Online Scheduling for Buffering Problems

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

Diplom-Informatiker

MATTHIAS ENGLERT

aus Warendorf

Berichter: Dr. Matthias Westermann
Universitätsprofessor Dr. Ir. Joost-Pieter Katoen
Universitätsprofessor Dr. Rolf Niedermeier

Tag der mündlichen Prüfung: 30. Mai 2008

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.
Abstract

In a scheduling problem, tasks have to be assigned to resources in such a way that some specified objective is accomplished. Often times, tasks either can or have to be stored in a buffer before they are assigned to a resource. In these cases, a buffer management strategy has to constantly facilitate decisions as to which tasks to store in the buffer, which tasks to execute, and which tasks to delete from the buffer. If the tasks arrive over time, these decisions have to be made online, that is, without knowledge of the future.

The predominant method to investigate online algorithms is the competitive analysis. An online algorithm is $c$-competitive if, for every input, the solution returned by the algorithm is at most by a factor of $c$ worse than a solution given by an optimal offline algorithm. We study four different online scheduling problems, in which buffers are a crucial component, in a competitive analysis.

First, we introduce and study reordering buffers, which are used to reorder a stream of tasks, requests, or jobs in such a way that they can be served more efficiently. This concept can be applied to various scheduling problems in order to improve performance.

To demonstrate the power of reordering buffers and to show how they can be efficiently used, we apply reordering buffers to two exemplary scheduling problems. In the first problem, the reordering buffer is used to minimize the sum of the distances between consecutive elements in a sequence of points from a metric space. We design the first algorithm achieving a polylogarithmic competitive ratio for general metric spaces. In the second problem, the reordering buffer is used to obtain improved competitive ratios for the well-known online minimum makespan scheduling problem. For the identical machine model, we present matching upper and lower bounds on the competitive ratio which are significantly lower than the bounds for the classic online minimum makespan scheduling problem without reordering buffers. This is somewhat surprising considering that, for more than four machines, no tight bounds are known for the problem without reordering, despite the great effort that was spent on this problem.

Buffers cannot only be an optional tool for improving performance for various scheduling problems, they can also be a problem-specific necessity. We investigate two
different scheduling problems that are motivated by the problem of packet forwarding in network switches that have so-called Quality-of-Service (QoS) capabilities, i.e., switches which are able to treat different kind of packets with different priority. Since a network switch may not be able to instantly forward every arriving data packet, network switches are equipped with buffers to temporarily store not yet forwarded data packets. The different packet priorities in the QoS scenario are abstracted by assigning each packet a certain value which reflects its priority. A scheduling strategy is used to decide which packets from the buffers are to be sent at any given time.

First, we study a scenario in which the buffers in the network switch have limited capacity and packets have to be sent in the order they arrive. Since the capacity of the outgoing link is also limited, buffer overflow events may occur. In case of a buffer overflow, packets have to be dropped and cannot be forwarded anymore. In order to avoid dropping very valuable packets, it can make sense to preemptively drop packets of lower value at a point in time where it would otherwise not be necessary to drop packets at all. The challenge is to design algorithms that drop the right packets at the right time to achieve the best possible performance.

In the second scenario we study, the capacity of buffers is unbounded and packets can be sent in arbitrary order. However, each packet has a deadline by which it has to be either sent or dropped. In this model, there is a trade-off between sending packets which are to expire shortly and sending packets with large values.

We completely solve both problems for the case that only two different packet values appear in the input sequence and improve the previous bounds for the general case. For the first problem variant, we study the so-called preemptive greedy strategy, which is currently the only algorithm achieving a competitive ratio below 2. We analyze this algorithm more carefully and show improved upper and lower bounds on the competitive ratio of preemptive greedy. For the second problem variant, we introduce the novel concept of suppressed packets and demonstrate the potential of this approach by, among other things, presenting an algorithm achieving the currently best known competitive ratio.

For many optimization problems that are interesting in an online setting, computing an optimal solution is intractable under reasonable complexity theoretic assumptions. Thus, even if we had clairvoyant abilities, it might still be impractical to compute an optimal solution. The optimal offline algorithm which the online algorithm is compared to might have an unacceptable worst-case running time. This is the case, for example, for the reordering buffer problems we study. These problems are NP-hard in their respective offline variant. In contrast, all online algorithms investigated in this thesis are relatively simple and can be implemented efficiently. Hence, we do not focus on the running times of the algorithms and instead concentrate on the quality of the solution obtained.

All the bounds we present are the currently best known for the specific problem. Our bounds for the online minimum makespan scheduling problem for identical machines with reordering and the bounds for packet forwarding in switches with two packet values are optimal. For the remaining problems, we improve the known upper bounds considerably.
First of all, I would like to thank my advisor Matthias Westermann with whom I searched for proofs and Indian restaurants, who always supported me, and who openly shared his insights into computer science, economics, politics, and life with me. I also thank Joost-Pieter Katoen and Rolf Niedermeier for agreeing to act as reviewers for this thesis.

I thank Berthold Vöcking for creating a research group with an atmosphere that was not only extremely productive but also fun. In fact, I thank all the people of the Algorithms and Complexity Group in Aachen for that. I especially would like to thank Heiko Röglin for fruitful collaborations, for proofreading this thesis, and for always staying calm even if I disturbed him during his work. I am also indebted to Harald Räcke with whom I spend a very productive and entertaining summer despite the fact that the climate in our office was closer to that of a Finnish sauna than to that of an office.

This work was supported by DFG grant WE 2848/1.
Scheduling problems are among the most extensively studied problems in computer science and emerge naturally in computer systems, economics, and everyday life. In a scheduling problem, tasks have to be assigned to resources in such a way that some specified objective is accomplished. In many situations, tasks arrive over time and scheduling decisions have to be made *online*, that is, without (full) knowledge of the future. However, there are numerous scheduling problems in which tasks can be delayed for a certain amount of time. In these cases they may not have to be processed in the order they emerge. In addition, it is often possible to ignore some tasks entirely which, however, may result in some kind of penalty. These problem characteristics can be utilized to partly counteract the lack of information about the future, e.g., in order to improve performance in disk scheduling, serving requests for data on a web server more efficiently, or improving the transmission of data packets at network nodes.

It is no coincident that buffers play an important role in these applications. On one hand, to delay tasks they have to be temporarily stored inside some kind of buffer. On the other hand, if tasks are stored in a buffer of limited capacity, there is a need for overflow resolution strategies. These usually result in dropping and hence ignoring some tasks. Therefore, an efficient and well performing management of buffers is crucial for these applications.

Depending on the specific problem, a buffer management strategy has to constantly facilitate decisions as to which tasks to store in the buffer, which tasks to execute, and which tasks to delete from the buffer. This thesis deals with four different online scheduling problems in which buffers are a crucial component.

In the first part of the thesis, we introduce and study *reordering buffers* which are used to reorder a stream of tasks, requests, or jobs in such a way that they can be served more efficiently. This concept can be applied to various scheduling problems in order to improve performance. A reordering buffer is a buffer that can store a fixed number of requests. Each arriving request is first stored in the buffer. If the buffer is completely filled with requests and a new request arrives, one request has to be removed from the buffer and served such that the arriving request can take its place.
The first problem we investigate is the reordering buffer problem. In this problem, each request is associated with a point from a metric space. The cost induced by serving a sequence of requests is defined as the sum over the distances of consecutive requests in the sequence, measured in the underlying metric space. A reordering buffer can be used to reorder the incoming stream of requests to reduce the cost for serving the sequence. Since the cost of the solution is completely determined by the order of the requests, the question is not how to serve the requests but in which order to serve them. Therefore, this is probably the most fundamental problem involving a reordering buffer.

Minimum makespan scheduling is a classic and extremely well-studied problem in which jobs with different processing times have to be assigned to a number of parallel machines such that the time until all jobs are completely processed is minimized. In the online variant of the problem, the jobs arrive one by one and have to be assigned without any knowledge of jobs arriving in the future.

We show how a reordering buffer can be used to significantly improve the quality of the solution achievable by an online scheduling algorithm. In this scenario, the cost of the solution is not determined by the order of the requests. Instead, the reordering is only used to facilitate the actual scheduling task. This means that we need one scheduling strategy to manage the reordering buffer and a second one to choose a machine for each of the jobs removed from the buffer.

In the second part of the thesis, we investigate two different scheduling problems that are motivated by the problem of packet forwarding in network switches that have so-called Quality-of-Service (QoS) capabilities, i.e., switches which are able to treat different kinds of packets with different priority. Network switches are equipped with buffers to temporarily store data packets. To abstract the QoS scenario, we assume that each packet has a certain value which reflects its priority. A scheduling strategy is used to decide which packets from the buffers are to be sent at any given time.

First, we study a scenario in which the buffers in the network switch have limited capacity and packets have to be sent in the order they arrive. Since the capacity of the outgoing link is also limited, buffer overflow events may occur. In case of a buffer overflow, packets have to be dropped and cannot be forwarded anymore. In order to avoid dropping very valuable packets, it can make sense to preemptively drop packets of lower value at a point in time where it would otherwise not be necessary to drop packets at all. The challenge is to design algorithms that drop the right packets at the right time to achieve the best possible performance.

In the second scenario we study, the capacity of buffers is unbounded and packets can be sent in arbitrary order. However, each packet has a deadline by which it has to be either sent or dropped. In this model, there is a trade-off between sending packets which are to expire shortly and sending packets with large values.

We study all the aforementioned problems in a competitive analysis, which is the predominant method for investigating online algorithms for optimization problems. In a competitive analysis the cost of the solution given by the online algorithm is compared to the optimal cost achieved by an optimal offline algorithm. Let $C_A(\sigma)$ denote the cost of the solution given by algorithm $A$ on the input sequence $\sigma$ and let $\text{OPT}$ denote an optimal offline algorithm that gets to know the whole input sequence in advance. For minimization problems, an online algorithm $A$ is said to
be $c$-competitive if and only if $C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + \kappa$ for every input sequence $\sigma$, where $\kappa$ is a term that does not depend on $\sigma$. Similarly, an online algorithm $A$ for a maximization problem is $c$-competitive if and only if $c \cdot C_A(\sigma) \geq C_{OPT}(\sigma) - \kappa$ for every input sequence $\sigma$. We call the smallest $c$ such that $A$ is $c$-competitive the competitive ratio of $A$ and we say that $A$ achieves a competitive ratio of $c$ if $A$ is $c$-competitive. If $\kappa$ can be chosen to be 0, $A$ is said to be strictly $c$-competitive. All upper bounds on competitive ratios in this thesis show strict competitiveness. Thus, for simplicity, we do not always explicitly mention this fact.

Graham [Gra66] was probably the first to analyze an algorithm in a competitive analysis, although he did not use any of today’s terminology. Section 2.2 is devoted to the problem Graham studies in his work: the minimum makespan scheduling problem. Competitive analysis was popularized by Sleator and Tarjan [STS85]. Today, it is the most common approach to investigating online algorithms. For an overview of online computation and competitive analysis the reader is referred to the book by Borodin and El-Yaniv [BEY98], the book edited by Fiat and Woeginger [FW98], and the survey article by Albers [Alb03].

Often, competitive analysis is viewed as a game between an online player and an adversary. The online player chooses an online algorithm. Thereafter, the adversary generates an input sequence in order to maximize the competitive ratio of this online algorithm. Studying randomized online algorithms in a competitive analysis is less straightforward. In this case, following Ben-David et al. [BDBK+94], one usually distinguishes between three different adversary models: oblivious, adaptive-online, and adaptive-offline. The oblivious adversary knows the online algorithm the online player has picked but is not aware of the random bits the algorithm uses. The adaptive-online adversary is aware of the online algorithm’s past behavior and may choose the next request based on this information. However, the adversary also has to serve the generated request sequence online, that is, without knowing future random choices of the online algorithm. The adaptive-offline adversary on the other hand can serve the generated request sequence offline. An adaptive-offline adversary is so powerful that randomization yields no improvement over deterministic choices [BDBK+94].

In this thesis, we study randomization in the oblivious adversary model. Following the notations for deterministic online algorithms, a randomized online algorithm $A$ for a minimization problem is said to be strictly $c$-competitive if and only if $E[C_A(\sigma)] \leq c \cdot C_{OPT}(\sigma)$ for every input sequence $\sigma$.

For many optimization problems that are interesting in an online setting, computing an optimal solution is intractable under reasonable complexity theoretic assumptions. Thus, even if we had clairvoyant abilities, it might still be impractical to compute an optimal solution. The optimal offline algorithm which the online algorithm is compared to might have an unacceptable worst-case running time. This is the case, for example, for the reordering buffer problem and the minimum makespan scheduling problem we study. Both problems are NP-hard in their respective offline variant. In contrast, all online algorithms investigated in this thesis are relatively simple and can be implemented efficiently. Hence, we do not focus on the running times of the algorithms and instead concentrate on the quality of the solution obtained.
1.1 Reordering Buffers

The reordering of requests can greatly help to improve solutions for online scheduling problems. Of course, allowing an online algorithm to reorder the input sequence of requests arbitrarily before the actual processing is started somewhat contradicts the notion of an online problem. Therefore, we only allow restricted reorderings that can be performed online.

To realize this online reordering we use the concept of a reordering buffer. A reordering buffer can store a limited number of requests. Thereby their processing is delayed. Whenever the reordering buffer is completely filled with requests, one of them has to be removed from the buffer and processed before the next request arrives.

We believe that this simple and universal framework of a reordering buffer has many potential applications in computer science and economics. In this thesis, we investigate two exemplary scheduling problems involving a reordering buffer. In the first problem, the reordering itself constitutes the scheduling task. In the second problem, the buffer is only used to facility the main task, namely the scheduling of jobs on a set of parallel machines such that the makespan is minimized.

1.1.1 The Reordering Buffer Problem

In many scheduling problems the order in which a set of tasks is performed heavily influences the overall performance. One prominent application scenario exhibiting this behavior is disk scheduling. Given a sequence of read and write requests to locations on a hard disk, the time needed for serving a request is mainly determined by the time it takes to move the head from its current to the new location, hence this time depends on the location of the previous request.

In the reordering buffer problem a reordering buffer is used to reorder the input sequence in a restricted fashion so as to construct an output sequence with lower service cost. This problem is motivated by various problems from practice.

As described above, in hard disks, the latency of an access is mainly induced by the movement of the disk head to the position where the requested data block is stored. The latencies are the dominating factor for the performance of a hard disk. A reordering buffer can be used to rearrange the incoming sequence of accesses in such a way that latencies are reduced. This problem is known as disk scheduling (see, e.g., [TP72]). Although, in practice, disk scheduling is much more complex and exhibits many more aspects than the reordering buffer problem, one can hope that advances in knowledge about the latter can also be beneficial to the former.

Another application for reordering buffers is in the area of rendering. In computer graphics, a rendering system displays a 3D scene which is composed of primitives. A significant factor for the performance of a rendering system are the state changes performed by the graphics hardware. A state change occurs when two consecutively rendered primitives differ in their attribute values, e.g., in their texture or shader program. Krokowski et al. [KRSW04] propose to include a small reordering buffer between application and graphics hardware to rearrange the incoming sequence of primitives in such a way that the cost of the state changes is reduced. In their experimental evaluation this method typically reduces the number of state changes by an order of magnitude and the rendering time by roughly 30% if the buffer can
1.1 Reordering Buffers

hold 30 primitives, the 3D scene is stored in an octree, and the sequence of primitives results from an in-order traversal of that tree. The rendering time for a completely presorted sequence of primitives is often only slightly lower than the rendering time in the reordering buffer setting.

An application scenario beyond the realm of computer science can be found in the automotive industry. In the painting shop of a car manufacturing plant, car bodies traverse the final layer painting where each car body is painted with its own top coat. If two consecutive cars have to be painted in different colors, a color change is required which causes non-negligible set-up and cleaning cost. This cost can be reduced by preceding the final layer painting with a reordering buffer (see, e.g., [GSV04]).

Formal Description of the Model

Our cost function is based on a metric space. A pair \((V, d)\) of a nonempty set of points \(V\) and a distance function \(d : V \times V \rightarrow \mathbb{R}_{\geq 0}\) is called a metric space if for all \(x, y, z \in V\) the following properties are satisfied:

- \(d(x, y) = 0\) if and only if \(x = y\) (reflexivity),
- \(d(x, y) = d(y, x)\) (symmetry),
- \(d(x, z) \leq d(x, y) + d(y, z)\) (triangle inequality).

If \((V, d)\) is a metric space, then \(d\) is called a metric on \(V\).

In the reordering buffer problem, we are given an input sequence of requests for service each of which corresponds to a point in a metric space \((V, d)\), that is, a request \(r\) corresponds to a point \(p(r) \in V\). Serving a request \(r\) following the service to a request \(q\) induces cost \(d(p(r), p(q))\), i.e., the distance between the points corresponding to the two requests.

A buffer is used to rearrange the sequence in order to reduce the total service cost. At each point in time, the reordering buffer contains the first \(k\) requests of the input sequence that have not been processed so far. A scheduling strategy has to decide which request to serve next. Upon its decision, the corresponding request is removed from the buffer and appended to the output sequence, and thereafter the next request in the input sequence takes its place.

An equivalent but slightly more abstract formulation of the problem is the following. An input sequence \(\sigma = \sigma_1 \sigma_2 \cdots \sigma_l\) of requests that correspond to points in a metric space \((V, d)\) is given. This input sequence can be reordered such that an output sequence \(\sigma_\pi = \sigma_{\pi(1)} \sigma_{\pi(2)} \cdots \sigma_{\pi(l)}\) that is a permutation \(\pi\) of the input sequence satisfying \(\pi(i) < i + k\) is produced. The objective is to minimize the cost of the output sequence \(\sum_{i=1}^{l-1} d(p(\sigma_{\pi(i)}), p(\sigma_{\pi(i+1)}))\). At first sight, it may not be obvious that this problem formulation is indeed equivalent but a straightforward proof for this was given in [Eng05].

Metric spaces are closely related to undirected weighted graphs. Every undirected weighted graph \(G = (V, E)\) induces a metric \(d\) on each subset of its set of vertices \(V' \subseteq V\) by defining the distance \(d(x, y)\) between \(x \in V'\) and \(y \in V'\) as the length of the shortest path between \(x\) and \(y\) in the graph \(G\). Conversely, every metric space
(V, d) can be interpreted as a (complete) undirected weighted graph (V, E) where the weight of the edge connecting two vertices x ∈ V and y ∈ V is defined as d(x, y).

This yields the following, somewhat more intuitive formulation of the reordering buffer problem: An undirected weighted graph G = (V, E) and an input sequence of requests which correspond to vertices in G is given. At each point in time the first k unprocessed requests from the input sequence are located at their respective vertices in the graph. The scheduling algorithm controls one server that moves through the graph. The initial position of the server can be chosen arbitrarily from the vertices that contain at least one of the first k requests from the input sequence without cost. Thereafter, to serve a request, the server has to be moved to the vertex containing the request. The cost of this movement is given by the length of the chosen path. If a request is served, it is removed from the graph and the next request from the input sequence is placed at its corresponding vertex. The objective is to process all requests from the input sequence while minimizing the total distance the server moves.

If the buffer size k is equal to the number of requests in the input sequence, we allow arbitrary reorderings. Therefore the problem is NP-hard in general, which can be shown by a straightforward reduction from the NP-hard metric traveling salesperson problem [GJ79]. On the other hand it is known that an optimal solution can be computed via dynamic programming in time O(tk+1) [KP04].

In this thesis we are interested in the online variant of the reordering buffer problem. In this variant the algorithm has to decide which of the requests in the buffer to process next without knowledge about the requests arriving in the future or, in other words, σπ(1) has to be determined based only on σ1σ2 ··· σi+k−1.

1.1.2 Online Minimum Makespan Scheduling with Reordering

In the classic minimum makespan scheduling problem, we are given an input sequence of jobs with processing times. A scheduling algorithm has to assign the jobs to m machines. The objective is to minimize the makespan, which is the time it takes until all jobs are processed. More precisely, we are given a sequence of jobs. Each job Ji is described by a vector (p_{i,1},...,p_{i,m}), where pi,j is the processing time of job Ji on machine Mj. The load L(Mj) of a machine Mj is the sum of the processing times of the jobs assigned to that machine, that is, L(Mj) = \sum_{i:Ji} p_{i,j}. The makespan of a schedule is the maximum max{L(M1),...,L(Mm)} of the loads. The goal is to assign each job to one of the machines such that the makespan is minimized.

Usually, four different machine models are distinguished. For m identical machines, the processing time of a job is the same on every machine, i.e., for each i, p_{i,1} = ··· = p_{i,m}. The uniformly related machines model is an extension of the identical machine case in which the machines have different speeds. There are positive constants α1,...,αm such that, for each i, there is a positive pi such that, for each j, p_{i,j} = pi/αj. The constant αj can be seen as the speed of machine Mj and pi as the size of job Ji. Another extension of the identical machine model is the restricted assignment case. There, machines are essential identical but not every job can be assigned to every machine. In this case, for each i and j, p_{i,j} ∈ {pi,∞}, for some pi (∞ may be replaced by a large number). Finally, the unrelated machine model is the most general one in which the p_{i,j} are arbitrary positive numbers. The minimum makespan scheduling
1.1 Reordering Buffers

problem is NP-hard in the strong sense even in the most special case of \( m \) identical machines [GJ79].

In this thesis, we restrict our attention to identical and uniformly related machines. Therefore, we assume that each job \( J_i \) has a size \( L(J_i) \) and each machine \( M_j \) has a speed \( \alpha_j \). The load \( L(M_j) \) of machine \( M_j \) is the sum of the sizes of the jobs scheduled on \( M_j \) divided by the speed \( \alpha_j \) of \( M_j \), that is, \( L(M_j) = \sum_{i \in J_i \text{ is assigned to } M_j} L(J_i) / \alpha_j \). In the special case of \( m \) identical machines all speeds are equal to 1.

We consider online scheduling algorithms without preemption. As before, an online algorithm does not have knowledge about the input sequence in advance. Instead, it gets to know the input sequence job by job. However, we do not require that arriving jobs are assigned immediately to one of the machines. Instead, a reordering buffer can be used to reorder the input sequence in a restricted fashion. The functionality of the reordering buffer is the same as explained in the previous section. Each arriving job is stored in a buffer in which up to \( k \) jobs can be held. If the buffer contains \( k \) jobs, one job has to be removed and assigned to a machine before the next job arrives.

If we have to remove the jobs from the buffer in the order they arrive, this corresponds to having a lookahead of \( k \) available. A lookahead alone, however, does not suffice to improve the quality of the solution since an adversary can always render the lookahead window useless by flooding it with unimportant, arbitrarily small jobs. Nevertheless, we can significantly benefit from the buffer if we allow jobs to be removed in arbitrary order like we do in our reordering buffer framework.

1.1.3 Our Results

For the reordering buffer problem, we present the first polylogarithmic competitive online algorithm for general metric spaces. Previous work on the reordering buffer problem only considered very restricted metric spaces like line metrics [GS07, KP06a, KP06b] and star metrics [ERW06, EW05, RSW02]. A star is a weighted tree of height one.

We obtain our result by first developing the deterministic algorithm PAY for tree metric spaces. The algorithm is inspired by the MAP strategy for star metrics introduced in [EW05]. However, our algorithm is not a generalization of this strategy as the behavior of both algorithms on a star metric can be different. In fact, analyzing the PAY algorithm for the case of a star metric would lead to a simpler proof of a logarithmic competitive ratio for this special case.

In Section 2.1.3, we analyze PAY for tree metric spaces and show that PAY is \( O(D \log k) \)-competitive for the shortest path metric space induced by a weighted tree whose unweighted diameter is bounded by \( D \). Here \( k \) denotes the size of the reordering buffer and the unweighted diameter of a tree is the maximum number of edges on a simple path connecting two nodes.

In Section 2.1.2, we then show how to improve the analysis for the special case that the underlying metric space is the shortest path metric induced by the leaf nodes of a hierarchically well-separated tree (HST).

**Definition 1.1.** An HST is a weighted rooted tree such that

- all leaf nodes are at the same distance from the root,
An example of a hierarchically well-separated tree.

- all edges on the same level, i.e., distance from the root, have the same length, and
- the length of each edge connecting a level $i$ node to a level $i+1$ node is half the length of an edge connecting a level $i-1$ node to a level $i$ node.

An example of an HST is given in Figure 1.1.

We show that PAY is $O(\log^2 k)$-competitive for metric spaces that can be represented as the shortest path metric induced by the leaf nodes of an HST. We then apply a seminal metric embedding result by Fakcharoenphol, Rao, and Talwar [FRT04] to transform PAY into a randomized algorithm for general metric spaces achieving a competitive ratio of $O(\log n \log^2 k)$ in expectation against an oblivious adversary. Here $n = |V|$ denotes the number of distinct points in the metric space. Note that the length of the input sequence $l$ can be much larger than $n$.

There are various, slightly different definitions of HSTs in the literature. This is, however, not of central importance to our work since the HSTs used in the metric embedding by Fakcharoenphol, Rao, and Talwar have the above structure. In fact, to the best of our knowledge all the different HSTs from the literature can be transformed into an HST with the aforementioned structure without changing the distance between any two points by more than a constant factor. This implies that our analysis can be applied to all these kinds of HSTs. Using this technique we can also remove the restriction that requests can only appear at leaf nodes.

In Section 2.2, we introduce the online minimum makespan scheduling problem with reordering. As main result, we give, for $m$ identical machines, tight and, in comparison to the problem without reordering, much improved bounds on the competitive ratio for minimum makespan scheduling with reordering buffers. Depending on $m$, the achieved competitive ratio lies between $4/3$ and approximately $1.4659$. This optimal ratio is achieved with a buffer of size $\Theta(m)$. A buffer of size $\Omega(m)$ is necessary to achieve this competitive ratio (Deniz Özmen, personal communication, November 2007) and we show that larger buffer sizes do not result in an additional advantage.

More precisely, for identical machines, we present the following results.
We prove a lower bound of $r(m)$ on the competitive ratio of this problem with $m$ identical machines and a reordering buffer whose size does not depend on the input sequence, where $r(m)$ is the smallest positive solution to

$$r(m) - \frac{r(m)}{m} \cdot \left\lfloor \frac{m}{r(m)} \right\rfloor + (r(m) - 1) \sum_{i=m-[m/r(m)]}^{m-1} \frac{1}{i} = 1.$$ 

The values of $r(m)$ for $2 \leq m \leq 30$ are depicted in Figure 1.2. For example, $r(2) = 4/3$, $r(3) = 15/11$, and

$$\lim_{m \to \infty} r(m) = \frac{\text{LambertW}_{-1}(-1/e^2)}{1 + \text{LambertW}_{-1}(-1/e^2)} \approx 1.4659,$$

where LambertW_{-1}(-1/e^2) is the smallest real solution to $x \cdot e^x = -1/e^2$ (for an in depth discussion of the Lambert W function the reader is referred to [CGH+96] and the references therein).

We introduce a fairly simple scheduling algorithm for $m$ identical machines matching this lower bound with a reordering buffer of size $\lceil (1 + 2/r(m)) \cdot m \rceil + 2$.

For $m$ uniformly related machines, i.e., for $m$ machines with different speeds, we give a scheduling algorithm that achieves a competitive ratio of $2 + \varepsilon$, for any constant $\varepsilon > 0$, with a reordering buffer of size $m$. Considering that the best known competitive ratio for uniformly related machines without reordering is 5.828 [BCK00], this result emphasizes the power of online reordering further more.
Extensive work has been done to narrow the gap between upper and lower bounds for online minimum makespan scheduling. Increasingly sophisticated algorithms and involved analyses were developed. For presorted input sequences, the greedy algorithm, which is known as the Longest Processing Time (LPT) algorithm in this context, gives significantly improved bounds on the competitive ratio [Gra69]. Our results show that a completely presorted input sequence is not necessary to achieve much improved bounds. Moreover, our algorithms and proofs for the upper and lower bounds are surprisingly simple and the bounds are tight.

In the following table, we compare, for identical machines, the competitive ratios of our algorithm and the LPT algorithm and the best known lower and upper bounds on the competitive ratios for the case that reordering is not allowed. Here and in the following we give rounded numeric values instead of symbolic terms for easier comparison.

<table>
<thead>
<tr>
<th>$m$</th>
<th>LPT presorted</th>
<th>our results reordering buffer</th>
<th>lower bounds no reordering</th>
<th>upper bounds no reordering</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1.1667</td>
<td>1.3333</td>
<td>1.5</td>
<td>1.5</td>
</tr>
<tr>
<td>3</td>
<td>1.2222</td>
<td>1.3636</td>
<td>1.6667</td>
<td>1.6667</td>
</tr>
<tr>
<td>4</td>
<td>1.25</td>
<td>1.375</td>
<td>1.7321</td>
<td>1.7333</td>
</tr>
<tr>
<td>$\rightarrow \infty$</td>
<td>1.3333</td>
<td>1.4659</td>
<td>1.8800</td>
<td>1.9201</td>
</tr>
</tbody>
</table>

Note that our upper bounds are optimal, i.e., we show matching lower bounds, whereas, there are still gaps between the upper and lower bounds for the problem without reordering buffers.

For uniformly related machines, the LPT algorithm achieves a competitive ratio of 1.66 and a lower bound of 1.52 on its competitive ratio is known [Fri87]. Without presorted input sequences, the best known upper and lower bounds on the competitive ratio are 5.828 and 2.438 [BCK00], respectively. For $m$ uniformly related machines, our online algorithm achieves a competitive ratio of $2 + \varepsilon$ with a reordering buffer of size $m$. Our algorithm and analysis is extremely simple. It has to be noted though that we make use of the polynomial time approximation scheme by Hochbaum and Shmoys [HS88] to schedule the last $m - 1$ jobs remaining in the buffer.

### 1.1.4 Related Work

The paradigm of online reordering has been used in various works. The reordering buffer problem with uniform metric spaces in which two points are either at distance 0 or at distance 1, was introduced by Räcke, Sohler, and Westermann [RSW02]. This setting intends to model, e.g., the paint shop and the rendering scenario. Two requests are at distance 1, if the corresponding cars are to be painted in different colors or the corresponding primitives have different attribute values, and at distance 0, otherwise. With this definition the total distance traveled by the server is equal to the total number of color changes.

The authors show that several standard strategies are unsuitable for a reordering buffer. Further, they present the deterministic Bounded Waste algorithm (BW) and prove that BW achieves a competitive ratio of $O(\log^2 k)$. 
1.1 Reordering Buffers

This has subsequently been improved by Englert and Westermann [EW05] to a competitive ratio of $O(\log k)$, which also holds for a slightly more general class of metric spaces, the class of so-called star metrics, which can be represented as the shortest path metric space induced by weighted trees of height one. Note that our PAY strategy is $O(\log k)$-competitive for trees of constant height, that is, it also obtains the best known bound for this special case.

Khandekar and Pandit [KP06b] analyze the reordering buffer problem for $n$ uniformly spaced points on a line with the motivation that this scenario models the disk scheduling problem well: Requests are categorized according to their destination cylinder on the disk, and the costs are defined as the distance between start and destination cylinder. They present a randomized algorithm achieving a competitive ratio of $O(\log^2 n)$ in expectation against an oblivious adversary. Gamzu and Segev [GS07] improve this by presenting a deterministic $O(\log n)$-competitive strategy which can also be used to derive an algorithm for the continuous line. However, the performance then depends polylogarithmically on the length of the input sequence. In addition, they give, for the line metric space, a lower bound of $1 + \frac{2}{\sqrt{3}} \approx 2.1547$ on the competitive ratio of any deterministic algorithm. This is the only non-trivial lower bound known so far.

In terms of approximating the offline scenario most of the work has been done in the maximization version of the problem where the goal is to maximize the total cost-savings that result from reordering the sequence. Note that in terms of an optimal solution the minimization and maximization scenario are identical. However, in terms of approximation they behave quite differently. For uniform metric spaces, Kohrt and Pruhs [KP04] present an approximation algorithm with approximation ratio 20. Bar-Yehuda and Laserson [BYL07] improve on this result to an approximation guarantee of 9.

Khandekar and Pandit [KP06a] investigate the offline version of the minimization problem for line metric spaces. They obtain a constant factor approximation guarantee with an algorithm that runs in quasi-polynomial time. To the best of our knowledge, the best polynomial time approximation algorithms for the minimization problem in the different scenarios discussed above are actually the corresponding online algorithms.

Krokowski et al. [KRSW04] examine the previously mentioned rendering application. They use a small reordering buffer (storing 30 references) to rearrange the incoming sequence of primitives online in such a way that the number of state changes is reduced. Due to its simple structure and its low memory requirements, this method can easily be implemented in software or even hardware. Their work is heavily based on the theoretical studies in [RSW02] and suggests that the concept of a reordering buffer is not merely of theoretical interest but has applications in practice as well.

Alborzi et al. [ATUW01] analyze the $k$-client problem in which we are given $k$ clients, each of which generates an input sequence of requests for service in a metric space. At each point in time a scheduling strategy has to decide which client’s request to serve next. The authors present a deterministic strategy that achieves a competitive ratio of $2k - 1$. Further, they give a lower bound of $\Omega(\log k)$ on the competitive ratio of any deterministic strategy. The $k$-client problem is closely related to our problem, in the sense that in each time step a scheduling strategy has to choose between $k$ requests in a metric space. At least for the online algorithm both problems look more
or less identical as in each time step it chooses a request to be appended to the output sequence and a new request appears. A crucial difference, however, is that in the $k$-client problem an optimal offline algorithm can take into account that processing different requests results in different requests to be released next. The adversary can leverage this to its advantage, and therefore the known bounds on the competitive ratio for the $k$-client problem are much larger.

