the algorithm *proactively* drops the first packet in the queue whose value is within a fraction $1/\beta$ of the value of p, for some $\beta \geq 1$. The resulting algorithm is called Preemptive-Greedy (PG). However, the additional rule of PG makes it a non-comparison-based algorithm. Kesselman et al. show that PG is 1.983-competitive. They also show a lower bound of $(1+\sqrt{5})/2 \approx 1.618$ on the competitive ratio of PG and a lower bound of 1.419 on the competitive ratio of any deterministic algorithm. Englert and Westermann [EW09] show that PG is in fact 1.732-competitive.

In Section 2.1, we give more insight into the competitiveness of comparison-based algorithms through the introduction of the zero-one principle. We also show that GREEDY cannot be better than 2-competitive, even if packets arrive in a specific order. In Section 2.2, we show a lower bound of $1 + 1/\sqrt{2} \approx 1.707$ on the competitive ratio of any deterministic comparison-based algorithm. Thus, our lower bound improves upon the lower bound of 1.419 for general deterministic algorithms. We then consider the case of monotonic sequences in Section 2.3 and present a comparison-based algorithm, CPG, that matches our lower bound in this case. We further study the competitive ratio of CPG for general sequences and show that its competitive ratio in this case is at least 1.829.

We conclude our discussion with another intriguing question of whether a comparison-based algorithm with a competitive ratio close to $1 + 1/\sqrt{2} \approx 1.707$ could exist. If so, this would mean that we do not need to know the actual values of packets in order to compete with PG, the best non-comparison-based algorithm so far. If not, and in particular if 2 is the right lower bound for any comparison-based algorithm, the desired robustness of this kind of algorithms must come at a price, namely, a significantly degraded performance.

2.1 The 0/1 Principle

Azar and Richter [AR04] introduce the 0/1 principle for the analysis of comparison-based algorithms in a variety of buffering models. They show the following theorem.

Theorem 2.1 ([AR04])**.** *Let* ALG *be a comparison-based switching algorithm (deterministic or randomized).* ALG *is c-competitive if and only if* ALG *is c-competitive for all input sequences of packets with values 0 and 1, under every possible way of breaking ties between equal values.*

The 0/1 principle is by its own an interesting motivation to consider comparison-based algorithms due to the significant simplification it brings into the analysis of buffer management algorithms. So, for such an algorithm ALG, it is sufficient by Theorem 2.1 to show the competitive ratio of ALG for only 0/1 sequences. We denote a packet of value 0 as *0-packet*, and of value 1 as *1-packet*.

As a warm-up, we first use the 0/1 principle to show a lower bound of asymptotically 2 on the competitive ratio of GREEDY.

Theorem 2.2. *The competitive ratio of* GREEDY *is at least* $2 - 1/B$.

Proof. We generate an input sequence σ in the following way. In the first time step, $B - 1$ 0-packets arrive, followed by a single 1-packet. In time steps $2, \ldots, B - 1$, one 1-packet arrives in each step. After that, a burst of B 1-packets arrive in the B-th time step.

Clearly, the optimal algorithm accepts all 1-packets and rejects all 0-packets. Hence, it makes a total benefit of $2\,B - 1$. On the other hand, GREEDY accepts all the 0-packets arriving in the first time step. It also accepts all the 1-packets arriving in the first $B - 1$ steps. However, since no overflow occurs in these time steps, none of these 1-packets can be sent before the burst of 1-packets arrives in the last time step. Therefore, GREEDY can keep only B 1-packets from all 1-packets that arrive in this sequence, and thus its competitive ratio is $(2B - 1)/B = 2 - 1/B$. □

Notice that in the sequence used in the construction of the above lower bound, packets arrive in a non-decreasing order of values, i.e., for any two packets p and q, $v(p) \leq v(q)$ if and only if p arrives before q. We call sequences of this kind *monotonic*. In Section 2.3, we give a comparison-based algorithm that is optimal in the case of monotonic sequences.

The 0/1 principle has been used to show several results in the buffer management problem. For our model of a single FIFO queue, Andelman [And05] employs the 0/1 principle to give a randomized comparison-based algorithm with a competitive ratio of 1.75. In fact, this is the only randomized algorithm known for this model. In a related model with multiple FIFO queues, Azar and Richter [AR04] use it to show a comparison-based algorithm with a competitive ratio of 3.

In a different model, where the buffer is not FIFO and packet values are not known for the online algorithm, Azar et al. [ACG13] use it to show a randomized algorithm with a competitive ratio of 1.69. This algorithm is modified to a 1.55-competitive randomized algorithm, and a lower bound of 1.5 on the competitive ratio of any randomized algorithm is shown for that model [AC15].

2.2 Lower Bound

The following theorem shows that no deterministic comparison-based algorithm can be better than $1 + 1/\sqrt{2} \approx 1.707$.

Theorem 2.3. *The competitive ratio of any deterministic comparison-based algorithm is at least* $1 + 1/\sqrt{2} \approx 1.707$.

Proof. Fix an online algorithm ONL. The adversary constructs a sequence of packets with non-decreasing values over a number of iterations. The 0/1 values corresponding to the packets' real values are revealed only when the sequence stops. In each iteration, the adversary generates a burst of B packets in one time slot followed by a number of individual packets, each in one time slot. We call a slot with B arrivals a *bursty* slot, and a slot with one arrival a *light* slot. A construction routine is repeated by the adversary until the desired lower bound is obtained. For $i \geq 0$, let f_i denote the i-th bursty slot, and let t_i denote the number of time slots that ONL takes to send and preempt all packets that it has in slot f_i.

As initialization, the adversary generates B packets in the first time slot. Thus, the first slot is f_0. After that, the adversary generates t_0 light slots, i.e., one packet arrives in each slot. Now, starting with $i = 0$, the adversary constructs the rest of the sequence by the following routine which is repeated until $t_i \geq B/\sqrt{2}$.

1. Generate the bursty slot f_{i+1}.

2. If $t_i \geq B/\sqrt{2}$, stop the sequence. At this point, all packets that arrive between f_0 and f_i (inclusive) are revealed as 0-packets and all packets after that are revealed as 1-packets, i.e., the 1-packets are those which arrive in the t_i light slots and in the bursty slot f_{i+1}. Clearly, the optimal algorithm, denoted as OPT, will send all the 1-packets while ONL will gain only the B 1-packets which it has in slot f_{i+1}. Notice that ONL sends only 0-packets in the t_i light slots. Hence, provided that $t_i \geq B/\sqrt{2}$,

$$\begin{aligned} \frac{\text{OPT}}{\text{ONL}} &= \frac{t_i + B}{B} \\ &\geq \frac{B/\sqrt{2} + B}{B} \ . \end{aligned}$$

3. If $t_i < B/\sqrt{2}$, continue the sequence after f_{i+1} by generating t_{i+1} light slots.

   (a) If $t_{i+1} \leq t_i$, stop the sequence. At this point, all packets that arrive between f_0 and f_i (inclusive) are revealed as 0-packets and all packets after that are revealed as 1-packets, i.e., the 1-packets

are those which arrive in the t_i and t_{i+1} light slots and in the bursty slot f_{i+1}. Clearly, OPT will send all the 1-packets while ONL will send only t_{i+1} packets of the B 1-packets which it has in slot f_{i+1} and also the t_{i+1} 1-packets which it collects after f_{i+1}. Hence, provided that $B > \sqrt{2} \cdot t_i$ and $t_i \geq t_{i+1}$,

$$\begin{aligned}\frac{\text{OPT}}{\text{ONL}} &= \frac{t_i + B + t_{i+1}}{t_{i+1} + t_{i+1}} \\ &\geq \frac{t_i + \sqrt{2} \cdot t_i + t_{i+1}}{2 \cdot t_{i+1}} \\ &\geq \frac{(1+\sqrt{2})t_{i+1} + t_{i+1}}{2 \cdot t_{i+1}} \quad .\end{aligned}$$

(b) If $t_{i+1} > t_i$, set $i = i + 1$ and repeat the routine.

Obviously, the above routine terminates eventually, because a new iteration is invoked only when $t_{i+1} > t_i$, and thus the amount of t_i is strictly increased in each iteration. Therefore, there must exist i such that $t_i \geq B/\sqrt{2}$. □

2.3 Algorithm CPG

We present a comparison-based preemptive greedy (CPG) algorithm. It follows a similar rule of preemption as PG, but without addressing the actual values of packets: Roughly speaking, once you have a set S of β packets in the queue with a packet r in front of them, such that r is less valuable than each packet in S, preempt r.

CPG is described more precisely in Algorithm 1. To avoid using the same set of packets to preempt many other packets, it associates with each arriving packet p a non-negative *credit*, denoted by $c(p)$. For a set S of packets, $c(S)$ will also denote the total credit of all packets in S. We now describe the above preemption rule in more details.

First, we present the notations of *preemptable* packets and *preempting* sets. Assume that a packet p arrives at time t. Let $Q(t)$ be the set of packets in CPG's queue immediately before t. For any packet $r \in Q(t)$, if there exists a set $S \subseteq (Q(t) \cup \{p\}) \setminus \{r\}$ such that (i) $p \in S$, (ii) $c(S) \geq \beta$, and (iii) for each packet $q \in (S)$, $\text{arr}(q) \geq \text{arr}(r)$ and $v(q) \geq v(r)$, then we say that r is *preemptable* by p. Furthermore, we call S a *preempting* set of r.

A packet r is preempted upon the arrival of another packet p if r is the first packet in the queue (in the FIFO order) such that r is preemptable by p and the value of r is less than the value of the packet that is behind r in the queue (if any). After a packet r is preempted, CPG invokes a subroutine CHARGE to deduct a total of β units from the credits of the preempting packets of r. This charging operation can be done arbitrarily, but subject to the non-negative constraint of credits, i.e., $c(p) \geq 0$, for any packet p. After

Algorithm 1: CPG

arrival event. A packet p arrives at time t:

$c(p) \leftarrow 1$;

Let r be the first packet in the queue such that r is preemptable by p and the value of r is less than the value of the packet that is behind r (if any).

if *r exists* **then**
  let S be a preempting set of r;
  drop r and CHARGE(S);

if *the queue is not full* **then**
  insert p;
else
  let q be the packet with the smallest value in the queue;
  if $v(q) < v(p)$ **then**
    drop q and insert p;
  else
    reject p;

that, the algorithm proceeds similarly to GREEDY: It inserts the arriving packet p into the queue if the queue is not full or p is more valuable than the packet with the least value in the queue. In the latter case, the packet with the least value is dropped. Otherwise, p is rejected. Finally, in send events, CPG simply sends the packet at the head of the queue.

Lost Packets. We distinguish between three types of packets lost by CPG:

1. Rejected packets: An arriving packet p is rejected if the queue is full and no packet in the queue is less valuable than p.

2. Evicted packets: An enqueued packet q is evicted by an arriving packet p if the queue is full and q is the least valuable among p and the packets in the queue.

3. Preempted packets: An enqueued packet r is preempted upon the arrival of another packet p if r is the first packet in the queue such that r is preemptable by p and the value of r is less than the value of the packet that is behind r (if any).

Notice that a 1-packet can only be evicted by a 1-packet. Also, if a 1-packet q is preempted, the preempting packets of q are all 1-packets.

2.3.1 Monotonic Sequences

In this section, we consider input sequences in which packets arrive with non-decreasing values, i.e., for any two packets p and q, $v(p) \leq v(q)$ if and only if p arrives before q. As shown in Section 2.1, the competitive ratio of GREEDY is asymptotically 2 in this case.

Theorem 2.4. *Choosing* $\beta = \sqrt{2} + 1$*, the competitive ratio of* CPG *is at most* $1 + 1/\sqrt{2} \approx 1.707$*.*

For the rest of the analysis, we fix an event sequence σ of only 0- and 1-packets. Furthermore, let $Q(t)$ (resp. $Q^*(t)$) denote the set of 1-packets in the queue of CPG (resp. OPT) at time t.

Assumptions on the Optimal and the Online Algorithms. Notice that OPT, in contrast to CPG, can determine whether a packet of σ has value 0 or 1. Therefore, we can assume that OPT accepts arriving 1-packets as long as its queue is not full, and rejects all 0-packets. In send events, it sends 1-packets (in FIFO order) unless its queue is empty.

We further assume that no packets arrive after the first time in which the queue of CPG becomes empty. This assumption is also without loss of generality as we can partition σ into phases such that each phase satisfies this assumption and the queues of CPG and OPT are both empty at the start and the end of the phase. Then, it is sufficient to show the competitive ratio on any arbitrary phase. Consider for example the creation of the first phase. Let t be the first time in which the queue of CPG becomes empty. We postpone the packets arriving after t until OPT's queue is empty as well, say at time t', so that OPT and CPG are both empty at t'. This change can only increase the benefit of OPT. Clearly, t' defines the end of the first phase, and the next arrival event in σ defines the start of the second phase. The remaining of σ can be further partitioned in the same way.