Web caching with request reordering extends the classic paging model by allowing reordering of requests under the constraint that a request is delayed by no longer than a predetermined number of time steps. Albers [Alb04] presents a deterministic strategy that achieves an optimal competitive ratio of $k + 1$, where $k$ denotes the size of the cache. Feder et al. [FMP+04] introduce a randomized strategy that achieves an asymptotically optimal competitive ratio of $\Theta(\log k)$.

Divakaran and Saks [DS00] consider an online scheduling problem with job set-ups. Each job has a release time, a processing time, and a type. Processing a job takes its processing time and in addition a job-type specific set-up time. However, this set-up time is not needed if the previously processed job was of the same type. Their objective is to minimize the maximum flow time. They present an $O(1)$-competitive online algorithm for this problem.

Minimum makespan scheduling has been extensively studied. The reader is referred to the survey by Pruhs, Sgall, and Torng [PST04] for an overview.

For $m$ identical machines, Graham [Gra66] shows that the greedy algorithm, which schedules each arriving job on a machine with minimum load, is $(2 - 1/m)$-competitive. This is optimal for $m \leq 3$ [FKT89]. However, better bounds are known for larger $m$. For $m = 4$, the best known lower and upper bounds on the competitive ratio are $1.7321$ [RC03] and $1.7333$ [CvVW94], respectively. For large $m$, the best known lower bound on the competitive ratio was improved from $1.837$ [BKR94] over $1.852$ [Alb99] to $1.880$ [Rud01]. The first upper bound on the competitive ratio below $2$ was $1.986$ [BFKV95]. This upper bound was improved to $1.945$ [KPT96], then to $1.923$ [Alb99], and finally to $1.9201$ [FW00].

Faigle, Kern, and Turán [FKT89] give lower bounds of $3/2$ and $1 + 1/\sqrt{2}$ for $m = 2$ and $m \geq 4$, respectively. Deniz Özmen recently observed that these lower bounds still hold in our model if the buffer size is at most $\lfloor m/2 \rfloor$ and $\lfloor m/8 \rfloor$, respectively (personal communication, November 2007). To show this, the input sequences used in [FKT89] have to be adapted only slightly.

For uniformly related machines, Aspnes et al. [AAF+97] present the first algorithm achieving a constant competitive ratio. Due to Berman, Charikar and Karpinski [BCK00], the best known lower and upper bounds on the competitive ratio are $2.438$ and $5.828$, respectively.

In the semi-online variant of the problem, the jobs arrive in decreasing order of their processing time. To the best of our knowledge, only the greedy LPT algorithm, which assigns each job to a machine with minimum load, was considered in this setting. Graham [Gra69] shows that the LPT algorithm achieves a competitive ratio of $4/3 - 1/(3m)$ for $m$ identical machines. For uniformly related machines, Friesen shows that the LPT algorithm is $1.66$-competitive. He also shows a lower bound of $1.52$ on the competitive ratio of LPT. A detailed and tight analysis for two uniformly
related machines is given by Mireault, Orlin, and Vohra [MOV97] and Epstein and Favrholdt [EF02].

To the best of our knowledge only Kellerer et al. [KKST97] studied reordering buffers in the context of minimum makespan scheduling previously. They gave, for two identical machines, an algorithm achieving a competitive ratio of $4/3$ with a reordering buffer of size 2, i.e., the smallest buffer size allowing reordering. They also showed that this bound is tight.

1.2 Buffer Management for Switches Supporting Quality of Service

Forwarding data packets is the main task of network routers and switches. Since network traffic may be bursty and the bandwidth of incoming links can be larger than the bandwidth of an outgoing link, it may be infeasible to instantly forward every arriving data packet. Therefore, not yet forwarded packets are usually stored in buffers.

We consider the output-queued switch architecture in which each output port is equipped with its own buffer. Incoming packets are directly routed to their destination output port and stored in the respective ports buffer. A scheduler is needed to manage these egress buffers, e.g., to handle overflow events. The simplest and most popular approach to overflow management is tail dropping: New arriving packets are dropped if there is no room in the buffer. This strategy is well suited if all packets have the same priority.

However, the desire to differentiate between different types of packets becomes more and more common. This, for instance, allows providers to address the requirements of customers by offering different levels of service. One possible approach is to specify a desired per-hop behavior for each data packet. The advantage of this method is that decisions can be made locally at each network node.

The IP-Protocol provides the possibility to define a desired per-hop behavior for every data packet. How to make use of this information in network nodes is an active area of research. A management strategy incessantly selects the packets that are sent and dropped. These decision should, among other things, depend on packet priorities and have to be made without knowing the characteristics of data packets arriving in the future.

We restrict our attention to the scenario where no load balancing is performed, that is, the output port to which a packet is routed does only depend on the source and destination of the packet and is independent of the occupancy of the buffers, the loads on the network links, and other traffic characteristics. In this scenario, we can assume without loss of generality that the switch has only one output port. Any algorithm to manage the buffer of a single output port can be applied in parallel to each output port since there is no interdependency between different output port buffers.

We study two different models for this problem and design and analyze online algorithms for them. In the FIFO model, the buffers have limited capacity and packet reordering is not allowed. In the bounded-delay model, packet reordering is allowed...
and there is no explicit bound on the number of packets that can be stored. Instead, each packet has a limited lifespan.

The differentiated service model is abstracted by attributing each packet $p$ with a value $v(p)$ according to its priority or service level. We discretize time into time steps. At the beginning of each time step, one packet can be sent and afterwards an arbitrary number of new packets arrive. The input can be regarded as a sequence of send and arrival events $\sigma_1\sigma_2\cdots\sigma_l$, where each sending of a packet corresponds to a send event and each arrival of a new packet corresponds to an arrival event. Obviously, the event sequence is partitioned into time steps, where the first time step starts with the first event and a new time step starts right before each send event. Using this form for the input sequence, we do not need to explicitly specify the arrival time of a packet. The time step, in which a packet arrives, is simply determined by the number of send events proceeding the packet in the input sequence.

1.2.1 The FIFO Model

In the FIFO Model, packets can be stored in a First-In-First-Out (FIFO) buffer, which we synonymously refer to as a queue, with storage capacity for $k$ packets. Initially, the buffer is empty. Packets stored in the buffer can be dropped at any time. These packets are removed from the buffer and never sent. Arriving packets can either be dropped or appended to the queue (possibly after dropping another packet to free space). Due to the FIFO property, the sequence of sent packets has to be a subsequence of the arriving packets, that is, if a packet $p$ is sent before a packet $p'$, $p$ has arrived before $p'$.

The goal of the buffer management strategy is to drop and sent packets in such a way that the sum of the values of sent packets is maximized.

1.2.2 The Bounded-Delay Model

In the bounded-delay model, packets can be stored in a buffer. Each packet $p$ has a deadline $d(p)$, which is greater than the time step in which $p$ arrives. If a packet $p$ is still stored in the buffer by the end of time step $d(p)$, $p$ is automatically dropped from the buffer at the end of this time step and never sent. Note that an explicit bound on the size of the buffer does not exist, instead the possible delay of each packet is bounded. Hence, this model is known as the bounded-delay model.

A buffer management strategy determines, in each time step, which packet from the buffer is to be sent. As in the FIFO model, the objective is to maximize the sum of the values of sent packets.

1.2.3 Our Results

We first investigate the FIFO model. In Section 3.1.1 we consider the case where packets can only have two different values, that is, for some fixed $\alpha$, each packet either has value 1 or $\alpha$. We introduce the account strategy (ACC) and prove that this strategy achieves, for each $\alpha$ and buffer size $k$, the optimal competitive ratio $r(k, \alpha)$. The precise definition of $r(k, \alpha)$ is given in Section 3.1.1 The values of $r(k, \alpha)$ for
1.2 Buffer Management for Switches Supporting Quality of Service

Figure 1.3: The values of \( r(k, \alpha) \) for \( 2 \leq k \leq 10 \) and \( 1 \leq \alpha \leq 10 \). Non-integral values of \( k \) are meaningless and only included for presentational purposes.

For example,

\[
r(2, \alpha) = \min\{1 + 1/\alpha, 3\alpha/(1 + 2\alpha)\} \leq (\sqrt{13} - 1)/2 \approx 1.303
\]

and

\[
\lim_{k \to \infty} r(k, \alpha) = \frac{2\alpha^2 + \alpha - 1 + \sqrt{4\alpha^4 + (\alpha - 1)^2}}{2\alpha(\alpha + 1)} \leq \sqrt{2} - (\sqrt{5} + 4\sqrt{2} - 3)/2 \approx 1.282.
\]

This improves upon a previous result by Lotker and Patt-Shamir [LPS03] and is the first non-trivial optimal result for the FIFO model. Moreover, our upper bound is not only optimal for a worst-case choice of \( \alpha \) or \( k \) but for arbitrary value combinations. For example, the achieved competitive ratio approaches 1 as \( \alpha \) tends towards 1 or \( \infty \).

In Section 3.1.2, the FIFO model with general packet values is considered. We study the preemptive greedy strategy (PG) introduced in [KMv05]. This is a simple strategy that can be implemented efficiently. We show that PG achieves a competitive ratio of \( \sqrt{3} \approx 1.732 \) which is the best known upper bound on the competitive ratio of this problem. In addition, we give a lower bound of \( 1 + 1/\sqrt{2} \approx 1.707 \) on the competitive ratio of PG which improves the previously known lower bound of \( (1 + \sqrt{5})/2 \approx 1.618 \). Hence, the gap between upper and lower bound for PG narrows to approximately 1/40. We conjecture that the lower bound is tight. As a consequence, we can conclude that new approaches are needed since the competitive ratio of PG cannot be further improved significantly.

Our algorithm for the bounded-delay model computes, after each time step, an optimal provisional schedule for the set of pending packets stored in the buffer. This schedule is optimal under the assumption that new packets do not arrive in the future. In Section 3.2.1 we briefly discuss some aspects of optimal provisional schedules.
Chapter 1 — Introduction

In Section 3.2.2, we make a first approach to design an online algorithm which is simple and natural. For each send event, define the first-packet as the packet that is earliest in the respective optimal provisional schedule and the max-packet as the packet that has maximum value. We study the natural approach to send either the first-packet or the max-packet, depending on the value of these two packets. This approach is very promising if only two packet values 1 and \( \alpha > 1 \) are possible. For this case, we achieve an optimal competitive ratio of \( \min\{1 + \alpha)/\alpha, 2\alpha/(\alpha + 1)\} \leq \sqrt{2} \). However, we also show that this approach is disappointing for general packet values, that is, we prove that this approach cannot achieve a competitive ratio better than 2. Note that there are two natural greedy strategies: The first greedy strategy tries to follow the optimal provisional schedules, that is, always sends the first-packet. The second greedy strategy maximizes the value in each step, that is, always sends the max-packet. These two greedy strategies already achieve a competitive ratio of 2 \([\text{CYO03, KLM+04}]\).

In Section 3.2.3, we enhance the first approach by introducing the concept of suppressed packets. Consider the optimal provisional schedule \( S \) for a set of pending packets \( P \). Suppose that a packet \( q \in P \) does not appear in \( S \), but it can be added to \( S \) if another packet \( p \in S \) is removed from \( S \). Then, \( q \) is called suppressed by \( p \). Obviously, if \( p \) is sent and \( p \) is not the first-packet, \( q \) can appear in the optimal provisional schedule. Hence, the sending of packets that are not first-packets can lead to the appearance of suppressed packets in the optimal provisional schedule. We present a deterministic strategy which considers suppressed packets in addition to the packets in the optimal provisional schedule and show that this strategy achieves a competitive ratio of \( 2\sqrt{2} - 1 \approx 1.828 \). Note that this the best known competitive ratio in the deterministic case. In addition, we present a memoryless version of this algorithm achieving a competitive ratio of approximately 1.893. Here, memoryless algorithms are strategies which base their decisions only on the weights of pending jobs. This is the first memoryless algorithm that achieves a competitive ratio less than 2 and demonstrates the potential of the concept of suppressed packets.

1.2.4 Related Work

Aiello et al. \([\text{AMRR05}]\) introduce a model of differentiated services for FIFO buffers without preemption. Mansour, Patt-Shamir, and Lapid \([\text{MPSL04}]\) add preemption and general packet values to this model. Kesselman and Mansour \([\text{KM03}]\) study the value of the lost packets instead of the value of the sent packets.

For the FIFO model, the natural greedy strategy never drops packets unless a buffer overflow occurs. In case of a buffer overflow, the \( k \) most valuable packets are kept. Kesselman et al. \([\text{KLM+04}]\) show that the competitive ratio of this greedy strategy is 2. Kesselman, Mansour, and van Stee \([\text{KMv05}]\) introduce the PG algorithm and prove that this strategy achieves a competitive ratio of approximately 1.983. In addition, they give a lower bound of \( (1 + \sqrt{5})/2 \approx 1.618 \) on the competitive ratio of the PG strategy. Bansal et al. \([\text{BFK+04}]\) study a modification of PG and show that this strategy achieves a competitive ratio of \( 7/4 \). Jawor \([\text{Jaw05}]\) notes that their modification does not change the performance of the algorithm. Therefore, an upper or lower bound for the modified version also holds for PG and vice versa. The best known lower bound on the competitive ratio of the problem is approximately 1.419.
For inputs restricted to only two different packet values, Lotker and Patt-Shamir [LPS03] present a strategy which achieves a competitive ratio of approximately 1.30448. Kesselman et al. [KLM+04] show a lower bound of approximately 1.282 on the competitive ratio of any deterministic algorithm for this problem variant. Andelman [And05] presents a 5/4-competitive randomized strategy and gives a lower bound of approximately 1.197 on the competitive ratio of any randomized strategy.

Azar and Richter [AR05] extend the buffer management problem to multi-queues, i.e., several incoming queues have to be served by delivering packets that arrive at these queues through one output port, one packet per time step. They present a generic technique that transforms a strategy for a single queue to a strategy for several queues. They show that the competitive ratio of the constructed strategy is at most twice the competitive ratio of the single queue strategy.

For the bounded-delay model, Kesselman et al. [KLM+04] show that the greedy strategy that always sends the available packet with maximum value achieves a competitive ratio of 2 and, if only the two packet values 1 and $\alpha > 1$ are possible, a better competitive ratio of $1 + 1/\alpha$. Chrobak et al. [CJST07] present a 64/33-competitive strategy. Concurrently and independently of our work, Li, Sethuraman, and Stein [LSS07] developed the DP (for dummy packets) algorithm which achieves a competitive ratio of $6/(\sqrt{5} + 1) \approx 1.854$. Similar to our approach they use an optimal provisional schedule and identify two packets similar to our first- and max-packet. However, instead of considering suppressed packets they manipulate the buffer contents to store information about the past. In some situations, the value of a packet is artificially reduced by a certain factor, and a dummy packet is added to the buffer and linked to a real packet stored in the buffer. Dummy packets are not sent but they influence the behavior of the strategy. Their proof is, in contrast to our proof, not explicitly based on a potential function. Instead, the buffer of the optimal offline strategy is modified after each step.

Andelman, Mansour, and Zhu [AMZ03], Chin and Fung [CF03], and Hajek [Haj01] show a lower bound of $(\sqrt{5}+1)/2 \approx 1.618$ on the competitive ratio of any deterministic algorithm for the bounded-delay model. Chin et al. [CCF+06] present a randomized strategy achieving a competitive ratio of $e/(e - 1) \approx 1.582$ and Chin and Fung [CF03] present a lower bound of 5/4 on the competitive ratio of any randomized strategy.

Several restricted variants of the bounded-delay model have been considered. Define the span of a packet to be the difference between its deadline and the time step in which it arrives. An instance is $s$-bounded, if the span of each packet is at most $s$, and an instance is $s$-uniform, if the span of each packet is exactly $s$. Further, an instance has agreeable deadlines, if for each packets $p$ and each packet $p'$ that arrives after $p$, $d(p) \leq d(p')$. Note that $s$-uniform instances are a special case of instances with agreeable deadlines.

The lower bound of $(\sqrt{5}+1)/2$ on the competitive ratio of any deterministic strategy in [AMZ03, CF03] and the lower bound of 5/4 on the competitive ratio of any randomized strategy in [CF03] use only instances that are 2-bounded and therefore also have agreeable deadlines. For $s$-bounded instances, Chin et al. [CCF+06] present a strategy which achieves a competitive ratio of $2 - 2/s + o(1/s)$. This strategy achieves an optimal competitive ratio of $(\sqrt{5}+1)/2$ for $s \in \{2, 3\}$ and is $\sqrt{3}$-competitive for
s = 4. Further, for 2-bounded instances, they give a randomized strategy that achieves an optimal competitive ratio of 5/4. For 2-uniform instances, Chrobak et al. [CJST07] present a strategy which achieves a competitive ratio of approximately 1.377 and a matching lower bound. For instances with agreeable deadlines, Li, Sethuraman, and Stein [LSS05] give an algorithm achieving an optimal competitive ratio of \((\sqrt{5} + 1)/2\).

1.3 Bibliographic Notes

Some of the results in this thesis have been previously presented, in preliminary form, as joint work at different conferences and were published in the respective conference proceedings. Section 2.1 is based on joint work with Harald Räcke and Matthias Westermann [ERW07] and Chapter 3 is based on joint work with Matthias Westermann [EW06, EW07]. The results of Section 2.2 are based on recent work and have not been published previously.
Reordering Buffers

The power of reordering in online scheduling is studied in this chapter. We investigate two exemplary scheduling problems involving a reordering buffer and show how the reordering buffer can be utilized to beneficially reorder input sequences online. First, we present and analyze an online algorithm for the reordering buffer problem for general metric spaces. Then, we study the minimum makespan scheduling problem for identical and uniformly related machines. In this problem, it is not sufficient to only manage the reordering buffer. We also need an algorithm to assign jobs to a machine when they are removed from the buffer.

2.1 The Reordering Buffer Problem

We start by presenting the algorithm PAY for the reordering buffer problem for tree metric spaces. Then it is shown that this algorithm achieves a competitive ratio of $O(D \log k)$ for metric spaces that are induced by weighted trees with unweighted diameter $D$. In Section 2.1.2 an improved analysis of the algorithm for metrics induced by the leaf nodes of a hierarchically well-separated tree is given. In particular, it is shown that PAY achieves a competitive ratio of $O(\log^2 k)$ for these metric spaces. In Section 2.1.3 we then discuss how the metric approximation result by Fakcharoenphol, Rao, and Talwar [FRT04] can be applied to obtain a randomized algorithm for general metric spaces.

2.1.1 Tree Metric Spaces

In the following, we present the PAY algorithm for the reordering buffer problem for tree metric spaces. Initially the first $k$ requests from the input sequence are stored in the reordering buffer. The server is placed at an arbitrary point corresponding to one of the $k$ requests. PAY works in phases where each phase consists of a selection step and a processing step. To simplify the presentation of the algorithm and the analysis, the selection step is described as a continuous process. For this, we describe
the behavior of the algorithm for infinitesimal short time intervals \([t, t + dt]\). Note that our notion of time is only used to describe the selection step and despite the continuous nature of our description the selection step can be easily discretized and implemented efficiently.

The two steps work as follows:

- **Selection Step**

  In this step, PAY selects a set of requests to be removed from its buffer and to be appended to the output sequence. This selection is done as follows. We assign a variable \( \text{pay}(e) \) to each edge \( e \) of the tree, which at any given point in time has a value between 0 and the length \( \ell(e) \) of the edge. We call an edge \( e \) a \textit{paid edge} if \( \text{pay}(e) = \ell(e) \). Otherwise, we call \( e \) an \textit{unpaid edge}.

  During the selection process, the requests currently stored in the buffer are buying edges towards \( v_{\text{pay}} \), where \( v_{\text{pay}} \) denotes the current position of PAY's server in the tree. This is done in the following continuous process: In a time interval \([t, t + dt)\) each request at each node \( u \) increases the payment \( \text{pay}(e) \) by \( dt \), where \( e \) is the first unpaid edge on the path from \( u \) to \( v_{\text{pay}} \). This process continues until there exists a connected component induced by paid edges that contains \( v_{\text{pay}} \).

- **Processing Step**

  In this step, PAY outputs all requests within the connected component. The order in which these requests are visited is not important. The online algorithm only has to ensure that each edge of the component is traversed at most twice and that the final position \( \hat{v}_{\text{pay}} \), i.e., the new position of the PAY server for the next phase, is a node in the component that is farthest away from \( v_{\text{pay}} \). Note that requests appearing during the processing step are ignored and will not be served in this processing step.

  After serving the requests the payment counter \( \text{pay}(e) \) on edges of the component is reset to 0. Note however that the payment counter of edges not in the component is not reset and that this payment will influence the selection step in future phases. This ends the phase.

These steps above are repeated as long as there exist at least \( k \) unprocessed requests. If the number of unprocessed requests drops below \( k \), PAY starts a \textit{clean-up phase}, during which it simply processes all remaining requests in an optimal fashion.

At first glance, the requirement that the new position of the server is a node that is farthest away from the previous position may seem like a subtlety, but it will be used in the proof of the following theorem and is, in fact, critical for achieving a sublinear competitive ratio.

**Theorem 2.1.** PAY achieves a competitive ratio of \( O(D \log k) \) for metric spaces that are induced by weighted trees with unweighted diameter \( D \).

**Proof.** Fix an input sequence \( \sigma \). For the analysis of the algorithm, we fix an optimal offline algorithm OPT, and we compare the performance of OPT to the performance of our algorithm, which is denoted as PAY. We view OPT and PAY as working in a
2.1 The Reordering Buffer Problem

synchronized manner. After a phase of PAY during which \( f \) requests were processed, i.e., appended to the output sequence, we simulate OPT until OPT processed \( f \) requests as well. Then we start the next phase of PAY.

Fix an input sequence and a tree \( T \) which induces the metric space. Throughout the analysis, we use \( v_{\text{opt}} \) to denote the current position of the optimal server in the tree, i.e., the position of the last request that was appended to OPT’s output sequence, and we use \( v_{\text{pay}} \) to denote the current position of PAY’s server.

If the PAY server traverses an edge, it traverses the edge either towards the OPT server or away from it. Due to the following observation it is sufficient to derive a bound for traversals away from OPT.

**Observation 2.2.** Let \( \text{PAY}_{\text{away}}(e) \) and \( \text{PAY}_{\text{towards}}(e) \) denote the cost induced by traversal of edge \( e \) from the PAY server away and towards OPT, respectively. Let \( \text{OPT}(e) \) denote the cost induced by traversals of edge \( e \) from the OPT server. For each edge \( e \),

\[
\text{PAY}_{\text{away}}(e) + 2 \cdot \text{OPT}(e) \geq \text{PAY}_{\text{towards}}(e).
\]

**Proof.** Fix an edge \( e \). First suppose that the PAY and the OPT server both start on the same side of \( e \). In this case,

\[
\text{PAY}_{\text{away}}(e) + \text{OPT}(e) \geq \text{PAY}_{\text{towards}}(e).
\]

We study how the left and the right side of the inequality change over time. In the beginning, \( \text{PAY}_{\text{away}}(e) = \text{OPT}(e) = \text{PAY}_{\text{towards}}(e) = 0 \). The left side of the inequality is increased first (by \( \ell(e) \)) since the PAY and OPT server both start on the same side of \( e \).

Every but the last increase of the right side of the inequality by \( \ell(e) \) is followed by an increase of the left side of the inequality by \( \ell(e) \). To see this, suppose PAY traverses \( e \) towards the OPT server. After the traversal both servers are on the same side of \( e \). Thus, the next traversal of \( e \) is either a traversal from the OPT server or a traversal from the PAY server away from the OPT server.

Now assume that the PAY and the OPT server start on different sides of \( e \). Observe that the OPT server has to traverse \( e \) at least once since there is at least one request on the side of \( e \) on which the PAY server starts. Thus, \( \text{OPT}(e) \geq \ell(e) \).

After \( e \) is traversed by the OPT or the PAY server once, both servers reside on the same side of \( e \). Ignoring the cost of the first traversal of \( e \) and using the arguments above, we have \( \text{PAY}_{\text{away}}(e) + \text{OPT}(e) \geq \text{PAY}_{\text{towards}}(e) \). Taking the first traversal into account yields

\[
\text{PAY}_{\text{away}}(e) + 2 \cdot \text{OPT}(e) \geq \text{PAY}_{\text{away}}(e) + \text{OPT}(e) + \ell(e) \geq \text{PAY}_{\text{towards}}(e). \qedhere
\]
We introduce a \texttt{collected}(e) counter, for each edge $e$, to collect the payment on the edges that are traversed away from OPT. Hence, this counter should reflect the cost that was produced by PAY traversals over $e$ away from OPT. Hence, in principal, we need to increase \texttt{collected}(e) by $\ell(e)$ for all edges in the connected component that are not on the $v_{\text{pay}}$–$x$ path.

The final goal of our analysis is to relate the total payment that is collected to the cost of OPT. The idea is to fix an edge $e$ and to analyze the payment that is collected on $e$ between two consecutive traversals of $e$ by OPT. If we can show that this payment is comparable to the length $\ell(e)$ of the edge, we have our desired result since the payment reflects the cost on the edge produced by traversals from the PAY server away from OPT.

However, our analysis cannot handle increases of \texttt{collected}(e) on edges on the $x$–$v_{\text{opt}}$ path. Thus, we do not increase \texttt{collected}(e) for those edges and instead increase the \texttt{collected}(e) counters for edges on the $x$–$\hat{v}_{\text{pay}}$ path by $2\ell(e)$. This is possible, i.e., does not decrease the total sum of collected payment since, due to the fact that $\hat{v}_{\text{pay}}$ is one of the farthest nodes from $v_{\text{pay}}$ in the connected component, the $x$–$\hat{v}_{\text{pay}}$ path is at least as long as the intersection of the $x$–$v_{\text{opt}}$ path with the component.

Unfortunately, this approach still fails as one can easily construct scenarios in which OPT can avoid using some edge for a long time at the cost of using other edges much more frequently. Therefore, we cannot compare the optimal and online cost on an edge-by-edge basis.

In order to account for this, we introduce the notion of \textit{discount}. In the selection step of the algorithm, requests generate payment in a continuous process. Similarly, we now let requests stored in PAY’s or OPT’s buffer generate discount.

Fix a selection step and a request $p$ at position $v_p$. Let $E_p$ denote the set of edges on the path from $v_p$ to $v_{\text{pay}}$ that have been last traversed from the PAY server after $p$ has arrived. In other words, let $e = \{u, v\}$ be an edge on the path between $v_p$ and $v_{\text{pay}}$. Let $T_u$ and $T_v$ denote the trees obtained by deleting $e$ from $T$ and assume that $v_p$ is located in $T_u$ and $v_{\text{pay}}$ is located in $T_v$. The edge $e$ is contained in $E_p$ if $p$ was already in the buffer of PAY or OPT when the PAY server was located in $T_u$ last.

We say that a request $p$ that is in PAY’s or OPT’s buffer at time $t$ generates a \textit{discount} of $dt/(8D)$ during the time interval $[t, t + dt)$ on all edges in $E_p$. Note that this discount generation is only used for the analysis. Hence, we can assume that OPT and especially OPT’s buffer content is known.

The introduction of this discount and the fact that we do not increase \texttt{collected}(e) for edges on the $x$–$v_{\text{opt}}$ path, yield to the following counter changes.

\begin{itemize}
  \item[(i)] For edges on the $v_{\text{pay}}$–$x$ path, we do not change \texttt{collected}(e) and reset \texttt{discount}(e) to 0.
  \item[(ii)] For edges in the intersection of the component with the $x$–$v_{\text{opt}}$ path, we do not change \texttt{collected}(e) and reset \texttt{discount}(e) to 0.
  \item[(iii)] For edges on the $x$–$\hat{v}_{\text{pay}}$ path, we increase \texttt{collected}(e) by $2\ell(e) - \texttt{discount}(e)$ and reset \texttt{discount}(e) to 0.
  \item[(iv)] For the remaining edges in the connected component, we increase \texttt{collected}(e) by $\ell(e) - \texttt{discount}(e)$ and reset \texttt{discount}(e) to 0.
\end{itemize}
2.1 The Reordering Buffer Problem

Figure 2.1 gives an overview over the different types of edges.

With these counter changes the total collected payment after the whole input has been processed is \( \sum_e \text{collected}(e) \geq \sum_e \text{PAY}_{\text{away}}(e) - \text{discount} \), where \( \text{discount} \) denotes the total generated discount. In order to derive a meaningful bound from this, we need an upper bound on \( \text{discount} \).

**Observation 2.3.** The total generated discount \( \text{discount} \) is at most \( \mathcal{C}_{\text{PAY}}(\sigma)/4 \).

**Proof.** The total number of requests that generate discount is \( 2k \). Each of these requests generates discount on at most \( D \) edges. This means that in a time interval of length \( dt \) a total discount of at most \( k \cdot dt/4 \) is generated. On the other hand the \( k \) requests stored in PAY’s buffer generate a payment of \( k \cdot dt \) in each time interval of length \( dt \). Hence, the total generated discount is at most a fourth of the total generated payment.

However, the total generated payment is at most the total cost of PAY since payment is not removed from an edge \( e \) unless PAY moves over \( e \) and, after the whole input has been processed, all \( \text{pay}(e) \) counters are 0 (otherwise there has to be an unprocessed request that is responsible for the remaining payment).

It remains to show that \( O(D \log k) \cdot \sum_e \text{OPT}(e) \geq \sum_e \text{collected}(e) \) since this yields

\[
\begin{align*}
\mathcal{C}_{\text{PAY}}(\sigma) &= 4 \cdot (\mathcal{C}_{\text{PAY}}(\sigma)/2 - \mathcal{C}_{\text{OPT}}(\sigma) - \mathcal{C}_{\text{PAY}}(\sigma)/4) + 4 \cdot \mathcal{C}_{\text{OPT}}(\sigma) \\
&\leq 4 \cdot \left( \sum_e \text{PAY}_{\text{away}}(e) - \mathcal{C}_{\text{PAY}}(\sigma)/4 \right) + 4 \cdot \mathcal{C}_{\text{OPT}}(\sigma) \\
&\leq 4 \cdot \left( \sum_e \text{PAY}_{\text{away}}(e) - \text{discount} \right) + 4 \cdot \mathcal{C}_{\text{OPT}}(\sigma) \\
&\leq 4 \cdot \sum_e \text{collected}(e) + 4 \cdot \mathcal{C}_{\text{OPT}}(\sigma) \\
&\leq O(D \log k) \cdot \sum_e \text{OPT}(e) + 4 \cdot \mathcal{C}_{\text{OPT}}(\sigma) \\
&= O(D \log k) \cdot \mathcal{C}_{\text{OPT}}(\sigma),
\end{align*}
\]

where the second step holds due to Observation 2.2 and the fact that \( \mathcal{C}_{\text{PAY}}(\sigma) = \sum_e (\text{PAY}_{\text{away}}(e) + \text{PAY}_{\text{towards}}(e)) \). The third step holds due to Observation 2.3.
We need to compare the collected payment on an edge \( e = \{u, v\} \) to the cost of OPT on \( e \). Note that whenever \( \text{collected}(e) \) is changed, it increases by at most \( 2\ell(e) \). However, it may also decrease depending on the value of the \( \text{discount}(e) \)-counter.

In order for the inequality \( \text{collected}(e) \leq O(D \log k) \cdot \text{OPT}(e) \) to be violated there need to be long sequences of changes to the counter \( \text{collected}(e) \) — and many of these changes have to increase the counter — without OPT visiting \( e \), as this would increase \( \text{OPT}(e) \) by \( \ell(e) \). The following lemma forms the crucial part of our analysis and shows that this is not possible.

**Lemma 2.4.** Let \([i_{\text{start}}, \ldots, i_{\text{end}}]\) denote a sequence of consecutive phases during which OPT does not traverse edge \( e \). Then the number of phases \( i \in [i_{\text{start}}, \ldots, i_{\text{end}}] \) in which the counter \( \text{collected}(e) \) increases is at most \( O(D \log k) \).

**Proof.** Let \( T_u \) and \( T_v \) denote the trees obtained when deleting \( e \) from \( T \), and assume without loss of generality that at the beginning of the phase \( i_{\text{start}} \) OPT’s server is located in \( T_u \). We call a request \( \text{opt-exclusive} \) (in phase \( i \)) if at the beginning of the phase the request is in OPT’s buffer but not in PAY’s buffer. Similarly, we call a request \( \text{pay-exclusive} \) if it is held by PAY but not by OPT.

Let \( \text{pay-excl}_i(T_u) \) and \( \text{opt-excl}_i(T_u) \) denote the number of pay-exclusive and opt-exclusive requests, respectively, that are in sub-tree \( T_u \) at the beginning of phase \( i \). Note that during phases in \([i_{\text{start}}, \ldots, i_{\text{end}}]\) the number of pay-exclusive requests in \( T_v \) cannot increase and the number of opt-exclusive requests in \( T_u \) cannot decrease, as this would require OPT to visit the sub-tree.

Let \( i_{\text{first}} \geq i_{\text{start}} \) denote the first phase in which the \( \text{collected}(e) \)-counter changes. If such a phase does not exist, then the lemma obviously holds. The following proposition shows that an increase in the counter \( \text{collected}(e) \) occurring after \( i_{\text{first}} \) is always accompanied by either a large decrease in \( \text{pay-excl}_i(T_v) \) or a large increase in \( \text{opt-excl}_i(T_v) \). This allows us to derive a bound on the total number of increases of the \( \text{collected}(e) \)-counter during phases in \([i_{\text{start}}, \ldots, i_{\text{end}}]\).

**Proposition 2.5.** Let \( i \in [i_{\text{first}} + 1, \ldots, i_{\text{end}}] \) denote a phase in which the counter \( \text{collected}(e) \) increases. Then either

\[
\text{opt-excl}_{i+1}(T_u) > \left(1 + \frac{1}{16D}\right) \cdot \text{opt-excl}_i(T_u)
\]

or

\[
\text{pay-excl}_{i+1}(T_v) < \left(1 - \frac{1}{16D}\right) \cdot \text{pay-excl}_i(T_v)
\]

**Proof.** First observe that in the beginning of the phase \( i \) the PAY server is located in \( T_u \), as otherwise \( e \) lies either on the \( v_{\text{pay}}-x \) path or on the \( x-v_{\text{opt}} \) path, and hence \( \text{collected}(e) \) would not be increased.

Let \( n_{\text{rem}} \) denote the number of requests that generate payment on \( e \) in phase \( i \). Note that since PAY’s server is located in \( T_u \) all these payment generating requests are in \( T_v \). Further, observe that all these requests are removed from the online buffer at the end of phase \( i \). Let \( n_{\text{rem}}^{\text{opt}} \leq n_{\text{rem}} \) denote the number of payment generating requests that are held by OPT and by PAY, and let \( n_{\text{rem}}^{\text{pay-excl}} \) denote the number of \( \text{pay-exclusive} \) requests that generate payment on \( e \). Note that

\[
n_{\text{rem}} = n_{\text{rem}}^{\text{opt}} + n_{\text{rem}}^{\text{pay-excl}}.
\]
Observe that all requests contributing to $n_{\text{opt}}^{\text{rem}}$ are held by OPT and are removed from PAY’s buffer at the end of phase $i$. Hence, these requests become opt-exclusive for phase $i + 1$. Similarly, requests contributing to $n_{\text{pay-excl}}^{\text{rem}}$ are removed from PAY’s buffer and decrease pay-excl($T_v$) accordingly. Hence,

\[ \text{opt-excl}_{i+1}(T_v) - \text{opt-excl}_i(T_v) = n_{\text{opt}}^{\text{rem}} \quad \text{and} \quad \text{pay-excl}_i(T_v) - \text{pay-excl}_{i+1}(T_v) = n_{\text{pay-excl}}^{\text{rem}}. \]

Now assume for contradiction that

\[ \frac{\text{opt-excl}_i(T_v)}{16D} + \frac{\text{pay-excl}_i(T_v)}{16D} \geq n_{\text{rem}}^{\text{opt}} + n_{\text{rem}}^{\text{pay-excl}} = n_{\text{rem}}. \]

Let $i_{\text{first}} \leq j < i$ be the most recent phase before phase $i$ during which PAY visited $T_v$. The requests contributing to $n_{\text{rem}}$ are the only requests that generate payment on $e$ during the phases $j + 1, \ldots, i$. All the requests contributing to opt-excl$_i(T_v)$ and pay-excl$_i(T_v)$ generate discount on $e$ during these phases. Note that the number of opt-exclusive and pay-exclusive requests in $T_v$ does not change during the phases $j + 1, \ldots, i - 1$ since PAY’s and OPT’s server are both located in $T_u$ during these phases. Therefore the total discount generated on $e$ during these phases is at least

\[
\text{discount}(e) \geq \frac{\text{pay}(e)}{n_{\text{rem}}} \cdot \frac{\text{opt-excl}_i(T_v) + \text{pay-excl}_i(T_v)}{8D} \\
\geq \frac{\text{pay}(e)}{n_{\text{rem}}} \cdot \frac{16D \cdot n_{\text{rem}}}{8D} \\
= 2 \cdot \text{pay}(e) \\
= 2 \cdot \ell(e),
\]

where the first inequality follows since $\text{pay}(e) = 0$ at the beginning of phase $j + 1$, $\text{pay}(e) = \ell(e)$ right before the processing step of phase $i$, and only $n_{\text{rem}}$ requests generate payment on $e$.

However, the counter collected($e$) increases by at most $2\ell(e) - \text{discount}(e)$, and hence collected($e$) does not increase in phase $i$. This contradiction completes the proof of the proposition.

Now, we can deduce the lemma from the proposition above. Since the number of opt-exclusive and pay-exclusive requests in $T_v$ are both bounded by $k$, the counter collected($e$) can only increase $O(D \log k)$ times.

The lemma directly implies $O(D \log k) \cdot \text{OPT}(e) \geq \text{collected}(e)$. As previously outlined, this shows that PAY achieves a competitive ratio of $O(D \log k)$.
2.1.2 HST Metric Spaces

In this section, we give an improved analysis for the competitive ratio of our online algorithm on metric spaces that can be represented as the shortest path metric induced by the leaf nodes of hierarchically well-separated trees.

**Theorem 2.6.** PAY achieves a competitive ratio of $O(\log^2 k)$ for shortest path metrics induced by the leaf nodes of a hierarchically well-separated tree.

**Proof.** The key idea for improving the analysis of the previous section for the special case of HSTs is to generate and distribute the discount in a more sophisticated manner. The goal is to increase the amount of discount an edge receives in a time interval $[t, t + dt)$ by a single request from $dt/(8D)$ to $dt/\Theta(\log k)$. If we can do this while otherwise maintaining the properties of the discount distribution, Theorem 2.1 will improve to a competitive ratio of $O(\log^2 k)$.

More precisely, we change the discount generation in such a way that Observation 2.3 still holds. Observation 2.2 holds independently of the discount generation process. Then the competitive ratio of the algorithm only hinges upon Lemma 2.4, that is, how often the counter $\text{collected}(e)$ of an edge $e$ can increase without OPT traversing the edge.

Consider a selection step. We change the discount generation by defining that a request does not generate discount on edges that are ancestor edges of $v_{\text{pay}}$ in the tree. This means that a request $p$ at position $v_p$ generates discount on an edge $e$ only if

- $e \in E_p$, that is, $e$ lies on the $v_p$–$v_{\text{pay}}$ path and the request $p$ was already stored in the buffer of OPT or PAY, when the PAY server last traversed $e$, and
- $e$ is an ancestor edge of $v_p$.

Unfortunately, changing the discount generation in this way creates a problem in our analysis when we collect payment on an edge that is an ancestor edge of $v_{\text{pay}}$. Lemma 2.4 may not hold anymore. (Informally stated: If we still collect payment when traversing such edges, $\text{collected}(e)$ may increase very often because there is no discount generated on these edges anymore.) An edge $e$ that at some point is an ancestor edge of $v_{\text{pay}}$, and hence gets a reduced discount, must lie on the $v_{\text{pay}}$–$\hat{v}_{\text{pay}}$ path the next time it is contained in the connected component since $\hat{v}_{\text{pay}}$ is chosen as a farthest node from $v_{\text{pay}}$. In the case that $e$ lies on the $v_{\text{pay}}$–$x$ portion of this path the discount on $e$ is not used and simply set to 0 (Case (i)). This means that the reduced discount that $e$ receives does no harm.

The problematic edges lie on the $x$–$\hat{v}_{\text{pay}}$ path by the next time they are contained in the connected component (Case (iii)). To deal with these edges, we modify our counter changes in the following way. Let $r$ denote the root of the connected component, i.e., the node on the lowest level in the component. We split Case (iii), i.e., edges on the $x$–$\hat{v}_{\text{pay}}$ path, into two sub-cases, namely edges in the intersection of the $v_{\text{pay}}$–$r$ path and the $x$–$\hat{v}_{\text{pay}}$ path (upward edges) and edges in the intersection of the $r$–$\hat{v}_{\text{pay}}$ path and the $x$–$\hat{v}_{\text{pay}}$ path (downward edges) (see Figure 2.2).

(iii.a) If the edge $e$ is an ancestor edge of $v_{\text{pay}}$, i.e., $e$ lies in the intersection of the $v_{\text{pay}}$–$r$ path and the $x$–$\hat{v}_{\text{pay}}$ path, reset the counters $\text{discount}(e)$ to 0 without increasing the counter $\text{collected}(e)$. 

26
2.1 The Reordering Buffer Problem

CASE (i): Edges on the \( v_{\text{pay}} - x \) path.

CASE (ii): Edges on the \( x - v_{\text{opt}} \) path.

CASE (iii.a): Upward edges on \( x - \hat{v}_{\text{pay}} \).

CASE (iii.b): Downward edges on \( x - \hat{v}_{\text{pay}} \).

CASE (iv): Other edges of the component.

Edges not in the component.

Figure 2.2: The different types of edges considered in the HST analysis.

(iii.b) If the edge \( e \) is not an ancestor edge of \( v_{\text{pay}} \), i.e., \( e \) lies in the intersection of the \( r - \hat{v}_{\text{pay}} \) path and the \( x - \hat{v}_{\text{pay}} \) path, increase the counter \( \text{collected}(e) \) by \( 4\ell(e) - \text{discount}(e) \), and then reset the counters \( \text{discount}(e) \) to 0. This means that the increase of the counter \( \text{collected}(e) \) exceeds the previously needed increase of \( 2\ell(e) - \text{discount}(e) \) by \( 2\ell(e) \).

This excess is used to counteract the now omitted increase of \( \text{collected}(e') \) on each edge \( e' \) in the intersection of the \( v_{\text{pay}} - r \) and the \( x - \hat{v}_{\text{pay}} \) path (Case (iii.a)). First observe that \( \text{collected}(e') \) was at most increased by \( 2\ell(e') \) previously, i.e., we only need to show that the total length of edges generating excess (Case (iii.b) edges) is at least as large as the length of edges for which the \( \text{collected}(e') \) counter is not increased anymore (Case (iii.a) edges).

First, assume that the root \( r \) does not lie on the \( x - \hat{v}_{\text{pay}} \) path, i.e., there are no edges corresponding to Case (iii.a). Thus, trivially, the total length of edges corresponding to Case (iii.a) is at most the total length of edges corresponding to Case (iii.b).

Now, assume that the root \( r \) of the component lies on the \( x - \hat{v}_{\text{pay}} \) path. In this case, edges corresponding to Case (iii.a) are the edges on the \( x - r \) path and edges corresponding to Case (iii.b) are edges on the \( r - \hat{v}_{\text{pay}} \) path. The latter is at least as long as the former since the \( x - r \) path is completely contained in the \( v_{\text{pay}} - r \) path and the \( v_{\text{pay}} - r \) path has the same length as the \( r - \hat{v}_{\text{pay}} \) path.

This shows that the new counter changes still fulfill

\[
\sum_e \text{collected}(e) \geq \sum_e \text{PAY}_{\text{away}}(e) - \text{discount}.
\]

These changes lead to the following result.

**Observation 2.7.** If a request \( p \) at position \( v_p \) generates a discount of \( dt/\Theta(\log k) \) on each edge \( e \) for which

- \( e \in E_p \) and
- \( e \) is an ancestor edge of \( v_p \),

then Lemma 2.4 holds with a bound of \( O(\log^2 k) \).
Proof. It can easily be seen that the rate of discount generation directly influences the bound in Lemma 2.4. Since we now need a discount of $4\ell(e)$ instead of $2\ell(e)$ in order to avoid an increase in the collected counter, Proposition 2.5 becomes slightly weaker and can only show an increase (decrease) of $\text{opt-excl}_i(T_v)$ ($\text{pay-excl}_i(T_v)$) by a factor of $1 + 1/(32D)$ ($1 - 1/(32D)$). However, this only worsens Lemma 2.4 by a constant factor.

Of course, if we generate discount at a rate of $dt/\Theta(\log k)$, Observation 2.3 does not hold anymore. To further improve the discount generation process, we observe that a discount of more than $4\ell(e)$ on an edge is useless and will never be used. The reason is that Lemma 2.4 (the central part of our argument) counts how often collected is increased. This is never the case if the discount on an edge exceeds $4\ell(e)$ (see Case (iii.b) and Case (iv)).

The idea now is to change the discount generation in such a way that for some very small edges much less discount is generated but that the generation is still sufficient to guarantee that by the next counter change the accumulated discount on such an edge $e$ will be at least $4\ell(e)$.

Fix a request $p$ at position $v_p$, and let $r$ denote the node with lowest level on the $v_p$–$v_{\text{pay}}$ path. We only generate discount on edges on the path from $r$ to $v_p$, and only on edges in $E_p$. We call the first $\log k + 7$ edges on the $r$–$v_p$ path long edges and the remaining edges on this path short edges. Now we define that in a time interval of length $dt$, the request $p$ generates a discount of $dt/16(\log k + 7)$ on every long edge and a discount of $k \cdot dt \cdot 4\ell(e)/\ell(e_{\text{max}})$ on every short edge $e$, where $e_{\text{max}}$ denotes the longest edge on the $r$–$v_p$ path. Altogether, the total discount generated by this request is at most

$$\sum_{\text{long edge } e} \frac{dt}{16(\log k + 7)} + \sum_{\text{short edge } e} \frac{k \cdot dt \cdot 4\ell(e)}{\ell(e_{\text{max}})} \leq \frac{dt}{16} + \frac{4k \cdot dt}{\ell(e_{\text{max}})} \cdot \sum_{\text{short edge } e} \ell(e) \leq \frac{dt}{8},$$

where the last step follows since in an HST the edge lengths are decreasing by a factor of 2, and hence $\sum_{\text{short edge } e} \ell(e) \leq \ell(e_{\text{max}})/(64k)$. This implies that Observation 2.3 holds, i.e., the total generated discount is at most one fourth of the total cost of PAY.

Further, every long edge receives discount at a rate of $dt/\Theta(\log k)$. The following lemma shows that the reduced discount on short edges does not cause any additional increases to the collected(e)-counters.

**Lemma 2.8.** If an edge $e = \{u, v\}$, where $u$ is the parent of $v$, receives discount as a short edge for some request $p$ in $T_v$, the accumulated discount on $e$ is at least $4\ell(e)$ by the next time $e$ is contained in the connected component.

This means that if an edge receives discount for being short, the next counter change to collected(e) will not be an increase, i.e., it does not hurt at all that short edges receive less discount. Hence, the total number of increases during a sequence of phases in which OPT does not traverse $e$ is $O(\log^2 k)$ due to Observation 2.7. This is
2.1 The Reordering Buffer Problem

because we generate the same number of increases as if each request would generate
discount at rate \( dt/\Theta(\log k) \) on every edge, that is, on long edges as well as on short ones.

**Proof of Lemma 2.8.** Assume that by the time of the next counter change on \( e \) the
smallest contribution rate by request \( p \) to the discount of \( e \) has been
\( k \cdot dt \cdot 4\ell(e)/\ell(e_{\max}) \), where \( e_{\max} = \{u', v'\} \) is some ancestor edge of \( e \) and \( u' \) is the parent node of \( v' \). Now, consider the most recent traversal of \( e_{\max} \) by PAY in direction from \( v' \) to \( u' \).

At this point the request \( p \) already exists in \( T_{v'} \) and generates a discount of
\( k \cdot dt \cdot 4\ell(e)/\ell(e_{\max}) \) on \( e \) in every time interval of length \( dt \). Right after PAY moved
over \( e_{\max} \) there is no payment on this edge, and in order for PAY to return into the
sub-tree \( T_{v'} \) the edge \( e_{\max} \) has to be paid for. However, in a time interval of length \( dt \)
only a total payment of \( k \cdot dt \) is generated by the \( k \) requests stored in PAY’s buffer.
Hence, by the time PAY returns into \( T_{v'} \) a discount of at least \( 4\ell(e) \) has been generated
by the request \( p \) on the edge \( e \).

This concludes the proof of the theorem.

2.1.3 General Metric Spaces

In 1996, Bartal [Bar96] introduced the concept of probabilistic approximations of
metric spaces.

**Definition 2.9.** A set of metric spaces \( S \) over a non-empty set \( V \) of points and a
probability distribution over the metrics in \( S \), \( \alpha \)-probabilistically approximate a metric
space \( M = (V, d_M) \) if

- for every metric \( N = (V, d_N) \in S \) and every \( u, v \in V \), \( d_N(u, v) \geq d_M(u, v) \) and
- for every \( u, v \in V \), \( E[d_N(u, v)] \leq \alpha \cdot d_M(u, v) \), where \( N = (V, d_N) \) is chosen
  from \( S \) according to the specified probability distribution.

Fakcharoenphol, Rao, and Talwar [FRT04] give, for an arbitrarily fixed metric
space \( M = (V, d_M) \), a probability distribution over a set of HSTs which \( O(\log |V|) \)-
probabilistically approximates \( M \). In fact, they present an efficient randomized
algorithm to generate an HST for a given metric space \( M \) such that each leaf node
of the HST corresponds to one element in \( V \), the distance between two leaf nodes
corresponding to \( u, v \in V \) is at least as large as \( d_M(u, v) \), and the expected distance
of the two leaf nodes is bounded by \( O(\log |V|) \cdot d_M(u, v) \).

To obtain an algorithm for general metric spaces, we first run the algorithm
from [FRT04] to obtain an HST and then execute PAY on this tree. This yields the
following result.

**Theorem 2.10.** The randomized strategy described above achieves a competitive ratio
of \( O(\log n \log^2 k) \) in expectation against an oblivious adversary for \( n \)-point metric
spaces.

**Proof.** The proof only reiterates the proof of a fundamental theorem by Bartal [Bar96,
Theorem 4]. Fix an arbitrary metric space \( M = (V, d_M) \) with \( |V| = n \) and an input
sequence \( \sigma \) for the reordering buffer problem. Let \( N \) denote the metric space induced
Chapter 2 — Reordering Buffers

by the HST chosen by our randomized algorithm. Consider an optimal solution for \( \sigma \) (according to the original metric space \( M \)) and denote its cost measured in the metric space \( M \) by \( C^M_{\text{OPT}}(\sigma) \). Denote the cost of this solution measured in \( N \) as \( C^N_{\text{OPT}}(\sigma) \).

Due to the \( O(\log n) \)-probabilistic approximation property,

\[
E[C^N_{\text{OPT}}(\sigma)] \leq O(\log n) \cdot C^M_{\text{OPT}}(\sigma)
\]

Denote by \( C^N_{\text{PAY}}(\sigma) \) and \( C^M_{\text{PAY}}(\sigma) \) the cost of our algorithm for the input sequence \( \sigma \) measured in the metric space \( N \) and \( M \), respectively. Since \( \text{PAY} \) is \( O(\log^2 k) \)-competitive for HSTs, we also have

\[
E[C^N_{\text{PAY}}(\sigma)] \leq E[O(\log^2 k) \cdot C^N_{\text{OPT}}(\sigma)]
\]

This is due to the fact that \( C^M_{\text{OPT}}(\sigma) \) is at least as large as the cost of a solution that is optimal for \( N \). Finally, combining the previous inequalities yields

\[
E[C^N_{\text{PAY}}(\sigma)] \leq E[C^N_{\text{PAY}}(\sigma)] \leq E[O(\log^2 k) \cdot C^M_{\text{OPT}}(\sigma)] \leq O(\log n \log^2 k) \cdot C^M_{\text{OPT}}(\sigma),
\]

where the first inequality holds because distances in \( N \) are at least as large as in \( M \).

2.2 Online Minimum Makespan Scheduling with Reordering

In this section, we give, for every \( m \), tight bounds for the online minimum makespan problem with reordering for \( m \) identical machines. This is somewhat surprising considering that, for \( m \geq 4 \), no matching bounds are known for the problem without reordering, despite the great effort that was spent on this problem.

We also present an algorithm for uniformly related machines. In this case we achieve a competitive ratio of \( 2 + \varepsilon \) in contrast to the best known upper bound of \( 3 + \sqrt{8} \approx 5.828 \) by Berman, Charikar, and Karpinski [BCK00] for the problem without reordering.

2.2.1 Identical Machines

The lower and upper bounds for identical machines are based on certain weights for the machines. The weight \( w_i \) of a machine \( M_i \) is defined as \( w_i := \min\{r(m)/m, (r(m) − 1)/i\} \) or equivalently

\[
w_i := \begin{cases} \frac{r(m)}{m}, & \text{if } 0 \leq i \leq \frac{r(m) - 1}{r(m)} \cdot m \\ \frac{r(m) - 1}{i}, & \text{if } \frac{r(m) - 1}{r(m)} \cdot m < i \leq m - 1 \end{cases}
\]

For the optimal competitive ratio \( r(m) \) for \( m \) identical machines the weights of all machines sum up to 1, that is, \( r(m) \) is the smallest positive solution to \( \sum_{i=0}^{m-1} w_i = 1 \). Our algorithm and our lower bound construction make use of this fact.

We start by showing a lower bound of \( r(m) \) on the competitive ratio of any deterministic strategy.
Theorem 2.11. For \( m \) identical machines, no deterministic online algorithm can achieve a competitive ratio less than \( r(m) \) with a reordering buffer whose size does not depend on the input sequence.

Proof. Assume for contradiction that there exists an online algorithm \( A \) that achieves a competitive ratio \( r' < r(m) \) with a reordering buffer of size \( k \). Consider the following input sequence. At first, \( (1/\varepsilon + k) \) jobs of size \( \varepsilon \) arrive, where \( \varepsilon \) is chosen appropriately small. Since only \( k \) of these jobs can be stored in the reordering buffer, \( 1/\varepsilon \) of them have to be scheduled on machines. Let \( M_0, \ldots, M_{m-1} \) denote the \( m \) identical machines with \( L(M_0) \geq \cdots \geq L(M_{m-1}) \). Then, there exists a machine \( M_j \) with load at least \( w_j \) since otherwise, the total scheduled load would be strictly less than \( \sum_{i=0}^{m-1} w_i = 1 \).

We distinguish two different cases.

- If \( w_j = r(m)/m \), no more jobs arrive. In the optimal schedule, all jobs are evenly distributed between the machines. Hence, the optimal makespan is at most \( (1 + k \cdot \varepsilon)/m + \varepsilon \). As a consequence, the competitive ratio of \( A \) is at least
  \[
  \frac{r(m)/m}{1 + (k + m) \cdot \varepsilon}/m = \frac{r(m)}{1 + (k + m) \cdot \varepsilon},
  \]
  which is strictly larger than \( r' \) if \( \varepsilon \) is chosen appropriately small.

- If \( w_j = (r(m) - 1)/j \), \( (m - j) \) additional jobs of size \( 1/j \) arrive. It is possible, to assign each of the \( (m - j) \) additional jobs to a different machine and to evenly distribute the remaining \( (1/\varepsilon + k) \) jobs between the remaining \( j \) machines. Hence, the optimal makespan is at most \( (1 + k \cdot \varepsilon)/j + \varepsilon \).

  If \( A \) schedules two jobs of size \( 1/j \) on the same machine, the competitive ratio of \( A \) is at least
  \[
  \frac{2/j}{1 + (k + j) \cdot \varepsilon}/j = \frac{2}{1 + (k + j) \cdot \varepsilon},
  \]
  which is strictly larger than \( r \) if \( \varepsilon \) is chosen appropriately small.

  Otherwise, \( A \) schedules at least one of the jobs of size \( 1/j \) on a machine that already has load at least \( (r(m) - 1)/j \). The competitive ratio of \( A \) is at least
  \[
  \frac{r(m)/j}{1 + (k + j) \cdot \varepsilon}/j = \frac{r(m)}{1 + (k + j) \cdot \varepsilon},
  \]
  which is strictly larger than \( r' \) if \( \varepsilon \) is chosen appropriately small.

This concludes the proof of the theorem. \( \square \)

Our algorithm for scheduling a sequence of jobs on \( m \) identical machines uses a reordering buffer of size \( k \geq m \). The algorithm consists of two different phases. Initially, the first \( k - 1 \) jobs are stored in the reordering buffer. Then, the algorithm iterates the iteration phase as long as new jobs arrive.

- **Iteration phase:** When a new job arrives, store this new job in the reordering buffer, and remove a job \( J \) of smallest size from the buffer. Let \( M_i \) be a machine with load at most
  \[
  w_i \cdot (T + m \cdot L(J)) - L(J),
  \]
where \( T = \sum_{j=0}^{m-1} L(M_j) \) denotes the total scheduled load at this point in time. (Due to Observation 2.12, there always exists such a machine.) Then, schedule job \( J \) on machine \( M_i \), that is, \( J \) is assigned to \( M_i \) and the load \( L(M_i) \) on \( M_i \) as well as the total scheduled load \( T \) increase by \( L(J) \).

After all jobs have arrived, the algorithm schedules the remaining jobs in the clean-up phase. In contrast to the PAY algorithm for the reordering buffer problem, this clean-up phase is a major and critical component of the algorithm.

- **Clean-up phase**: This phase consists of two steps.

  In the first step, some of the \( k - 1 \) remaining jobs in the reordering buffer are virtually scheduled on \( m \) empty machines \( M'_0, \ldots, M'_{m-1} \). For this, the jobs are considered in descending order of their size and greedily assigned to a machine with minimum load. After a job is assigned, two checks are performed:

  1. Is there a machine with three jobs?
  2. Is there a machine with load at least three times as large as the size of the smallest assigned job?

  If one of these is true, the process aborts and the smallest job (which was assigned last) is removed from its machine.

  Hence, at most two jobs are assigned to each machine in the virtual schedule, and the makespan of the virtual schedule is larger than the sum of the sizes of the three smallest jobs in the schedule.

  Assume that \( L(M'_0) \leq \cdots \leq L(M'_{m-1}) \). Then, for each \( 0 \leq i \leq m - 1 \), schedule the jobs from \( M'_i \) on the real machine \( M_i \).

  In the second step, schedule the remaining jobs on the machines \( M_0, \ldots, M_{m-1} \) according to the greedy algorithm, which allocates each job on a machine with minimum load.

**Observation 2.12.** There always exists a machine \( M_i \) with load at most \( w_i \cdot (T + m \cdot L(J)) - L(J) \).

*Proof.* Assume for contradiction that, for each \( 0 \leq i \leq m - 1 \), machine \( M_i \) has load strictly greater than \( w_i \cdot (T + m \cdot L(J)) - L(J) \). This yields the following contradiction

\[
T > \sum_{i=0}^{m-1} (w_i \cdot (T + m \cdot L(J)) - L(J)) = (T + m \cdot L(J)) - m \cdot L(J) = T
\]

since by definition \( \sum_{i=0}^{m-1} w_i = 1 \).

We first prove that the above algorithm is \( r(m) \)-competitive if a reordering buffer of size \( k = 3m \) is used. Thereafter, using a more careful analysis, we show that this competitive ratio is already achieved with a buffer of size \( k = \lceil (1 + 2/r(m)) \cdot m \rceil + 2 \).

**Theorem 2.13.** For \( m \) identical machines, our online algorithm achieves the optimal competitive ratio \( r(m) \) with a reordering buffer of size \( k = 3m \).
2.2 Online Minimum Makespan Scheduling with Reordering

Proof. Fix an input sequence of jobs $\sigma$. Suppose that this sequence of jobs is scheduled by our online algorithm on $m$ identical machines with a reordering buffer of size $k = 3m$. Let $C_{\text{OPT}}(\sigma)$ denote the minimum makespan achieved by an optimal offline algorithm. We show that the makespan of our algorithm is at most $r(m) \cdot C_{\text{OPT}}(\sigma)$.

At the end of the iteration phase, for each $0 \leq i \leq m-1$, the load of machine $M_i$ is at most

$$w_i \cdot (T + (m-1) \cdot L(J_i)),$$

where $T$ denotes the total scheduled load at the end of the iteration phase and $J_i$ denotes the last job scheduled on machine $M_i$. Let $L_{\text{min}}$ denote the smallest size of all remaining jobs in the reordering buffer at the end of the iteration phase. Obviously, $L(J_i) \leq L_{\text{min}}$ and hence, for each $0 \leq i \leq m-1$,

$$w_i \cdot (T + (m-1) \cdot L(J_i)) \leq w_i \cdot (T + (m-1) \cdot L_{\text{min}}).$$

In the clean-up phase, the algorithm schedules the remaining $3m-1$ jobs in the reordering buffer. This phase consists of two steps. First, we analyze, for a fixed $0 \leq i \leq m-1$, the load on machine $M_i$ at the end of the first step. In this step, some of the remaining jobs in the buffer are virtually scheduled on $m$ empty machines. Let $M'_0, \ldots, M'_{m-1}$ denote the machines in the final virtual schedule with $L(M'_0) \leq \cdots \leq L(M'_{m-1})$.

The set of jobs appearing in the virtual schedule are scheduled optimally. This is due to the fact that at most two of these jobs are scheduled on the same machine. Scheduling three of the jobs on the same machine cannot improve the makespan. By design of our algorithm, the combined size of the three smallest jobs in the virtual schedule would lead to a larger makespan. It is a well-known fact that the LPT algorithm produces an optimal schedule if, in an optimal schedule, at most two jobs are scheduled per machine. Hence, for each $0 \leq j \leq m-1$, $L(M'_j) \leq C_{\text{OPT}}(\sigma)$.

At the end of the first step, for each $0 \leq j \leq m-1$, the jobs from $M'_j$ are scheduled on the real machine $M_j$. Thus, the load of machine $M_i$ is at most

$$w_i \cdot (T + (m-1) \cdot L_{\text{min}}) + L(M'_i).$$

It remains to show that

$$w_i \cdot (T + (m-1) \cdot L_{\text{min}}) + L(M'_i) \leq r(m) \cdot C_{\text{OPT}}(\sigma).$$

Clearly,

$$T + (m-1) \cdot L_{\text{min}} + \sum_{j=0}^{m-1} L(M'_j) \leq C_{\text{OPT}}(\sigma)$$

since at least $m-1$ jobs remain in the buffer at the end of the first step and the size of each of these jobs is at least $L_{\text{min}}$. Thus, for each $0 \leq \ell \leq m-1$,

$$T + (m-1) \cdot L_{\text{min}} \leq m \cdot C_{\text{OPT}}(\sigma) - \sum_{j=0}^{m-1} L(M'_j) \leq m \cdot C_{\text{OPT}}(\sigma) - (m-\ell) \cdot L(M'_\ell). \quad (2.1)$$

We distinguish two cases.
we argue that Inequality (2.1) indeed holds for these $\ell$

In both cases, the makespan is at most $r$.

Let $k$ be the number of machines.

In the beginning of the first step, there always exists a machine with load at most $m/r$.

To improve upon Theorem 2.13, we observe that the proof even goes through if Inequality (2.1) only holds for $[(r(m) - 1) \cdot m/r(m)] \leq \ell \leq m - 1$. In the following, we argue that Inequality (2.1) indeed holds for these $\ell$ if we only have a reordering buffer of size $k = [(1 + 2/r(m)) \cdot m] + 2$.

In the beginning of the first step, $[(1 + 2/r(m)) \cdot m] + 1$ jobs are stored in the reordering buffer. Let $n'$ denote the number of jobs scheduled in the final virtual
2.2 Online Minimum Makespan Scheduling with Reordering

schedule. The number of jobs that are stored in the reordering buffer and that are not scheduled on the virtual machines \( M'_{[(r(m)-1)\cdot m/r(m)], \ldots, M'_{m-1}} \) is at least

\[
\frac{r(m) + 2}{r(m)} \cdot m + 1 - n' + \max \left\{ 0, n' - 2 \left( \frac{m}{r(m)} + 1 \right) \right\} \geq m - 1
\]

since \( m - [(r(m)-1) \cdot m/r(m)] \leq \ell \leq m - 1 \),

\[
T + (m-1) \cdot L_{\min} \leq m \cdot C_{OPT}(\sigma) - \sum_{j=\lfloor (r(m)-1) \cdot m/r(m) \rfloor}^{m-1} L(M'_j) \leq m \cdot C_{OPT}(\sigma) - (m-\ell) \cdot L(M'_\ell) .
\]

Hence, the proof of Theorem 2.13 goes through if we only have a reordering buffer of size \( k = \lceil (1 + 2/r(m)) \cdot m \rceil + 2 \).

**Theorem 2.14.** For \( m \) identical machines, our online algorithm achieves the optimal competitive ratio \( r(m) \) with a reordering buffer of size \( k = \lfloor (1 + 2/r(m)) \cdot m \rfloor + 2 \).

2.2.2 Uniformly Related Machines

The algorithm for scheduling a sequence of jobs on \( m \) uniformly related machines \( M_0, \ldots, M_{m-1} \) uses a reordering buffer of size \( m \). For each \( 0 \leq i \leq m-1 \), let \( \alpha_i \) denote the speed of machine \( M_i \). The objective is to minimize the makespan, i.e., the maximum load.

The algorithm consists of two different phases. Initially, the first \( m - 1 \) jobs are stored in the reordering buffer. Then, the algorithm iterates the iteration phase as long as new jobs arrive.

- **Iteration phase:** When a new job arrives, store this new job in the reordering buffer, and remove a job \( J \) of smallest size from the buffer. Let \( M_i \) be a machine with load at most

\[
\frac{1}{\sum_{j=0}^{m-1} \alpha_j} \cdot (T + m \cdot L(J)) - L(J) ,
\]

where \( T = \sum_{j=0}^{m-1} L(M_j) \cdot \alpha_j \) denotes the total scheduled load at this point in time. (Due to Observation 2.12, there always exists such a machine.) Then, schedule job \( J \) on machine \( M_i \), that is, \( J \) is assigned to \( M_i \) and the load \( L(M_i) \) on \( M_i \) increases by \( L(J)/\alpha_j \) and the total scheduled load \( T \) increase by \( L(J) \).