Overflow Time Slot. We call a time slot in which CPG rejects or evicts 1-packets an overflow time slot. Assume for the moment that at least one overflow time slot occurs in σ. For the rest of the analysis, we will use f to denote the last overflow slot, and t_f to denote the time immediately before this slot ends. Obviously, rejection and eviction of 1-packets can happen only when the queue of CPG is full of 1-packets. Let t'_f be the point of time immediately before the first rejection or eviction in f takes place. Thus, the number of 1-packets in the queue at time t'_f is B. Thereafter, between t'_f and t_f, any 1-packet that is evicted or preempted is replaced by the 1-packet whose arrival invokes that eviction or preemption. Thus, the size of the queue does not change between t'_f and t_f, and hence the following observation.

Remark 2.5. $|Q(t_f)| = B$.

Furthermore, the following lemma shows that the B 1-packets in the queue at time t_f can be used to preempt at most one 1-packet in later arrival events.

Lemma 2.6. *Consider any arrival event e. Let t be the time immediately after e and let $D(t)$ denote the set of packets in the queue at time t except the head packet. Then, $c(D(t)) < \beta$.*

Proof. We show the lemma by contradiction. Let e be the first arrival event in σ, such that immediately after e, say at time t, $c(D(t)) \geq \beta$. Hence, immediately before e, say at time t', $\beta > c(D(t')) \geq \beta - 1$, since the total credit of the queue cannot increase by more than 1 in each arrival event.

Now, let p be the packet arriving in e and let q be the head packet at the arrival of p. Recall that σ is monotonic. Thus, the packets behind q in the queue and packet p are all at least as valuable as q. Hence, adding the credit of p to $c(D(t'))$, these packets would preempt q upon the arrival of p, and thus the total credit would decrease by 1. Therefore, $c(D(\cdot))$ does not change between t' and t which contradicts the definition of e. □

Before we proceed, we introduce further notations. Let $\textsc{arr}(t, t')$ denote the set of 1-packets that arrive in σ between time t and t'. Furthermore, let $\textsc{sent}(t, t')$ and $\textsc{lost}(t, t')$ denote the set of 1-packets that CPG sends and loses, respectively, between time t and t'. Similarly, we define $\textsc{sent}^*(t, t')$ and $\textsc{lost}^*(t, t')$ for OPT.

Lemma 2.7. *It holds that*

$$|\textsc{lost}(0, t_f)| - |\textsc{lost}^*(0, t_f)| + |Q(t_f)| - |Q^*(t_f)| = |\textsc{sent}^*(0, t_f)| - |\textsc{sent}(0, t_f)|.$$

Proof. The lemma follows from this simple observation:

$$\begin{aligned} &|Q(t_f)| + |\textsc{sent}(0, t_f)| + |\textsc{lost}(0, t_f)| \\ &\quad = |\textsc{arr}(0, t_f)| \\ &\quad = |Q^*(t_f)| + |\textsc{sent}^*(0, t_f)| + |\textsc{lost}^*(0, t_f)| \ . \end{aligned}$$

□

The following lemma is crucial for the analysis of CPG. It essentially upper-bounds the number of 1-packets that CPG loses between the start of the sequence and the end of the overflow slot.

Lemma 2.8. $|\textsc{lost}(0, t_f)| - |\textsc{lost}^*(0, t_f)| + |Q(t_f)| - |Q^*(t_f)| \leq \frac{\beta}{\beta+1} B$.

Proof. First, we present further notations. If an algorithm ALG does not send anything in a sent event t, we say that ALG sends a $\emptyset$-packet in t. We call a send event in which OPT sends an x-packet and CPG sends a y-packet an x/y send event, where x and y take on values from $\{0, 1, \emptyset\}$. Furthermore, we denote by $\delta_{x/y}(t, t')$ the number of x/y send events that occur between time t and time t'.

Now, observe that

$$|\text{SENT}^*(0, t_f)| = \delta_{1/0}(0, t_f) + \delta_{1/1}(0, t_f) + \delta_{1/\emptyset}(0, t_f) \ ,$$
$$|\text{SENT}(0, t_f)| = \delta_{0/1}(0, t_f) + \delta_{1/1}(0, t_f) + \delta_{\emptyset/1}(0, t_f) \ .$$

Recall that OPT does not send 0-packets and that, by assumption, the queue of CPG does not get empty before t_f. Thus, $\delta_{0/1}(0, t_f) = \delta_{1/\emptyset}(0, t_f) = 0$, and therefore

$$|\text{SENT}^*(0, t_f)| - |\text{SENT}(0, t_f)| = \delta_{1/0}(0, t_f) - \delta_{\emptyset/1}(0, t_f) \leq \delta_{1/0}(0, t_f) \ .$$

Hence, given Lemma 2.7, it suffices to show that $\delta_{1/0}(0, t_f) \leq \lfloor \frac{\beta}{\beta+1} B \rfloor$.

Assume for the sake of contradiction that $\delta_{1/0}(0, t_f) > \lfloor \frac{\beta}{\beta+1} B \rfloor$. Let M_1 (resp. M_0) be the set of 1-packets (resp. 0-packets) that OPT (resp. CPG) sends in these 1/0 send events. Thus,

$$|M_1| = |M_0| \geq \lfloor \frac{\beta}{\beta+1} B \rfloor + 1 > \frac{\beta}{\beta+1} B \ . \tag{2.3.1}$$

Let p (resp. q) denote the first arriving packet in M_1 (resp. M_0). Furthermore, let r be the last arriving packet in M_0 and denote the time in which it is sent by t_r. Recall that σ is monotonic. Thus, all the 1-packets of M_1 arrive after r. Moreover, since CPG's buffer is FIFO, none of these 1-packets is sent before t_r. Also, since r, which is a 0-packet, is before them in the queue and is eventually sent, CPG does not either reject, evict or preempt any 1-packet from M_1 before t_r. Therefore, all the 1-packets of M_1 must be in the queue of CPG at time t_r.

Let's now look closely at the queue of CPG immediately after the arrival of p. Let that time be denoted as t_p. Since q is sent with p in the same 1/0 send event and since r is between q and p (by the above argument), q and r must be in the queue as well at time t_p. Moreover, since r is the last arriving 0-packet in M_0, the remaining 0-packets of M_0 must also be in the queue at t_p. Hence, the queue of CPG contains all the packets of M_0 along with p at time t_p.

Next, notice that all the 1-packets of M_1 are inserted in CPG's queue after r (which is a 0-packet) without preempting it. Since the credits of packets are used only in preemption, the credits of these 1-packets must be used to preempt other packets before r. Let R be the set of these preempted packets. Obviously,

$$|R| \geq \lfloor |M_1|/\beta \rfloor > |M_1|/\beta - 1 \ . \tag{2.3.2}$$

Since the packets of R cannot be preempted before the arrivals of the packets of M_1, all of them must be then before r in the queue at time t_p. Thus, the queue of CPG contains the packets of both M_0 and R along with p at time t_p. Clearly, $M_0 \cap R \cap \{p\} = \emptyset$. Hence, given Inequalities 2.3.1 and 2.3.2, the size of CPG's queue at t_p is at least

$$|M_0| + |R| + 1 > |M_0| + |M_1|/\beta = \frac{\beta+1}{\beta} M_0 > \frac{\beta+1}{\beta} \frac{\beta}{\beta+1} B = B \ ,$$

which is strictly larger than B, and hence a contradiction. □

So far, our discussion has been focused on one half of the scene; namely, the one between the start of the sequence and the end of the last overflow slot. We shall now move our focus to the second half which extends from time t_f until the end of the sequence.

First, let t_0 be defined as follows: $t_0 = 0$ if no overflow slot occurs in σ, and $t_0 = t_f$ otherwise. Notice that in both cases, no 1-packet is rejected or evicted by CPG after t_0. Moreover, let T denote the first time by which the sequence stops and the queues of OPT and CPG are both empty. Thus, the benefits of OPT and CPG are given by $|\textsc{sent}^*(0,T)|$ and $|\textsc{sent}(0,T)|$, respectively.

The following lemma is the main ingredient of the proof of the competitive ratio.

Lemma 2.9. $|\textsc{sent}(0,T)| \geq (\beta-1)\left(|\textsc{lost}(0,T)| - |\textsc{lost}^*(0,T)|\right)$.

Proof. Obviously, we can write $|\textsc{sent}(0,T)|$ as follows:

$$\begin{aligned} |\textsc{sent}(0,T)| &= |\textsc{sent}(0,t_0)| + |Q(t_0)| + |\textsc{arr}(t_0,T)| - |\textsc{lost}(t_0,T)| \\ &\geq |Q(t_0)| + |\textsc{arr}(t_0,T)| - |\textsc{lost}(t_0,T)| \ . \end{aligned}$$

Due to the fact that no 1-packet is rejected or evicted by CPG after t_0, all packets in $\textsc{lost}(t_0,T)$ are lost by preemption. We further notice that all these packets are preempted using packets that arrive after t_0. This is trivial in case $t_0 = 0$, and follows from Lemma 2.6 in case $t_0 = t_f$. (In fact, in the latter case, at most one packet of $\textsc{lost}(t_0,T)$ can be preempted using the credits of packets that are in the queue at time t_f, but this anomaly can be covered by introducing an additive constant in the competitive ratio of CPG.) Since preempting a packet requires a credit of β, preempting the packets of $\textsc{lost}(t_0,T)$ implies the arrival of at least new $\beta\,|\textsc{lost}(t_0,T)|$ 1-packets that are inserted into the queue after t_0. Thus, $|\textsc{arr}(t_0,T)| \geq \beta\,|\textsc{lost}(t_0,T)|$, and hence we can rewrite $|\textsc{sent}(0,T)|$ in the following way:

$$\begin{aligned} |\textsc{sent}(0,T)| &\geq |Q(t_0)| + \beta\,|\textsc{lost}(t_0,T)| - |\textsc{lost}(t_0,T)| \\ &= |Q(t_0)| + (\beta-1)\,|\textsc{lost}(t_0,T)| \\ &\geq |Q(t_0)| + (\beta-1)\,\left(|\textsc{lost}(t_0,T)| - |\textsc{lost}^*(t_0,T)|\right) \ . \end{aligned}$$

Now, if $t_0 = 0$, then $|Q(t_0)| = 0$ and thus the lemma follows immediately. If $t_0 = t_f$, we continue as follows:

$$\begin{aligned}
|\textsc{sent}(0,T)| \geq B+ & \\
&(\beta-1)\Big(|\textsc{lost}(t_f,T)| - |\textsc{lost}^*(t_f,T)| - |Q(t_f)| + |Q^*(t_f)|\Big) \\
&\geq \frac{\beta+1}{\beta}\Big(|\textsc{lost}(0,t_f)| - |\textsc{lost}^*(0,t_f)| + |Q(t_f)| - |Q^*(t_f)|\Big) \\
&\quad + (\beta-1)\Big(|\textsc{lost}(t_f,T)| - |\textsc{lost}^*(t_f,T)| - |Q(t_f)| + |Q^*(t_f)|\Big) \\
&= (\beta-1)\Big(|\textsc{lost}(0,T)| - |\textsc{lost}^*(0,T)|\Big) \ ,
\end{aligned}$$

where the first inequality follows from Remark 2.5, the second inequality from Lemma 2.8, and the equality from the fact that $\beta - 1 = (\beta+1)/\beta$, for $\beta = \sqrt{2}+1$. □

Now, we use Lemma 2.9 to show that $|\textsc{sent}^*(0,T)| \leq \frac{\beta}{\beta-1}\,|\textsc{sent}(0,T)|$, which obviously completes the proof of Theorem 2.4:

$$\begin{aligned}
|\textsc{sent}^*(0,T)| &= |\textsc{arr}(0,T)| - |\textsc{lost}^*(0,T)| \\
&= |\textsc{sent}(0,T)| + |\textsc{lost}(0,T)| - |\textsc{lost}^*(0,T)| \\
&\leq |\textsc{sent}(0,T)| + \frac{1}{\beta-1}\,|\textsc{sent}(0,T)| \\
&= \frac{\beta}{\beta-1}\,|\textsc{sent}(0,T)| \ .
\end{aligned}$$

2.3.2 General Sequences

Theorem 2.4 shows that CPG is an optimal comparison-based algorithm in the case of monotonic sequences. In this section, we investigate how this algorithm performs on general sequences.

We notice that Lemma 2.6 does not necessarily hold for general sequences. Therefore, after an overflow of 1-packets takes place, the total credit of the 1-packets in the online buffer can significantly exceed β and thus some of these packets may be used in a subsequent time steps to preempt other packets from the same group, i.e., the group of the B 1-packets from the overflow slot. Consequently, the lower bound of B on the number of CPG's sent 1-packets may no longer hold in the general case, resulting in a competitive ratio worse than 1.707. Such a bad scenario for CPG is illustrated in the proof of the following theorem and leads to a lower bound of 1.829 on its competitive ratio.