After all jobs have arrived, the algorithm schedules the remaining jobs in the clean-up phase.

- **Clean-up phase:** The \( m - 1 \) remaining jobs in the reordering buffer are virtually scheduled with the polynomial time approximation scheme due to Hochbaum and Shmoys [HS88] on \( m \) empty machines \( M'_0, \ldots, M'_{m-1} \), where, for each \( 0 \leq i \leq m-1 \), machine \( M'_i \) has speed \( \alpha_i \). With this scheme an \((1 + \varepsilon)\)-approximation is achieved. Then, for each \( 0 \leq i \leq m-1 \), schedule the jobs from \( M'_i \) on the real machine \( M_i \).
Chapter 2 — Reordering Buffers

Theorem 2.15. For \( m \) uniformly related machines and any constant \( \varepsilon > 0 \), our algorithm achieves the competitive ratio \( 2 + \varepsilon \) with a reordering buffer of size \( m \).

Proof. Fix an input sequence of jobs. Suppose that this sequence of jobs is scheduled by our online algorithm on \( m \) uniformly related machines with a reordering buffer of size \( m \). Let \( C_{OPT}(\sigma) \) denote the minimum makespan achieved by an optimal offline algorithm.

At the end of the iteration phase, for each \( 0 \leq i \leq m - 1 \), the load on machine \( M_i \) is at most

\[
\frac{1}{\sum_{j=0}^{m-1} \alpha_j} \cdot (T + (m - 1) \cdot L(J_i))
\]

where \( T \) denotes the total scheduled load at the end of the iteration phase and \( J_i \) denotes the last job scheduled on machine \( M_i \) in the iteration phase. Obviously, for each \( 0 \leq i \leq m - 1 \),

\[
\frac{1}{\sum_{j=0}^{m-1} \alpha_j} \cdot (T + (m - 1) \cdot L(J_i)) \leq C_{OPT}(\sigma)
\]

since \( m - 1 \) jobs are stored in the reordering buffer at the end of the iteration phase and the size of each of these jobs is at least \( L(J_i) \).

In the clean-up phase, for each \( 0 \leq i \leq m - 1 \), the load of the machine \( M'_i \) in the virtual schedule is at most \( (1 + \varepsilon) \cdot C_{OPT}(\sigma) \), due to the polynomial time approximation scheme. As a consequence, the makespan of our algorithm is at most

\( (2 + \varepsilon) \cdot C_{OPT}(\sigma) \). \qed
CHAPTER 3

Buffer Management for Switches Supporting Quality of Service

In this chapter we develop online algorithms to manage buffers in switches with QoS capabilities. In the model we consider, an arbitrary number of data packets can arrive in each time step but only one data packet can be sent. In Section 3.1 we investigate the FIFO model in which up to $k$ packets can be stored in a buffer but the order of the packets has to be preserved. Section 3.2 deals with the bounded-delay model where packets do not have to be sent in the order they arrive. The buffer in this model has unbounded capacity. However, each packet has a deadline by which it either has to be sent or dropped. In both models each packet has a value which reflects the packet’s priority. The objective is to maximize the sum of the values of the sent packets.

In the following we discuss some properties of schedules for both models. A schedule specifies which packet is sent in which time step. We call a schedule feasible if it satisfies all problem constraints. In the FIFO model this means that no packet is sent before it arrives, packets are sent in the order they arrive, and the buffer never contains more than $k$ packets. In the bounded-delay model it means that no packet is sent before it arrives and no packet is sent after its deadline.

It is easy to determine the set of packets sent in a feasible schedule as a schedule only specifies which packet is to be sent in which time step. It actually suffices to specify this set of sent packets, as one can easily construct a feasible schedule from such a set. Every arriving packet that is not contained in the set is dropped immediately. The remaining packets are stored in the buffer. In the FIFO model, in each time step, the first packet in the buffer, i.e., the packet that is stored the longest in the buffer, is sent. In the bounded-delay model, in each time step, a packet that has the earliest deadline among the packets stored in the buffer is sent. If the buffer contains no packet, no packet is sent.

We call a set of packets feasible if it is possible to send every packet from the set while adhering to the problem constraints. Due to the above observation, we use the terms “set of sent packets” and “schedule” synonymously in the following.

For the bounded-delay model it is well known that the set of feasible schedules is a matroid over the set of packets in the input sequence [CLRS01]. The same was
shown to be the case for the FIFO model by Kesselman et al. [KLM+04].

A collection of subsets $\mathcal{I}$ of a finite set $\mathcal{S}$ is called a matroid over $\mathcal{S}$ if it satisfies the following three properties:

- $\emptyset \in \mathcal{I}$.
- If $J \subset I$ and $I \in \mathcal{I}$, then $J \in \mathcal{I}$.
- If $I, J \in \mathcal{I}$ and $|I| < |J|$, then there exists an $p \in J \setminus I$ such that $I \cup \{p\} \in \mathcal{I}$.

In our scenario the packets, that is, the elements of $\mathcal{S}$, have values. There is a corresponding notion for matroids. A weighted matroid is a matroid with a weight function which assigns a weight (or value) $v(p)$ to each element $p \in \mathcal{S}$. The weight of a subset $A \in \mathcal{S}$ is given by the sum $\sum_{p \in A} v(p)$ of the weights of the elements contained in $A$.

It is well known that a greedy algorithm can be used to compute a maximum weight set $I$ in $\mathcal{I}$ if $\mathcal{I}$ is a matroid over a finite set $\mathcal{S}$. The greedy algorithm considers the elements from $\mathcal{S}$ in descending order of their weight for inclusion into the set $I$. An element $p$ is added to $I$ if $I \cup \{x\} \in \mathcal{I}$. Otherwise, the element is not added to $I$. After all elements in $\mathcal{S}$ have been considered for inclusion, the set $I$ is a maximum weight set in $\mathcal{I}$. As a consequence, optimal solutions for both the FIFO and the bounded-delay model can be computed efficiently offline.

### 3.1 The FIFO Model

This section is devoted to the study of the FIFO model. Due to the limited capacity $k$ of the buffer, a scheduling algorithm has to deal with buffer overflow events. The most reasonable approach is to keep the $k$ most valuable packets and drop the packets of less value. However, it can also make sense to preemptively drop packets to avoid buffer overflows altogether or at least reduce them and the impact they have. We call such preemptively dropped packets preempted. The approach to handle buffer overflows when they occur is clear and easy but it is much less obvious how to perform the preemption, that is, to decide which packets should be preempted, and despite the intensive effort that has been made to study the FIFO model there are still considerable gaps between upper and lower bounds for the problem.

We first investigate the case where only two different packet values 1 and $\alpha$ appear in the input. We present an optimal online algorithm for this scenario. This improves upon a previous result by Lotker and Patt-Shamir [LPS03] showing a close to optimal bound on the competitive ratio for the worst case choice of $\alpha$. Thereafter, we improve the upper and lower bounds for the PG algorithm by Kesselman, Mansour, and van Stee [KMv05]. PG is currently the only algorithm known to achieve a competitive ratio below 2 for the FIFO model with general packet values.

#### 3.1.1 Two Packet Values

We start by shortly reiterating the lower bound construction from Kesselman et al. [KLM+04]. In the remainder of this section a packet of value 1 is called a 1-packet and a packet of value $\alpha$ is called an $\alpha$-packet.
3.1 The FIFO Model

Fix any deterministic online algorithm $A$. The lower bound consists of two input sequences. First, consider the following sequence: In the first time step, $k-1$ 1-packets and thereafter one $\alpha$-packet arrive. In each of the following time steps, only one $\alpha$-packet arrives until we reach a time step $t$ in which $A$ sends an $\alpha$-packet. Note that this means that $A$ has no longer stored any of the 1-packets. From time step $t$ onwards, no more packets arrive. Our second input sequence is a slight modification of the first one: In time step $t-1$, $k$ $\alpha$-packets arrive instead of just one.

An online algorithm has to find the optimal trade-off between dropping and sending 1-packets. If one $\alpha$-packet arrives in time step $t-1$, the optimal solution would be to send all $k-1$ 1-packets as well as all $t$ $\alpha$-packets whereas $A$ sends all the $\alpha$-packets but only $t-1$ of the 1-packets. If $k$ $\alpha$-packets arrive in time step $t-1$, the optimal solution would be to send no 1-packet and to send all $t-1+k$ $\alpha$-packets whereas $A$ sends $t-1$ 1-packets but only $k$ of the $\alpha$-packets. Thus, the optimal number of 1-packets $s$ to send can be calculated as the solution to

$$\frac{k-1+(s+1)\cdot\alpha}{s+(s+1)\cdot\alpha} = \frac{\alpha \cdot (s+k)}{s+k \cdot \alpha}.$$ (3.1)

Since $s$ may not be an integer, each algorithm has to compromise by either sending $\lfloor s \rfloor$ or $\lceil s \rceil$ of the 1-packets. In the former case the adversary chooses the first input sequence from the lower bound as it yields the larger ratio in the latter case the second input sequence is chosen. This yields the following lower bound on the competitive ratio for a buffer of size $k$ and two packet values 1 and $\alpha$,

$$r(k, \alpha) := \min \left\{ \frac{k-1+(\lfloor s \rfloor +1)\cdot\alpha}{\lfloor s \rfloor+(\lfloor s \rfloor +1)\cdot\alpha}, \frac{\alpha \cdot (k+\lceil s \rceil)}{\lceil s \rceil+k \cdot \alpha} \right\}.$$

**Theorem 3.1.** The competitive ratio of any deterministic online algorithm for the FIFO model with a buffer of size $k$ and two possible packet values 1 and $\alpha > 1$ is at least $r(k, \alpha)$.

Our account strategy (ACC) tries to preempt 1-packets from the buffer in order to avoid losing too many $\alpha$-packets in case of a buffer overflow. The number of preempted 1-packets has to be chosen carefully. Obviously, in the end, the total number of preempted 1-packets should not exceed $(x-1)$ times the total value of sent packets if we want to achieve a competitive ratio of $x$. Hence, one basic idea of ACC is to preempt at most $(x-1) \cdot \alpha$ 1-packets for each $\alpha$-packet entering the buffer and at most $(x-1)$ 1-packets for each sent 1-packet. ACC tries to preempt as many 1-packets as possible without violating this constraint.

We define ACC($x$) with one parameter $x \geq 1$ which is the competitive ratio we aim for and which is therefore used to determine how aggressive the strategy is with respect to preemption. ACC($x$) uses two accounts $a$ and $a'$ which are initially set to 0. Basically, each packet sent by ACC($x$) increases the account $a$ by $(x-1)$ times its own value, and each preempted 1-packet decreases the account $a$ by 1 and increases $a'$ by 1. More precisely, for each time step, ACC($x$) does the following.

1. For each arriving packet $p$, do the following:

   (a) If there is an unoccupied location in the buffer, store $p$. Otherwise, if a 1-packet is stored in the buffer, drop the 1-packet which is closest to the front of the buffer and store $p$. 39
Chapter 3 — Buffer Management for Switches Supporting Quality of Service

(b) If \( p \) is an \( \alpha \)-packet that is stored in the buffer (observe that stored \( \alpha \)-packets are never dropped), increase the account \( a \) by \((x - 1) \cdot \alpha\).

(c) If the buffer is completely filled with \( \alpha \)-packets, reset the accounts \( a \) and \( a' \) to 0.

(d) As long as the buffer contains more than \( k - a' \) packets and there is a 1-packet stored in the buffer, drop the 1-packet which is closest to the front of the buffer.

2. After all packets have arrived, do the following:

(a) As long as the first packet is a 1-packet and \( a \geq 1 \), drop this packet, which is called preempted, decrease the account \( a \) by 1, and increase \( a' \) by 1.

(b) Send the first packet. If this packet is a 1-packet, increase the account \( a \) by \((x - 1)\).

(c) If no packet is stored in the buffer, reset the accounts \( a \) and \( a' \) to 0.

The following theorem states that ACC achieves an optimal competitive ratio for all values of \( k \) and \( \alpha \).

**Theorem 3.2.** ACC\((r(k, \alpha))\) is \( r(k, \alpha) \) competitive for the FIFO model with a buffer of size \( k \) and two possible packet values 1 and \( \alpha > 1 \).

**Proof.** For simplicity, we use a slightly modified definition of a time step in the proof of this theorem. Instead of defining that a new time step starts right before every send event, we now define that a new time step starts right after every send event. In other words, in each time step, first, an arbitrary number of packets arrive and finally, one packet can be sent.

Similar to [LPS03], we define a particular optimal offline strategy OPT. For each input sequence, the set of feasible schedules is a matroid. Hence, a greedy strategy can compute an optimal solution. First, OPT considers all \( \alpha \)-packets in increasing order of their arrival, and thereafter, OPT considers all 1-packets in increasing order of their arrival for inclusion into the schedule.

We call a time step in which ACC’s buffer is completely filled with \( \alpha \)-packets an \( \alpha \)-overflow time step. In the following we argue that the analysis can be restricted to input sequences that satisfy the following two properties.

1. In each time step, except for the last \( k - 1 \) ones, ACC sends a packet, where \( k \) denotes the buffer size.

2. In each \( \alpha \)-overflow time step exactly \( k \) \( \alpha \)-packets and no 1-packets arrive.

The following two observations show that we can assume without loss of generality that each input sequence satisfies the two properties.

**Observation 3.3.** For every input sequence \( \sigma \), there exists an input sequence on which ACC has at least the same competitive ratio as on \( \sigma \) and that satisfies the first property.
3.1 The FIFO Model

**Proof.** After each time step in $\sigma$ in which the buffer of ACC is empty, insert $k - 1$ additional time steps in which no packets arrive. The set of packets sent by ACC does not change and the value of an optimal solution can only increase. Hence, the competitive ratio of ACC for the altered input sequence is at least as large as for the original sequence $\sigma$.

Now, we partition the input sequence into subsequences. A new subsequence starts after $k - 1$ consecutive time steps in which no new packets arrive. Obviously, we can assume that there are never more than $k - 1$ consecutive time steps in which no new packets arrive.

Fix a subsequence $\sigma^{(i)}$. The buffers of ACC and OPT are empty at the beginning of $\sigma^{(i)}$ since any packet stored in the buffers of size $k$ is sent during one of the previous $k$ time steps and no new packets arrive in between. Furthermore, the buffers of ACC and OPT are empty at the end of $\sigma^{(i)}$. However, the buffer of ACC is only empty for the last $k - 1$ time steps of $\sigma^{(i)}$, due to the construction of the subsequences. In all other time steps, a packet is sent.

Finally note that the competitive ratio of ACC for one of the subsequences is at least as large as for the original sequence $\sigma$. \qed

**Observation 3.4.** For each input sequence $\sigma$, there exists an input sequence on which ACC has at least the same competitive ratio as on $\sigma$ and that satisfies both properties.

**Proof.** In each $\alpha$-overflow time step of $\sigma$, add $k \alpha$-packets to the arriving packets. None of these $\alpha$-packets can be stored by ACC. The set of packets sent by ACC does not change and the value of an optimal solution can only increase. Hence, the competitive ratio of ACC for the altered input sequence is at least as large as for the original sequence $\sigma$.

For each $\alpha$-overflow time step, we remove all arriving packets except for $k \alpha$-packets. The sets of packets sent by ACC and OPT do not change since in each time step only the $k$ most valuable arriving packets are relevant. \qed

Now, fix an input sequence $\sigma$ that satisfies both properties. We partition $\sigma$ into time intervals. A time interval ends with an $\alpha$-overflow time step, and the next time interval begins with the time step following this $\alpha$-overflow. Let $P_i$ denote the set of packets arriving in the $i$-th time interval, and let $m$ denote the total number of different time intervals, i.e., each arriving packet in $\sigma$ is in $\bigcup_{i=1}^{m} P_i$.

Let $\text{ACC}^1(P_i)$ ($\text{ACC}^\alpha(P_i)$) denote the subset of 1-packets ($\alpha$-packets) in $P_i$ that are sent by ACC and let $\text{OPT}^1(P_i)$ ($\text{OPT}^\alpha(P_i)$) denote the subset of 1-packets ($\alpha$-packets) in $P_i$ that are sent by OPT. In order to show the theorem, we prove the claimed competitive ratio for each set of packets $P_i$, i.e., we show, for each $P_i$,

$$\frac{|\text{OPT}^1(P_i)| + \alpha \cdot |\text{OPT}^\alpha(P_i)|}{|\text{ACC}^1(P_i)| + \alpha \cdot |\text{ACC}^\alpha(P_i)|} \leq r(k, \alpha). \tag{3.2}$$

The following two lemmata give upper bounds on the number of packets sent by OPT.

**Lemma 3.5.** ACC sends the same number of packets as OPT from each set $P_i$ with $i < m$. 

41
Proof. We prove the lemma by induction over $i$. Fix an $i < m$ and assume that ACC sends the same number of packets as OPT from each set $P_j$ with $j < i$. As a consequence, ACC and OPT start sending packets from $P_i$ in the same time step.

Let $t$ denote the last time step in which a packet from $P_i$ arrives, i.e., the $\alpha$-overflow time step. In time step $t + k - 1$, ACC sends a packet from $P_i$ since in time step $t$ the buffer of ACC is completely filled with $\alpha$-packets and the last $\alpha$-packet in the buffer is a packet from $P_i$.

- OPT does not send more packets from $P_i$ than ACC.
  
  Each packet is stored in the buffer for at most $k - 1$ time steps. As a consequence, after time step $t + k - 1$, OPT can only send packets that arrive after time step $t$, and hence, these packets are not in $P_i$.

- OPT does not send less packets from $P_i$ than ACC.
  
  Assume for contradiction that OPT sends less packets from $P_i$ than ACC. As a consequence, in time step $t + k - 1$ a packet from $P_j$ with $j > i$ is sent by OPT. Hence, OPT does not send all $\alpha$-packets from $P_i$ since $k \alpha$-packets arrive in time step $t$. When one of these $\alpha$-packets not sent by OPT was considered to be included in the schedule of OPT, it could have been added without making the schedule infeasible. This is a contradiction to our definition of OPT.

This concludes the proof of the lemma.

Let $D \subseteq P_m$ denote the set of preempted 1-packets from $P_m$, i.e., $D := \{p \in P_m \mid p$ is preempted by ACC$\}$.

**Lemma 3.6.** $\sum_{i=1}^{m}(|\text{OPT}^1(P_i)| + |\text{OPT}^\alpha(P_i)|) \leq \sum_{i=1}^{m}(|\text{ACC}^1(P_i)| + |\text{ACC}^\alpha(P_i)|) + |D|$.

**Proof.** In the following, we add packets to the schedule of ACC, such that the resulting schedule is maximal, i.e., the schedule becomes infeasible if another packet is added. As a consequence, the schedule of OPT contains the same number of packets as our modified schedule since the set of feasible schedules is a matroid.

Obviously, no packet from $P_i$ with $i < m$ can be added to the schedule without rendering it infeasible. Therefore we concentrate on packets in $P_m$ in the following. Consider the last arrival event $\sigma_t$ in which a 1-packet is dropped from the buffer or an arriving 1-packet is not stored in the buffer. Let $y$ denote the number of packets stored in the buffer of ACC at this point in time that are not contained in the schedule of ACC, i.e., which are preempted at a later time. Recall that the number of unoccupied slots in the buffer is at most the value of account $a'$ at this time. Clearly, at most $a' + y$ of the packets that arrived in or prior to $\sigma_t$ can be added to the schedule of ACC without rendering the schedule infeasible.

Due to our choice of $\sigma_t$, every packet that is not sent by ACC and arrives after $\sigma_t$ is preempted by ACC. Let $y'$ denote the number of these packets. Then, in total, at most $a' + y + y'$ packets can be added to the schedule of ACC without rendering it infeasible.

Since the value of $a'$ equals the number of 1-packets from $P_m$ that were preempted prior to $\sigma_t$ and $y$ is the number of 1-packets that arrived prior to $\sigma_t$ but are preempted
3.1 The FIFO Model

after $a$, the total number of packets from $P_m$ that are preempted by ACC is $|D| = a' + y + y'$.

We conclude that by adding at most $|D|$ packets to the schedule of ACC we obtain a maximal schedule, which implies the lemma.

Now, we are able to show Inequality $(3.2)$ for $P_m$. Combining Lemma 3.5 and Lemma 3.6 yields $|OPT^1(P_m)| + |OPT^\alpha(P_m)| \leq |ACC^1(P_m)| + |ACC^\alpha(P_m)| + |D|$. Since ACC sends all $\alpha$-packets from $P_m$, $|OPT^\alpha(P_m)| = |ACC^\alpha(P_m)|$. Hence,

$$|OPT^1(P_m)| + \alpha \cdot |OPT^\alpha(P_m)| \leq |ACC^1(P_m)| + \alpha \cdot |ACC^\alpha(P_m)| + |D| \quad (3.3)$$

When the last packet of $P_{m-1}$ arrives, the buffer of ACC is completely filled with $\alpha$-packets and the account $a$ is reset to 0. Hence, the preemption of later arriving packets, i.e., packets in $P_m$, is caused by packets from $P_m$ that are sent by ACC. As a consequence,

$$|D| \leq (r(k, \alpha) - 1) \cdot (|ACC^1(P_m)| + \alpha \cdot |ACC^\alpha(P_m)|) .$$

In combination with Inequality $(3.3)$, this yields

$$|OPT^1(P_m)| + \alpha \cdot |OPT^\alpha(P_m)| \leq |ACC^1(P_m)| + \alpha \cdot |ACC^\alpha(P_m)| + (r(k, \alpha) - 1) \cdot (|ACC^1(P_m)| + \alpha \cdot |ACC^\alpha(P_m)|)

= r(k, \alpha) \cdot (|ACC^1(P_m)| + \alpha \cdot |ACC^\alpha(P_m)|) .$$

To show Inequality $(3.2)$ for each $P_i$ with $i < m$, we need to know by how much the number of $\alpha$-packets sent by OPT exceeds the number of $\alpha$-packets sent by ACC.

Consider a $P_i$ with $i < m$. ACC sends at least $k$ $\alpha$-packets from $P_i$. The only $\alpha$-packets that cannot be sent by ACC are packets arriving in the $\alpha$-overflow time step. For each $\alpha$-packet in the buffer of ACC at this time step that is already sent by OPT, OPT can store one additional $\alpha$-packet that cannot be sent by ACC.

The following lemma gives an upper bound on the number of $\alpha$-packets sent by OPT but not by ACC.

**Lemma 3.7.** Consider a set $P_i$ with $i < m$. At most

$$\left\lceil \frac{k - 1 + r(k, \alpha)}{r(k, \alpha) - 1} \cdot \alpha + r(k, \alpha) \right\rceil - 1$$

$\alpha$-packets in the buffer of ACC are already sent by OPT right before the $\alpha$-overflow time step of $P_i$.

*Proof.* Consider the latest time step $t$ in which the number of $\alpha$-packets in the buffer of ACC that are already sent by OPT is increased from $n - 1$ to $n$. Hence, ACC sends a 1-packet $p$ and OPT sends an $\alpha$-packet that arrived after $p$ and is stored in the buffer of ACC. Each $\alpha$-packet in the buffer of ACC arrived later than $p$. Let $q$ denote the first of these $\alpha$-packets, i.e., the one that arrived earliest, and let $t'$ denote the time step in which $q$ arrives.

Let $a'_{t'}$ and $a_{t'}$ denote the values of the accounts $a'$ and $a$ respectively, after the arrival of $q$ in time step $t'$. At this time, there are at most $k - 1 - a'_{t'}$ other packets in
the buffer of ACC and all of them arrived earlier than \( q \). Otherwise \( p \) would have been dropped in step 1.(d) of the algorithm. Let \( y \) be the number of \( \alpha \)-packets from \( P_i \) stored in ACC’s buffer at this time (including \( q \)). Each of these packets has increased the account \( a \) by \((r(k, \alpha) - 1)\alpha\). Hence, \( a_{\alpha'} + a_{\alpha'}' \geq y \cdot (r(k, \alpha) - 1)\alpha\).

At least \( n - 1 \) \( \alpha \)-packets arrive after \( q \). Each of these increases the account \( a \) by \((r(k, \alpha) - 1)\alpha\). In addition, the account \( a \) is increased by \( z \cdot (r(k, \alpha) - 1) \), where \( z \) denotes the number of 1-packets sent by ACC from \( t' \) to \( t \).

However, the value of the account \( a \) is less than 1 right before \( p \) is sent by ACC since otherwise \( p \) would have been preempted. Hence, more than \((n - 1) \cdot (r(k, \alpha) - 1) \cdot \alpha + z \cdot (r(k, \alpha) - 1) + a_{\alpha'} - 1\) 1-packets are preempted from \( t' \) to \( t \). All the preempted 1-packets arrive before \( p \).

Since only one \( \alpha \)-packet can be sent by OPT in each time step and ACC and OPT start sending packets from \( P_i \) in the same time step, at least \( n - 1 \) packets from \( P_i \) are sent from \( t' \) to \( t \). Thus, \( y - 1 + z \geq n - 1 \).

Altogether, \( n - 1 \) of the \( k - a_{\alpha'} \) packets that are stored in the buffer after the arrival of \( q \) are sent until time step \( t \), more than \((n - 1) \cdot (r(k, \alpha) - 1) \cdot \alpha + z \cdot (r(k, \alpha) - 1) + a_{\alpha'} - 1\) of them are preempted, and \( p \) and \( q \) are still stored in the buffer. Hence,

\[
n - 1 + (n - 1) \cdot (r(k, \alpha) - 1) \cdot \alpha + z \cdot (r(k, \alpha) - 1) + a_{\alpha'} - 1 + 2 < k - a_{\alpha'}',
\]

which simplifies to

\[
k > n + ((n - 1)\alpha + z) \cdot (r(k, \alpha) - 1) + a_{\alpha'} + a_{\alpha'}'.
\]

Finally,

\[
k - 1 + r(k, \alpha) > n + ((n - 1)\alpha + z) \cdot (r(k, \alpha) - 1) + a_{\alpha'} + a_{\alpha'}' - 1 + r(k, \alpha)
\geq n + ((n - 1)\alpha + z) \cdot (r(k, \alpha) - 1) + y \cdot (r(k, \alpha) - 1)\alpha - 1 + r(k, \alpha)
= n + (n\alpha + z) \cdot (r(k, \alpha) - 1) + (y - 1) \cdot (r(k, \alpha) - 1)\alpha - 1 + r(k, \alpha)
\geq n + (n\alpha + z + y - 1) \cdot (r(k, \alpha) - 1) - 1 + r(k, \alpha)
\geq n + (n\alpha + n - 1) \cdot (r(k, \alpha) - 1) - 1 + r(k, \alpha)
= n + (n\alpha + n) \cdot (r(k, \alpha) - 1)
= n(\alpha \cdot (r(k, \alpha) - 1) + r(k, \alpha)),
\]

which concludes the proof of the lemma. \( \square \)

Using Lemma 3.7, we now prove the following lemma which, in turn, is then used to conclude the proof the theorem.

**Lemma 3.8.** For every \( i < m \),

\[
(\alpha - r(k, \alpha)) \cdot |\text{OPT}^\alpha(P_i)| \leq r(k, \alpha) \cdot (\alpha - 1) \cdot |\text{ACC}^\alpha(P_i)|.
\]

**Proof.** Fix an \( i < m \). Due to Lemma 3.7,

\[
|\text{OPT}^\alpha(P_i)| \leq |\text{ACC}^\alpha(P_i)| + \left[ \frac{k - 1 + r(k, \alpha)}{(r(k, \alpha) - 1) \cdot \alpha + r(k, \alpha)} \right] - 1.
\]

\(^4\)From \( t' \) to \( t \) denotes the time interval from \( t' \) to \( t \) excluding time step \( t \).
3.1 The FIFO Model

Hence, it remains to show

\[(\alpha - r(k, \alpha)) \cdot \left( |\text{ACC}^\alpha(P_i)| + \left\lceil \frac{k - 1 + r(k, \alpha)}{(r(k, \alpha) - 1) \cdot \alpha + r(k, \alpha)} \right\rceil - 1 \right) \leq r(k, \alpha) \cdot (\alpha - 1) \cdot |\text{ACC}^\alpha(P_i)| ,\]

which is equivalent to

\[(\alpha - r(k, \alpha)) \cdot \left( \left\lceil \frac{k - 1 + r(k, \alpha)}{(r(k, \alpha) - 1) \cdot \alpha + r(k, \alpha)} \right\rceil - 1 \right) - (r(k, \alpha) - 1) \cdot \alpha \cdot k \leq 0 . \] (3.4)

Since the i-th time interval contains an \(\alpha\)-overflow time step, \(|\text{ACC}^\alpha(P_i)| \geq k\).

Therefore, it suffices to show

\[(\alpha - r(k, \alpha)) \cdot \left( \left\lceil \frac{k - 1 + r(k, \alpha)}{(r(k, \alpha) - 1) \cdot \alpha + r(k, \alpha)} \right\rceil - 1 \right) - (r(k, \alpha) - 1) \cdot \alpha \cdot k \leq 0 . \] (3.4)

Depending on the value of \(r(k, \alpha)\) we distinguish two cases.

- If \(r(k, \alpha) = \frac{k - 1 + (\lfloor s \rfloor + 1) - k}{\lceil s \rceil + (\lfloor s \rfloor + 1) - k}\), substituting \(r(k, \alpha)\) in Inequality (3.4) yields

\[(\alpha - r(k, \alpha)) \cdot \left( \left\lceil \frac{k - 1 + r(k, \alpha)}{(r(k, \alpha) - 1) \cdot \alpha + r(k, \alpha)} \right\rceil - 1 \right) - (r(k, \alpha) - 1) \cdot \alpha \cdot k = \frac{\alpha^2 |s|^2 + (1 - \alpha + \alpha^2 - k + k\alpha)|s| - k^2\alpha + k\alpha}{|s|(1 + \alpha) + \alpha} .\]

The denominator of the above fraction is obviously positive. However, the numerator is not positive since \(\alpha^2\) and \((1 - \alpha + \alpha^2 - k + k\alpha)\) are both positive and hence,

\[\alpha^2 |s|^2 + (1 - \alpha + \alpha^2 - k + k\alpha)|s| - k^2\alpha + k\alpha \leq \alpha^2 s^2 + (1 - \alpha + \alpha^2 - k + k\alpha)s - k^2\alpha + k\alpha = 0 .\]

We conclude

\[\frac{\alpha^2 |s|^2 + (1 - \alpha + \alpha^2 - k + k\alpha)|s| - k^2\alpha + k\alpha}{|s|(1 + \alpha) + \alpha} \leq 0 .\]

- If \(r(k, \alpha) = \frac{\alpha + (\lfloor s \rfloor + s)}{\lceil s \rceil + k\alpha}\), we first observe that

\[\frac{(k - 1 + \alpha)|s| + k^2\alpha}{\alpha(\alpha |s| + k)} - (s + 1) = \frac{(-s - 1)\alpha^2 + k - 1 + \alpha}{\alpha(\alpha |s| + k)} \leq \frac{(-s - 1)\alpha^2 + k - 1 + \alpha s + k^2 - (s + 1)\alpha k}{\alpha(\alpha |s| + k)} = 0 .\]
We now turn our attention to the FIFO model with arbitrary packet values. Kesselman, Mansour, and van Stee [KMv05] introduce the preemptive greedy strategy (PG) with the parameter $\beta > 0$. This concludes the proof of the lemma.

Due to Lemma 3.8,

$$\frac{|\text{OPT}^1(P_i)| + \alpha \cdot |\text{OPT}^\alpha(P_i)|}{|\text{ACC}^1(P_i)| + \alpha \cdot |\text{ACC}^\alpha(P_i)|} \leq \frac{\alpha \cdot |\text{OPT}^\alpha(P_i)|}{(\alpha - 1) \cdot |\text{ACC}^\alpha(P_i)| + |\text{OPT}^\alpha(P_i)|}.$$ 

And finally, due to Lemma 3.8,

$$\frac{\alpha \cdot |\text{OPT}^\alpha(P_i)|}{(\alpha - 1) \cdot |\text{ACC}^\alpha(P_i)| + |\text{OPT}^\alpha(P_i)|} \leq \frac{\alpha \cdot |\text{OPT}^\alpha(P_i)|}{r(k,\alpha) \cdot |\text{OPT}^\alpha(P_i)| + |\text{OPT}^\alpha(P_i)|}$$

which concludes the proof of the theorem.

3.1.2 General Packet Values and the Preemptive Greedy Strategy

We now turn our attention to the FIFO model with arbitrary packet values. Kesselman, Mansour, and van Stee [KMv05] introduce the preemptive greedy strategy (PG) with the parameter $\beta > 1$ for this problem. When a packet $p$ arrives, PG does the following.

1. Find the first packet, i.e., the packet closest to the front of the buffer, $p'$, with $v(p') \leq v(p)/\beta$. If such a packet $p'$ exists, drop it ($p'$ is called preempted by $p$).

2. If there is an unoccupied location in the buffer, store $p$ in the buffer.

3. Otherwise, find a packet $p'$ with the smallest value among the packets in the buffer. If $v(p') < v(p)$, drop $p'$ ($p'$ is called ejected by $p$) and store $p$ in the buffer. Otherwise, drop $p$ ($p$ is called rejected).
3.1 The FIFO Model

Bansal et al. [BFK+04] study a modified version of PG. The only difference is that step 1 of PG is substituted by the following.