Theorem 2.10. *For any value of β,* CPG *cannot be better than 1.829-competitive.*

Proof. The adversary generates one of the following two sequences based on the value of β:

Case 1. $\beta \leq 2.206$**:** In the first time slot, B 1-packets are generated in an increasing order (with respect to their original values). After that, no more packets arrive. Clearly, OPT sends all the B packets, while in CPG, every β packets preempt a packet from the front. Thus, CPG preempts B/β in total. Hence, its competitive ratio is given by

$$\frac{\text{OPT}}{\text{CPG}} = \frac{B}{B - B/\beta} = \frac{\beta}{\beta - 1} \geq 1.829 \ .$$

Case 2. $\beta > 2.206$**:** In the first time slot, $(B-1)$ 0-packets are generated followed by a single 1-packet. Then, over the next $\beta B/(\beta+1)-1$ time slots, a single 1-packet is generated in each slot. Let M_1 denote the set of those 1-packets that arrive in the first $\beta B/(\beta+1)$ time slots. After that, in slot number $\beta B/(\beta+1)+1$, B 1-packets arrive at once. Let M_2 denote the set of these packets. Finally, in the next $B/(\beta(\beta+1))$ time slots, a single 1-packet arrives in each slot. Let M_3 denote the set of these packets. After that, no more packets arrive.

Clearly, OPT sends all the 1-packets in the sequence. To minimize the number of 1-packets sent by CPG, the adversary can choose the original values of the 1-packets in the following malicious way. First, the values of packets in M_2 are all strictly less than the smallest value in M_1. Let M_2' denote the set of the first $B/(\beta+1)$ packets in M_2. The packets of M_2' are ordered as follows. For each group of β packets, starting from the earliest, the first packet is strictly smaller than the $\beta-1$ packets behind it, and all the β packets of this group are strictly smaller than all packets before them in M_2'. For example, for $\beta = 3$, theses groups may look like $|50, 51, 51|40, 41, 41|30, 31, 31|\ldots$. For the rest of M_2, i.e., the set $M_2 \setminus M_2'$, packets are given values that are strictly less than the smallest value in M_2'. Finally, the packets in M_3 are all assigned a value that is equal to the greatest value in M_2'.

Obviously, CPG accepts all the $\beta B/(\beta+1)$ packets of M_1 and uses them to preempt $B/(\beta+1)$ 0-packets. Meanwhile, the rest of the B 0-packets are sent in the first $\beta B/(\beta+1)$ time slots. Thus, the packets of M_1 will be all in the queue of CPG when the packets of M_2 arrive. Clearly, this leads to an overflow of 1-packets and only the packets of M_2' can be accepted in this time slot. These packets are inserted with full credits into the queue, and thus when each packet from M_3 arrives, it groups with $\beta-1$ packets from M_2' to preempt the first packet in one β-group of M_2', according to the above description of M_2'. Therefore, CPG sends a total of B 1-packets only,

and hence its competitive ratio is given by

$$\begin{aligned}\frac{\text{OPT}}{\text{CPG}} &= \frac{|M_1| + |M_2| + |M_3|}{B} \\ &= \frac{\beta}{\beta+1} + 1 + \frac{1}{\beta(\beta+1)} \\ &= \frac{\beta(\beta+1) + \beta^2 + 1}{\beta(\beta+1)} \geq 1.829 \ .\end{aligned}$$

□

Finally, we make an observation on the preemption rule of CPG. It can be easily verified that CPG remains optimal in the monotonic-sequence case even with the following change of the preemption rule: "Let r be the first packet in the queue such that r is preemptable by p." Thus, without any regard to packets that follow r in the queue. Let's call this changed version of CPG the *relaxed version*. We next show that the relaxed version of CPG is no better than 2-competitive in the case of general sequences. Thus, the preemption rule as it is defined in Algorithm 1 is necessary to show a competitive ratio below 2 for CPG. In fact, this definition of CPG corresponds to the modified version of PG given by Bansal et al. [BFK+04].

Theorem 2.11. *For any value of β, the relaxed version of* CPG *cannot be better than 2-competitive.*

Proof. Assume that CPG has the following configuration of the queue at the end of the first slot: $B/(\beta+2)$ 0-packets in the front of the queue, followed by $B/(\beta+2)$ 1-packets, but these 1-packets are placed among another set of 0-packets in a specific way, so that they all get preempted before the next overflow slot. Specifically, each 1-packet of those is preceded by $(\beta-1)$ 0-packets and followed by one 0-packet. Assume that OPT has these 1-packets in its queue at the end of the first slot as well.

After the first time slot, nothing arrives until CPG sends all the $B/(\beta+2)$ 0-packets in the front of the queue. In these time steps, OPT sends the $B/(\beta+2)$ 1-packets. After that, the rest of the buffer is filled again with 0-packets, and then a single 1-packet arrives in each slot of the next $\beta B/(\beta+1)$ time slots. It is over these time slots where CPG preempts the $B/(\beta+2)$ 1-packets of the first slot while sending only 0-packets. Notice that OPT sends the single 1-packets over these time slots. Then the overflow of B 1-packets happens and we just continue after that in the same way as we did in the proof of Theorem 2.10.

Summing up all the 1-packets of OPT results in

$$\frac{B}{\beta+2} + \frac{\beta B}{\beta+1} + B + \frac{B}{\beta(\beta+1)} \,,$$

while CPG sends only B 1-packets (after the overflow). Clearly, that leads to a competitive ratio above 2 for any value of β. □

Chapter 3

Packet Scheduling with Priority Queues

In this chapter, we consider the switch model where m queues are used to store packets of values $V = \{v_1 < \cdots < v_m\}$, such that the i-th queue stores only packets of value v_i.

In Section 3.1, We derive an upper bound for the competitive ratio of a natural greedy algorithm, GREEDY, which sends in each time step a packet with the greatest value. Specifically, we show that GREEDY is $(1 + r)$-competitive, where $r = \max_{1 \le i \le m-1}\{v_i/v_{i+1}\}$. Clearly, the competitive ratio of GREEDY is strictly below 2 and it tends to 1 as r tends to 0. Deriving a competitive ratio that is a function of packet values allows us to tune these values so as to minimize the competitive ratio. For example, choosing the packet values to be powers of 2 makes GREEDY 3/2-competitive.

We assume that all queues have the same capacity B. In fact, GREEDY is 2-competitive even if each queue has its own capacity. This can be shown by an argument similar to the proof of Theorem 1.4 in Chapter 1 using the fact that GREEDY sends the packet with the greatest value in each send event.

In Section 3.2, we show a lower bound of of $2 - v_m/\sum_{i=1}^{m} v_i$ on the competitive ratio of any deterministic algorithm. Thus, when r tends to 1, the differences between packet values will shrink to 0 and thus the lower bound will tend to $2 - 1/m$, which is asymptotically 2. Clearly, when r tends to 1, the competitive ratio of GREEDY is also asymptotically 2 and thus GREEDY is optimal in this case.

3.1 Algorithm GREEDY

We define GREEDY as follows. At arrive events, GREEDY accepts packets of any value until the respective queue becomes full. At send events, it serves the non-empty queue, if any, with the highest packet value, i.e., a packet of value v_i is sent only when all queues Q_j are empty, for $i < j \le m$.

Let $r = \max_{1 \le i \le m-1}\{v_i/v_{i+1}\}$. The following theorem shows that GREEDY is $(1+r)$-competitive.

Theorem 3.1. *The competitive ratio of* GREEDY *is at most* $1 + r$.

First, we fix an event sequence σ. We call a packet of value v_i a *v_i-packet*, and the i-th queue is denoted as the *v_i-queue*. Let A_i and A_i^* denote the total number of v_i-packets accepted by GREEDY and OPT, respectively. Hence, $\text{GREEDY}(\sigma) = \sum_{i=1}^{m} v_i A_i$, and $\text{OPT}(\sigma) = \sum_{i=1}^{m} v_i A_i^*$.

We begin by showing that OPT and GREEDY send the same number of v_m-packets.

Lemma 3.2. $A_m^* = A_m$.

Proof. By definition of GREEDY, v_m-packets enjoy absolute priority at sending. Hence, the number of v_m-packets that GREEDY sends is maximum, i.e., $A_m \ge A_m^*$.

Assume that A_m becomes greater than A_m^* for the first time at arrive event t. This means that OPT rejects at t a v_m-packet that GREEDY accepts. Hence, OPT's v_m-queue was full immediately before t but GREEDY's had at least one vacancy. Let $j = 1, \ldots, m-1$. Since $A_m = A_m^*$ immediately before t, there must have been a send event t' before t where OPT sent a v_j-packet while its v_m-queue was not empty. Change OPT's schedule by sending a v_m-packet at t' instead of the v_j-packet. Clearly, this yields an increase in OPT's benefit and the rejected v_m-packet at time t can be accepted. □

The following lemma shows an upper bound on the total number of packets that OPT accepts but GREEDY rejects.

Lemma 3.3. *For any* $1 \le i \le m-1$,

$$\sum_{j=i}^{m-1} \left(A_j^* - A_j\right) \le \sum_{j=i+1}^{m} A_j.$$

Proof. For any $1 \le i \le m-1$, we define the following notion of *time interval*. A time interval I ends with a send event, and the next time interval starts with the first arrive event after the end of I. We call a time interval I *red interval* (or r-interval) if the value of any packet sent by GREEDY in I is in $\{v_i, \ldots, v_m\} \subseteq V$, and *green interval* (or g-interval) if the value of any packet sent by GREEDY in I is in $\{v_1, \ldots, v_{i-1}\} \subseteq V$ or I contains send events in which GREEDY does not send any packet. We partition σ into r- and g-intervals such that no two consecutive intervals are of the same color. Clearly, this partitioning is feasible.

For the rest of the proof, let j be any number in $\{i, \ldots, m\}$. The following observation follows from the definition of g-intervals; otherwise, GREEDY would send v_j-packets in a g-interval and thus it is no longer a g-interval.

Observation 3.4. *In any g-interval, any v_j-queue of* GREEDY *is empty and no v_j-packets arrive.*

Let $A_j(I)$ (resp., $A_j^*(I)$) denote the total number of v_j-packets accepted by GREEDY (resp., OPT) in time interval I, and let $\Re$ denote the set of all r-intervals. Given Observation 3.4, $A_j = \sum_{I \in \Re} A_j(I)$ and $A_j^* = \sum_{I \in \Re} A_j^*(I)$. Hence, it suffices to prove the lemma for any r-interval. Thus, we fix an r-interval I and show that $\sum_{j=i}^{m-1}(A_j^*(I) - A_j(I)) \leq \sum_{j=i+1}^{m} A_j(I)$.

Let $\delta_j(t)$ denote the total number of v_j-packets sent by GREEDY between the first and the t-th event of I, inclusive. Let $b_j(t)$ denote the size of GREEDY's v_j-queue right after the t-th event of I. By Observation 3.4, interval I starts with GREEDY's v_j-queue empty. Thus, if no further v_j-packets arrive in I after event t, we get that $A_j(I) = \delta_j(t) + b_j(t)$. For OPT, we define $\delta_j^*(t)$ as the total number of v_j-packets *that arrive in* I and are sent by OPT between the first and the t-th event of I, inclusive; and $b_j^*(t)$ as the number of packets *that arrive in* I and still reside in OPT's v_j-queue right after the t-th event of I. Thus, if no further v_j-packets arrive in I after event t, we also get that $A_j^*(I) = \delta_j^*(t) + b_j^*(t)$.

We now define a potential function $\varphi : \mathbb{Z} \mapsto \mathbb{Z}$ as follows.

$$\varphi(t) = \sum_{j=i}^{m} (\delta_j(t) + b_j(t)) + \sum_{j=i+1}^{m-1} (\delta_j(t) + b_j(t)) - \sum_{j=i}^{m-1} \left(\delta_j^*(t) + b_j^*(t)\right)$$

Clearly, it suffices to prove that $\varphi(t) \geq 0$ for any $t \geq 0$. We conduct an induction proof over the number of arrive and send events of I. (Notice that $\varphi(t)$ changes on arrive and send events only.) For the induction basis, $\varphi(0) = 0$ as nothing arrives in I before its first event. Assume that $\varphi(t) \geq 0$ for $t \leq k-1$. We will show that $\varphi(k) \geq 0$.

First, assume that the k-th event is a send event. Since I is an r-interval, GREEDY sends a v_l-packet, where $i \leq l \leq m$. We show that $\delta_j(k) + b_j(k) = \delta_j(k-1) + b_j(k-1)$, for any $j = i, \ldots, m$. This is obvious for $j \neq l$ as nothing has changed in v_j-queue since event $k-1$. For $j = l$, $\delta_j(k) = \delta_j(k-1) + 1$ as the number of sent v_j-packets increases by 1, and $b_j(k) = b_j(k-1) - 1$ as the size of v_j-queue decreases by 1. Thus, $\delta_j(k) + b_j(k) = \delta_j(k-1) + b_j(k-1)$, as required. Similarly, if OPT sends a v_j-packet that arrives in I, one can show that $\delta_j^*(k) + b_j^*(k) = \delta_j^*(k-1) + b_j^*(k-1)$, for any $j = i, \ldots, m-1$. Therefore, $\varphi(k) = \varphi(k-1) \geq 0$.