1. Find the first packet $p'$, with $v(p') \leq v(p)/\beta$ and $v(p')$ is not larger than the value of the packet stored after $p'$ in the buffer. If such a packet exists, drop it.

However, this modification does not improve the performance of the strategy [Jaw05].

First, we show a lower bound of $1 + 1/\sqrt{2} \approx 1.707$ on the competitive ratio of PG which improves upon the previously best lower bound of $(1 + \sqrt{5})/2 \approx 1.618$ by Kesselman, Mansour, and van Stee [KMv05]. Then, we improve the upper bound on the competitive ratio of PG to $\sqrt{3} \approx 1.732$. Previously, the best bound known for the problem was the upper bound of $7/4$ on the competitive ratio of the modified PG strategy by Bansal et al. [BFK+04].

**Theorem 3.9.** The competitive ratio of PG is at least $1 + 1/\sqrt{2} \approx 1.707$.

**Proof.** In the following we assume that the buffer size $k$ is even. Depending on $\beta$, we distinguish the following two cases.

- Suppose that $\beta \leq 2 + \sqrt{2}$.

The input sequence consists of $n$ consecutive phases defined as follows.

- Phase $1 \leq i < n$ consists of $k/2$ time steps. In time step 1 of the $i$-th phase, at first $k$ packets of value $\varepsilon$ and finally $k/2$ packet of value $\beta^i$ arrive. In the remaining $k/2 - 1$ time steps, new packets do not arrive.
- Phase $n$ consists of one time step. In this time step, $k$ packets of value $\beta^{n-1}$ arrive.

For this input sequence, PG produces value

$$\lim_{\varepsilon \to 0} \sum_{i=1}^{n-1} \left( \frac{k}{2} \cdot \varepsilon \right) + k \cdot \beta^{n-1} = k \cdot \beta^{n-1},$$

and the optimal value is

$$\sum_{i=1}^{n-1} \left( \frac{k}{2} \cdot \beta^i \right) + k \cdot \beta^{n-1} = k \cdot \frac{3\beta^n - 2\beta^{n-1} - \beta}{2(\beta - 1)}.$$

Hence, the competitive ratio is

$$\lim_{n \to \infty} \frac{3\beta^n - 2\beta^{n-1} - \beta}{2(\beta^n - \beta^{n-1})} = 1 + \frac{\beta}{2(\beta - 1)} \geq 1 + \frac{1}{\sqrt{2}}.$$

- Suppose that $\beta > 2 + \sqrt{2}$.

The input sequence consists of $n$ consecutive phases defined as follows.

- Phase 1 consists of $k - 1$ time steps. In time step 1, at first $k - 1$ packets of value 1 and finally one packet of value $\alpha < \beta$ arrive. In each of the remaining $k - 2$ time steps, one packet of value $\alpha$ arrives.
Chapter 3 — Buffer Management for Switches Supporting Quality of Service

– Phase 1 < \( i < n \) consists of \( k - 1 \) time steps. In each of these time steps, one packet of value \( \alpha^i \) arrives.
– Phase \( n \) consists of one time step. In this time step, \( k \) packets of value \( \alpha^{n-1} \) arrive.

For this input sequence, \( \text{PG} \) produces value
\[
\sum_{i=0}^{n-2} ((k-1) \cdot \alpha^i) + k \cdot \alpha^{n-1} = k \cdot \frac{\alpha^n - \frac{1}{k} \cdot \alpha^{n-1} - \frac{k-1}{k} \cdot \alpha}{\alpha - 1},
\]
and the optimal value is
\[
\sum_{i=1}^{n-1} ((k-1) \cdot \alpha^i) + k \cdot \alpha^{n-1} = k \cdot \frac{(2 - \frac{1}{k}) \cdot \alpha^n - \alpha^{n-1} - \frac{k-1}{k} \cdot \alpha}{\alpha - 1}.
\]

Hence, the competitive ratio is
\[
\lim_{\alpha \to \beta} \lim_{n \to \infty} \lim_{k \to \infty} \frac{2 \alpha^n - \alpha^{n-1} - \frac{k-1}{k} \cdot \alpha}{\alpha^n - \frac{1}{k} \cdot \alpha^{n-1} - \frac{k-1}{k} \cdot \alpha} = \lim_{\alpha \to \beta} \frac{2 \alpha - 1 - \frac{\beta - 1}{\beta}}{\alpha - 1} = 1 + \frac{\beta - 1}{\beta} \geq 1 + \frac{1}{\sqrt{2}}.
\]

This concludes the proof of the theorem.

The above lower bound does not hold for an arbitrary fixed buffer size. Instead, the buffer size has to tend to infinity. However, a similar lower bound holding for arbitrary buffer sizes cannot be expected. Kesselman et al. [KLM+04] show that if \( \beta \) is chosen infinitely large, that is, \( \text{PG} \) never preempts a packet, \( \text{PG}' \)’s competitive ratio is \( 2 - \frac{1}{k} \) which is already smaller than \( 1 + \frac{1}{\sqrt{2}} \) for \( k \leq 3 \).

The following theorem states an upper bound on the competitive ratio of \( \text{PG} \).

**Theorem 3.10.** \( \text{PG} \) achieves a competitive ratio of \( \sqrt{3} \approx 1.732 \) for \( \beta = 2 + \sqrt{3} \).

**Proof.** Let \( \text{OPT} \) denote an optimal offline strategy. We assume that \( \text{OPT} \) only stores packets in its buffer that are sent by \( \text{OPT} \). Further, we assume that, at the arrival of each packet, the buffer of \( \text{PG} \) is completely filled with packets. If there are unoccupied locations in the buffer of \( \text{PG} \), it is assumed that dummy packets of value 0 are stored at these locations which are always at the end of the buffer. Hence, each arriving packet either preempts another packet, ejects another packet, or is rejected.

Fix an input sequence of arriving packets. This input sequence can also be regarded as a sequence \( \sigma = \sigma_1 \sigma_2 \cdots \sigma_t \) of arrival and send events, where each arrival of a new packet corresponds to an arrival event and each sending of a packet corresponds to a send event.

Let \( S_t^{\text{PG}} (S_t^{\text{OPT}}) \) denote the set of packets sent by \( \text{PG} \) (OPT) by the end of event \( \sigma_t \), i.e., all packets sent in the events \( \sigma_1, \ldots, \sigma_t \) (including \( \sigma_t \)). Let \( B_t^{\text{PG}} (B_t^{\text{OPT}}) \) denote the set of packets stored in the buffer of \( \text{PG} \) (OPT) at the end of \( \sigma_t \). For a packet \( p \in B_t^{\text{PG}} \), we call \( c_t(p) \) the charge of \( p \) at the end of \( \sigma_t \). Further, we call \( D_t \) the set of packets with a deposit at the end of \( \sigma_t \). Note that charges and deposits are two
3.1 The FIFO Model

independent concepts we use. Initially, \( D_0 := \emptyset \). The goal is to choose \( c_t(p) \) and \( D_t \) in such a way that, for each event \( \sigma_t \), the main inequality

\[
 r \sum_{p \in S_{OPT}^t \cup D_t} v(p) + \sum_{p \in B_{OPT}^t} c_t(p) \geq \sum_{p \in S_{OPT}^{t-1} \cup D_{t-1}} v(p)
\]

is true, with \( r := \sqrt{3} \). This directly implies the theorem.

Let \( \Delta_{pg}^t \) (\( \Delta_{opt}^t \)) denote the alterations of the left (right) side of the main inequality at the event \( \sigma_t \), i.e.,

\[
\Delta_{pg}^t := r \sum_{p \in S_{OPT}^t \setminus S_{pg}^{t-1}} v(p) + \sum_{p \in B_{pg}^t} c_t(p) - \sum_{p \in B_{pg}^{t-1}} c_{t-1}(p) \quad \text{and}
\]

\[
\Delta_{opt}^t := \sum_{p \in (S_{OPT}^t \cup D_t) \setminus (S_{OPT}^{t-1} \cup D_{t-1})} v(p)
\]

Obviously, the main inequality is true before the first event since packets have not been sent so far and the buffers and the set of packets with a deposit are empty. Hence, it is sufficient to show, for each event \( \sigma_t \), \( \Delta_{pg}^t \geq \Delta_{opt}^t \) since this yields the main inequality.

First, we give an intuition for the basic ideas of the proof. Then, we present the formal proof. The basic idea for the set \( D_t \) is simple. Packets stored exclusively in the buffer of OPT at the end of event \( \sigma_t \), especially packets already sent by PG, could be a problem if PG cannot send a packet because the buffer is empty, when those packets are sent by OPT. The left side of the main inequality is not increased at these events, and it is crucial for the proof that the same is true for the right side of the main inequality. Hence, these packets have to be contained in \( D_t \). Intuitively, PG has already gained enough value to pay these packets in advance, that is, before they are sent by OPT.

The basic idea for \( c_t(p) \) is the following. In case of a send event \( \sigma_t \) in which OPT sends a much more valuable packet than PG that is not in \( D_{t-1} \), the right side of the main inequality is increased by a large amount and we have to compensate this by increasing the charge of packets stored in the buffer of PG. It is fairly unproblematic to charge a packet up to \((r - 1)\) times its own value because if such a packet is sent by PG and OPT in the same send event, the left side of the main inequality is still increased by the same amount as the right side of the main inequality. In any case, larger charges are only allowed for packets that are exclusively in the buffer of PG.

In case of a buffer overflow in the buffer of PG in which a charged packet is ejected, this charge has to be transferred to another packet in the buffer of PG. This is problematic for an ejected packet that is charged by more than \((r - 1)\) times its own value since the charge might be transferred to a packet that is also stored in OPT’s buffer and hence not exclusively stored in PG’s buffer. Therefore we introduce the concept of buddies. A packet stored exclusively in the buffer of PG might be charged by \(2(r-1)\) times its own value only if there is another packet in the buffer of PG that is not charged at all. We call the packet with no charge buddy for the packet with the high charge.

Unfortunately, the precise definition of charges is slightly more complicated. Before we define the charges in detail, we need some preliminaries. For each two packets \( p \) and
\[ s_t(p) \quad c_t(p) \quad \text{comment} \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>BC</td>
<td>((r - 2) \cdot v(p))</td>
<td>buddy with credit</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>buddy</td>
</tr>
<tr>
<td>U</td>
<td>((r/\beta) \cdot v(p) + (2 - r) \cdot v_t^{min}(p))</td>
<td>unproblematic</td>
</tr>
<tr>
<td>E</td>
<td>((r - 1) \cdot v(p))</td>
<td>exclusively in (B_t^{PG}), i.e., not in (B_t^{OPT})</td>
</tr>
<tr>
<td>EB</td>
<td>(2(r - 1) \cdot v(p))</td>
<td>exclusively in (B_t^{PG}) with buddy</td>
</tr>
</tbody>
</table>

Figure 3.1: Definition of the charge \(c_t(p)\) of a packet \(p \in B_t^{PG}\) at the end of event \(\sigma_t\). The charges are listed in increasing order, e.g., a packet in state \(E\) is at least as charged as a packet of same value in state \(U\). Note that the charge in case of state \(BC\) is negative. Further, note that \(v_t^{min}(p) \leq v(p)\). If \(v_t^{min}(p) = v(p)\), the charges in state \(U\) and \(E\) are the same for packet \(p\).

\(p'\), we write \(p \prec p'\) if \(p\) arrives before \(p'\) in the input sequence. Further, for each packet \(p\) and the undefined symbol \(\perp\), \(p \prec \perp\), \(\perp \prec p\), and \(\perp \prec \perp\). Each \(p \in B_t^{PG}\) can have assigned another \(p' \in B_t^{PG}\) as buddy if \(p \prec p'\). Each \(p' \in B_t^{PG}\) is assigned as buddy for at most one other packet. If \(p \in B_t^{PG}\) has assigned another \(p' \in B_t^{PG}\) as buddy at the end of event \(\sigma_t\), define \(b_t(p) := p'\), otherwise, define \(b_t(p) := \perp\). Further, for each \(p \notin B_t^{PG}\), \(b_t(p) := \perp\). Finally, for each \(p \in B_t^{PG}\), define \(v_t^{min}(p) := \min\{v(p') | B_t^{PG} \ni p' \geq p\}\).

Each \(p \in B_t^{PG}\) is in one of the five states \(BC\), \(B\), \(U\), \(E\), and \(EB\). Let \(s_t(p)\) denote the state of \(p\) at the end of event \(\sigma_t\), and define \(s_t(\perp) := \perp\). Let \(BC_t\), \(B_t\), \(U_t\), \(E_t\), and \(EB_t\) denote the set of packets that are in state \(BC\), \(B\), \(U\), \(E\), and \(EB\), respectively, at the end of event \(\sigma_t\). The initial state of each packet is \(B\), and dummy packets of value \(0\) are always in state \(B\). The charge \(c_t(p)\) of a packet \(p\) at the end of event \(\sigma_t\) is defined in Figure 3.1. Note that the charge of a packet, except for packets in state \(U\), does not change as long as this packet stays in the same state. The charge of a packet in state \(U\) can only increase since \(v_t^{min}(p) \leq v_t^{min+1}(p)\).

Let \(P_t\) denote the set of packets that are preempted by \(PG\) by the end of event \(\sigma_t\). For each packet \(p\), if \(p\) preempts another packet \(p'\), define \(d(p) := p'\), otherwise, define \(d(p) := \perp\). A packet \(p\) transitively preempts another packet \(p'\), if either \(d(p) = p'\), \(p\) preempts a packet that transitively preempts \(p'\), or \(p\) ejects a packet that transitively preempts \(p'\). For each \(p' \notin P_t\), if \(p'\) is transitively preempted by a packet \(p \in B_t^{PG}\), define \(\hat{d}_t(p') := p\), otherwise, define \(\hat{d}_t(p') := \perp\). For each \(p' \notin P_t\), define \(\hat{d}_t(p') := \perp\). Figure 3.2 gives an overview of our notations.

In order to prove the theorem, we show the following five invariants by induction over the event sequence \(\sigma\). To shorten notation, we define \(X_t := (P_t \cup S_t^{PG}) \cap (B_t^{OPT} \setminus D_t)\).

1. \(\Delta_t^{PG} \geq \Delta_t^{OPT}\).
2. If \(p \in E_t \cup EB_t\), then \(p \notin B_t^{OPT}\).
3. If \(p \in EB_t\), then \(b_t(p) \in BC_t \cup B_t\).
4. If \(p \in X_t\), then \(\hat{d}_t(p) \in BC_t \cup B_t\).
5. If \(p \in B_t^{PG} \setminus BC_t\), then \(b_t^{-1}(p) \prec d(p)\).
3.1 The FIFO Model

<table>
<thead>
<tr>
<th>notation</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_{t}^{PG}, S_{t}^{OPT}$</td>
<td>The set of packets sent by PG and OPT by the end of $\sigma_{t}$.</td>
</tr>
<tr>
<td>$B_{t}^{PG}, B_{t}^{OPT}$</td>
<td>The set of packets stored in the buffer of PG and OPT at the end of $\sigma_{t}$.</td>
</tr>
<tr>
<td>$P_{t}$</td>
<td>The set of packets preempted by PG by the end of $\sigma_{t}$.</td>
</tr>
<tr>
<td>$D_{t}$</td>
<td>The set of packets with a deposit at the end of $\sigma_{t}$.</td>
</tr>
<tr>
<td>$X_{t}$</td>
<td>Short notation for $(P_{t} \cup S_{t}^{PG}) \cap (B_{t}^{OPT} \setminus D_{t})$.</td>
</tr>
<tr>
<td>$c_{t}(p)$</td>
<td>The charge of the packet $p \in B_{t}^{PG}$ at the end of $\sigma_{t}$, which is determined by its state.</td>
</tr>
<tr>
<td>$s_{t}(p)$</td>
<td>The state of the packet $p \in B_{t}^{PG}$. Each packet $p \in B_{t}^{PG}$ is in one of the five states BC, B, U, E, or EB.</td>
</tr>
<tr>
<td>$v_{t}^{min}(p)$</td>
<td>The value of the least valuable packet stored in the buffer of PG in front of $p \in B_{t}^{PG}$.</td>
</tr>
<tr>
<td>$b_{t}(p)$</td>
<td>The buddy packet of the packet $p$. Equals ⊥ if $p$ has no buddy or $p \notin B_{t}^{PG}$.</td>
</tr>
<tr>
<td>$b_{t}^{-1}(p)$</td>
<td>The packet for which $p$ is a buddy. Equals ⊥ if $p$ is not a buddy for another packet.</td>
</tr>
<tr>
<td>$d(p)$</td>
<td>The packet that is preempted by $p$. Equals ⊥ if $p$ does not preempt another packet.</td>
</tr>
<tr>
<td>$d_{t}(p)$</td>
<td>The packet $p' \in B_{t}^{PG}$ that transitively preempted $p$. Equals ⊥ if $p$ was not preempted, i.e., $p \notin P_{t}$, or there is no packet in the buffer of PG which transitively preempted $p$.</td>
</tr>
<tr>
<td>$p \prec p'$</td>
<td>The packet $p$ arrives before the packet $p'$.</td>
</tr>
</tbody>
</table>

Figure 3.2: Informal overview of our notations.

Observe that the invariants have only to be verified in the following cases.

I1: Always.

I2: For each packet $p \in (E_{t} \cup EB_{t}) \setminus (E_{t-1} \cup EB_{t-1})$.

I3: For each packet $p$ with $(p \in EB_{t} \setminus EB_{t-1}) \lor (b_{t-1}(p) \in (BC_{t-1} \cup B_{t-1})) \setminus (BC_{t} \cup B_{t}) \lor (b_{t-1}(p) \neq b_{t}(p))$.

I4: For each packet $p$ with $(p \in X_{t} \setminus X_{t-1}) \lor (d_{t-1}(p) \in (BC_{t-1} \cup B_{t-1})) \setminus (BC_{t} \cup B_{t}) \lor (d_{t-1}(p) \neq d_{t}(p))$.

I5: For each packet $p$ with $(p \in (B_{t}^{PG} \setminus BC_{t}) \setminus (B_{t-1}^{PG} \setminus BC_{t-1})) \lor (b_{t-1}(p) \neq b_{t}^{-1}(p))$.

The following lemma is used to dramatically reduce the number of cases we have to consider. Whenever we encounter a situation during the induction where $B_{t}^{OPT} \not\subseteq P_{t} \cup S_{t}^{PG} \cup B_{t}^{PG}$, we manipulate the buffer contents of OPT in such a way that $B_{t}^{OPT} \subseteq P_{t} \cup S_{t}^{PG} \cup B_{t}^{PG}$. The five invariants continue to hold after this manipulation. Thereafter, we can continue the induction.
Lemma 3.11. Assume that $\sigma_t$ is the first event with $B_t^\text{OPT} \not\subseteq P_t \cup S_t^\text{pg} \cup B_t^\text{pg}$. Then, the buffer contents of OPT can be manipulated in such a way that $B_t^\text{OPT} \subseteq P_t \cup S_t^\text{pg} \cup B_t^\text{pg}$ and the five invariants continue to hold.

Proof. Assume that $\sigma_t$ is the first event with $B_t^\text{OPT} \not\subseteq P_t \cup S_t^\text{pg} \cup B_t^\text{pg}$, i.e., the buffer of OPT contains a packet that was ejected or rejected by PG. Since $\sigma_t$ is the first event with $B_t^\text{OPT} \not\subseteq P_t \cup S_t^\text{pg} \cup B_t^\text{pg}$, a packet $p$ must have been ejected or rejected by PG in $\sigma_t$. This also implies that $\sigma_t$ is an arrival event. In the following, we assume that $p$ is rejected by PG but stored in the buffer of OPT. The arguments for the case that $p$ is ejected are analogous.

Since OPT stores $p$ in its buffer and the buffer of PG is completely filled with packets, there exists a packet $q \in B_t^\text{pg} \setminus B_t^\text{OPT}$. The value $v(q)$ of $q$ has to be at least as large as $v(p)$. Otherwise, $q$ would have been ejected by PG and $p$ would have been stored in the buffer of PG. Define $v := v(p)$.

After $p$ arrived, we manipulate the buffer contents of OPT in the following way: The arrival time of $p$ is set to the arrival time of $q$, i.e., the packets stored in the buffer of OPT are reordered such that $p$ is placed at the position of $q$ if $q$ would be contained in the buffer of OPT. This reordering does not change the set of packets sent by OPT and hence, does not change the total value gained by OPT.

In addition, we manipulate the value of $p$. We increase the value of $p$ to the value of $q$. After both manipulations, the attributes of the packet $p \in B_t^\text{OPT} \setminus B_t^\text{pg}$ are identical to the packet $q \in B_t^\text{pg} \setminus B_t^\text{OPT}$. As a consequence, $p$ can be identified with $q$, i.e., we can assume that $p$ is actually the packet $q$ and therefore stored in the buffer of PG.

The Invariants I3, I4, and I5 are not effected by our manipulation since changes are not made in the buffer of PG and $q \not\in P_t \cup S_t^\text{pg}$. If $s_t(q) \not\in \{E, EB\}$, Invariant I2 is not effected either. Otherwise, set $s_t(q) := U$ and, if $q$ was in state EB, set $s_t(b_{t-1}(q)) := U$ (due to I3 $b_{t-1}(q)$ exists and is in state BC or B in this case). Thus, Invariants I2–I5 continue to hold.

It remains to study the effect of our manipulation on the main inequality.

- If $s_t(q) \not\in \{E, EB\}$ the main inequality does not change.
- If $q$ was in state E and its state changed to U, the left side of the main inequality is decreased by at most $(r-1) \cdot v(q) - ((r/\beta) \cdot v(q) + (2-r) \cdot v_{t-1}^\text{min}(q)) = (2-r) \cdot (v(q) - v_{t-1}^\text{min}(q)) \leq v(q) - v$ since $r/\beta = 2r - 3$ and $p$ is rejected at $\sigma_t$.
- If $q$ was in state EB and its state changed to U, the left side of the main inequality is decreased by at most $2(r-1) \cdot v(q) - ((r/\beta) \cdot v(q) + (2-r) \cdot v_{t-1}^\text{min}(q)) = v(q) + (r-2) \cdot v_{t-1}^\text{min}(q) \leq v(q) + (r-2) \cdot v$. In this case, the state of $b_{t-1}$ changed from BC to B or U. This increases the left side of the main inequality by at least $(r/\beta) \cdot v(b_{t-1}) + (2-r) \cdot v_{t-1}^\text{min}(b_{t-1}) \geq (r/\beta) \cdot v + (2-r) \cdot v \geq (r-1) \cdot v$. Hence, in total the left side of the main inequality is decreased by at most $v(q) + (r-2) \cdot v - (r-1) \cdot v = v(q) - v$.

Hence, the left side of the main inequality is decreased by at most $v(q) - v$. As a consequence, we can only guarantee that

$$v(q) - v + \sum_{p' \in S_t^\text{pg}} r \cdot v(p') \geq \sum_{p' \in S_t^\text{OPT}} v(p')$$
after the last event $\sigma_1$ in the sequence of events $\sigma$. This is not sufficient to show the theorem. Fortunately, by virtually increasing the value of $p$ we have also increased $\sum_{p' \in S_{t}^{OPT}} v(p')$ by $v(q) - v$, i.e., the real total value of $OPT$ is smaller by $v(q) - v$. Thus, we have

$$v(q) - v + r \cdot PG(\sigma) = v(q) - v + \sum_{p' \in S_{t}^{pg}} r \cdot v(p') \geq \sum_{p' \in S_{t}^{OPT}} v(p') = OPT(\sigma) + v(q) - v,$$

which concludes the proof of the lemma.

Even after applying the above lemma the induction consists of an extensive, quite involved, and somewhat tedious case distinction. Therefore, this case distinction and the choices for the set $D_t$, the states of the packets, and their buddy relations are deferred to Appendix A.

### 3.2 The Bounded-Delay Model

In this section we study the bounded-delay model. As in the proof of Theorem 3.9 we use the following notations. For a given sequence of events $\sigma_1 \sigma_2 \cdots \sigma_l$ and a buffer management strategy A, let $S_t^A$ denote the set of packets sent by A by the end of event $\sigma_t$, and let $B_t^A$ denote the set of packets stored in the buffer of A at the end of event $\sigma_t$. Initially, $S_0^A := \emptyset$ and $B_0^A := \emptyset$.

#### 3.2.1 Provisional Schedules

We introduce the basic concept of a provisional schedule. First, we define a canonical order $\prec$ on the packets. In contrast to the FIFO model, the canonical order is not primarily based on the arrival times of the packets but on their deadlines. We say $p$ comes before $q$ in the canonical order, that is, $p \prec q$,

- if $d(p) < d(q)$, or
- if $d(p) = d(q)$ and $v(p) > v(q)$, or
- if $d(p) = d(q)$, $v(p) = v(q)$, and the arrival event of $p$ is before the arrival event of $q$.

The last condition only ensures that ties are broken in some arbitrary but consistent way.

A provisional schedule $S$ for a set of pending packets $P$ specifies which packet should be sent in which time step. To simplify notation, a provisional schedule $S$ is sometimes regarded as a set of packets, e.g., we write $p \in S$ to indicate that the packet $p$ is scheduled in $S$. Let $S(p)$ denote the time step at which a packet $p \in S$ is scheduled in a provisional schedule $S$. Obviously, only one packet can be scheduled at each single time step and, for each $p \in S$, $S(p) \leq d(p)$. A provisional schedule $S$ is called a schedule for a time step $\tau$ if all packets in $S$ are scheduled after time step $\tau$, i.e., for each $p \in S$, $S(p) \geq \tau + 1$.

After each event $\sigma_t$, our strategies compute the optimal provisional schedule $S_t$ for the set of pending packets $B_t^{ONL}$ stored in the buffer at the end of $\sigma_t$ as follows: Start
with an empty set $S$. Consider the packets in $B^\text{ONL}$ for inclusion into $S$ in descending order of their value (ties are broken in favor of packets that come first in the canonical order). A packet $p$ is added to the set $S$ if

$$\left|\{p' \in S \cup \{p\} \mid d(p') \leq \tau'\}\right| \leq \tau' - \tau,$$

for each $\tau' \geq \tau$ with $\tau$ denoting the time step the event $\sigma_t$ belongs to.

The final set $S$ can be interpreted as the optimal provisional schedule $S_t$: Let $p_i \in S$ denote the $i$-th smallest packet in $S$ according to the canonical order. Then, $p_i$ can be scheduled for the time step $\tau + i$ since $d(p_i) \geq \tau + i$ due to

$$\left|\{p' \in S \mid d(p') \leq \tau'\}\right| \leq \tau' - \tau,$$

for each $\tau' \geq \tau$.

The optimality of this schedule is a simple consequence of the aforementioned fact that the set of feasible schedules for a set of packets $P$ is a matroid. Of course, the optimal provisional schedule $S_t$ is computed under the assumption that no new packets arrive in the future. Further, note that $S_t$ is a schedule for the time step $\tau$ if the event $\sigma_t$ belongs to $\tau$. This is important since we are not allowed to schedule packets for the past. Finally, observe that the packets are scheduled in $S_t$ in canonical order, that is, for each pair of packets $p, q \in S_t$, $S_t(p) < S_t(q)$ if $p < q$.

### 3.2.2 First Approach

In this section, we consider a first approach which is simple and natural. For each send event $\sigma_t$, define the first-packet $p_f \in S_{t-1}$ as the packet in $S_{t-1}$ that comes first in the canonical order and the max-packet $p_m := \arg\max_{p' \in S_{t-1}} v(p')$ (ties are broken in favor of packets that come first in the canonical order). There is a trade-off between sending the first-packet, which is a packet that expires next, and the max-packet, which may be more valuable than the first-packet but may also be available for a longer time. We study the natural approach to send either the first-packet or the max-packet, depending on the value of these two packets. This approach is very promising if only two different packet values are possible. However, this approach is disappointing for general packet values.

There are two natural greedy strategies: Either always send the first-packet or always send the max-packet. These two greedy strategies achieve a competitive ratio of $2$ [CY03, KLM+04]. The following natural strategy uses a parameter $\beta > 1$ and either sends the first-packet $p_f$ or the max-packet $p_m$, depending on the value of these two packets.

- If $v(p_f) \geq v(p_m)/\beta$, send the first-packet $p_f$.
- Otherwise, send the max-packet $p_m$.

Consider an input sequence with only two different packet values $1$ and $\alpha > 1$. Depending on the values of $\alpha$ and $\beta$, the above strategy either always sends the first-packet or always sends the max-packet. The following theorem states an upper bound on the competitive ratio of this strategy. For the case that the max-packet is always sent, the straightforward proof of the competitive ratio $1 + 1/\alpha$ can be found.
3.2 The Bounded-Delay Model

in [KLM+04]. For the case that the first-packet is always sent, the competitive ratio $2\alpha/(\alpha + 1)$ follows directly from the definition of the strategy.

**Theorem 3.12.** If only two packet values 1 and $\alpha > 1$ are possible, the above strategy achieves a competitive ratio of $\min\{1 + 1/\alpha, 2\alpha/(\alpha + 1)\} \leq \sqrt{2}$ for $\beta = \sqrt{2} + 1$.

The input sequences used in a proof of a lower bound on the competitive ratio for 2-bounded instances by Kesselman et al. [KLM+04] also yield a lower bound on the competitive ratio of any deterministic strategy matching the above upper bound.

**Theorem 3.13.** If only two packet values 1 and $\alpha > 1$ are possible, the competitive ratio of any deterministic strategy is at least $\min\{1 + 1/\alpha, 2\alpha/(\alpha + 1)\}$.

Even though, the first approach yields an optimal strategy if only two different packet values appear in the input sequence, the following theorem shows that it does not yield an advantage over the two greedy strategies in the general case.

**Theorem 3.14.** The competitive ratio of the above strategy is at least 2.

*Proof. Depending on $\beta$, we distinguish the following two cases.*

- Suppose that $\beta > 2$.

  The input sequence consists of $n + 1$ consecutive phases defined as follows.

  - Phase $1 \leq i \leq n$ consists of $2^n-i$ time steps. In the first time step of each phase, $2^n-i$ packets with value $2^{i-1}$ and deadline $2^n - 2^n-i+1 + 1, \ldots, 2^n - 2^n-i$, respectively, and $2^n-i$ packets with value $2^i$ and deadline $2^n - 2^n-i + 1, \ldots, 2^n$, respectively, arrive. In the remaining $2^n-i - 1$ time steps, new packets do not arrive.

  - Phase $n + 1$ consists of one time step. In this time step, one packet with value $2^n$ and deadline $2^n$ arrives.

  For this input sequence, the above strategy produces value $\sum_{i=1}^{n}(2^n-i \cdot 2^{i-1}) + 2^n$, and the optimal value is $\sum_{i=1}^{n}(2^n-i \cdot 2^i) + 2^n$. Hence, the competitive ratio is

  $$\lim_{n \to \infty} \frac{\sum_{i=1}^{n} 2^n + 2^n}{\sum_{i=1}^{n} 2^n-i + 2^n} = 2$$

- Suppose that $\beta \leq 2$.

  The input sequence is an extension of the previous one. It consists of $n + 1$ consecutive phases defined as follows.

  - Phase $1 \leq i \leq n$ consists of $2^n-i$ time steps. In the first time step of each phase, $2^n-i$ packets with value $2^{i-1}$ and deadline $2^n - 2^n-i+1 + 1, \ldots, 2^n - 2^n-i$, respectively, $2^n-i$ packets with value $2^i$ and deadline $2^n - 2^n-i+1 + 1, \ldots, 2^n$, respectively, and $2^n-i$ packets with value $(2+\varepsilon) \cdot 2^{i-1}$ arrive. In the remaining $2^n-i - 1$ time steps, new packets do not arrive.

  - Phase $n + 1$ consists of one time step. In this time step, one packet with value $2^n$ and deadline $2^n$ arrives.
$\tau + 1$ $\tau + 6$ $\tau + 11$

Figure 3.3: The optimal provisional schedule for a time step $\tau$ with 8 packets and additional dummy packets of value 0. The marked packet is scheduled on its deadline $\tau + 6$ and this is a tight time step. In addition, the set of levels $L = \{(\tau + 3, 5), (\tau + 5, 2), (\tau + 6, 3), (\tau + 12, 1)\}$ and $\delta_L^{\max}$ are depicted. In the second step of our strategy, the level $(\tau + 6, 3)$ would be added to the set of levels for the tight time step $\tau + 6$. Since the level $(\tau + 5, 2)$ is dominated by the level $(\tau + 6, 3)$, it does not need to be retained.

For this input sequence, the above strategy produces value $\sum_{i=1}^{n}(2^{n-i} \cdot (2 + \varepsilon) \cdot 2^{i-1}) + 2^n$, and the optimal value is $\sum_{i=1}^{n}(2^{n-i} \cdot (4 + \varepsilon) \cdot 2^{i-1}) + 2^n$. Hence, the competitive ratio is

$$\lim_{n \to \infty} \lim_{\varepsilon \to 0} \frac{\sum_{i=1}^{n}(2^{n-i} \cdot (4 + \varepsilon) \cdot 2^{i-1}) + 2^n}{\sum_{i=1}^{n}(2^{n-i} \cdot (2 + \varepsilon) \cdot 2^{i-1}) + 2^n} = \lim_{n \to \infty} \frac{\sum_{i=1}^{n}2^{n+1} + 2^n}{\sum_{i=1}^{n}2^n + 2^n} = 2.$$  

This concludes the proof of the theorem.

### 3.2.3 Suppressed Packets and Our Strategies

We enhance the natural approach to send either the first-packet or the max-packet by introducing the concept of suppressed packets. Consider the optimal provisional schedule $S$ for a set of pending packets $P$. Suppose that a packet $q \in P$ does not appear in $S$, but it can be added to $S$ if another packet $p \in S$ is removed from $S$. Then, $q$ is called *suppressed* by $p$.

More precisely, consider the optimal provisional schedule $S_t$ at the end of event $\sigma_t$ for the set of pending packet $B_t^{\text{ONL}}$. For each $p \in S_t$, let $S_t^p$ denote the optimal provisional schedule at the end of event $\sigma_t$ for the set of pending packets without $p$, $B_t^{\text{ONL}} \setminus \{p\}$. If $S_t^p \setminus S_t \neq \emptyset$, let $s_t(p) := S_t^p \setminus S_t$ denote the packet that is suppressed by $p \in S_t$. Note that $s_t(p)$ is well defined since $|S_t^p \setminus S_t| \leq 1$ which follows from the fact that the set of feasible schedules is a matroid. For simplicity, if $S_t^p \setminus S_t = \emptyset$, let $s_t(p)$ be a dummy packet with value 0 and an infinite deadline.

Before we introduce our algorithm, we need some additional preliminaries. Consider the optimal provisional schedule $S$ for a time step $\tau$. A time step $\tau' > \tau$ is called a *tight time step* in $S$ if

$$|\{p' \in S \mid d(p') \leq \tau'\}| = \tau' - \tau.$$

Roughly speaking, a tight time step is a time step that prevents further packets with an earlier deadline from being added to the schedule. Another characterization is
the following. A tight time step is a time step in which a packet is scheduled on its
deadline, i.e., \( \tau' \) is a tight time step in \( S \) if and only if a packet \( p \in S \) exists with
\( S(p) = d(p) = \tau' \). Let \( p' \) be a suppressed packet and \( \tau' \geq d(p') \) be the earliest tight
time step after the deadline of \( p' \). Then, \( v(p') \leq v(p) \) for each packet \( p \in S \) with
\( S(p) \leq \tau' \).

For a time step \( \tau \) and a packet value \( \delta \), \((\tau, \delta)\) is called a level. For a set of levels \( L \)
and a time step \( \tau \), let \( \delta_{\max}(\tau) \) denote the value of the level in \( L \) with maximum value
that contains time step \( \tau \), i.e.,
\[
\delta_{\max}(\tau) := \max\{\delta' \mid (\tau', \delta') \in L, \tau \leq \tau'\}.
\]
If \( \{\delta' \mid (\tau', \delta') \in L, \tau \leq \tau'\} = \emptyset \), define \( \delta_{\max}(\tau) := 0 \). The function
\( \delta_{\max} \) describes the upper envelope of all levels in \( L \). Figure 3.3 depicts an optimal provisional schedule
for a time step \( \tau \) including a tight time step. In addition, a set of levels \( L \) and \( \delta_{\max} \) are depicted.

Our strategy uses a parameter \( \beta > 1 \). For each event \( \sigma_t \), a set of levels \( L_t \) is
defined. Initially, define \( L_0 := \emptyset \). For each event \( \sigma_t \), our strategy does the following.

1. If \( \sigma_t \) is the send event of a time step \( \tau \):
   Define \( p_f \in S_{t-1} \) as the first packet in \( S_{t-1} \) according to the canonical order and
   \[
p_m := \arg\max_{p' \in S_{t-1}} (v(p') + (\beta - 1) \cdot v(s_{t-1}(p')))\]
   (ties are broken in favor of packets that come first in the canonical order).
   If
   \[
   \max\{v(p_f), \delta_{\max}^{L_{t-1}}(\tau)\} \geq \frac{v(p_m) + (\beta - 1) \cdot v(s_{t-1}(p_m))}{\beta},
   \]
   send \( p_f \). Otherwise, send \( p_m \).

2. After event \( \sigma_t \), i.e., after a packet has been sent or has arrived:
   Compute \( S_t \) and set
   \[
   L_t := L_{t-1} \cup \{\{\tau', \min\{v(p) \mid p \in S_t, d(p) \leq \tau'\}\} \mid \tau' \text{ is a tight time step in } S_t\}.
   \]
   Note that this strategy does not have to compute the optimal provisional schedule
completely new at each event. Instead, it suffices to remove and to insert the respective
packets. Further, note that this strategy does not have to accumulate all levels. Instead,
it suffices to retain only the values of \( \delta_{\max}^{L_t} \) for future time steps.

Our strategy achieves the best known competitive ratio in the deterministic case.

**Theorem 3.15.** The above strategy achieves the competitive ratio \( r := 2\sqrt{2} - 1 \approx 1.8284 \) for \( \beta := 1 + 1/\sqrt{2} \).

The proof of this theorem follows in the next section.

The above strategy can easily be transformed into the following memoryless strategy
which does not have to store \( \delta_{\max}^{L_t} \). For each event \( \sigma_t \), our memoryless strategy does the following.
1. If $\sigma_t$ is the send event of a time step $\tau$:

   Define $p_f \in S_{t-1}$ as the first packet in $S_{t-1}$ according to the canonical order and
   
   $$p_m := \arg\max_{p' \in S_{t-1}} (v(p') + v(s_{t-1}(p')))/2$$
   
   (ties are broken in favor of packets that come first in the canonical order).

   If
   
   $$v(p_f) \geq \frac{v(p_m) + v(s_{t-1}(p_m))}{\beta}$$
   
   send $p_f$. Otherwise, send $p_m$.

2. After event $\sigma_t$, i.e., after a packet has been sent or has arrived:

   Compute $S_t$.

This strategy is the first memoryless strategy which achieves a competitive ratio below 2.

**Theorem 3.16.** The memoryless strategy achieves the competitive ratio $r := (2\beta^2 + \beta - 5)/2 \approx 1.893$ for $\beta := 4 \cos((\pi - \arccos(3\sqrt{3}/16))/3)/\sqrt{3}$ ($\beta$ is the largest real root of $X^3 - 4X + 1$).

The proof of this theorem follows in the next section. It answers the question posed by Chrobak et al. [CJST07] whether it is possible for a memoryless strategy to achieve a competitive ratio below 2.

We first give the proof of Theorem 3.15. Thereafter it is explained how this proof can be adapted to show Theorem 3.16.

**Proof of Theorem 3.15** Let OPT denote an optimal offline strategy, and let ONL denote our online strategy. We assume without loss of generality that the sequence of packets $p_1 p_2 \cdots$ sent by OPT is in canonical order, i.e., for each $i < j$, either $p_i < p_j$ or $p_i$ is sent before $p_j$ arrives. Note that each sequence of sent packets can easily be converted into canonical order by rearranging its packets.

For simplicity, we assume that ONL and OPT both send a packet in each send event. If the buffer of one of these strategies is empty at a send event, we suppose that a dummy packet of value 0 is sent. Further, we assume that there is a packet scheduled for every time step in the optimal provisional schedule. This can be achieved by assuming that the schedule is filled up with dummy packets with an infinite deadline and value 0 (see Figure 3.3). The packets $p_f$ and $p_m$ are newly defined for each send event. Nevertheless, we refer to $p_f$ and $p_m$ without explicitly referencing a send event. It is always obvious from the context which send event is meant.

Our proof is based on a potential function argument. In the following, we give some basic ideas. If we could show, for each time step, that the value of the packet sent by OPT is at most $r$ times larger than the packet sent by ONL in this step, the theorem would follow immediately. Of course this is not true: Time steps can exist such that the value of the packet sent by OPT is much larger than the value of the packet sent by ONL in this step. There are two basic scenarios.
3.2 The Bounded-Delay Model

In the first scenario, OPT sends a packet $p$ that was not sent by ONL yet, that is, $p$ is still stored in the buffer of ONL. In this case, value is lent on $p$, i.e., a $(r - 1)/2$ fraction of the value of $p$ is allocated to this time step. This lent value has to be amortized by the time step in which $p$ leaves the buffer of ONL (either because it is sent or because its deadline is reached). We can only amortize the lent value of one packet in each time step. Hence, we maintain, for each time step, the invariant that all packets on which value is lent can be scheduled in a feasible schedule (see Lemma 3.19). This guarantees that the deadline of at most one packet on which value is lent expires in every time step.

In the second scenario, OPT sends a packet $p$ that was sent by ONL in a previous time step. In this case, we allocate value that is amortized in previous time steps as follows. For a certain level $(\tau, \delta)$, ONL provides an increase of the total value of sent packets by at least $\delta$ in each time step less or equal to $\tau$. If in one of these time steps the actual value of the sent packet is less than $\delta$, ONL can nevertheless guarantee the claimed increase by amortization (see the $V(L_t, S_t)$ term in the potential function which is defined later). Hence, ONL can guarantee the value $\delta_{L_t}^{\max}(d(p))$ at the send event $\sigma_t$ when OPT sends $p$. It remains to amortize the value $v(p) - \delta_{L_t}^{\max}(d(p))$ in previous time steps. This is the task of the $A(L_t, B^\text{OPT}_t \setminus B^\text{ONL}_t)$ term in the potential function which is defined later.

In the following, these basic ideas are formalized. To shorten notation, define, for a set of levels $L$ and a packet $p$,

$$m_L(p) := \min\{v(p), \delta_{L_t}^{\max}(d(p))\} .$$

Then, for a set of levels $L$ and a set of packets $P$, define

$$A(L, P) := \sum_{p \in P} (v(p) - m_L(p)) .$$

The following observation states an upper bound on

$$A(L_t, B^\text{OPT}_t \setminus B^\text{ONL}_t) - A(L_{t-1}, B^\text{OPT}_{t-1} \setminus B^\text{ONL}_{t-1}) .$$

**Observation 3.17.** Fix an event $\sigma_t$ and define

$$\Delta A := A(L_t, B^\text{OPT}_t \setminus B^\text{ONL}_t) - A(L_{t-1}, B^\text{OPT}_{t-1} \setminus B^\text{ONL}_{t-1}) .$$

- Suppose that $\sigma_t$ is an arrival event. Then
  $$\Delta A \leq 0 .$$
- Suppose that $\sigma_t$ is a send event in which ONL sends the packet $p$ and OPT sends the packet $q$. Then

$$\Delta A \leq \begin{cases} 
  v(p) - m_{L_{t-1}}(p) & \text{if } q \notin B^\text{ONL}_{t-1} \text{ and } p \in B^\text{OPT}_t, \\
  - (v(q) - m_{L_{t-1}}(q)) & \text{if } q \in B^\text{ONL}_{t-1} \text{ and } p \in B^\text{OPT}_t, \\
  v(p) - m_{L_{t-1}}(p) & \text{if } q \notin B^\text{ONL}_{t-1} \text{ and } p \notin B^\text{OPT}_t, \\
  - (v(q) - m_{L_{t-1}}(q)) & \text{if } q \in B^\text{ONL}_{t-1} \text{ and } p \notin B^\text{OPT}_t, \\
  0 & \text{if } q \notin B^\text{ONL}_{t-1} \text{ and } p \notin B^\text{OPT}_t.
\end{cases}$$
Proof. For a set of levels $L$, a set of packets $P$, and a level $(\tau', \delta')$, obviously $A(L \cup (\tau', \delta'), P) \leq A(L, P)$. The successive application of this argument yields

$$A(L_t, B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}}) \leq A(L_{t-1}, B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}}) \ . \tag{3.5}$$

Suppose that $\sigma_1$ is an arrival event. Then, $B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}} = B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}}$. Hence, $A(L_{t-1}, B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}}) = A(L_{t-1}, B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}})$. Together with Inequality (3.5), this yields the first statement of the observation.

Suppose that $\sigma_1$ is a send event in which ONL sends the packet $p$ and OPT sends the packet $q$. Then, $p \in B_{t-1}^{\text{ONL}} \setminus B_{t-1}^{\text{OPT}}$ and $q \in B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}}$.

$$B_{t}^{\text{OPT}} \setminus B_{t}^{\text{ONL}} = \begin{cases} 
\{p\} \cup (B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}}) \setminus \{q\} & \text{if } q \not\in B_{t-1}^{\text{ONL}} \text{ and } p \in B_{t-1}^{\text{OPT}}, \\
\{p\} \cup (B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}}) & \text{if } q \in B_{t-1}^{\text{ONL}} \text{ and } p \in B_{t-1}^{\text{OPT}}, \\
(B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}}) \setminus \{q\} & \text{if } q \not\in B_{t-1}^{\text{ONL}} \text{ and } p \not\in B_{t-1}^{\text{OPT}}, \\
(B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}}) & \text{if } q \in B_{t-1}^{\text{ONL}} \text{ and } p \not\in B_{t-1}^{\text{OPT}}.
\end{cases}$$

Together with Inequality (3.5), this yields the second statement of the observation. \(\square\)

For a set of levels $L$ and the optimal provisional schedule $S$ for a time step $\tau$, define

$$V(L, S) := \sum_{p \in S} (\delta_L^{\text{max}}(S(p)) - m_L(p)) \ .$$

Note that $\sum_{p \in S} \delta_L^{\text{max}}(S(p)) = \sum_{\tau' \geq \tau+1} \delta_L^{\text{max}}(\tau')$ since in every time step from time step $\tau + 1$ onwards there is a packet scheduled in $S$. The following observation states an upper bound on $V(L_t, S_t) - V(L_{t-1}, S_{t-1})$. The proof of this observation is similar to the proof of Observation 3.17.

**Observation 3.18.** Fix an event $\sigma_1$ in a time step $\tau$ and define

$$\Delta V := V(L_t, S_t) - V(L_{t-1}, S_{t-1}) \ .$$

- If $\sigma_1$ is an arrival event,
  $$\Delta V \leq 0 \ .$$

- If $\sigma_1$ is a send event in which ONL sends $p_f$,
  $$\Delta V \leq m_{L_{t-1}}(p_f) - \delta_{L_{t-1}}^{\text{max}}(\tau) \ .$$

- If $\sigma_1$ is a send event in which ONL sends $p_m$,
  $$\Delta V \leq m_{L_{t-1}}(p_m) - v(s_{t-1}(p_m)) + m_{L_{t-1}}(p_f) - \delta_{L_{t-1}}^{\text{max}}(\tau) \ .$$

**Proof.** As in Observation 3.17, we first show

$$V(L_t, S_t) \leq V(L_{t-1}, S_t) \ . \tag{3.6}$$

In order to show this we prove, for a set of levels $L$, an optimal provisional schedule $S$, and a level $(\tau', \delta')$, where $\tau'$ is a tight time step in $S$ and $\delta' \leq v(p)$ for every packet $p \in S$ with $d(p) \leq \tau'$, that

$$V(L \cup (\tau', \delta'), S) \leq V(L, S) \ .$$
The successive application of this argument yields Inequality (3.6).

The inequality $V(L \cup (\tau', \delta'), S) \leq V(L, S)$ follows immediately if, for each $p \in S$,

$$
\delta_{L \cup (\tau', \delta')}^\text{max}(S(p)) - m_{L \cup (\tau', \delta')}^L(p) \leq \delta_{L \cup (\tau', \delta')}^\text{max}(S(p)) - m_L(p) \ .
$$

(3.7)

Obviously, for each $p \in S$, $m_{L \cup (\tau', \delta')}^L(p) \geq m_L(p)$. Hence, Inequality (3.7) is true if, for each $p \in S$, $\delta_{L \cup (\tau', \delta')}^\text{max}(S(p)) \leq \delta_{L \cup (\tau', \delta')}^\text{max}(S(p))$.

Suppose that $p \in S$ exists with $\delta_{L \cup (\tau', \delta')}^\text{max}(S(p)) > \delta_{L \cup (\tau', \delta')}^\text{max}(S(p))$. Then,

$$
\delta_{L \cup (\tau', \delta')}^\text{max}(S(p)) = \delta' \quad \text{and} \quad S(p) \leq \tau' \ .
$$

Hence, $d(p) \leq \tau'$ since $\tau'$ is a tight time step in $S$. This implies $\delta_{L \cup (\tau', \delta')}^\text{max}(d(p)) \geq \delta'$. Then, $\min\{v(p)\, \delta_{L \cup (\tau', \delta')}^\text{max}(d(p))\} \geq \delta'$ since $v(p) \geq \delta'$ due to the definition of $\delta'$. As a consequence,

$$
\delta_{L \cup (\tau', \delta')}^\text{max}(S(p)) - m_{L \cup (\tau', \delta')}^L(p) = \delta' - \min\{v(p)\, \delta_{L \cup (\tau', \delta')}^\text{max}(d(p))\} \leq \delta' - \delta' = 0 \ .
$$

Further, $\delta_{L \cup (\tau', \delta')}^\text{max}(S(p)) - m_L(p) \geq 0$ since $\delta_{L \text{max}}(S(p)) \geq \delta_{L \text{max}}(d(p))$. Altogether, this yields Inequality (3.7).

In the following, we show the three statements of the observation. Fix an event $\sigma_t$ in a time step $\tau$.

- **Suppose that $\sigma_t$ is an arrival event in which a packet $p$ arrives.**

  Due to Inequality (3.6), it remains to show that $V(L_{t-1}, S_t) \leq V(L_{t-1}, S_{t-1})$.

  Obviously,

  $$
  \sum_{p' \in S_t} \delta_{L_{t-1}}^\text{max}(S(p')) = \sum_{\tau' \geq \tau + 1} \delta_{L_{t-1}}^\text{max}(\tau') = \sum_{p' \in S_{t-1}} \delta_{L_{t-1}}^\text{max}(S(p')) \ .
  $$

  Hence, it remains to show that

  $$
  \sum_{p' \in S_{t-1} \setminus S_t} m_{L_{t-1}}(p) \geq \sum_{p' \in S_{t-1} \setminus S_t} m_{L_{t-1}}(p) \ .
  $$

  Since the set of feasible schedules is a matroid, only the following three possibilities exist for $S_t$: $S_t = S_{t-1}$, $S_t = \{p\} \cup S_{t-1}$, or $S_t = \{p\} \cup S_{t-1} \setminus \{s_t(p)\}$. The above inequality follows immediately in the first and second case. In the third case, we have to show that

  $$
  \min\{v(p), \delta_{L_{t-1}}^\text{max}(d(p))\} \geq \min\{v(s_t(p)), \delta_{L_{t-1}}^\text{max}(d(s_t(p)))\} \ .
  $$

  Obviously, $v(p) \geq v(s_t(p))$. Further, a tight time step $\tau' \geq d(p)$ exists in $S_{t-1}$ such that each packet in $S_{t-1}$ with a deadline smaller or equal to $\tau'$ has a value of at least $v(s_t(p))$. Hence, a level $(\tau', \delta')$ with $\delta' \geq v(s_t(p))$ exists in $L_{t-1}$. As a consequence, $\delta_{L_{t-1}}^\text{max}(d(p)) \geq v(s_t(p))$.

- **Suppose that $\sigma_t$ is a send event in which ONL sends $p_f$.**

  Note that $S(p_f) = \tau$. Due to Inequality (3.6), it remains to show that

  $$
  V(L_{t-1}, S_t) \leq V(L_{t-1}, S_{t-1}) + m_{L_{t-1}}(p_f) - \delta_{L_{t-1}}^\text{max}(\tau) \ .
  $$

  61
Obviously, $S_t = S_{t-1} \setminus \{p_f\}$ and, for each $p \in S_t$, $S_t(p) = S_{t-1}(p)$. Hence,

$$\sum_{p \in S_t} \left( \delta_{L_{t-1}}^{\max}(S_t(p)) - m_{L_{t-1}}(p) \right)$$

$$= \sum_{p \in S_{t-1}} \left( \delta_{L_{t-1}}^{\max}(S_{t-1}(p)) - m_{L_{t-1}}(p) \right) - \left( \delta_{L_{t-1}}^{\max}(\tau) - m_{L_{t-1}}(p_f) \right).$$

- Suppose that $\sigma$ is a send event in which ONL sends $p_m$.

Note that $S(p_m) = \tau$. Due to Inequality (3.6), it remains to show that

$$V(L_{t-1}, S_t) \leq V(L_{t-1}, S_{t-1}) + m_{L_{t-1}}(p_m) - v(s_{t-1}(p_m))$$

$$+ m_{L_{t-1}}(p_f) - \delta_{L_{t-1}}^{\max}(\tau).$$

Obviously,

$$\sum_{p \in S_{t-1}} \delta_{L_{t-1}}^{\max}(S_{t-1}(p)) = \delta_{L_{t-1}}^{\max}(\tau) + \sum_{p \in S_t} \delta_{L_{t-1}}^{\max}(S_t(p)).$$

Hence, it remains to show that

$$m_{L_{t-1}}(p_f) + m_{L_{t-1}}(p_m) + \sum_{p \in S_t} m_{L_{t-1}}(p) \geq v(s_{t-1}(p_m)) + \sum_{p \in S_{t-1}} m_{L_{t-1}}(p). \tag{3.8}$$

Only the following two possibilities exist for $S_t$: $S_t = S_{t-1} \setminus \{p_m\}$ or $S_t = \{s_{t-1}(p_m)\} \cup S_{t-1} \setminus \{p_m, f\}$, where $f$ is the packet in $S_{t-1}$ with minimum value and a deadline smaller or equal to the first tight time step in $S_{t-1}$ (ties are broken in favor of packets that come later in the canonical order).

- Suppose that $S_t = S_{t-1} \setminus \{p_m\}$.

A tight time step $\tau'$ exists in $S_{t-1}$ that prevents $s_{t-1}(p_m)$ from being scheduled in $S_{t-1}$. The value of each packet in $S_{t-1}$ with a deadline smaller or equal to $\tau'$ is at least $v(s_{t-1}(p_m))$. Hence, a level $(\tau', \delta')$ with $\delta' \geq v(s_{t-1}(p_m))$ exists in $L_{t-1}$. This implies that $\delta_{L_{t-1}}^{\max}(d(p_f)) \geq v(s_{t-1}(p_m))$.

Then,

$$\min\{v(p_f), \delta_{L_{t-1}}^{\max}(d(p_f))\} \geq v(s_{t-1}(p_m))$$

since $v(p_f) \geq v(s_{t-1}(p_m))$. This yields Inequality (3.8).

- Suppose that $S_t = \{s_{t-1}(p_m)\} \cup S_{t-1} \setminus \{p_m, f\}$.

Due to the definition of $f$, $v(p_f) \geq v(f)$ and a level $(\tau', \delta')$ with $\tau' \geq d(f)$ and $\delta' \geq v(f)$ exists in $L_{t-1}$. Hence,

$$\min\{v(p_f), \delta_{L_{t-1}}^{\max}(d(p_f))\} \geq \min\{v(f), \delta_{L_{t-1}}^{\max}(d(f))\}. \tag{3.8}$$

Further, a tight time step $\tau'' \geq d(s_{t-1}(p_m))$ exists in $S_{t-1}$ that prevents $s_{t-1}(p_m)$ from being scheduled in $S_{t-1}$. Hence, a level $(\tau'', \delta'')$ with $\delta'' \geq v(s_{t-1}(p_m))$ exists in $L_{t-1}$. This implies that

$$\min\{v(s_{t-1}(p_m)), \delta_{L_{t-1}}^{\max}(d(s_{t-1}(p_m)))\} = v(s_{t-1}(p_m)).$$

Altogether, this yields Inequality (3.8).
This concludes the proof of the observation.

For each event $\sigma_t$, define the potential function

$$\Phi_t := r \sum_{p \in S^\text{ONL}_t} v(p) - \sum_{p \in S^\text{OPT}_t} v(p) - A(L_t, B^\text{OPT}_t \setminus B^\text{ONL}_t) - V(L_t, S_t) + \frac{r - 1}{2} \sum_{p \in C_t} v(p) \ ,$$

where $C_t \subseteq B^\text{ONL}_t \setminus B^\text{OPT}_t$ is specified later (see Lemma 3.19). Initially, define $C_0 := \emptyset$. In order to prove the theorem, we show that the potential function $\Phi_t$ is monotonically increasing in $t$, for appropriately chosen sets of packets $C_t \subseteq B^\text{ONL}_t \setminus B^\text{OPT}_t$.

Obviously, $\Phi_0 = 0$ since $S^\text{ONL}_0 = S^\text{OPT}_0 = B^\text{OPT}_0 = B^\text{ONL}_0 = L_0 = C_0 = \emptyset$ by definition. Then, if the potential function $\Phi_t$ is monotonically increasing in $t$, $\Phi_t \geq 0$, where $\sigma_t$ is the last event. As a consequence, $r \sum_{p \in S^\text{ONL}_t} v(p) \geq \sum_{p \in S^\text{OPT}_t} v(p)$ since $A(L_t, B^\text{OPT}_t \setminus B^\text{ONL}_t) \geq 0$ and $V(L_t, S_t) \geq 0$, for each event $\sigma_t$, and $C_t \subseteq B^\text{ONL}_t \setminus B^\text{OPT}_t = \emptyset$. This yields the theorem.

It remains to show that the potential function $\Phi_t$ is monotonically increasing in $t$, for appropriately chosen sets of packets $C_t \subseteq B^\text{ONL}_t \setminus B^\text{OPT}_t$. The following lemma states that it is possible to choose, for each event $\sigma_t$, $C_t \subseteq B^\text{ONL}_t \setminus B^\text{OPT}_t$ such that certain lower bounds on $\sum_{p' \in C_t} v(p') - \sum_{p' \in C_{t-1}} v(p')$ hold.

**Lemma 3.19.** Define

$$\Delta C_t := \sum_{p' \in C_t} v(p') - \sum_{p' \in C_{t-1}} v(p') \ .$$

For each event $\sigma_t$, the set of packets $C_t \subseteq B^\text{ONL}_t \setminus B^\text{OPT}_t$ can be chosen such that the following is true.

(a) If $\sigma_t$ is a send event in which ONL and OPT send the same packet $p$,

$$\Delta C_t \geq -v(p_f) \ .$$

(b) If $\sigma_t$ is a send event in which ONL sends a packet $p \in C_{t-1}$ and OPT sends a packet $q \not\in S_{t-1}$,

$$\Delta C_t \geq -v(p) - v(p_f) \ .$$

(c) If $\sigma_t$ is a send event in which ONL sends a packet $p \not\in C_{t-1}$ and OPT sends a packet $q \not\in S_{t-1}$,

$$\Delta C_t \geq -v(p_f) \ .$$

(d) If $\sigma_t$ is a send event in which ONL sends $p_f$ and OPT sends a packet $q \in S_{t-1} \setminus \{p_f\}$,

$$\Delta C_t \geq -2 \cdot v(p_f) + v(q) \ .$$

(e) If $\sigma_t$ is a send event in which ONL sends $p_m \not\in C_{t-1}$ and OPT sends a packet $q \in S_{t-1} \setminus \{p_m\}$,

$$\Delta C_t \geq -v(p_f) - v(s_{t-1}(q)) + v(q) \ .$$
In order to show the lemma, we maintain, for each event

\[ \sigma, \]

the statements in the lemma hold. Suppose that the invariant is true for

\[ C \]

appropriately choose the set of packets \( C \) deadline of at most one packet in \( \tau \) at most one packet with a deadline less or equal to \( C \). i.e., we can build a feasible schedule containing all packets in \( C \).

Obviously, the invariant is true for \( C_0 = \emptyset \). By induction over \( t \), we show how to appropriately choose the set of packets \( C_t \subseteq B_t^{ONL} \setminus B_t^{OPT} \) such that the invariant and the statements in the lemma hold. Suppose that the invariant is true for \( \sigma_{t-1} \), i.e.,

\[ \forall \tau' \geq (\tau - 1) : |\{ p' \in C_{t-1} | d(p') \leq \tau' \}| \leq \tau' - (\tau - 1) . \]

Let \( \sigma_t \) be a send event in time step \( \tau \). According to statements in the lemma, we distinguish the following cases.

(a) Suppose that ONL and OPT send the same packet \( p \) in \( \sigma_t \).

Obviously, \( p \notin C_{t-1} \) since \( p \in B_{t-1}^{OPT} \). Hence, \( C_{t-1} \subseteq B_t^{ONL} \setminus B_t^{OPT} \).

If no packet \( f \in C_{t-1} \) with \( v(f) \leq v(p_f) \) exists, set \( C_t := C_{t-1} \). Then, for each \( \tau' \geq d(p_f) \),

\[
|\{ p' \in C_t | d(p') \leq \tau' \}| = |\{ p' \in C_{t-1} | d(p') \leq \tau' \}|
\leq |\{ p' \in S_{t-1} \setminus \{ p_f \} | d(p') \leq \tau' \}|
= |\{ p' \in S_{t-1} \setminus \{ p_f \} | d(p') \leq \tau' \}| - 1
\leq \tau' - (\tau - 1) - 1
\]

since \( C_{t-1} \subseteq S_{t-1} \setminus \{ p_f \} \). On the other hand, for each \( \tau \leq \tau' < d(p_f) \),

\[
|\{ p' \in C_t | d(p') \leq \tau' \}| = |\{ p' \in C_{t-1} | d(p') \leq \tau' \}|
\leq |\{ p' \in S_{t-1} \setminus \{ p_f \} | d(p') \leq \tau' \}|
= 0 \leq \tau' - (\tau - 1) - 1 .
\]

Otherwise, if there is a packet in \( C_{t-1} \) with value at most \( v(p_f) \), set \( C_t := C_{t-1} \setminus \{ f \} \), where \( f \) denotes the first packet in \( C_{t-1} \) according to the canonical order with \( v(f) \leq v(p_f) \). Then, for each \( \tau' \geq d(f) \),

\[
|\{ p' \in C_t | d(p') \leq \tau' \}| = |\{ p' \in C_{t-1} | d(p') \leq \tau' \}| - 1
\leq \tau' - (\tau - 1) - 1 ,
\]
for each $d(p_f) \leq \tau' < d(f)$,
\[
\left|\{p' \in C_t \mid d(p') \leq \tau'\}\right| = \left|\{p' \in C_{t-1} \mid d(p') \leq \tau'\}\right|
\leq \left|\{p' \in S_{t-1} \setminus \{p_f\} \mid d(p') \leq \tau'\}\right|
= \left|\{p' \in S_{t-1} \mid d(p') \leq \tau'\}\right| - 1
\leq \tau' - (\tau - 1) - 1,
\]
and, for each $\tau \leq \tau' < d(p_f)$,
\[
\left|\{p' \in C_t \mid d(p') \leq \tau'\}\right| = \left|\{p' \in C_{t-1} \mid d(p') \leq \tau'\}\right|
\leq \left|\{p' \in S_{t-1} \setminus \{p_f\} \mid d(p') \leq \tau'\}\right|
= 0 \leq \tau' - (\tau - 1) - 1.
\]

In both cases, $\sum_{p' \in C_t} v(p') - \sum_{p' \in C_{t-1}} v(p') \geq -v(p_f)$.

(b) Suppose that ONL sends a packet $p \in C_{t-1}$ and OPT sends a packet $q \not\in S_{t-1}$ in $\sigma_t$.

This case follows analogously to case (a) except that $C_t$ is set to $C_{t-1} \setminus \{p\}$ or $C_{t-1} \setminus \{f, p\}$ instead of $C_{t-1}$ or $C_{t-1} \setminus \{f\}$.

(c) Suppose that ONL sends a packet $p \not\in C_t$ and OPT sends a packet $q \not\in S_t$ in $\sigma_t$.

This case follows analogously to case (a).

(d) Suppose that ONL sends $p_f$ and OPT sends a packet $q \in S_{t-1} \setminus \{p_f\}$ in $\sigma_t$.

We distinguish the following two cases.

- Suppose that $p_f \in C_{t-1}$.
  
  If $C_{t-1} \subseteq S_{t-1}$, set $C_t := \{q\} \cup C_{t-1} \setminus \{p_f\}$. Then, for each $\tau' \geq \tau$,
  \[
  \left|\{p' \in C_t \mid d(p') \leq \tau'\}\right| = \left|\{p' \in \{q\} \cup C_{t-1} \setminus \{p_f\} \mid d(p') \leq \tau'\}\right|
  \leq \left|\{p' \in S_{t-1} \setminus \{p_f\} \mid d(p') \leq \tau'\}\right|
  = \max\{0, \left|\{p' \in S_{t-1} \mid d(p') \leq \tau'\}\right| - 1\}
  \leq \tau' - (\tau - 1) - 1,
  \]
  since $\{q\} \cup C_{t-1} \subseteq S_{t-1}$.
  
  Otherwise, set $C_t := \{q\} \cup C_{t-1} \setminus \{f, p_f\}$, where $f$ denotes the first packet in $C_{t-1} \setminus S_{t-1}$ according to the canonical order. Note that $v(f) \leq v(p_f)$.
  Then, for each $\tau' \geq d(f)$,
  \[
  \left|\{p' \in C_t \mid d(p') \leq \tau'\}\right| \leq \left|\{p' \in C_{t-1} \mid d(p') \leq \tau'\}\right| - 1
  \leq \tau' - (\tau - 1) - 1
  \]
  since $d(p_f) \leq d(q)$, and, for each $\tau \leq \tau' < d(f)$,
  \[
  \left|\{p' \in C_t \mid d(p') \leq \tau'\}\right| = \left|\{p' \in \{q\} \cup C_{t-1} \setminus \{p_f\} \mid d(p') \leq \tau'\}\right|
  \leq \left|\{p' \in S_{t-1} \setminus \{p_f\} \mid d(p') \leq \tau'\}\right|
  = \max\{0, \left|\{p' \in S_{t-1} \mid d(p') \leq \tau'\}\right| - 1\}
  \leq \tau' - (\tau - 1) - 1.
  \]
In both cases, $\sum_{p' \in C_t} v(p') - \sum_{p' \in C_{t-1}} v(p') \geq -v(p_f) - v(f) + v(q) \geq -2v(p_f) + v(q)$.