Assume now that the k-th event is an arrive event. We first observe that in arrive events, $\delta_j(k) = \delta_j(k-1)$ and $\delta_j^*(k) = \delta_j^*(k-1)$ as nothing has been sent since the last event. Moreover, since I is an r-interval, the arriving packet must be a v_l-packet, where $i \leq l \leq m$. Now, we distinguish between four cases: (1) Both OPT and GREEDY reject the packet, (2) both accept it, (3) GREEDY accepts and OPT rejects, or (4) OPT accepts and GREEDY rejects. The potential function does clearly not change in the first case. In

the second and third case, v_l-queue increases by one in GREEDY and by at most one in OPT (it does not increase in OPT in the third case). Thus, $\varphi(k)$ will increase by at least 1 due to change in GREEDY's queue (it increases by 2 if $i+1 \leq l \leq m-1$) and will decrease by at most 1 due to change in OPT's queue. Hence, $\varphi(k) \geq \varphi(k-1)$.

Now, consider the last case where OPT accepts but GREEDY rejects. First, we rewrite $\varphi(k)$ as

$$\varphi(k) = \sum_{j=i}^{m} \delta_j(k) + \sum_{j=i}^{m} b_j(k) + \sum_{j=i+1}^{m-1} (\delta_j(k) + b_j(k)) - \sum_{j=i}^{m-1} \delta_j^*(k) - \sum_{j=i+1}^{m-1} b_j^*(k) - b_i^*(k)\,.$$

Since the arriving v_l-packet is rejected by GREEDY, GREEDY's v_l-queue must be full. Thus, $\sum_{j=i}^{m} b_j(k) \geq B$. However, the size of OPT's v_i-queue can never exceed B. Hence

$$\sum_{j=i}^{m} (b_j(k)) - b_i^*(k) \geq 0. \tag{3.1.1}$$

Let η be the number of send events in interval I up to the current arrive event k. Clearly, $\sum_{j=i}^{m-1} \delta_j^*(k) \leq \eta$. By definition of r-intervals, GREEDY has sent exactly η packets of value $v_j \in \{v_i, \ldots, v_m\}$. Thus, $\sum_{j=i}^{m} \delta_j(k) = \eta$. Hence,

$$\sum_{j=i}^{m} \delta_j(k) - \sum_{j=i}^{m-1} \delta_j^*(k) \geq 0. \tag{3.1.2}$$

Recall that b_j^* counts OPT's v_j-packets that it accepts only in I. Thus, for $i+1 \leq j \leq m-1$, at least $b_j^*(k)$ packets must arrive in I up to event k. Since GREEDY starts I with an empty v_j-queue, and since $b_j^*(k) \leq B$, GREEDY must be able to accept at least $b_j^*(k)$ packets of all v_j-packets arriving in I. Thus, recalling that $\delta_j(k) + b_j(k)$ represents the number of v_j-packets that GREEDY accepts in I up to k, we get that $\delta_j(k) + b_j(k) \geq b_j^*(k)$, and hence,

$$\sum_{j=i+1}^{m-1} (\delta_j(k) + b_j(k)) - \sum_{j=i+1}^{m-1} b_j^*(k) \geq 0. \tag{3.1.3}$$

By summing up inequalities 3.1.1 - 3.1.3, we get that $\varphi(k) \geq 0$, and thus the lemma follows. □

Next, we show a weighted version of Lemma 3.3 when $i = 1$, which is, along with Lemma 3.6, essential to improve the competitive ratio of GREEDY from 2 to $1 + r$.

Lemma 3.5. *It holds that*

$$\sum_{j=1}^{m-1} v_j(A_j^* - A_j) \leq \sum_{j=1}^{m-1} v_j A_{j+1}.$$

Proof. For simplicity of exposition, let $m = 4$. By Lemma 3.3,

$$\begin{aligned}
i = 1: \quad (A_1^* - A_1) + (A_2^* - A_2) + (A_3^* - A_3) &\leq A_2 + A_3 + A_4,\\
i = 2: \quad (A_2^* - A_2) + (A_3^* - A_3) &\leq A_3 + A_4,\\
i = 3: \quad (A_3^* - A_3) &\leq A_4.
\end{aligned}$$

Recall that $v_i < v_{i+1}$, for all $1 \leq i \leq m-1$. Thus, multiplying both sides of $(i = 1)$ by v_1, both sides of $(i = 2)$ by $(v_2 - v_1)$, both sides of $(i = 3)$ by $(v_3 - v_2)$, and adding up all the resulting inequalities, we get

$$v_1(A_1^* - A_1) + v_2(A_2^* - A_2) + v_3(A_3^* - A_3) \leq v_1 A_2 + v_2 A_3 + v_3 A_4.$$

Clearly, the argument above can be generalized for any $m \geq 2$. □

The following lemma is the last ingredient of the proof of Theorem 3.1.

Lemma 3.6. *It holds that*

$$\frac{\sum_{j=1}^{m-1} v_j A_{j+1}}{\sum_{j=1}^{m-1} v_{j+1} A_{j+1}} \leq r.$$

Proof. Let $r = v_k/v_{k+1}$, for some $1 \leq k \leq m-1$. Hence, by definition of r, for any $1 \leq j \leq m-1$,

$$\frac{v_j}{v_{j+1}} \leq \frac{v_k}{v_{k+1}},$$

which can be rewritten as $v_{k+1} \cdot v_j \leq v_k \cdot v_{j+1}$. Thus, multiplying by A_{j+1} and then summing over all j, we get

$$v_{k+1} \sum_{j=1}^{m-1} v_j A_{j+1} \leq v_k \sum_{j=1}^{m-1} v_{j+1} A_{j+1},$$

from which the lemma follows. □

Now, we conclude the proof of Theorem 3.1 as follows.

$$\begin{aligned}
\frac{\text{OPT}(\sigma)}{\text{GREEDY}(\sigma)} &= \frac{\sum_{j=1}^{m} v_j A_j^*}{\sum_{j=1}^{m} v_j A_j} \\
&= 1 + \frac{\sum_{j=1}^{m} v_j (A_j^* - A_j)}{\sum_{j=1}^{m} v_j A_j} \\
&= 1 + \frac{\sum_{j=1}^{m-1} v_j (A_j^* - A_j)}{\sum_{j=1}^{m} v_j A_j} \\
&\leq 1 + \frac{\sum_{j=1}^{m-1} v_j A_{j+1}}{\sum_{j=1}^{m} v_j A_j} \\
&\leq 1 + \frac{\sum_{j=1}^{m-1} v_j A_{j+1}}{\sum_{j=1}^{m-1} v_{j+1} A_{j+1}} \\
&\leq 1 + r,
\end{aligned}$$

where the last equality follows from Lemma 3.2, while the first and last inequalities follow from Lemma 3.5 and Lemma 3.6, respectively.

3.2 Lower Bound

Next, we show a lower bound on the competitive ratio of any deterministic online algorithm.

Theorem 3.7. *The competitive ratio of any deterministic online algorithm is at least* $2 - v_m/(\sum_{i=1}^{m} v_i)$.

Proof. We first assume that any online and offline algorithm in this model is *work-conserving*: It must send a packet if it has a non-empty queue at sending time. Moreover, since accepting a packet of one value does not interfere with packets of other values, we may extend the concept of work-conserving and assume that any algorithm must accept a packet arriving at a queue if this queue has residual capacity. Notice that each algorithm, which is not work-conserving can be transformed into a work-conserving algorithm, without changing the benefit. This is because the work-conserving algorithm can send everything the other algorithm sends, but earlier; and it can accept everything the other algorithm accepts, but also earlier.

Let ALG be any deterministic online algorithm, which will have to compete against an offline algorithm OPT. Furthermore, assume that all queues are of size 1.

We construct an adversarial instance σ in the following way: In each time step $1 \leq i \leq m$, packets of *distinct* values arrive. We denote the set of values of the packets arriving in step i by $V_i \subseteq V$. Let $V_1 = V$. By work-conserving, we may assume that ALG accepts all packets in step 1. Let

s_1 be the packet value sent by ALG in the send event of step 1. Since ALG is online and deterministic, this value is well-defined. Define $V_2 = V_1 \setminus \{s_1\}$ and let s_2 be the packet value sent by ALG if V_2 arrives in step 2. Now define $V_3 = V_2 \setminus \{s_2\}$ and in general $V_{t+1} = V_t \setminus \{s_t\}$ for $t = 1, \ldots, m-1$. Clearly, $V_1 \supset V_2 \supset \cdots \supset V_m$.

Observe that all packets arriving in the time steps $2, \ldots, m$ are of values corresponding to non-empty queues in ALG. So ALG must reject all of these packets. Hence $\text{ALG}(\sigma) = \sum_{i=1}^{m} v_i$ since ALG accepts all packets arriving in the first step.

Now, we define the optimal algorithm. OPT accepts all packets V_1 in step 1, but sends the packet with value s_2 in the send event of that step. Since s_2 is still in the queues of ALG in step 2, we have $s_2 \in V_2$. Now OPT accepts the packet with value s_2 in step 2 and rejects all other packets. In the send event of step 2, OPT sends a packet with value s_3. In general, in the send event of step t, it sends a packet with value s_{t+1} for $t = 1, \ldots, m-1$; and, for $t = 2, \ldots, m$, OPT accepts the packet with value s_t in step t and rejects all other packets.

Observe that before the send event of step m, OPT still has one packet of each value in its queues and no further packets will arrive. So, OPT sends those packets in the send events of steps $m, \ldots, 2m-1$, in any order. Therefore, if ALG sends one packet of value v, OPT sends two packets of value v, except for the one packet with value s_1. Thus we have $\text{OPT}(\sigma) = 2 \cdot \sum_{i=1}^{m} v_i - s_1 \geq 2 \cdot \sum_{i=1}^{m} v_i - v_m$.

Therefore,

$$\begin{aligned} \frac{\text{OPT}(\sigma)}{\text{ALG}(\sigma)} &\geq \frac{2 \cdot \sum_{i=1}^{m} v_i \quad - v_m}{\sum_{i=1}^{m} v_i} \\ &= 2 - \frac{v_m}{\sum_{i=1}^{m} v_i}. \end{aligned}$$

□

Chapter 4

Forwarding Packets in CIOQ Switches

In this chapter, we study the buffer management problem in an $N \times N$ CIOQ switch. This time our objective is twofold: To devise online algorithms that are both competitive and efficient. All online algorithms known for this problem are based on computing 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.

All algorithms that we present in this chapter are based on greedy matching computations, i.e., we construct a matching incrementally by adding edges, one by one, until no more edges can be added. The resulting matching is called maximal. Clearly, a maximal matching can be built in time linear to the number of edges in the graph, and this is more efficient than computing a maximum matching, which is known to take $O(N^{5/2})$ time [HK73]. Moreover, computing maximal matchings complies more with the current practice in distributed systems where packet scheduling has to perform in real time.

With respect to competitiveness, we show in Section 4.1 that our algorithm for the unit-value case is 3-competitive for any speedup, and thus it achieves the best competitive ratio known for this problem [KR06]. In Section 4.2, we improve on a previous algorithm that is 6-competitive [KR08] in the general-value case, and show that our algorithm has a competitive

ratio of $3 + 2\sqrt{2} \approx 5.828$ for any speedup.

To obtain these results in an elegant way, we use the technique of modifying the optimal algorithm that is described in Section 1.5.1, but in a way that is far more involved. This technique has been used also by Jeż et al. [JLSS12] for another buffer management related problem. However, in that work, buffers are manipulated so that the optimal algorithm and the online algorithm always have identical buffer configurations—a property that is relaxed in our analysis.

4.1 Unit-value Case

In this case, all packets have value of 1. Our goal is thus to maximize the number of transmitted packets. We present the following algorithm which we call **G**reedy **M**atching (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 $U = \{u_1, \ldots, u_N\}$, $V = \{v_1, \ldots, v_N\}$ 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 4.1. *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 or sending them as early as possible does not change 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 4.1.1. *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 4.1.2. *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., 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 $|P^*| + |S^*| \leq 3\,|S|$.

We now show how to derive the competitive ratio of 3. First, we show in Lemma 4.2 how Modifications 4.1.1 and 4.1.2 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, $|S^*| \leq |S|$. After that, we show by Lemma 4.4 that $|P^*| \leq 2\,|S|$. Thus, the proof of Theorem 4.1 follows directly from these two lemmas.

Lemma 4.2. *For any* $i, j \in \{1, \ldots, N\}$ *and any time* t, *the following inequalities hold:*

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

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

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 before time t. 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 4.1.1. 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 4.1.2.

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 4.1.2 takes place, GM must transfer a packet from the same input port.