- Suppose that $p_f \notin C_t$.

  If $C_{t-1} \subseteq S_{t-1}$, set $C_t := \{q\} \cup C_{t-1}$. If $|C_{t-1} \setminus S_{t-1}| = 1$, set $C_t := \{q\} \cup C_{t-1} \setminus (C_{t-1} \setminus S_{t-1})$. The invariant follows analogously to the previous case.

  Otherwise, set $C_t := \{q\} \cup C_{t-1} \setminus \{f_1, f_2\}$, where $f_1$ and $f_2$ denote the first two packets in $C_{t-1} \setminus S_{t-1}$ according to the canonical order with $f_1 < f_2$. Then, for each $\tau' \geq d(f_2)$,

  $$|\{p' \in C_t \mid d(p') \leq \tau'\}| = |\{p' \in \{q\} \cup C_{t-1} \mid d(p') \leq \tau'\}| - 2 \leq |\{p' \in C_{t-1} \mid d(p') \leq \tau'\}| - 1 \leq \tau' - (\tau - 1) - 1,$$

  and, for each $\tau \leq \tau' < d(f_2)$,

  $$|\{p' \in C_t \mid d(p') \leq \tau'\}| = |\{p' \in \{q\} \cup C_{t-1} \setminus \{f_1\} \mid d(p') \leq \tau'\}| \leq |\{p' \in S_{t-1} \setminus \{f_1, f_2\} \mid d(p') \leq \tau'\}| \leq \max\{0, |\{p' \in S_{t-1} \mid d(p') \leq \tau'\}| - 1\} \leq \tau' - (\tau - 1) - 1.$$

  In all cases, $\sum_{p' \in C_t} v(p') - \sum_{p' \in C_{t-1}} v(p') \geq -v(f_1) - v(f_2) + v(q) \geq -2v(p_f) + v(q)$.

- Suppose that ONL sends $p_m \notin C_t$ and OPT sends a packet $q \in S_t \setminus \{p_m\}$ in $\sigma_t$.

Let $\tau_1$ denote the latest tight time step in $S_{t-1}$ with $\tau_1 < d(q)$. If no such time step exists, define $\tau_1 := 0$. Let $p_1 \in C_{t-1} \setminus S_{t-1}$ denote a packet with the earliest deadline greater than $\tau_1$. If no such packet exists, let $p_1$ denote a dummy packet with value 0 and an infinite deadline. Note that $v(p_1) \leq v(s_{t-1}(q))$ since $p_1$ can be added to the schedule $S_{t-1}$ if $q$ is removed.

Define $C' := \{q\} \cup C_{t-1} \setminus \{p_1\}$. In the following, we show that, for each $\tau' \geq \tau - 1$, $|\{p' \in C' \mid d(p') \leq \tau'\}| \leq \tau' - (\tau - 1)$. Obviously, this is true if $\tau' < d(q)$ or $\tau' \geq d(p_1)$. Hence, it is sufficient to show, for each $d(q) \leq \tau' < d(p_1)$,

- $|\{p' \in C' \mid d(p') \leq \tau'\}| \leq \tau' - (\tau - 1)$.

First, we prove, for each $d(q) \leq \tau' < d(p_1)$, $|\{p' \in C_{t-1} \mid d(p') \leq \tau'\}| \leq \tau' - (\tau - 1) - 1$. Suppose for contradiction that, for some $d(q) \leq \tau' < d(p_1)$, $|\{p' \in C_{t-1} \mid d(p') \leq \tau'\}| = \tau' - (\tau - 1)$. The invariant already provides $|\{p' \in C_{t-1} \mid d(p') \leq \tau'\}| \leq \tau' - (\tau - 1)$. For $\tau_1 \neq 0$, this yields the following
3.2 The Bounded-Delay Model

contradiction:

\[
\tau' - \tau_1 = \tau' - (\tau - 1) - (\tau_1 - (\tau - 1)) \\
\leq \tau' - (\tau - 1) - |\{p' \in C_{t-1} \mid d(p') \leq \tau'\}| \\
= |\{p' \in C_{t-1} \mid \tau_1 < d(p') \leq \tau'\}| \\
\leq |\{p' \in S_{t-1} \setminus \{q\} \mid \tau_1 < d(p') \leq \tau'\}| \\
= |\{p' \in S_{t-1} \mid \tau_1 < d(p') \leq \tau'\}| - 1 \\
= |\{p' \in S_{t-1} \mid d(p') \leq \tau'\}| - |\{p' \in S_{t-1} \mid d(p') \leq 1\}| - 1 \\
\leq \tau' - (\tau - 1) - (\tau_1 - (\tau - 1)) - 1 \\
= \tau' - \tau_1 - 1 .
\]

For \(\tau_1 = 0\), this yields the following contradiction:

\[
\tau' - (\tau - 1) = |\{p' \in C_{t-1} \mid d(p') \leq \tau'\}| \\
\leq |\{p' \in S_{t-1} \setminus \{q\} \mid d(p') \leq \tau'\}| \\
= |\{p' \in S_{t-1} \mid d(p') \leq \tau'\}| - 1 \\
\leq \tau' - (\tau - 1) - 1 .
\]

Then, for each \(d(q) \leq \tau' < d(p_1)\),

\[
|\{p' \in C' \mid d(p') \leq \tau'\}| = |\{p' \in \{q\} \cup C_{t-1} \setminus \{p_1\} \mid d(p') \leq \tau'\}| \\
= |\{p' \in \{q\} \cup C_{t-1} \mid d(p') \leq \tau'\}| \\
= |\{p' \in C_{t-1} \mid d(p') \leq \tau'\}| + 1 \\
\leq \tau' - (\tau - 1) .
\]

If no packet \(f \in C'\) with \(v(f) \leq v(p_f)\) exists, set \(C_t := C'\). Otherwise, set \(C_t := C_{t-1} \setminus \{f\}\), where \(f\) denotes the first packet in \(C_{t-1}\) according to the canonical order with \(v(f) \leq v(p_f)\). Then, this case follows analogously to case (a).

(f) Suppose that ONL sends \(p_m \in C_{t-1}\) and OPT sends a packet \(q \in S_{t-1} \setminus \{p_m\}\) in \(\sigma_t\).

This case follows analogously to case (e) except that \(C_t\) is set to \(C' \setminus \{p_m\}\) or \(C' \setminus \{f, p_m\}\) instead of \(C'\) or \(C' \setminus \{f\}\).

Let \(\sigma_t\) be an arrival event in time step \(\tau\).

(g) Set \(C_t := C_{t-1}\). Obviously, the invariant is true, and

\[
\sum_{p' \in C_t} v(p') - \sum_{p' \in C_{t-1}} v(p') = 0 .
\]

This concludes the proof of the lemma. \(\square\)
For each $\sigma_t$, the upper bounds on $A(L_t, B_{OPT}^t \setminus B_{ONL}^t) - A(L_{t-1}, B_{OPT}^{t-1} \setminus B_{ONL}^{t-1})$ from Observation 3.17, the upper bounds on $V(L_t, S_t) - V(L_{t-1}, S_{t-1})$ from Observation 3.18, and the lower bounds on $\sum_{p' \in C_t} v(p') - \sum_{p' \in C_{t-1}} v(p')$ from Lemma 3.19 are used in a straightforward case analysis to show $\Phi_t - \Phi_{t-1} \geq 0$. This case analysis is deferred to Appendix B.1.

In our non-memoryless strategy either $p_f$ or $p_m$ is sent depending on the condition

$$\max\{v(p_f), \delta_{L_t-1}(\tau)\} \geq \frac{v(p_m) + (\beta - 1) \cdot v(s_{t-1}(p_m))}{\beta}.$$ 

The only difference of our memoryless strategy to our non-memoryless strategy is that this condition is replaced by

$$v(p_f) \geq \frac{v(p_m) + v(s_{t-1}(p_m))/2}{\beta}.$$ 

The fact that either $p_f$ or $p_m$ is sent based on the aforementioned condition is only exploited in the case analysis in Appendix B.1 of the proof of Theorem 3.15. Other parts of the proof are not affected by a change of this condition. Note that $A(L_t, B_{OPT}^t \setminus B_{ONL}^t)$ and $V(L_t, S_t)$ depend on $L_t$ and that our memoryless strategy does not compute $L_t$. However, it is sufficient to define $L_t$ in the proof.

Using the above observations, we can adopt the proof of Theorem 3.15 to show Theorem 3.16. For each event $\sigma_t$, the potential function has to be redefined

$$\Phi_t := r \sum_{p \in S_{t-1}^{ONL}} v(p) - \sum_{p \in S_{t}^{OPT}} v(p) - A(L_t, B_{t}^{OPT} \setminus B_{t}^{ONL}) - V(L_t, S_t) + \alpha \sum_{p \in C_t} v(p),$$

with $\alpha := (\beta^2 - 3)/2$.

For each event $\sigma_t$, the upper bounds from Observation 3.17, the upper bounds from Observation 3.18, and the lower bounds from Lemma 3.19 are used in a straightforward case analysis to show $\Phi_t - \Phi_{t-1} \geq 0$, which implies Theorem 3.16. The case analysis is deferred to Appendix B.2.
Conclusions

We studied four different online scheduling problems that involve the management of a buffer. First, we investigated reordering buffers and showed how they can be applied in different settings. For the reordering buffer problem, we presented a provable well-performing strategy for general metric spaces, answering the open question arising out of numerous results for special metric spaces. In case of the online minimum makespan scheduling problem, we showed that reordering buffers are a powerful tool which can lead not only to significantly improved competitive ratios, but also to much simpler proofs. Hopefully, in the future, there will be a broad and systematic study of the power of reordering in different settings. Of course, such a study should not be restricted to reordering buffers. It ought to incorporate other aspects like, for example, avoiding starvation by bounding the amount of time for which a request can be stored in the buffer. (Although PAY already somewhat avoids starvation since each request buys edges towards PAY’s server and will eventually be contained in a connected component.)

The second part of the thesis was devoted to two problems motivated by packet forwarding in QoS switches. Here, the buffer was not so much an optional tool for improving performance as a problem-specific necessity. We completely solved both problems for the case that only two different packet values appear in the input sequence. For the general case we improved the previous bounds.

All the bounds presented in this thesis are the currently best known for the specific problem. Our bounds for the online minimum makespan scheduling problem with identical machines and reordering and the bounds for the FIFO and the bounded-delay model with two packet values are optimal. For the remaining problems no tight bounds are known. This naturally leads us to a number of open questions.

4.1 Open Problems

In the following, we discuss some specific open questions related to the four problems of this thesis. One obvious question which concerns all these problems is how random-
Conclusions

In this thesis, we have mainly studied deterministic strategies but, of course, studying the possibilities and limits of randomization is also intriguing. For none of the problems, tight bounds are known for randomized online algorithms.

4.1.1 The Reordering Buffer Problem

No algorithms are known to achieve a constant competitive ratio for the reordering buffer problem for any non-trivial metric space. However, the largest and in fact only non-trivial lower bound on the reordering buffer problem was given by Gamzu and Segev [GS07] for line metrics. They show that no deterministic online algorithm can achieve a competitive ratio below \( 1 + 2/\sqrt{3} \approx 2.154 \). For some naïve algorithms non-constant lower bounds are known. For the PAY strategy, however, we do not know any lower bound.

This poses the question whether it is possible to improve the upper bound on the competitive ratio of PAY. As long as we use the metric embedding result devised by Fakcharoenphol, Rao, and Talwar [FRT04], we lose an \( \Theta(\log n) \)-factor in the competitive ratio. Hence, we cannot hope for a significantly improved upper bound for general metric spaces without a wholly new technique for avoiding the use of this metric embedding as a black box tool. In light of this observation, we turn our attention to more special metric spaces. Probably the most basic metric is the uniform metric.

A uniform metric space can essentially be described by a tree of height one where the root node is connected to its children by edges of length 1/2. Excluding the root node, any two nodes have distance 1 from each other. We can run our PAY algorithm on such a tree. It follows from Theorem 2.1 that we achieve a competitive ratio of \( O(\log k) \) for this metric space since \( D = 2 \).

In [ERW06] we give strong empirical evidence that our proof techniques (and in fact all previously used proof techniques) are inherently incapable of showing significantly improved upper bounds for uniform metrics. We give evidence that there is a family of instances on which the cost of a solution given by an optimal offline strategy with a buffer of size \( k \) is at least by an \( \Omega(\sqrt{\log k}) \) factor larger than the cost of a solution given by an optimal offline strategy with a buffer of size \( 4k \).

This has the following interesting consequence: Suppose a proof technique is used to show that an online algorithm \( A \) with a buffer of size \( k \) is \( f(k) \)-competitive against an optimal offline algorithm with the same buffer size. If the same proof technique can be used to show that, for some constant \( c \), \( A \) is \( c \cdot f(k) \)-competitive against an optimal offline algorithm with a buffer of size \( 4k \), the technique is, most likely, inherently incapable of showing an \( o(\sqrt{\log k}) \) upper bound on the competitive ratio of \( A \) against an optimal offline strategy that has the same buffer size as \( A \), that is, most likely, \( f(k) = \Omega(\sqrt{\log k}) \).

Stated more informally: Every proof technique that is “robust” against increases of OPT’s buffer size is most likely incapable of showing an \( o(\sqrt{\log k}) \) upper bound on the competitive ratio.

This applies to all known upper bound proof techniques, including our analysis for PAY. Hence, although we do not know of any non-constant lower bound, proving a constant upper bound will probably require fundamentally new ideas. Apart from
improving the upper bounds, it would also be interesting to show a non-constant lower bound, for example, for general metric spaces.

Although PAY does not need to know the tree metric space in advance, our algorithm for the reordering buffer problem for general metric spaces has to know the complete metric space from the start. This is an implication of the used metric embedding which requires the choice of a random permutation of all points in the metric space. Without this random permutation, e.g., considering the points in the metric only over time as they are revealed, the guarantee that distances are stretched by a factor of at most $O(\log n)$ in expectation cannot be given anymore. If we use the metric embedding technique devised by Bartal [Bar96], we can solve this problem at least partially. His embedding technique yields an $O(\log^2 \delta)$-probabilistic approximation of general metrics by HSTs when the points of the metric arrive over time, that is, when the metric embedding is constructed online. Here $\delta$ denotes the aspect ratio of the metric space which is the maximal ratio of any two distances in the metric space. Using this embedding yields a competitive ratio of $O(\log^2 \delta \cdot \log^2 k)$ in expectation against an oblivious adversary.

Finally, we could ask ourselves if it is worth considering the reordering buffer problem for arbitrary distance functions which do not need to be symmetric or do not need to satisfy the triangle inequality.

It can easily be seen that for distance functions not satisfying the triangle inequality, there is no online algorithm with a competitive ratio below $\Omega(\delta)$, where $\delta$ denotes the aspect ratio. Consider the following symmetric distance function over four points $V = \{p_1, \ldots, p_4\}$: $d(p_1, p_2) = d(p_2, p_3) = d(p_1, p_4) = 1$ and all other distances are $\delta$. Now, given an online algorithm $A$ with a reordering buffer of size 2, consider the input sequence of points $p_1, p_2, x$, where $x$ is either $p_3$ or $p_4$. When $x$ arrives, $A$ can have only one of the points $p_1$ and $p_2$ left in the buffer. If $p_1$ is left in the buffer, $x$ is chosen to be $p_3$ otherwise, $x$ is chosen to be $p_4$. Therefore, the last pair in the output sequence of $A$ has cost $\delta$. The optimal cost, however, is 2 since $p_1, p_2, p_3$ is an optimal solution for $x = p_3$ and $p_2, p_1, p_4$ is an optimal solution for $x = p_4$. With slight modifications this can be iterated arbitrarily often and yields a lower bound of $\Omega(\delta)$ on the competitive ratio of $A$.

For non-symmetric distance functions, the server of an algorithm moves in a directed instead of an undirected graph. In this situation, it is much less obvious whether reasonable competitive ratios can be achieved. This remains as an interesting open problem.

### 4.1.2 Online Minimum Makespan Scheduling with Reordering

We gave tight bounds for the online minimum makespan scheduling problem with reordering for any number $m$ of identical machines. Nevertheless, one question remains. How large does the reordering buffer have to be in order to achieve this bound? We showed that a buffer of $\lceil 5m/2 \rceil + 2$ is sufficient in every case. Slightly smaller buffer sizes are possible for larger values of $m$. As observed by Deniz Özmen, a buffer of size $\left\lceil m/2 \right\rceil$ is not sufficient (personal communication, November 2007).

Seeing the positive effect a reordering buffer has for the problem with identical machines in terms of both the competitive ratio and the complexity of proofs, it is natural to further investigate this method and to apply reordering buffers to other
problems as well. Numerous scheduling problems with varying machine models, scheduling constraints, and objectives can be considered in this setting. We started this line of research by also considering $m$ uniformly related machines.

### 4.1.3 Buffer Management for Switches Supporting Quality of Service

Improving the known bounds on the competitive ratios is the apparent and challenging open problems of packet forwarding in network switches in the FIFO and the bounded-delay model.

We hope that the concept of suppressed packets, that is, considering not only the packets in the provisional optimal schedule but also the other packets in the buffer, will be helpful for designing an optimal strategy for the bounded-delay model.

For the FIFO model, new approaches are needed. The PG algorithm is the only strategy known to achieve a competitive ratio below 2. We showed that the competitive ratio of PG lies between $1 + 1/\sqrt{2} \approx 1.707$ and $\sqrt{3} \approx 1.732$. Therefore, we conclude that the competitive ratio of PG cannot be further improved to any significant degree. A basic concept of PG is that, for each arriving packet $p$, the first packet whose value is at most $v(p)/\beta$ is preempted. At first sight, it seems more reasonable that the packet with the smallest value is preempted instead. In fact, however, the preemption of the first packet whose value is sufficiently small is a crucial property to achieve a competitive ratio below 2. However, preempting the first possible packet can turn out to be a great disadvantage as the first input sequence of our lower bound construction shows. This disadvantage diminishes with increasing $\beta$. On the other hand, too few packets are preempted for larger $\beta$. Increasing $\beta$ but allowing to combine the values of more than one packet for preemption might be a possible approach. At least in the case where only two different packet values appear in the input, this aggregation of values enabled us to design an optimal algorithm.

Finally, we would like to pose a technical question. The proofs in Chapter 3 and, in particular, the proof of the upper bound on the competitive ratio of PG, are rather involved (especially in contrast to the proofs presented in the first part of this thesis). This is not specific to our proofs. For these problems, all the known proofs showing a competitive ratio below 2 seem to be rather complicated and require case analyses, heavy notations, or other elements that somewhat obfuscate the intuition. To give simplified, more accessible proofs for our results would certainly be of great interest.
Verification of the Invariants from the Proof of Theorem 3.10

In the following two sections we verify that the five invariants hold for \( \sigma_t \) if they hold for \( \sigma_{t-1} \). For this verification, we distinguish 21 cases. For each case, we specify how the set \( D \), the states of the packets, and the buddy relations are chosen. If appropriate, we state, in parentheses, some simple facts that are used in the respective case. Thereafter, Invariant I1 is verified. The Invariants I2–I5 are verified in separate figures.

A.1 Cases for Arrival Events

Fix an arrival event \( \sigma_t \) in which a packet \( p \) arrives. We distinguish the following cases. If not mentioned otherwise, everything remains unchanged at event \( \sigma_t \). We only consider the Invariant I1. For the verification of the Invariants I2–I5, see Figure A.1

- \( p \) preempts another packet \( q \)
  - a1: \( q \in B_{t-1} \cup BC_{t-1} \)
    - Changes: \( b_t(b^{-1}_{t-1}(q)) := p \) and \( s_t(p) := B \)
    - II: \( \Delta^{pg}_t \geq 0 = \Delta^{opt}_t \)

  - a2: \( q \in E_{t-1} \cup EB_{t-1} \)
    - Changes: \( s_t(p) := U \)
    - II: \( \Delta^{pg}_t \geq (r/\beta) \cdot v(p) - 2(r-1) \cdot v(q) \)
      \[ \geq (r/\beta) \cdot v(p) - 2(r-1) \cdot v(p)/\beta \]
      \[ = ((2-r)/\beta) \cdot v(p) > 0 = \Delta^{opt}_t \]

  - a3: \( q \in U_{t-1} \)
Appendix A — Verification of the Invariants from the Proof of Theorem 3.10

Changes: $s_t(p) := U$ and $D_t := D_{t-1} \cup \{q\}$

I1: $\Delta_{t}^{\text{pg}} \geq (r/\beta) \cdot v(p) - ((r/\beta) \cdot v(q) + (2 - r) \cdot v(q))$
$\geq (r/\beta) \cdot (\beta \cdot v(q)) - (r - 1) \cdot v(q) = v(q) = \Delta_{t}^{\text{opt}}$

• $p$ ejects another packet $q$

a4: $q \in B_{t-1} \cup BC_{t-1}$
Changes: $s_t(p) := B$ and $b_t(b_{t-1}(q)) := p$

I1: $\Delta_{t}^{\text{pg}} \geq 0 = \Delta_{t}^{\text{opt}}$

a5: $q \in E_{t-1} \cup U_{t-1}$
Changes: $s_t(p) := U$

I1: $\Delta_{t}^{\text{pg}} \geq (r/\beta) \cdot v(p) + (2 - r) \cdot v_{t-1}^{\text{min}}(p) - (r - 1) \cdot v(q)$
$\geq (r/\beta) \cdot v(q) + (2 - r) \cdot v(q) - (r - 1) \cdot v(q)$
$= ((2r - 3) + (2 - r) - (r - 1)) \cdot v(q) = 0 = \Delta_{t}^{\text{opt}}$

a6: $q \in EB_{t-1}$
Changes: $s_t(p) := U$ and $s_t(b_{t-1}(q)) := U$
(Due to I3, $b_{t-1}(q) \in B_{t-1} \cup BC_{t-1}$.)

I1: $\Delta_{t}^{\text{pg}} \geq (r/\beta) \cdot v(p) + (2 - r) \cdot v_{t-1}^{\text{min}}(p) + (r/\beta) \cdot v(b_{t-1}(q))$  
$+ (2 - r) \cdot v_{t-1}^{\text{min}}(b_{t-1}(q)) - 2(r - 1) \cdot v(q)$
$\geq (r/\beta) \cdot v(q) + (2 - r) \cdot v(q)$
$+ (r/\beta) \cdot v(q) + (2 - r) \cdot v(q) - 2(r - 1) \cdot v(q)$
$= (2((2r - 3) + (2 - r) - 2(r - 1))) \cdot v(q) = 0 = \Delta_{t}^{\text{opt}}$

• $p$ is rejected
Changes: –
(Due to Lemma 3.11, $p$ is also not stored in the buffer of OPT.)
### A.1 Cases for Arrival Events

<table>
<thead>
<tr>
<th>Case</th>
<th>Packets Concerned</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>$q, b_{t-1}(q)$</td>
<td>$s_t(q) = \perp, b_t(b_{t-1}(q)) = p \in B_t$</td>
</tr>
<tr>
<td></td>
<td>$p, q$</td>
<td>$b_t^{-1}(p) = b_{t-1}^{-1}(q) \prec q = d(p), q \notin B_t^{\text{pg}}$</td>
</tr>
<tr>
<td>a2</td>
<td>$q, b_{t-1}(q)$</td>
<td>$s_t(q) = \perp, s_t(b_{t-1}^{-1}(q)) = s_{t-1}(b_{t-1}^{-1}(q))$</td>
</tr>
<tr>
<td></td>
<td>$q, {p'</td>
<td>\hat{d}_{t-1}(p') = q}$</td>
</tr>
<tr>
<td></td>
<td>$p, q$</td>
<td>$b_t^{-1}(p) = b_{t-1}^{-1}(q) \prec q = d(p), q \notin B_t^{\text{pg}}$</td>
</tr>
<tr>
<td>a3</td>
<td>$q, b_{t-1}(q)$</td>
<td>$s_t(q) = \perp, s_t(b_{t-1}^{-1}(q)) = s_{t-1}(b_{t-1}^{-1}(q))$</td>
</tr>
<tr>
<td></td>
<td>$q, {p'</td>
<td>\hat{d}_{t-1}(p') = q}$</td>
</tr>
<tr>
<td></td>
<td>$p, q$</td>
<td>$b_t^{-1}(p) = b_{t-1}^{-1}(q) \prec q = d(p), q \notin B_t^{\text{pg}}$</td>
</tr>
<tr>
<td>a4</td>
<td>$q, b_{t-1}(q)$</td>
<td>$s_t(q) = \perp, b_t(b_{t-1}^{-1}(q)) = p \in B_t$</td>
</tr>
<tr>
<td></td>
<td>$q, {p'</td>
<td>\hat{d}_{t-1}(p') = q}$</td>
</tr>
<tr>
<td></td>
<td>$p, q$</td>
<td>$b_t^{-1}(p) = b_{t-1}^{-1}(q) \prec q = d(p), q \notin B_t^{\text{pg}}$</td>
</tr>
<tr>
<td>a5</td>
<td>$q, b_{t-1}(q)$</td>
<td>$s_t(q) = \perp, s_t(b_{t-1}^{-1}(q)) = s_{t-1}(b_{t-1}^{-1}(q))$</td>
</tr>
<tr>
<td></td>
<td>$q, {p'</td>
<td>\hat{d}_{t-1}(p') = q}$</td>
</tr>
<tr>
<td></td>
<td>$p, q$</td>
<td>$b_t^{-1}(p) = b_{t-1}^{-1}(q) \prec q = d(p), q \notin B_t^{\text{pg}}$</td>
</tr>
<tr>
<td>a6</td>
<td>$q, b_{t-1}(q)$</td>
<td>$s_t(q) = \perp, s_t(b_{t-1}^{-1}(q)) = s_{t-1}(b_{t-1}^{-1}(q))$</td>
</tr>
<tr>
<td></td>
<td>$b_{t-1}(q)$</td>
<td>$b_t(b_{t-1}^{-1}(q)) = \perp \equiv d(b_{t-1}(q))$</td>
</tr>
<tr>
<td></td>
<td>$q, {p'</td>
<td>\hat{d}_{t-1}(p') = q}$</td>
</tr>
<tr>
<td></td>
<td>$b_{t-1}(q)$</td>
<td>$b_t(b_{t-1}^{-1}(q)) = \perp \equiv d(b_{t-1}(q))$</td>
</tr>
</tbody>
</table>

Figure A.1: Verification of the Invariants I2–I5 for the Cases a1–a6.
A.2 Cases for Send Events

Fix a send event $\sigma_t$ in which PG sends packet $p$ and OPT sends packet $q$. Note that due to Lemma 3.11, $q \in P_{t-1} \cup S_{t-1}^{pg} \cup B_{t-1}^{pg}$. Since a new dummy packet of value 0 is stored in the buffer of PG after a packet is sent, a packet $u_B \in B_{t}^{pg} \setminus B_{t-1}^{pg}$ exists with $s_t(u_B) = B$. We can assign $u_B$ as buddy to another packet at this event, since $u_B \not\in B_{t-1}^{pg}$. We distinguish the following cases. If not mentioned otherwise, everything remains unchanged at event $\sigma_t$. We only consider the Invariant I1. For the verification of the Invariants I2–I5, see Figure A.3, Figure A.4, and Figure A.5. In Figure A.2, we depict the possible state transitions at $\sigma_t$.

- $q \in P_{t-1} \cup S_{t-1}^{pg}$
  
  b1: $q \in D_{t-1}$ and $p \in B_{t-1} \cup BC_{t-1}$
  
  Changes: $D_t := D_{t-1} \cup \{p\} \cup \{p'|\hat{d}_{t-1}(p') = p\}$
  
  $I1: \Delta_{t}^{pg} \geq r \cdot v(p) \geq v(p) + \sum_{i=1}^{\infty} v(p) / \beta^i \geq \Delta_{t}^{opt}$

b2: $q \in D_{t-1}$ and $p \in B_{t}^{OPT} \setminus (B_{t-1} \cup BC_{t-1})$

Changes: $D_t := D_{t-1} \cup \{p\}$

(Due to I2, $p \in U_{t-1}$.)

$I1: \Delta_{t}^{pg} \geq r \cdot v(p) - ((r/\beta) \cdot v(p) + (2 - r) \cdot v_{t-1}^{min}(p)) \geq v(p) = \Delta_{t}^{opt}$

b3: $q \in D_{t-1}$ and $p \not\in B_{t}^{OPT} \cup (B_{t-1} \cup BC_{t-1})$

Changes: –

$I1: \Delta_{t}^{pg} \geq r \cdot v(p) - 2(r - 1) \cdot v(p) \geq 0 = \Delta_{t}^{opt}$

b4: $q \not\in D_{t-1}$ and $p \in B_{t-1} \cup BC_{t-1}$

Changes: $s_t(\hat{d}_{t-1}(q)) := U$, $D_t := D_{t-1} \cup \{p\} \cup \{p'|\hat{d}_{t-1}(p') = p\} \cup \{q' \neq q|\hat{d}_{t-1}(q') = \hat{d}_{t-1}(q)\}$, $b_t(b_{t-1}(\hat{d}_{t-1}(q))) := u_B$
A.2 Cases for Send Events

(Due to I4, $\hat{d}_{t-1}(q) \in BC_{t-1} \cup B_{t-1}$.)

I1: $\Delta^p_t \geq r \cdot v(p) + (r/\beta) \cdot v(\hat{d}_{t-1}(q))$

\[ \geq v(p) + \sum_{i=1}^{\infty} v(p)/\beta^i + \sum_{i=1}^{\infty} v(\hat{d}_{t-1}(q))/\beta^i \]

\[ \geq v(p) + \sum_{i=1}^{\infty} v(p)/\beta^i + \sum_{q', \hat{d}_{t-1}(q') = \hat{d}_{t-1}(q)} v(q') \geq \Delta^{\text{opt}}_t \]

b5: $q \notin D_{t-1}$ and $p \in B^{\text{OPT}}_t \setminus (B_{t-1} \cup BC_{t-1})$

Changes: $s_t(\hat{d}_{t-1}(q)) := U$, $D_t := D_{t-1} \cup \{p\} \cup \{q' \neq q|\hat{d}_{t-1}(q') = \hat{d}_{t-1}(q)\}$, $b_t(b_{t-1}^{-1}(\hat{d}_{t-1}(q))) := u_B$

(Due to I2, $p \in U_{t-1}$. Due to I4, $\hat{d}_{t-1}(q) \in BC_{t-1} \cup B_{t-1}$.)

I1: $\Delta^p_t \geq r \cdot v(p) - ((r/\beta) \cdot v(p) + (2 - r) \cdot v_{t-1}^{\min}(p))$

\[ + (r/\beta) \cdot v(\hat{d}_{t-1}(q)) \]

\[ \geq v(p) + (r/\beta) \cdot v(\hat{d}_{t-1}(q)) \]

\[ \geq v(p) + \sum_{i=1}^{\infty} v(\hat{d}_{t-1}(q))/\beta^i \geq \Delta^{\text{opt}}_t \]

b6: $q \notin D_{t-1}$ and $p \notin B^{\text{OPT}}_t \cup (B_{t-1} \cup BC_{t-1})$

Changes: $s_t(\hat{d}_{t-1}(q)) := U$, $D_t := D_{t-1} \cup \{q' \neq q|\hat{d}_{t-1}(q') = \hat{d}_{t-1}(q)\}$, $b_t(b_{t-1}^{-1}(\hat{d}_{t-1}(q))) := u_B$

(Due to I4, $\hat{d}_{t-1}(q) \in BC_{t-1} \cup B_{t-1}$.)

I1: $\Delta^p_t \geq r \cdot v(p) - 2(r-1) \cdot v(p) + (r/\beta) \cdot v(\hat{d}_{t-1}(q))$

\[ \geq (r/\beta) \cdot v(\hat{d}_{t-1}(q)) \]

\[ \geq \sum_{i=1}^{\infty} v(\hat{d}_{t-1}(q))/\beta^i \geq \Delta^{\text{opt}}_t \]

• b7: $q = p$

Changes: –

(Due to I2, $p \in U_{t-1} \cup BC_{t-1} \cup B_{t-1}$.)