Lemma 4.3. *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 $j' \neq j$.*

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 4.2, 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 $j' \neq j$, such that $(u_i, v_{j'}) \in M_{t[s]}$. Hence, a packet p' is transferred from $Q_{ij'}$ in $t[s]$. □

Lemma 4.4. *The following inequality holds:*

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

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 4.1.1, 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 4.3, 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 4.1.1 and 4.1.2, 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. □

4.2 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 [KR08]. 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 $\beta \geq 1$ 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 [KR08], PG computes a maximal weighted matching in each scheduling cycle rather than a maximum weighted matching.

Theorem 4.5. *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 it knows in advance which packets it is going to send, it does not matter for OPT in which order these packets are released from their 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 4.2.1. *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 4.2.2. *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 4.2.3. *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.*

Notice that Modifications 4.2.2 and 4.2.3 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 $v(p) \leq \beta v(q)$.

Lemma 4.6. *For any* $i, j \in \{1, \ldots, N\}$ *and any time* t, *the following inequalities hold:*

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

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

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 $v(p) \leq \beta v(q)$. 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 $v(p) \leq \beta v(q)$ [resp. $v(p) \leq v(q)$]. 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. Notice 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 4.2.1], 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 $v(q) \leq v(q')$, 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 $v(p) \leq v(p')$. Notice that the insertion of p' happens only in one of two cases: (i) if a packet r with $v(r) \geq v(p')$ is inserted into Q_j (by Modifications 4.2.2), or (ii) if Q_j is full in t and $v(p') \leq \beta v(l_j(t))$ (by Modifications 4.2.3). 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 $k' \leq k$, 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 $v(r) \leq v(q')$. Since $v(p) \leq v(p') \leq v(r)$, $v(p) \leq v(q')$. 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, $v(l_j(t)) \leq v(q')$. Moreover, since $v(p') \leq \beta v(l_j(t))$, $v(p) \leq v(p') \leq \beta v(q')$. 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 $v(l_j(t)) \geq v(p')$.]

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, notice that the insertion of p into Q_j^* happens only in one of two cases: (i) if a packet r with $v(r) \geq v(p)$ is inserted into

Q_j (by Modification 4.2.2), or (ii) if Q_j is full in t and $v(p) \leq \beta v(l_j(t))$ (by Modifications 4.2.3). In case (ii), since Q_j is full in t, p must be aligned with a packet q in t. Since $v(p) \leq \beta v(l_j(t))$, $v(p) \leq \beta v(q)$. 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 $v(r) \geq v(p)$. 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 $v(p) \leq v(p')$. Furthermore, let q' be the packet aligned with p' in time $t-1$. Thus, $v(p) \leq v(p') \leq \beta v(q')$. 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).

Notice 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 $q \neq \emptyset$ and that $v(q) \geq v(r) \geq v(p)$. Therefore, we have $v(p) \leq v(r)$ in the first case, $v(p) \leq \beta v(q')$ in the second, and $v(p) \leq v(q)$ 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 $v(l_j(t)) \geq v(p)$.] □

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 4.9 how to fix this problem.

Lemma 4.7. *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 $j' \neq j$ and $v(p') \geq v(p)$.*

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 4.6, 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 $v(q) > \beta v(l_j(t[s]))$. Therefore, 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 4.7. We present it for the sake of completeness.

Lemma 4.8. *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 $v(p') \geq v(p)$.*

Proof. Assume that Q_j is not full in $t[s]$ or $v(p) > \beta v(l_j(t[s]))$. By Inequality I1 of Lemma 4.6, since OPT transfers p from Q^*_{ij} in $t[s]$, PG must have at the head of Q_{ij} a packet r with $v(r) \geq v(p)$. Thus, if $v(p) > \beta v(l_j(t[s]))$, then it must also hold that $v(r) > \beta v(l_j(t[s]))$. Hence, Q_j is not full in $t[s]$ or $v(r) > \beta v(l_j(t[s]))$, and therefore the edge (u_i, v_j) must be in E. Nevertheless, as PG does not transfer any packet to Q_j, (u_i, v_j) is not in $M_{t[s]}$. Hence, the node u_i must have been matched before considering (u_i, v_j) in the matching computation. 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). Clearly, this results in the transfer of a packet p' from $Q_{ij'}$. Moreover, $w(u_i, v_{j'}) \geq w(u_i, v_j)$, and therefore $v(p') \geq v(r) \geq v(p)$. □

Now, recall Inequality I2 of Lemma 4.6. 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 4.9. *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 4.2.1, PG transfers a packet p' from Q_{ij} in $t[s]$, and by Inequality I1 of Lemma 4.6, $v(p) \leq v(p')$. 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 4.7, 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 4.8, 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 $x(q_m)$. Notice 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:

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

Solving this recursion, we obtain that

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

Notice 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:

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

□

Chapter 5

Forwarding Packets in Buffered Crossbar Switches

As we saw in Chapter 1, the model of buffered crossbar switches is obtained from the CIOQ model by adding further queues at the crosspoints of the switching fabric. Crossbar queues are also non-FIFO. A crossbar queue that is placed at the crosspoint of input port i ($i = 1, \ldots, N$) and output port j ($j = 1, \ldots, N$) is denoted by C_{ij}.

As we did in Chapter 4, we study two variants of this model. In Section 5.1, we show a 3-competitive algorithm for any speedup, which is an improvement on a previous competitive ratio of 4 [KKS12a]. In Section 5.2, we also improve on a previous result of 16.24-competitiveness in the general-value case, and present a new algorithm with a competitive ratio of $12 + 2\sqrt{2} \approx 14.828$ for any speedup.

The analysis of these algorithms is similar to the analysis of their counterparts in the CIOQ model. We show that the technique of modifying the adversary can also be applied here, however not in a straightforward way, to obtain better competitive ratios.

5.1 Unit-value Case

First, recall that 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.

Kesselman et al. [KKS12a] consider the following algorithm, which we call Crossbar Greedy Unit (CGU), for the case where all packets have the

same value.

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: 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.

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

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

Theorem 5.1. *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 5.1.1. *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 5.1.2. *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 5.1.1 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 5.1.3. *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.*

Notice that the trick of extra packets is not used in the analysis of the algorithms presented in Chapter 4 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 4.1.

Lemma 5.2. *For any time* t *and any* $i, j \in \{1, \ldots, N\}$*, 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 before time t. 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 anything from Q^*_{ij}. However, either Q^*_{ij} is empty in this case or it cannot happen due to Modification 5.1.1. For I2, the first critical case is when CGU inserts a packet into C_{ij} while OPT does not insert anything into C^*_{ij}. However, either C^*_{ij} is full in this case or it cannot happen due to Modification 5.1.2. The second critical case for I2 is when OPT transfers a packet from C^*_{ij} while CGU does not transfer anything from C_{ij}. However, either C_{ij} is empty in this case or the size of C^*_{ij} does not decrease due to Modification 5.1.3. □

In the following, we use $S^*_{t[s]}$ to denote the set of OPT's *normal* packets in the input subphase of cycle $t[s]$. These are packets that OPT schedules through the normal channels, i.e., not privileged, and are part of the original input sequence, i.e., 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 5.3. *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 5.2, 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 input queue $Q_{ij'}$ 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 5.1.1, 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 5.1.2, 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 5.1.3, 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 5.4. *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 in t 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 5.4, $|Q_j^*(t)| = B(Q_j^*)$ 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 5.5. *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 $t[s]$, for $i \neq i'$. 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 5.1.2, since an extra packet p' is inserted into C_{ij}^*, C_{ij}^* cannot be full in $t[s]$. However, by Modification 5.1.1, p must be transferred to C_{ij}^* in this case and not directly to outside the switch. Hence, by Modification 5.1.2, 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, notice that $\text{OPT}(\sigma) = \sum_{t[s]} |S_{t[s]}^*| + |P_{t[s]}^*|$. Therefore, the proof of Theorem 5.1 follows immediately from Lemma 5.3 and 5.5.

5.2 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. [KKS12a].

Recall the notations $g_{ij}(t)$, $l_{ij}(t)$, and $l_j(t)$ that we used with algorithm PG (Section 4.2). 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 [KKS12a]. However, we show that to minimize the competitive ratio for this algorithm, these two parameters must take on different values.

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: 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.

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

Notice that all ties in CPG are broken arbitrarily.

Theorem 5.6. *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 analysis of CPG is carried out in a similar way as PG. We extend Assumptions A1 - A3 to include crossbar queues as well, and modify OPT in a slightly different way. 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 5.2.1. *Suppose that* CPG *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 5.2.2. *Suppose that* OPT *transfers a packet p to C^*_{ij} and* CPG *does not transfer any packet to C_{ij} in $t[s]$. If C_{ij} is not full in $t[s]$ or $v(p) > \beta\, v(lc_{ij}(t[s]))$, 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.*

Modification 5.2.3. *Suppose that* CPG *transfers a packet from C_{ij} and* OPT *does not transfer any packet from C^*_{ij} in $t[s]$. If C^*_{ij} is not empty in $t[s]$, we release the head packet p of C^*_{ij}, i.e., the packet with the greatest value in C^*_{ij}, and send it 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.*

Notice that Modifications 5.2.1 and 5.2.2 occur in the input subphase of $t[s]$, while Modification 5.2.3 occurs in the output subphase.

The following lemma extends Lemma 4.6 to include crossbar queues. We similarly use $\gamma_{ij}(k,t)$ (resp. $\gamma^*_{ij}(k,t)$) to denote the packet at position k in C_{ij} (resp. C^*_{ij}) at time t.

Lemma 5.7. *For any $i, j \in \{1, \ldots, N\}$ and any time t, the following inequalities hold:*

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

I2. $v(\gamma^*_{ij}(k,t)) \leq \beta\, v(\gamma_{ij}(k,t))$*, for any position* $k = 1, \ldots, |C^*_{ij}(t)|$

I3. $v(\delta^*_j(k,t)) \leq \alpha\beta\, v(\delta_j(k,t))$*, for any position* $k = 1, \ldots, |Q^*_j(t)|$

Proof. Inequalities $I1$ - $I3$ 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 all inequalities 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 $I2$ and $I3$. The argument for $I1$ is the same as in the proof of Lemma 4.6.

Before we start with $I2$, we say that a packet $p \in C^*_{ij}(t)$ is in a legal alignment if p is aligned in time t to a packet $q \in C_{ij}(t)$ with $v(p) \leq \beta v(q)$. Clearly, it suffices to show that any packet $p \in C^*_{ij}(t)$ is in a legal alignment. We distinguish between two cases:

Case I2.1 $p \in C^*_{ij}(t-1)$. Thus, by induction, p is aligned in $t-1$ to a packet $q \in C_{ij}(t-1)$ with $v(p) \leq \beta v(q)$. We need to show in this case that p either remains in the same alignment in t or it changes to another legal alignment.

The only critical case is when p moves one step back in t (other cases are the same as in the proof of Lemma 4.6). In this case, a packet p' must be inserted in front of p, implying that $v(p) \leq v(p')$. Here, we distinguish between two cases: (i) CPG transfers a packet r to C_{ij} as well, (ii) CPG does not transfer any packet to C_{ij}. Let k denote the position of the alignment (p,q) in time $t-1$. In case (i), due to Inequality $I1$ of this lemma, it must hold that $v(p') \leq v(r)$. Now, notice that either (1) r is inserted in a position $k' \leq k$, 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 $v(r) \leq v(q')$. Since $v(p) \leq v(p') \leq v(r)$, $v(p) \leq v(q')$. Hence, p is in a legal alignment in either case.

In case (ii), C_{ij} must be full in t and $v(p') \leq \beta v(lc_{ij}(t))$; otherwise, due to Modification 5.2.2, OPT would not insert any packet in C^*_{ij}. Thus, p must be aligned with some packet q' in t. Clearly, $v(lc_{ij}(t)) \leq v(q')$. Thus, $v(p) \leq v(p') \leq \beta v(q')$. Hence, p makes a legal alignment with q'.

Case I2.2 $p \notin C^*_{ij}(t-1)$. Thus, p is a new packet that is inserted in C^*_{ij} in time t. Again, we distinguish between two cases: (i) CPG transfers a packet r to C_{ij}, or (ii) CPG does not transfer any packet to C_{ij}. In case (ii), C_{ij} must be full in t and $v(p) \leq \beta v(lc_{ij}(t))$; otherwise, due to Modification 5.2.2, OPT would not insert any packet into C^*_{ij}. Thus, p must be aligned with a packet q in t. Clearly, $v(lc_{ij}(t)) \leq v(q)$. Thus, $v(p) \leq \beta v(q)$. Hence, p makes a legal alignment with q.

Now, consider case (i). Due to Inequality $I1$ of this lemma, it must hold that $v(p) \leq v(r)$. Let k denote the position at which p is inserted. If $k = 1$, p is aligned with the most valuable packet in C_{ij} in t. Since r is in C_{ij} in time t, p must be aligned with a packet of value at least $v(r) \geq v(p)$. Now suppose $k > 1$. Let p' be the packet that is directly in front of p in t. Clearly, $p' \in C^*_{ij}(t-1)$ and $v(p) \leq v(p')$. Furthermore, let q' be the packet aligned with p' in time $t-1$. Thus, $v(p) \leq v(p') \leq \beta v(q')$. additionally, let q be the packet at position k in C_{ij} in time $t-1$ (assume $q = \emptyset$ if this is an empty position in C_{ij}).

Notice 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 $q \neq \emptyset$ and that $v(q) \geq v(r) \geq v(p)$. Therefore, we have $v(p) \leq v(r)$ in the first case, $v(p) \leq \beta v(q')$ in the second, and $v(p) \leq v(q)$ in the third. Hence, p is in a legal alignment in any case.

Before we continue with $I3$, 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 $v(p) \leq \alpha\beta v(q)$. Clearly, it suffices to show that any packet $p \in Q^*_j(t)$ is in a legal alignment. We distinguish between two cases:

Case I3.1 $p \in Q^*_j(t-1)$. Thus, by induction, p is aligned in $t-1$ to a packet $q \in Q_j(t-1)$ with $v(p) \leq \alpha\beta v(q)$. We need to show in this case that p either remains in the same alignment in t or it changes to another legal alignment.

The only critical case is when p moves one step back in t (other cases are the same as in the proof of Lemma 4.6). In this case, a packet p' must be inserted in front of p, implying that $v(p) \leq v(p')$. Here, we distinguish between two cases: (i) CPG transfers a packet r to Q_j as well, or (ii) CPG does not transfer any packet to Q_j. Let k denote the position of the alignment (p, q) in time $t-1$. In case (i), due to Inequality $I2$ (of this lemma), it must hold that $v(p') \leq \beta v(r)$. Now, notice that either (1) r is inserted in a position $k' \leq k$, 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 $v(r) \leq v(q')$. Since $v(p) \leq v(p') \leq \beta v(r)$, $v(p) \leq \beta v(q')$. Hence, p is in a legal alignment in either case.

In case (ii), recall that p' is transferred by OPT from C^*_{ij}. Thus, due to Inequality $I2$ (of this lemma), C_{ij} is not empty. Therefore, since CPG does not transfer any packet to Q_j in this case, Q_j must be full in t and $v(gc_{ij}(t)) \leq \alpha v(l_j(t))$. Since $v(p') \leq \beta v(gc_{ij}(t))$ (again due to Inequality $I2$), $v(p') \leq \alpha\beta v(l_j(t))$. Now, since Q_j is full in t, p must be aligned with some packet q' in t. Clearly, $v(l_j(t)) \leq v(q')$. Thus, $v(p) \leq v(p') \leq \alpha\beta v(q')$. Thus, p makes a legal alignment with q'.

Case I3.2 $p \notin Q^*_j(t-1)$. Thus, p is a new packet that is inserted in the

queue in time t. Again, we distinguish between two cases: (i) CPG transfers a packet r to Q_j, or (ii) or (ii) CPG does not transfer any packet to Q_j. In case (ii), recall that p is transferred by OPT from C^*_{ij}. Thus, due to Inequality $I2$ of this lemma, C_{ij} is not empty. Therefore, since CPG does not transfer any packet to Q_j in this case, Q_j must be full in t and $v(gc_{ij}(t)) \leq \alpha v(l_j(t))$. Since $v(p) \leq \beta v(gc_{ij}(t))$ (again due to Inequality $I2$), $v(p) \leq \alpha\beta v(l_j(t))$. Now, since Q_j is full in t, p must be aligned with a packet q in t. Clearly, $v(l_j(t)) \leq v(q)$. Thus, $v(p) \leq \alpha\beta v(q)$. Thus, p makes a legal alignment with q.

Now, consider case (i). Due to Inequality $I2$ of this lemma, it must hold that $v(p) \leq \beta v(r)$. 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 $v(r) \geq v(p)$. Now suppose that $k > 1$. Let p' be the packet that is directly in front of p in t. Clearly, $p' \in Q^*_j(t-1)$ and $v(p) \leq v(p')$. Furthermore, let q' be the packet aligned with p' in time $t-1$. Thus, $v(p) \leq v(p') \leq \alpha\beta v(q')$. 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).

Notice 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 $q \neq \emptyset$ and that $v(r) \leq v(q)$, and thus $v(p) \leq \beta v(q)$. Therefore, we have $v(p) \leq \beta v(r)$ in the first case, $v(p) \leq \alpha\beta v(q')$ in the second, and $v(p) \leq \beta v(q)$ in the third. Hence, p is in a legal alignment in any case. □

The following lemma extends the claim of Lemma 4.8 to crossbar queues concerning the feasibility of mapping privileged packets of type 2.

Lemma 5.8. *Assume that* OPT *transfers a packet p from Q^*_{ij} to C^*_{ij} in $t[s]$ and* CPG *does not transfer any packet to C_{ij}. If C_{ij} is not full in $t[s]$ or $v(p) > \beta v(lc_{ij}(t[s]))$, then* CPG *transfers a packet p' from $Q_{ij'}$ in $t[s]$ with $j' \neq j$ and $v(p') \geq v(p)$.*

Proof. Assume that C_{ij} is not full in $t[s]$ or $v(p) > \beta v(lc_{ij}(t[s]))$. By Inequality I1 of Lemma 5.7, since OPT transfers p from Q^*_{ij} in $t[s]$, CPG must have at the head of Q_{ij} a packet r with $v(r) \geq v(p)$. Thus, if $v(p) > \beta v(lc_{ij}(t[s]))$, then it must also hold that $v(r) > \beta v(lc_{ij}(t[s]))$. Hence, C_{ij} is not full in $t[s]$ or $v(r) > \beta v(lc_{ij}(t[s]))$, and therefore r is eligible to be transferred to C_{ij}. Nevertheless, as CPG does not transfer any packet to C_{ij}, another eligible packet p' must be transferred from another input queue $Q_{ij'}$, where $j' \neq j$. Obviously, as CPG preferred p' over r, it must hold that $v(p') \geq v(r)$, and hence $v(p') \geq v(p)$. □

Now, recall Inequality I3 of Lemma 5.7. It implies that if OPT sends a packet of value v from some output queue at some time, CPG must send a

packet of at least $v/(\alpha\beta)$ from the same output queue at the same time. Let S (resp. S^*) denote the set of all packets that CPG (resp. OPT) sends from output queues. Thus,

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

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

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

Thus, we can conclude the competitive ratio of CPG as

$$\text{OPT}(\sigma) \leq \left(\alpha\beta + \frac{2\alpha\beta + \alpha\beta(\beta-1)}{(\alpha-1)(\beta-1)}\right) \text{CPG}(\sigma) \ .$$

It can be verified that this competitive ratio is minimized when $\beta = 2\sqrt{2}-1$ and $\alpha = 2\sqrt{2}$, resulting in a competitive ratio of $12 + 2\sqrt{2} \approx 14.828$.

Lemma 5.9. *The following inequality holds:*

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

Proof. We consider the following mapping scheme:

1. Let p be a privileged packet of Type 1 that is sent from Q^*_{ij} in $t[s]$. By Modification 5.2.1, CPG transfers a packet p' from Q_{ij} in $t[s]$, and by Inequality I1 of Lemma 5.7, $v(p) \leq v(p')$. Map p to p'.

2. Let p be a privileged packet of Type 2 that is sent from Q^*_{ij} in $t[s]$. By Lemma 5.8, CPG 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 from C^*_{ij} in $t[s]$. By Modification 5.2.3, CPG transfers a packet p' from C_{ij} in $t[s]$, and by Inequality I2 of Lemma 5.7, $v(p) \leq \beta v(p')$. Map p to p'.

4. Let q be a packet that is preempted from a crossbar or an output queue 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 $\frac{2\alpha\beta+\alpha\beta(\beta-1)}{(\alpha-1)(\beta-1)}v(p')$.

For any packet $p' \in S$, p' can get mapped in four cases: (1) when it is scheduled in an input subphase, (2) when it preempts a packet from a crossbar queue, (3) when it is scheduled in an output subphase, and (4) when it preempts a packet from an output queue. We first consider cases (1) and (2). Assume that p' is scheduled from $Q_{ij'}$ to $C_{ij'}$ in the input subphase t. Now, assume that OPT transfers a packet from Q^*_{ij} to C^*_{ij} in the same time. Clearly, if $j \neq j'$, a privileged packet, say p_1, of Type 1 can be sent from $Q^*_{ij'}$ in t, and the packet which OPT transfers from Q^*_{ij} can become a privileged packet, say p_2, of Type 2. Hence, at most two privileged packets may be sent in t from each input port i. Since privileged packets of Type 1 and 2 are mapped only to packets that are transferred by CPG from the same input port during the same input subphase, only p_1 and p_2 can be mapped to p' in t. 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 in an input subphase is at most $2\,v(p')$.

Assume now that p' preempts a packet from $C_{ij'}$. Using the same argument of preemption chains in the proof of Lemma 4.9, we can show that the total value of privileged packets that are mapped to p' when it preempts a packet from $C_{ij'}$ is at most $\frac{2\beta}{\beta-1}v(p')$.

Now, we consider cases (3) and (4). In case (3), p' is scheduled in an output subphase t to the output queue Q'_j. Additionally, assume that OPT does not transfer any packet from $C^*_{ij'}$ in t. Clearly, a privileged packet, say p_3, of Type 3 will be sent in this case from $C^*_{ij'}$. Since privileged packets of Type 3 are mapped only to packets that are transferred by CPG from the same crossbar queue in the same output subphase, only p_3 is mapped to p' in t. Furthermore, as shown in the mapping scheme above, the value of p_3 is at most β times the value of p'. Thus, the total value of privileged packets that are mapped to p' when it is scheduled in an output subphase is at most $(\beta + \frac{2\beta}{\beta-1})v(p')$.

Finally, assume that p' preempts a packet from Q'_j. Again, using the same argument of preemption chains in the proof of Lemma 4.9, we can show that the total value of privileged packets that are mapped to p' when it preempts a packet from Q'_j is at most $\frac{2\alpha\beta+\alpha\beta(\beta-1)}{(\alpha-1)(\beta-1)}v(p')$. □

Chapter 6

Scheduling Packets with Deadlines

In this chapter, we first consider the case where packets have arbitrary processing times and agreeable deadlines. We start with a negative result showing that no deterministic algorithm has a bounded competitive ratio in this case. Then, we consider the more restricted case where all processing times are equal and present an optimal online algorithm with a competitive ratio of 4.

Packets with deadlines are stored in a single queue with unlimited capacity. Each packet p needs $\rho(p)$ time steps to be completely sent. For ease of exposition, we call $\rho(p)$ the processing time of p. We also call a packet that is completely transmitted a *completed* packet. Our objective in this model is to maximize the total value of packets that are completed before their deadlines.

Recall that, for a packet p, $r(p)$, $d(p)$, and $v(p)$ denote the arrival time, deadline and value of p, respectively. Moreover, in each time step, only one packet can be processed, and a packet that is being processed can be preempted to process another packet. However, the processing of a preempted packet can be resumed from the last point of preemption. We say that a packet p is pending at time t if it has not been completed yet and can still be completed if it is immediately processed at t without preemption. We assume that the queue keeps only pending packets at any time.

This problem has its roots in the area of job scheduling. It is known that without any restriction on deadlines and processing times, no deterministic online algorithm can achieve a bounded competitive ratio, even if all packets have value of 1 [BHS94]. Therefore, only results for special cases have been pursued.

If packets have equal processing times and tight deadlines, i.e., for any packet p, $d(p) = r(p) + \rho(p)$, Woeginger [Woe94] shows that no deterministic algorithm can be better than 4-competitive. He also provides an algorithm

with a matching competitive ratio. If deadlines are arbitrary, we only know that the competitive ratio is between 4 and 4.24 [Kim11].

6.1 Unbounded Competitive Ratio

In this section, we consider the case of arbitrary processing times and agreeable deadlines. We show that no deterministic algorithm has a bounded competitive ratio in this case.

Theorem 6.1. *In the case of arbitrary processing times and agreeable deadlines, no deterministic algorithm has a bounded competitive ratio.*

Proof. Let K be a sufficiently large integer. We define two other integers $M = 2^{K^2}$ and $m = \log M + 1$. We consider a sequence of packets $p_0, \ldots, p_m$, where all packets have a deadline at time M. Packet p_0 arrives at time 0 and has a processing time of M. Furthermore, p_0 has a value of K and all other packets have a value of 1. For $1 \leq n < m$, packet p_n has a processing time $M/2^n$, and the last packet p_m has a processing time of 1. Moreover, $r(p_1) = 0$ and $r(p_n) = r(p_{n-1}) + \rho(p_{n-1})$, for $1 < n \leq m$. Clearly, the first packet corresponds to the time interval $[0, M]$, and the remaining m packets correspond to successive intervals of lengths $M/2, M/4, \ldots, 1, 1$ whose union is $[0, M]$. Thus, packets $p_1, \ldots, p_m$ collide with the large packet p_0.

Now, fix an online algorithm ALG. We construct an agreeable-deadline instance for ALG in the following adversarial way. Let the packet sequence proceed as long as ALG chooses in each time step to process p_0. If ALG switches to any other packet in a time step t, we cut the sequence immediately, i.e., if $t \in [r(p_n), r(p_{n+1}))$, for some $n > 0$, packets $p_{n+1}, \ldots, p_m$ do not appear in this instance.

Clearly, if ALG completes p_0, it cannot complete any other packet. The adversary on the other hand completes all packets except p_0. Thus, the competitive ratio of ALG in this case is at least $(\log M)/K = K$. In the other case, if p_n is the last packet released, where $n > 0$, ALG can complete only p_n as all previous packets are too large to be completed in the remaining interval $(r(p_n), M]$. In this case, the adversary completes p_0 only. Thus, the competitive ratio of ALG is again K. Clearly, K can be chosen arbitrarily large, and therefore the competitive ratio of ALG cannot be bounded. □

6.2 An Optimal Algorithm SG

In this section, we show that a natural greedy algorithm, which we call Semi-Greedy (SG), has a competitive ratio of 4 in the case of equal processing times and agreeable deadlines. Clearly, Woeginger's model is a special case of this model. Therefore, the lower bound of 4 holds in our model and thus

SG is optimal. For general deadlines, SG achieves a competitive ratio of 5 [CLTW04].

Algorithm SG.

- If SG is idle, i.e., it has just completed a packet or the queue is empty, process the packet with the maximum value among all pending packets.
- If a new packet p arrives while another packet q is being processed, process p if $v(q) < v(p)/2$. We say in this case that p preempts q.

Theorem 6.2. *The competitive ratio of* SG *is at most 4.*

First, we fix a sequence of packets with agreeable deadlines. Without loss of generality, we make the following assumption about OPT. Since the adversary is clairvoyant, it knows the set of "right" packets in advance, i.e., packets that constitute the optimal solution. Let OPT be the algorithm which schedules these packets without preemption, using the Earliest-Deadline-First (EDF) heuristic. Clearly, the schedule computed by OPT in this way is feasible.

We slightly abuse our notation and let OPT and SG denote the set of packets that are completed by OPT and SG, respectively. Furthermore, for a set S of packets, we denote by $v(S)$ the total value of the packets of S.

Next, we show a mapping scheme from OPT to SG. We show afterwards how this scheme is used to show a competitive ratio of 4.

Mapping Scheme. We scan the schedules of both OPT and SG, starting from time 0. In each time step t, we observe two kinds of events, start-events of OPT and preempt-events of SG.

- t **is a start-event of** OPT**.** OPT starts a packet q at t.
  1. SG is idle at t: Map q to itself. We call q a packet of Type 1.
  2. SG is processing a packet q' at t:
     (a) If SG completes q before t, map q to itself. We call q a packet of Type 1.
     (b) Otherwise (i.e., SG completes q after t or it does not complete q at all), map q to q'. We call q a packet of Type 2.
- t **is a preempt-event of** SG**.** A packet p preempts another packet p' at t. If a packet q of Type 2 or 3 is mapped to p', then q is re-mapped to p and its type is changed to 3.

Notice that we do not assume any transition time between packets, i.e., SG may complete (or preempt) a packet at a point of time and start (or resume) another packet at the same point of time. Therefore, in a start-event, if SG completes a packet in t and then becomes idle, t denotes in this case a point of time at which SG is idle. Similarly, if SG completes (or preempts) a packet in t and then starts (or resumes) another packet, then q' in Step 2 of the mapping scheme denotes the packet that SG starts or resumes. Thus, in start-events, t does not coincide with the completion or preemption time of a packet.

Lemma 6.3. *All packets of* OPT *are mapped to packets that are completed by* SG*.*

Proof. Clearly, all packets of OPT are considered in the mapping scheme. In Steps 1 and 2.(a) of a start-event, q is mapped to a completed packet. Notice that in Step 1, SG must have completed q by t; otherwise, it would not be idle at that time. In Step 2.(b) of a start-event and in a preempt-event, q is mapped to a packet that SG may not complete. However, in preempt-events, it is ensured that such a packet is re-mapped whenever a preemption takes place until it is eventually mapped to a packet that is completed by SG. □

The previous lemma shows that the mapping scheme is feasible. Thus, to show a competitive ratio of 4, it remains only to show the following lemma.

Lemma 6.4. *For any packet $p \in$ SG, if $A_p \subseteq$ OPT denotes the set of all packets that are mapped to p, then $v(A_p) \leq 4\, v(p)$.*

Before we show Lemma 6.4, we first show a number of lemmas that will constitute its proof.

Lemma 6.5. *For any packet $p \in$ SG, at most one packet of Type 1 and one packet of Type 2 are mapped to p.*

Proof. Clearly, any packet $q \in$ OPT is mapped to itself at most once in the mapping scheme. This shows the first part of the lemma.

We show now that at most one packet of Type 2 is mapped to p. Assume for contradiction that two packets q and q' of Type 2 are mapped to p, where OPT processes q before q'. Let t and t' be the times at which q and q' are started, respectively. Thus, two start-events occur at t and t' in which q and q' are respectively mapped to p. Clearly, SG processes p at both t and t'. Notice that, by the mapping scheme, t' does not coincide with the completion time of p, and thus p must be completed after t'. Recall that all packets have the same processing time. Thus, p must be preempted between t and t' by another packet p'; otherwise, p would be completed before or at t'. Hence, a preempt-event occurs between t and t' in which q is re-mapped to p'. Consequently, the type of q is changed to 3 and that leads to a contradiction. □

Lemma 6.6. *For any packet $p \in$ SG, if a packet q of Type 2 is mapped to p, then $v(q) \leq 2\,v(p)$. Furthermore, if a packet of Type 1 is additionally mapped to p, then $v(q) \leq v(p)$.*

Proof. We know from the mapping scheme that when OPT starts q, say at time t_q, q is pending in SG and SG is processing p at t_q. Let s_p be the latest time before t_q at which SG starts (or resumes) p. Thus, since SG prefers p over q between s_p and t_q, either $v(q) \leq v(p)$, in case q arrives before or at s_p, or $v(q) \leq 2\,v(p)$, in case q arrives between s_p and t_q.

To show the other part of the claim, we assume that, besides q, p is mapped to itself. Let t_p be the time at which OPT starts p. We distinguish between two cases: (i) OPT completes p first, and (ii) OPT completes q first. In the first case, since SG is still processing p at time t_q, it completes p after t_p. Also, SG must be processing some packet p' at time t_p; otherwise, SG would be idle at t_p and thus it would complete p before t_q. Thus, by Step 2.(b) of the mapping scheme, p must be mapped to p' and hence being of Type 2. Since packets of Type 2 can be changed to Type 3 only, p cannot be of Type 1 and thus Case (i) cannot occur.

We now consider Case (ii). By assumption, OPT schedules its packets in an EDF-order. Thus, since OPT completes q before p, $d(q) \leq d(p)$. Furthermore, since packets have agreeable deadlines, $r(q) \leq r(p)$ and thus q arrives before or at the same time as p. As shown above for the first part of the claim, since SG prefers p over q between s_p and t_q, either $v(q) \leq v(p)$ in case q arrives before or at s_p, or $v(q) \leq 2\,v(p)$ in case q arrives between s_p and t_q. However, q cannot arrive between s_p and t_q in this case. Thus, $v(q) \leq v(p)$. □

Lemma 6.7. *For any packet $p \in$ SG, if $T_p \subseteq$ OPT denotes the set of packets of Type 3 that are mapped to p, then $v(T_p) \leq 2\,v(p)$.*

Proof. Clearly, packets of Type 3 are mapped to a packet p only in preempt-events, i.e., when p preempts another packet p'.

Let $p_0, \ldots, p_m$ be a chain of packets, where packet p_n preempts packet p_{n-1}, for $0 < n \leq m$, and $p_m = p$. Let $x(p_n)$ denote the total value of packets of Type 2 and 3 that are mapped to packet p_n, for $0 \leq n \leq m$. Notice that a packet can preempt only one packet—at the time of its arrival. Hence, each packet p_n preempts packet p_{n-1} only, and thus packets of Type 3 that are mapped to p emerge only from Type-2 and Type-3 packets of p_{m-1}. Hence, the total value of packets of Type 3 that are mapped to p is given by $x(p_{m-1})$. Recursively, for $0 < n \leq m$, packets of Type 3 that are mapped to p_n emerge only from Type-2 and Type-3 packets of p_{n-1}. Notice also that p_0 does not preempt any packet and thus $x(p_0)$ gives only the total value of packets of Type 2 that are mapped to p_0. Furthermore, by Lemma 6.5 and 6.6, at most one packet of Type 2 can be mapped to any packet p_n and the

value of this packet is at most $2\,v(p_n)$. Thus, $x(p_{m-1})$ can be given by the following recursion:

$$\begin{aligned} x(p_0) &\leq 2\,v(p_0) \\ x(p_n) &\leq 2\,v(p_n) + x(p_{n-1}) \ , \text{ for } 0 < n \leq m-1. \end{aligned}$$

Solving this recursion, we obtain that

$$x(p_{m-1}) \leq 2 \sum_{n=0}^{m-1} v(p_n) \ .$$

Recall that, by algorithm, $v(p_{n-1}) < v(p_n)/2$, for $0 < n \leq m$. Hence, we can rewrite $x(p_{m-1})$ as follows:

$$\begin{aligned} x(p_{m-1}) &\leq 2\,v(p_{m-1}) \sum_{n=0}^{m-1} (1/2)^n \\ &\leq 4\,v(p_{m-1}) \\ &\leq 2\,v(p) \ . \end{aligned}$$

□

Proof of Lemma 6.4. The proof follows directly from the last three lemmas. By Lemma 6.5 and 6.6, the total value of packets of Type 1 and Type 2 that are mapped to p is at most $2\,v(p)$. Also, by Lemma 6.7, the total value of packets of Type 3 that are mapped to p is at most $2\,v(p)$. □

Chapter 7

Open Problems

We discuss in this chapter several questions that are still open in the problem of buffer management. In this work, we focus only on deterministic online algorithms, so one major question that is common for all our models is how randomization can be used to improve the upper bounds of the respective problems. In the deterministic setting, improving the current upper and lower bounds is another intriguing question for most of these models.

7.1 The FIFO Model

In the FIFO model, the only randomized algorithm known is due to Andelman [And05]. He presents a comparison-based algorithm which flips a coin between two deterministic algorithms, one of them is GREEDY presented in Chapter 2. Andelman shows that his algorithm is 1.75-competitive. Unfortunately, this is outperformed by the deterministic algorithm PG which is known to be 1.732-competitive [EW09]. Moreover, the best known lower bound on the competitive ratio of randomized algorithms is 1.25. Thus, showing better randomized upper and lower bounds seems to be a challenging task that is still much far from being accomplished.

For general deterministic algorithm in this model, it is still unclear how to reduce the gap between the current upper bound of 1.732 (of PG) and the current lower bound of 1.419 given by [KMvS05]. The competitive ratio of PG itself cannot be significantly improved as Englert and Westermann [EW09] show a lower bound of 1.707 on its competitive ratio. Thus, one needs to think of preemption rules that are cleverer than the preemption rule of PG. One observation that may help in this direction is that PG is a *memoryless* algorithm, i.e., its preemption decisions are based only on the current configuration of the queue. So how would a history of previous send events change the preemption decisions?

With respect to comparison-based algorithm, an interesting question is whether a comparison-based algorithm with a competitive ratio close to

$1+1/\sqrt{2} \approx 1.707$ could exist. If so, this would mean that we do not need to know the actual values of packets in order to compete with PG, the best non-comparison-based algorithm so far. If not, and in particular if 2 is the right lower bound for any comparison-based algorithm, the desired robustness of this kind of algorithms must come at a price, namely, a significantly degraded performance.

7.2 The Model of Priority Queues

Randomization in the model of priority queues is even more vital from a practical point of view. In practice, queuing as it is done in Chapter 3 is rarely used. The problem in this kind of *greedy* queuing is that queues corresponding to small packet values may suffer from starvation. In other words, if queues of high values are populated most of the time, the queues of low values will not be served frequently enough and will thus overflow, which leads to frequent packet loss. That will in turn lead to severely reduce the sender's transmission rate of the the lost packets as imposed by the TCP mechanism. Therefore, a certain level of fairness must be guaranteed in priority queuing policies.

In today's switches, a randomized algorithm called Weighted Fair Queuing (WFQ) is used. Simply stated, in each time step, WFQ sends a packet of value v_i from a non-empty queue with probability $v_i / \sum_{j \in A_t} v_j$, where $A_t \subseteq \{1, \ldots, m\}$ corresponds to the set of non-empty queues at time t. It is thus of great interest to assess the competitiveness of WFQ and other randomized algorithms in terms of lower and upper bounds.

7.3 The CIOQ and Buffered Crossbar Models

Despite the considerable interest that the switching problem in the CIOQ and buffered crossbar models has received, no result is known on any randomized algorithm in these models. Furthermore, we are not aware of any deterministic lower bounds that are constructed especially for these models.

As we showed in Section 1.7, the IQ model is a special case of these two models, and a lower bound of $2-1/N$ is known for that model [AR05]. Thus, this lower bound applies also to the CIOQ and buffered crossbar models. However, the gap between this lower bound and the upper bounds we show in this work is quite considerable, especially in the general-value case where we show an upper bound of 5.828 in the CIOQ model and 14.828 in the buffered crossbar model. Due to the complex interaction between input and output queues in these models, we consider this problem as one of the most challenging open problems in the area of buffer management—it is already intriguing for us whether a lower bound that is only $2+\epsilon$ is attainable, even in the unit-value case of the CIOQ model.

7.4 The Bounded-Delay Model

The main open problem in the model of packets with deadlines that we consider in this work is to reduce the gap between the long-standing lower bound of 4 [Woe94] and the most recent upper bound of 4.24 [Kim11] in the case of general deadlines. Before tackling this general case, one may first settle the problem in further special cases of deadlines, e.g., the s-bounded case, where the deadline of any packet is no more than s time steps after its arrival time, for a fixed s. Clearly, this is a special case of the agreeable-deadline model, so it is settled by our result in the case where packets have equal processing times. What about general processing times? Does it admit a bounded competitive ratio in that case? In fact, even if all packets have value 1, there is still a gap between a lower bound of 3/2 and an upper bound of 2 given by Baruah et al. [BHS94]. Closing this gap has been a challenging open problem.

The randomized version of this model is not settled either. The current upper bound for the case of general deadlines is 3 [FPZ14], which is implied from a 3-competitive algorithm for the so-called *preemptive model with restarts*. That is, when a "job" (in the machine scheduling jargon) is preempted, it cannot be processed again unless it is restarted from scratch. In other words, jobs can only be completed as a whole, without interruption. The current non-trivial lower bound is only known in the preemptive model with restarts for barely randomized algorithms, i.e., algorithms that choose between two deterministic algorithms at random. Fung et al. [FPZ14] show that no barely randomized algorithm can be better than 2-competitive in that model.

With respect to the variant of preemption with restarts, notice that the deterministic lower bound of 4 holds also in this model. Thus, a natural question in this case is whether our algorithm, SG, remains optimal with this different concept of preemption.

Finally, in the related model of packets of size 1 and general deadlines, a gap between the best known lower bound of 1.618 [Haj01, AMZ03, CF03] and the current upper bound of 1.828 [EW07] still persists, which may reflect an intrinsic difficulty in the case of general deadlines.

Bibliography

[AB10] Kamal Al-Bawani. Competitive algorithms for packet buffering in QoS networks. Master's thesis, Freiburg University, 2010.

[ABEW16] Kamal Al-Bawani, Matthias Englert, and Matthias Westermann. Comparison-based FIFO buffer management in QoS switches. In *Proc. of the 12th Latin American Theoretical Informatics Symposium (LATIN)*, pages 27–40, 2016.

[ABEWar] Kamal Al-Bawani, Matthias Englert, and Matthias Westermann. Online packet scheduling for CIOQ and buffered crossbar switches. In *Proc. of the 28th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2016 (to appear).

[ABS13] Kamal Al-Bawani and Alexander Souza. Buffer overflow management with class segregation. *Information Processing Letters*, 113(4):145–150, 2013.

[AC15] Yossi Azar and Ilan Reuven Cohen. Serving in the dark should be done non-uniformly. In *Proc. of the 42nd Int. Colloquium on Automata, Languages and Programming (ICALP)*, pages 91–102, 2015.

[ACG13] Yossi Azar, Ilan Reuven Cohen, and Iftah Gamzu. The loss of serving in the dark. In *Proc. of the 45th ACM Symp. on Theory of Computing (STOC)*, pages 951–960, 2013.

[AL06] Yossi Azar and Arik Litichevskey. Maximizing throughput in multi-queue switches. *Algorithmica*, 45(1):69–90, 2006.

[AM03] Nir Andelman and Yishay Mansour. Competitive management of non-preemptive queues with multiple values. In *Proc. of the 17th Int. Conf. on Distributed Computing (DISC)*, pages 166–180, 2003.

[AMRR00] William Aiello, Yishay Mansour, S. Rajagopolan, and Adi Rosén. Competitive queue policies for differentiated services.

In *Proc. Nineteenth IEEE Conf. on Computer Communications (INFOCOM)*, volume 2, pages 431–440, 2000.

[AMRR05] William Aiello, Yishay Mansour, S. Rajagopolan, and Adi Rosén. Competitive queue policies for differentiated services. *Journal of Algorithms*, 55(2):113–141, 2005.

[AMZ03] Nir Andelman, Yishay Mansour, and An Zhu. Competitive queueing policies for QoS switches. In *Proc. of the 14th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA)*, pages 761–770, 2003.

[And05] Nir Andelman. Randomized queue management for DiffServ. In *Proc. of the 17th ACM Symp. on Parallelism in Algorithms and Architectures (SPAA)*, pages 1–10, 2005.

[AR04] Yossi Azar and Yossi Richter. The zero-one principle for switching networks. In *Proc. of the 36th ACM Symp. on Theory of Computing (STOC)*, pages 64–71, 2004.

[AR05] Yossi Azar and Yossi Richter. Management of multi-queue switches in QoS networks. *Algorithmica*, 43:81–96, 2005.

[AR06] Yossi Azar and Yossi Richter. An improved algorithm for CIOQ switches. *ACM Transactions on Algorithms*, 2(2):282–295, 2006.

[AS06] Susanne Albers and Markus Schmidt. On the performance of greedy algorithms in packet buffering. *SIAM Journal on Computing*, 35(2):278–304, 2006.

[BCJ08] Marcin Bienkowski, Marek Chrobak, and Łukasz Jeż. Randomized algorithms for buffer management with 2-bounded delay. In *Proc. of the 6th Workshop on Approximation and Online Algorithms (WAOA)*, pages 92–104, 2008.

[BEY98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, New York, 1998.

[BFK+04] Nikhil Bansal, Lisa Fleischer, Tracy Kimbrel, Mohammad Mahdian, Baruch Schieber, and Maxim Sviridenko. Further improvements in competitive guarantees for QoS buffering. In *Proc. of the 31st Int. Colloquium on Automata, Languages and Programming (ICALP)*, pages 196–207, 2004.

[BHS94] Sanjoy K. Baruah, Jayant Haritsa, and Nitin Sharma. On-line scheduling to maximize task completions. In *Real-time systems symposium*, pages 228 – 236, 1994.

[BM08] Marcin Bienkowski and Aleksander 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.

[CCF+06] Francis Y. L. Chin, Marek Chrobak, Stanley P. Y. Fung, Wojciech Jawor, Jiří Sgall, and Tomáš Tichý. Online competitive algorithms for maximizing weighted throughput of unit jobs. *Journal of Discrete Algorithms*, 4(2):255–276, 2006.

[CF03] Francis Y. L. Chin and Stanley P. Y. Fung. Online scheduling for partial job values: Does timesharing or randomization help? *Algorithmica*, 37:149–164, 2003.

[cis06] *Internetworking Technology Handbook*, chapter 49. Cisco Systems, Inc., 2006.

[CJST07] Marek Chrobak, Wojciech Jawor, Jiří Sgall, and Tomáš Tichý. Online scheduling of equal-length jobs: Randomization and restarts help. *SIAM Journal on Computing*, 36(6):1709–1728, 2007.

[CLTW04] Wun-Tat Chan, Tak Wah Lam, Hing-Fung Ting, and Prudence W. H. Wong. New results on on-demand broadcasting with deadline via job scheduling with cancellation. In *Proc. of the 10th International on Computing and Combinatorics Conference*, pages 210 – 218, 2004.

[DJT12] Christoph Dürr, Łukasz Jeż, and Nguyen Kim Thang. Online scheduling of bounded length jobs to maximize throughput. *Journal of Scheduling*, 15:653 – 664, 2012.

[EJSvS16] Leah Epstein, Łukasz Jeż, Jiří Sgall, and Rob van Stee. Online scheduling of jobs with fixed start times on related machines. *Algorithmica*, 1(1):156–176, 2016.

[EW07] Matthias Englert and Matthias 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.

[EW09] Matthias Englert and Matthias Westermann. Lower and upper bounds on FIFO buffer management in QoS switches. *Algorithmica*, 53(4):523–548, 2009.

[FPZ14] S.P.Y. Fung, C.K. Poon, and F. Zheng. Improved randomized online scheduling of intervals and jobs. *Theory Comput. Syst*, 55(1):202–228, 2014.

[Gol10] Michael H. Goldwasser. A survey of buffer management policies for packet switches. *SIGACT News*, 41:100–128, 2010.

[Haj01] B. Hajek. On the competitiveness of online scheduling of unit-length packets with hard deadlines in slotted time. In *Proc. 35th Conf. in Information Sciences and Systems (CISS)*, pages 434–438, 2001.

[HK73] John E. Hopcroft and Richard M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2(4):225–231, 1973.

[IT06] Toshiya Itoh and Noriyuki 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.

[IY15] Toshiya Itoh and Seiji Yoshimoto. Buffer management of multi-queue QoS switches with class segregation. *Theoretical Computer Science*, 589:24–33, 2015.

[JLSS12] Łukasz Jeż, Fei Li, Jay Sethuraman, and Clifford Stein. Online scheduling of packets with agreeable deadlines. *ACM Transactions on Algorithms*, 9(1):Article 5, 2012.

[Kim11] Thang Nguyen Kim. Improved online scheduling in maximizing throughput of equal length jobs. In *Proc. of the 6th International Computer Science Symposium in Russia (CSR)*, pages 429 – 442, 2011.

[KKM15] Jun Kawahara, Koji M. Kobayashib, and Tomotaka Maeda. Tight analysis of priority queuing for egress traffic. *Computer Networks*, 91:614–624, 2015.

[KKS10] Alex Kesselman, Kirill Kogan, and Michael Segal. Packet mode and QoS algorithms for buffered crossbar switches with FIFO queuing. *Distributed Computing*, 23(3):163–175, 2010.

[KKS12a] Alex Kesselman, Kirill Kogan, and Michael Segal. Best effort and priority queuing policies for buffered crossbar switches. *Chicago Journal of Theoretical Computer Science*, 2012(5):1–14, 2012.

[KKS12b] Alex Kesselman, Kirill Kogan, and Michael Segal. Improved competitive performance bounds for CIOQ switches. *Algorithmica*, 63(1-2):411–424, 2012.

[KLM+04] Alexander Kesselman, Zvi Lotker, Yishay Mansour, Boal Patt-Shamir, Baruch Schieber, and Maxim Sviridenko. Buffer overflow management in QoS switches. *SIAM Journal on Computing*, 33(3):563–583, 2004.

[KMvS05] Alex Kesselman, Yishay Mansour, and Rob van Stee. Improved competitive guarantees for QoS buffering. *Algorithmica*, 43(1-2):97–111, 2005.

[KR06] Alex Kesselman and Adi Rosén. Scheduling policies for CIOQ switches. *Journal of Algorithms*, 60(1):60–83, 2006.

[KR08] Alex Kesselman and Adi Rosén. Controlling CIOQ switches with priority queuing and in multistage interconnection networks. *Journal of Interconnection Networks*, 9(1-2):53–72, 2008.

[KR12] James F. Kurose and Keith W. Ross. *Computer networking: a top-down approach*. Pearson Education, Inc., 6th edition, 2012.

[LPS03] Zvi Lotker and Boaz Patt-Shamir. Nearly optimal FIFO buffer management for two packet classes. *Computer Networks*, 42(4):481–492, 2003.

[LTWW94] Will E. Leland, Murad S. Taqqu, Walter Willinger, and Daniel V. Wilson. On the self-similar nature of ethernet traffic (extended version). *IEEE/ACM Transactions on Networking*, 2(1):1–15, 1994.

[MPSL00] Yishay Mansour, Boaz Patt-Shamir, and Ofer Lapid. Optimal smoothing schedules for real-time streams. In *Proc. 19th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 21–29, 2000.

[MPSL04] Yishay Mansour, Boaz Patt-Shamir, and Ofer Lapid. Optimal smoothing schedules for real-time streams. *Distributed Computing*, 17(1):77–89, 2004.

[PF95] Vern Paxson and Sally Floyd. Wide-area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[ST85] Daniel Sleator and Robert Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.

[VB00] Andras Veres and Miklós Boda. The chaotic nature of TCP congestion control. In *Proc. of the 19th IEEE Conf. on Computer Communications (INFOCOM)*, pages 1715–1723, 2000.

[Woe94] Gerhard J. Woeginger. On-line scheduling of jobs with fixed start and end times. *Theoretical Computer Science*, 130(1):5 – 16, 1994.

[Zhu04] An Zhu. Analysis of queueing policies in QoS switches. *Journal of Algorithms*, 53(2):137–168, 2004.