I1: $\Delta^p_t \geq r \cdot v(p) - ((r/\beta) \cdot v(p) + (2 - r) \cdot v_{t-1}^{\min}(p)) \geq v(p) = \Delta^{\text{opt}}_t$

• $q \in B^{\text{pg}}_t \setminus \{p\}$

b8: $q \in U_{t-1}$

Changes: $b_t(q) := u_B$, $s_t(q) := EB$

I1: $\Delta^p_t \geq r \cdot v(p) - c_{t-1}(p)$

\[ + 2(r-1) \cdot v(q) - ((r/\beta) \cdot v(q) + (2 - r) \cdot v_{t-1}^{\min}(q)) \]

\[ \geq r \cdot v(p) - 2(r-1) \cdot v(p) \]

\[ + 2(r-1) \cdot v(q) - ((r/\beta) \cdot v(q) + (2 - r) \cdot v(p)) \]

\[ = v(q) = \Delta^{\text{opt}}_t \]
Appendix A — Verification of the Invariants from the Proof of Theorem 3.10

b9: $q \in BC_{t-1}$

Changes: $b_t(b^1_{t-1}(q)) := u_B$, $s_t(q) := E$

\[ \Delta^B_t \geq r \cdot v(p) - c_{t-1}(p) + (r - 1) \cdot v(q) \geq v(q) = \Delta^B_t \]

b10: $q \in B_{t-1}$ and $v(p) < v(q)/\beta$

Changes: $b_t(q) := u_B$, $s_t(q) := E$

(Due to I5, $b^1_{t-1}(q) < d(q)$, i.e., $b^1_{t-1}(q) \notin B^B_{t-1}$, since $v(p) < v(q)/\beta$)

\[ \Delta^B_t \geq r \cdot v(p) - c_{t-1}(p) + 2(r - 1) \cdot v(q) > v(q) = \Delta^B_t \]

b11: $q \in B_{t-1}$ and $v(p) \geq v(q)/\beta$ and $p \notin EB_{t-1}$

Changes: $b_t(b^1_{t-1}(q)) := u_B$, $s_t(q) := EB$

\[ \Delta^B_t \geq r \cdot v(p) - c_{t-1}(p) + (r - 1) \cdot v(q) \]

\[ \geq v(p) + (r - 1) \cdot v(q) \]

\[ \geq (1/\beta + (r - 1)) \cdot v(q) = v(q) = \Delta^B_t \]

b12: $q \in B_{t-1}$ and $v(p) \geq v(q)/\beta$ and $p \in EB_{t-1}$ and $b^1_{t-1}(q) = \perp$

Changes: $b_t(q) := u_B$, $s_t(q) := E$

\[ \Delta^B_t \geq r \cdot v(p) - c_{t-1}(p) + 2(r - 1) \cdot v(q) \geq v(q) = \Delta^B_t \]

b13: $q \in B_{t-1}$ and $v(p) \geq v(q)/\beta$ and $p \in EB_{t-1}$ and $b^1_{t-1}(p) \leq b^1_{t-1}(q)$

Changes: $s_t(b^1_{t-1}(p)) := E$, $b_t(b^1_{t-1}(q)) := u_B$, $s_t(q) := E$

(Due to I5, $b^1_{t-1}(q) < d(q)$, i.e., $v(b_{t-1}(p)) \geq v(q)/\beta$. Due to I3, $b_{t-1}(p) \in B_{t-1} \cup BC_{t-1}$.)

\[ \Delta^B_t \geq r \cdot v(p) - 2(r - 1) \cdot v(p) + (r - 1) \cdot v(q) + (r - 1) \cdot v(b_{t-1}(p)) \]

\[ \geq (2 - r) \cdot v(q)/\beta + (r - 1) \cdot v(q) + (r - 1) \cdot v(q)/\beta \]

\[ = (1/\beta + (r - 1)) \cdot v(q) = v(q) = \Delta^B_t \]

b14: $q \in B_{t-1}$ and $v(p) \geq v(q)/\beta$ and $p \in EB_{t-1}$ and $b^1_{t-1}(q) < b^1_{t-1}(p)$ and $v(b_{t-1}(p)) > 2v(q)$

Changes: $s_t(b^1_{t-1}(p)) := U$, $D_t := D_{t-1} \cup \{p' \mid d_{t-1}(p') = b_{t-1}(p)\}$,

$b_t(b^1_{t-1}(q)) := u_B$, $s_t(q) := E$

(Due to I3, $b_{t-1}(p) \in B_{t-1} \cup BC_{t-1}$.)

\[ \Delta^B_t \geq r \cdot v(p) - c_{t-1}(p) + (r/\beta) \cdot v(b_{t-1}(p)) - c_{t-1}(b^1_{t-1}(p)) \]

\[ + (r - 1) \cdot v(q) \]

\[ \geq (2 - r) \cdot v(p) + (r/\beta) \cdot v(b_{t-1}(p)) + (r - 1) \cdot v(q) \]

\[ = (2 - r) \cdot v(p) + (3r - 5)/2 \cdot v(b^1_{t-1}(p)) \]

\[ + (r - 1) \cdot v(q) + v(b_{t-1}(p))/(\beta - 1) \]

\[ \geq (2 - r) \cdot v(q)/\beta + (3r - 5) \cdot v(q) + (r - 1) \cdot v(q) \]

\[ + v(b_{t-1}(p))/(\beta - 1) \]

\[ = v(q) + \sum_{i=1}^{\infty} v(b_{t-1}(p))/\beta^i \geq \Delta^B_t \]
A.2 Cases for Send Events

b15: $q \in B_{t-1}$ and $v(p) \geq v(q)/\beta$ and $p \in EB_{t-1}$ and $b_{t-1}^{-1}(q) < b_{t-1}(p)$ and $v(b_{t-1}(p)) \leq 2v(q)$

Changes: $s_t(b_{t-1}(p)) := BC$, $b_t(b_{t-1}^{-1}(q)) := b_{t-1}(p)$, $b_t(q) := u_B$, $s_t(q) := EB$

(Due to I3, $b_{t-1}(p) \in B_{t-1} \cup BC_{t-1}$.)

I1: $\Delta_{t}^{pg} = r \cdot v(p) - c_1(p) + (r - 2) \cdot v(b_{t-1}(p)) - c_{t-1}(b_{t-1}(p)) + 2(r - 1) \cdot v(q)$

$\geq (2 - r) \cdot v(p) + (r - 2) \cdot v(b_{t-1}(p)) + 2(r - 1) \cdot v(q)$

$\geq (2 - r) \cdot v(q)/\beta + (r - 2) \cdot 2v(q) + 2(r - 1) \cdot v(q)$

$= v(q) = \Delta_{t}^{opt}$
### Appendix A — Verification of the Invariants from the Proof of Theorem 3.10

<table>
<thead>
<tr>
<th>Case</th>
<th>Packets Concerned</th>
<th>Verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>b1</td>
<td>I2</td>
<td>$s_t(p) = \bot$, $b_{t-1}^{-1}(p) = \bot$</td>
</tr>
<tr>
<td></td>
<td>I3</td>
<td>$p, b_{t-1}^{-1}(p)$, $s_t(p) = \bot$, $b_{t-1}^{-1}(p) = \bot$</td>
</tr>
<tr>
<td></td>
<td>I4</td>
<td>$p \in D_t \implies p \notin X_t$</td>
</tr>
<tr>
<td></td>
<td>I5</td>
<td>$p, b_{t-1}(p)$, $u_B$ $d(u_B) = \bot$</td>
</tr>
<tr>
<td></td>
<td>$p \notin B_t^{\text{opt}}$, $b_{t-1}^{-1}(b_{t-1}(p)) = \bot \prec d(b_{t-1}(p))$</td>
<td></td>
</tr>
<tr>
<td>b2</td>
<td>I2</td>
<td>$s_t(p) = \bot$, $b_{t-1}^{-1}(p) = \bot$</td>
</tr>
<tr>
<td>and</td>
<td>I3</td>
<td>$p, b_{t-1}^{-1}(p)$, $s_t(p) = \bot$, $b_{t-1}^{-1}(p) = \bot$</td>
</tr>
<tr>
<td>b3</td>
<td>I4</td>
<td>$p \notin B_t^{\text{opt}} \setminus D_t \supseteq X_t$</td>
</tr>
<tr>
<td></td>
<td>I5</td>
<td>$p, b_{t-1}(p)$, $u_B$ $d(u_B) = \bot$</td>
</tr>
<tr>
<td></td>
<td>$p \notin B_t^{\text{opt}}$, $b_{t-1}^{-1}(b_{t-1}(p)) = \bot \prec d(b_{t-1}(p))$</td>
<td></td>
</tr>
<tr>
<td>b4</td>
<td>I2</td>
<td>$s_t(p) = \bot$, $b_{t-1}^{-1}(p) = \bot$</td>
</tr>
<tr>
<td></td>
<td>I3</td>
<td>$p, b_{t-1}^{-1}(p)$, $b_{t-1}^{-1}(d_{t-1}(q))$ $b_t(b_{t-1}^{-1}(d_{t-1}(q))) = u_B \in B_t$</td>
</tr>
<tr>
<td></td>
<td>I4</td>
<td>$p \in D_t \implies p \notin X_t$</td>
</tr>
<tr>
<td></td>
<td>I5</td>
<td>$p, b_{t-1}(p)$, $d_{t-1}(q)$, $u_B$ $d(u_B) = \bot$</td>
</tr>
<tr>
<td></td>
<td>$p \notin B_t^{\text{opt}}$, $b_{t-1}^{-1}(b_{t-1}(p)) = \bot \prec d(b_{t-1}(p))$</td>
<td></td>
</tr>
<tr>
<td>b5</td>
<td>I2</td>
<td>$s_t(p) = \bot$, $b_{t-1}^{-1}(p) = \bot$</td>
</tr>
<tr>
<td>and</td>
<td>I3</td>
<td>$p, b_{t-1}^{-1}(p)$, $b_{t-1}^{-1}(d_{t-1}(q))$ $b_t(b_{t-1}^{-1}(d_{t-1}(q))) = u_B \in B_t$</td>
</tr>
<tr>
<td>b6</td>
<td>I4</td>
<td>$p \in D_t \implies p \notin X_t$</td>
</tr>
<tr>
<td></td>
<td>I5</td>
<td>$p, b_{t-1}(p)$, $d_{t-1}(q)$, $u_B$ $d(u_B) = \bot$</td>
</tr>
<tr>
<td></td>
<td>$p \notin B_t^{\text{opt}}$, $b_{t-1}^{-1}(b_{t-1}(p)) = \bot \prec d(b_{t-1}(p))$</td>
<td></td>
</tr>
</tbody>
</table>

Figure A.3: Verification of the Invariants I2–I5 for the Cases b1–b6.
### A.2 Cases for Send Events

<table>
<thead>
<tr>
<th>case</th>
<th>packets concerned</th>
<th>verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>b7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>$p, b_{i-1}^{-1}(p)$</td>
<td>$s_i(p) = \bot, b_{i-1}^{-1}(p) = \bot$</td>
</tr>
<tr>
<td>13</td>
<td>$p$</td>
<td>$p \notin B^p \Rightarrow p \notin X_i,$</td>
</tr>
<tr>
<td>14</td>
<td>${p'</td>
<td>d_{i-1}(p') = p}$</td>
</tr>
<tr>
<td>15</td>
<td>$p, b_{i-1}(p)$</td>
<td>$p \notin B^p, b_i^{-1}(b_{i-1}(p)) = \bot &lt; d(b_{i-1}(p))$</td>
</tr>
<tr>
<td>$u_B$</td>
<td></td>
<td>$d(u_B) = \bot$</td>
</tr>
<tr>
<td>b8</td>
<td>$q$</td>
<td>$q \notin B^p$</td>
</tr>
<tr>
<td>12</td>
<td>$p, b_{i-1}^{-1}(p), b_{i-1}^{-1}(q)$</td>
<td>$s_i(p) = \bot, b_{i-1}^{-1}(p) = \bot, b_i^{-1}(q) = u_B \in B_i$</td>
</tr>
<tr>
<td>13</td>
<td>$p$</td>
<td>$p \notin B^p \Rightarrow p \notin X_i,$</td>
</tr>
<tr>
<td>14</td>
<td>${p'</td>
<td>d_{i-1}(p') = p}$</td>
</tr>
<tr>
<td>15</td>
<td>$p, b_{i-1}(p)$</td>
<td>$p \notin B^p, b_i^{-1}(b_{i-1}(p)) = \bot &lt; d(b_{i-1}(p))$</td>
</tr>
<tr>
<td>$q, u_B$</td>
<td></td>
<td>$b_i^{-1}(q) = \bot, d(u_B) = \bot$</td>
</tr>
<tr>
<td>b9</td>
<td>$q$</td>
<td>$q \notin B^p$</td>
</tr>
<tr>
<td>12</td>
<td>$p, b_{i-1}^{-1}(p), b_{i-1}^{-1}(q)$</td>
<td>$s_i(p) = \bot, b_{i-1}^{-1}(p) = \bot, b_i^{-1}(q) = u_B \in B_i$</td>
</tr>
<tr>
<td>13</td>
<td>$p$</td>
<td>$p \notin B^p \Rightarrow p \notin X_i,$</td>
</tr>
<tr>
<td>14</td>
<td>${p'</td>
<td>d_{i-1}(p') = p}$</td>
</tr>
<tr>
<td>15</td>
<td>$p, b_{i-1}(p)$</td>
<td>$p \notin B^p, b_i^{-1}(b_{i-1}(p)) = \bot &lt; d(b_{i-1}(p))$</td>
</tr>
<tr>
<td>$q, u_B$</td>
<td></td>
<td>$b_i^{-1}(q) = \bot, d(u_B) = \bot$</td>
</tr>
<tr>
<td>b10</td>
<td>$q$</td>
<td>$q \notin B^p$</td>
</tr>
<tr>
<td>12</td>
<td>$p, b_{i-1}^{-1}(p), b_{i-1}^{-1}(q)$</td>
<td>$s_i(p) = \bot, b_{i-1}^{-1}(p) = \bot$</td>
</tr>
<tr>
<td>13</td>
<td>$p$</td>
<td>$p &lt; q \Rightarrow p \notin B^p \supseteq X_i$</td>
</tr>
<tr>
<td>14</td>
<td>${p'</td>
<td>d_{i-1}(p') = p}$</td>
</tr>
<tr>
<td>15</td>
<td>$p, b_{i-1}(p)$</td>
<td>$p \notin B^p, b_i^{-1}(b_{i-1}(p)) = \bot &lt; d(b_{i-1}(p))$</td>
</tr>
<tr>
<td>$u_B$</td>
<td></td>
<td>$d(u_B) = \bot$</td>
</tr>
<tr>
<td>b11</td>
<td>$q$</td>
<td>$q \notin B^p$</td>
</tr>
<tr>
<td>12</td>
<td>$p, b_{i-1}^{-1}(p), b_{i-1}^{-1}(q)$</td>
<td>$s_i(p) = \bot, b_{i-1}^{-1}(p) = \bot, b_i^{-1}(q) = u_B \in B_i$</td>
</tr>
<tr>
<td>13</td>
<td>$p$</td>
<td>$p &lt; q \Rightarrow p \notin B^p \supseteq X_i$</td>
</tr>
<tr>
<td>14</td>
<td>${p'</td>
<td>d_{i-1}(p') = p}$</td>
</tr>
<tr>
<td>15</td>
<td>$p, b_{i-1}(p)$</td>
<td>$p \notin B^p, b_i^{-1}(b_{i-1}(p)) = \bot &lt; d(b_{i-1}(p))$</td>
</tr>
<tr>
<td>$u_B$</td>
<td></td>
<td>$d(u_B) = \bot$</td>
</tr>
</tbody>
</table>

Figure A.4: Verification of the Invariants I2–I5 for the Cases b7–b11.
<table>
<thead>
<tr>
<th>case</th>
<th>packets concerned</th>
<th>verification</th>
</tr>
</thead>
<tbody>
<tr>
<td>b12</td>
<td>I2 q</td>
<td>$q \not\in B_t^{OPT}$</td>
</tr>
<tr>
<td></td>
<td>I3 $p, b_{t-1}(p), q$</td>
<td>$s_t(p) = \bot, b_{t-1}(p) = \bot, b_t(q) = u_B \in B_t$</td>
</tr>
<tr>
<td></td>
<td>$b_{t-1}(q)$</td>
<td>$b_{t-1}(q) = \bot$</td>
</tr>
<tr>
<td></td>
<td>I4 $p$</td>
<td>$p &lt; q \Rightarrow p \not\in B_t^{OPT} \supseteq X_t$</td>
</tr>
<tr>
<td></td>
<td>${ p' \mid d_{t-1}(p') = p }$</td>
<td>$d_{t-1}(p') = p \Rightarrow p' &lt; p &lt; q \Rightarrow p' \not\in B_t^{OPT} \supseteq X_t$</td>
</tr>
<tr>
<td></td>
<td>${ p' \mid d_{t-1}(p') = q }$</td>
<td>$d_{t-1}(p') = q \Rightarrow p' &lt; q \Rightarrow p' \not\in B_t^{OPT} \supseteq X_t$</td>
</tr>
<tr>
<td></td>
<td>I5 $p, b_{t-1}(p)$, $u_B$</td>
<td>$p \not\in B_t^{OPT}, b_{t-1}^{-1}(b_{t-1}(p)) = \bot &lt; d(b_{t-1}(p))$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$d(u_B) = \bot$</td>
</tr>
<tr>
<td>b13</td>
<td>I2 q</td>
<td>$q \not\in B_t^{OPT}$</td>
</tr>
<tr>
<td></td>
<td>$b_{t-1}(p)$</td>
<td>$b_{t-1}(p) \leq b_{t-1}^{-1}(q) \Rightarrow q \iff b_{t-1}(p) \not\in B_t^{OPT}$</td>
</tr>
<tr>
<td></td>
<td>I3 $p, b_{t-1}(p), b_{t-1}^{-1}(q)$</td>
<td>$s_t(p) = \bot, b_{t-1}(p) = \bot, b_t(b_{t-1}^{-1}(q)) = u_B \in B_t$</td>
</tr>
<tr>
<td></td>
<td>I4 $p$</td>
<td>$p &lt; q \Rightarrow p \not\in B_t^{OPT} \supseteq X_t$</td>
</tr>
<tr>
<td></td>
<td>${ p' \mid d_{t-1}(p') = p }$</td>
<td>$d_{t-1}(p') = p \Rightarrow p' &lt; p &lt; q \Rightarrow p' \not\in B_t^{OPT} \supseteq X_t$</td>
</tr>
<tr>
<td></td>
<td>${ p' \mid d_{t-1}(p') = q }$</td>
<td>$d_{t-1}(p') = q \Rightarrow p' &lt; q \Rightarrow p' \not\in B_t^{OPT} \supseteq X_t$</td>
</tr>
<tr>
<td></td>
<td>I5 $p, b_{t-1}(p)$, $u_B$</td>
<td>$p \not\in B_t^{OPT}, b_{t-1}^{-1}(b_{t-1}(p)) = \bot &lt; d(b_{t-1}(p))$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$d(u_B) = \bot$</td>
</tr>
<tr>
<td>b14</td>
<td>I2 q</td>
<td>$q \not\in B_t^{OPT}$</td>
</tr>
<tr>
<td></td>
<td>I3 $p, b_{t-1}(p), b_{t-1}^{-1}(q)$</td>
<td>$s_t(p) = \bot, b_{t-1}(p) = \bot, b_t(b_{t-1}^{-1}(q)) = u_B \in B_t$</td>
</tr>
<tr>
<td></td>
<td>I4 $p$</td>
<td>$p &lt; q \Rightarrow p \not\in B_t^{OPT} \supseteq X_t$</td>
</tr>
<tr>
<td></td>
<td>${ p' \mid d_{t-1}(p') = p }$</td>
<td>$d_{t-1}(p') = p \Rightarrow p' &lt; p &lt; q \Rightarrow p' \not\in B_t^{OPT} \supseteq X_t$</td>
</tr>
<tr>
<td></td>
<td>${ p' \mid d_{t-1}(p') = q }$</td>
<td>$d_{t-1}(p') = q \Rightarrow p' &lt; q \Rightarrow p' \not\in B_t^{OPT} \supseteq X_t$</td>
</tr>
<tr>
<td></td>
<td>I5 $p, b_{t-1}(p)$, $u_B$</td>
<td>$p \not\in B_t^{OPT}, b_{t-1}^{-1}(b_{t-1}(p)) = \bot &lt; d(b_{t-1}(p))$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$d(u_B) = \bot$</td>
</tr>
<tr>
<td>b15</td>
<td>I2 q</td>
<td>$q \not\in B_t^{OPT}$</td>
</tr>
<tr>
<td></td>
<td>I3 $p, b_{t-1}(p), b_{t-1}^{-1}(q)$</td>
<td>$s_t(p) = \bot, b_{t-1}(p) = \bot, b_t(b_{t-1}^{-1}(q)) = b_{t-1}(p) \in BC_t$</td>
</tr>
<tr>
<td></td>
<td>I4 $p$</td>
<td>$p &lt; q \Rightarrow p \not\in B_t^{OPT} \supseteq X_t$</td>
</tr>
<tr>
<td></td>
<td>${ p' \mid d_{t-1}(p') = p }$</td>
<td>$d_{t-1}(p') = p \Rightarrow p' &lt; p &lt; q \Rightarrow p' \not\in B_t^{OPT} \supseteq X_t$</td>
</tr>
<tr>
<td></td>
<td>${ p' \mid d_{t-1}(p') = q }$</td>
<td>$d_{t-1}(p') = q \Rightarrow p' &lt; q \Rightarrow p' \not\in B_t^{OPT} \supseteq X_t$</td>
</tr>
<tr>
<td></td>
<td>I5 $p, b_{t-1}(p)$, $u_B$</td>
<td>$p \not\in B_t^{OPT}, b_{t-1}(p) \in BC_t, d(u_B) = \bot$</td>
</tr>
</tbody>
</table>

Figure A.5: Verification of the Invariants I2–I5 for the Cases b12–b15.
Omitted Case Analyses from Theorem 3.15 and 3.16

For each arrival event \( \sigma_t \), \( \Phi_t \leq \Phi_{t-1} \geq 0 \), since \( V(L_t, S_t) - V(L_{t-1}, S_{t-1}) \leq 0 \), \( A(L_t, B_t^{OPT} \setminus B_t^{ONL}) - A(L_{t-1}, B_{t-1}^{OPT} \setminus B_{t-1}^{ONL}) \leq 0 \), \( \sum_{p' \in C_t} v(p') - \sum_{p' \in C_{t-1}} v(p') = 0 \), \( S_t^{ONL} = S_{t-1}^{ONL} \), and \( S_t^{OPT} = S_{t-1}^{OPT} \).

For each send event \( \sigma_t \) of a time step \( \tau \) in which ONL sends a packet \( p \) and OPT sends a packet \( q \), we distinguish the cases presented in the following two sections. In each case, we use the appropriate bounds for \( V(L_t, S_t) - V(L_{t-1}, S_{t-1}) \), \( A(L_t, B_t^{OPT} \setminus B_t^{ONL}) - A(L_{t-1}, B_{t-1}^{OPT} \setminus B_{t-1}^{ONL}) \), and \( \sum_{p' \in C_t} v(p') - \sum_{p' \in C_{t-1}} v(p') \). To determine the right bounds, we have to consider certain facts concerning the packets \( p \) and \( q \), for example, whether \( p = p_f \) or \( p = p_m \), whether \( q \in B_t^{ONL} \), or whether \( p \in C_{t-1} \). Fortunately, most of these facts are clear from the respective case. Otherwise, supplementary notes are given.

In the analysis, we use several simple facts without explicit reference. The following facts can be used for Theorem 3.15 as well as for Theorem 3.16.

- If \( p \in C_{t-1} \), \( p \notin B_{t-1}^{OPT} \geq B_t^{OPT} \) since \( C_{t-1} \subseteq B_{t-1}^{ONL} \setminus B_t^{OPT} \).
- For each \( p' \in B_{t-1}^{ONL} \setminus S_{t-1} \), \( v(p') \leq v(p_f) \) since otherwise \( p' \) would have been added to \( S_{t-1} \) instead of \( p_f \). Also, \( v(p') \leq \delta_{L_{t-1}}(d(p')) \leq \delta_{L_{t-1}}(\tau) \), since a tight time step \( \tau' \geq d(p') \geq \tau \) in \( S_{t-1} \) exists, such that all packets in \( S_{t-1} \) with a deadline smaller or equal than \( \tau' \) have a value of at least \( v(p') \), and hence a level \((\tau', \delta') \in L_{t-1} \) exists with \( \delta' \geq v(p') \).
- For each packet \( p' \) with \( d(p') \geq \tau \),
  \[ \delta_{L_{t-1}}(\tau) \geq m_{L_{t-1}}(p') := \min\{ v(p'), \delta_{L_{t-1}}(d(p')) \} \].

B.1 Case Analysis for Theorem 3.15

In addition to the above facts, we know the following
Appendix B — Omitted Case Analyses from Theorem 3.15 and 3.16

- If \( p = p_f, \beta \cdot \max \{ v(p_f), \delta_{L_{t-1}}^{\max}(\tau) \} \geq v(p_m) + (\beta - 1) \cdot v(s_{t-1}(p_m)) \), due to the strategy.

- If \( p = p_m, \beta \cdot \max \{ v(p_f), \delta_{L_{t-1}}^{\max}(\tau) \} < v(p_m) + (\beta - 1) \cdot v(s_{t-1}(p_m)) \), due to the strategy.

- For each \( q \in S_{t-1}, v(p_m) + (\beta - 1) \cdot v(s_{t-1}(p_m)) \geq v(q) + (\beta - 1) \cdot v(s_{t-1}(q)) \geq v(q) \), due to the strategy.

In the following, we list some (in)equalities that hold for our choices of \( r \) and \( \beta \) and are used frequently.

- \( \frac{r-3}{2} \cdot \beta = -1 \)

- \( \frac{(r-1)(\beta-1)}{2 \beta} < (r-1) \cdot (\beta-1) < 1 \)

- \( (r-1) \cdot \beta = \frac{r+1}{2} \)

- \( \frac{(r+1)(\beta-1)}{2} = 1 \)

- \( \frac{r+1}{2} \cdot \beta = 2 + \frac{r-1}{2} \)

- \( \frac{r-3}{2} \cdot (\beta-1) = -\frac{r-1}{2} \)

- \( \frac{(r-3)(\beta-1)}{2} + 1 = (r-1) \cdot (\beta-1) = -\frac{r-3}{2} < 1 \)

Finally, we can distinguish the following cases to show \( \Phi_t - \Phi_{t-1} \geq 0 \):

(a) \( p = p_f = q \)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(p) - v(q) - 0 - m_{L_{t-1}}(p_f) + \delta_{L_{t-1}}^{\max}(\tau) - (r-1)/2 \cdot v(p_f) \\
\geq r \cdot v(p) - v(q) - (r-1)/2 \cdot v(p_f) \\
= (r-1 - (r-1)/2) \cdot v(p_f) \geq 0
\]

(b) \( p = p_f \in C_{t-1} \) and \( q \in B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}} \)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(p_f) - v(q) + v(q) - m_{L_{t-1}}(q) - m_{L_{t-1}}(p_f) + \delta_{L_{t-1}}^{\max}(\tau) \\
- (r-1)/2 \cdot (v(p_f) + v(p_f)) \\
= v(p_f) - m_{L_{t-1}}(q) - m_{L_{t-1}}(p_f) + \delta_{L_{t-1}}^{\max}(\tau) \\
\geq v(p_f) - m_{L_{t-1}}(p_f) \geq 0
\]

(c) \( p = p_f \notin C_{t-1} \) and \( q \in B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}} \)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(p_f) - v(q) - v(p_f) + m_{L_{t-1}}(p_f) + v(q) \\
- m_{L_{t-1}}(q) - m_{L_{t-1}}(p_f) + \delta_{L_{t-1}}^{\max}(\tau) - (r-1)/2 \cdot v(p_f) \\
= (r-1 - (r-1)/2) \cdot v(p_f) - m_{L_{t-1}}(q) + \delta_{L_{t-1}}^{\max}(\tau) \\
\geq (r-1)/2 \cdot v(p_f) \geq 0
\]
B.1 Case Analysis for Theorem 3.15

(d) \( p = p_f \in C_{l-1} \) and \( q \in B^\text{ONL}_{l-1} \setminus S_{l-1} \)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(p_f) - v(q) - 0 - m_{L_{t-1}}(p_f) \\
+ \delta^\text{max}_{L_{t-1}}(\tau) - (r - 1)/2 \cdot v(p_f) + v(p_f)) \\
\geq r \cdot v(p_f) - v(p_f) - (r - 1)/2 \cdot (v(p_f) + v(p_f)) = 0
\]

(e) \( p = p_f \not\in C_{l-1} \) and \( q \in B^\text{ONL}_{l-1} \setminus S_{l-1} \)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(p_f) - v(q) - v(p_f) + m_{L_{t-1}}(p_f) \\
+ m_{L_{t-1}}(p_f) \\
= r \cdot v(p_f) - v(q) - v(p_f) + \delta^\text{max}_{L_{t-1}}(\tau) - (r - 1)/2 \cdot v(p_f) \\
\geq r \cdot v(p_f) - v(p_f) - (r - 1)/2 \cdot v(p_f) \geq 0
\]

(f) \( p = p_f \) and \( q \in S_{l-1} \setminus \{p\} \)

Note that \( p \not\in B^\text{OPT}_t \), since \( p < q \).

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(p_f) - v(q) - 0 - m_{L_{t-1}}(p_f) + \delta^\text{max}_{L_{t-1}}(\tau) \\
-(r - 1)/2 \cdot (2 \cdot v(p_f) - v(q)) \\
= r \cdot v(p_f) - v(q) - m_{L_{t-1}}(p_f) + \delta^\text{max}_{L_{t-1}}(\tau) \\
\geq v(p_f) + (r - 3)/2 \cdot v(q) - m_{L_{t-1}}(p_f) + \delta^\text{max}_{L_{t-1}}(\tau) \\
= v(p_f) + (r - 3)/2 \cdot \beta \cdot \max\{v(p_f), \delta^\text{max}_{L_{t-1}}(\tau)\} - m_{L_{t-1}}(p_f) + \delta^\text{max}_{L_{t-1}}(\tau) \\
\geq v(p_f) - \max\{v(p_f), \delta^\text{max}_{L_{t-1}}(\tau)\} - \min\{v(p_f), \delta^\text{max}_{L_{t-1}}(\tau)\} + \delta^\text{max}_{L_{t-1}}(\tau) \\
\geq v(p_f) - v(p_f) - \delta^\text{max}_{L_{t-1}}(\tau) + \delta^\text{max}_{L_{t-1}}(\tau) = 0
\]

(g) \( p = p_m = q \)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) - 0 - m_{L_{t-1}}(p_m) + \delta^\text{max}_{L_{t-1}}(\tau) \\
+ \delta^\text{max}_{L_{t-1}}(\tau) - (r - 1)/2 \cdot v(p_f) \\
\geq r \cdot v(p_m) - v(p_m) + v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_f) - (r - 1)/2 \cdot v(p_f) \\
\geq r \cdot v(p_m) - v(p_m) + v(s_{t-1}(p_m)) - v(p_f) - (r - 1)/2 \cdot v(p_f) \\
\geq (r - 1) \cdot v(p_m) + (r - 1) \cdot (\beta - 1) \cdot v(s_{t-1}(p_m)) - (r - 1)/2 \cdot v(p_f) \\
\geq (r - 1) \cdot \beta \cdot \max\{v(p_f), \delta^\text{max}_{L_{t-1}}(\tau)\} - (r + 1)/2 \cdot v(p_f) \\
\geq (r - 1) \cdot \beta \cdot v(p_f) - (r + 1)/2 \cdot v(p_f) = 0
\]

(h) \( p = p_m \in C_{l-1} \) and \( q \in B^\text{OPT}_{l-1} \setminus B^\text{ONL}_{l-1} \)
\[ \Phi_t - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) + v(q) - m_{L_{t-1}}(p_m) + v(s_{t-1}(p_m)) \]
\[ - m_{L_{t-1}}(p_f) + \sigma^\text{max}_{L_{t-1}}(\tau) - (r - 1)/2 \cdot (v(p_m) + v(p_f)) \]
\[ = r \cdot v(p_m) - m_{L_{t-1}}(q) - m_{L_{t-1}}(p_m) + v(s_{t-1}(p_m)) \]
\[ - m_{L_{t-1}}(p_f) + \sigma^\text{max}_{L_{t-1}}(\tau) - (r - 1)/2 \cdot (v(p_m) + v(p_f)) \]
\[ \geq (r + 1)/2 \cdot v(p_m) - m_{L_{t-1}}(p_m) + v(s_{t-1}(p_m)) \]
\[ - m_{L_{t-1}}(p_f) - (r - 1)/2 \cdot v(p_f) \]
\[ = (r + 1)/2 \cdot (v(p_m) + (\beta - 1) \cdot v(s_{t-1}(p_m))) - m_{L_{t-1}}(p_m) \]
\[ - m_{L_{t-1}}(p_f) - (r - 1)/2 \cdot v(p_f) \]
\[ \geq (r + 1)/2 \cdot \beta \cdot \max\{v(p_f), \sigma^\text{max}_{L_{t-1}}(\tau)\} - m_{L_{t-1}}(p_m) \]
\[ - m_{L_{t-1}}(p_f) - (r - 1)/2 \cdot v(p_f) \]
\[ = 2 \cdot \max\{v(p_f), \sigma^\text{max}_{L_{t-1}}(\tau)\} + (r - 1)/2 \cdot \max\{v(p_f), \sigma^\text{max}_{L_{t-1}}(\tau)\} \]
\[ - m_{L_{t-1}}(p_f) - (r - 1)/2 \cdot v(p_f) \]
\[ \geq 2 \cdot \sigma^\text{max}_{L_{t-1}}(\tau) + (r - 1)/2 \cdot v(p_f) - m_{L_{t-1}}(p_m) - m_{L_{t-1}}(p_f) \]
\[ -(r - 1)/2 \cdot v(p_f) \geq 0 \]

(i) \( p = p_m \in C_t \) and \( q \in B^\text{QNL}_t \setminus S_t \)

\[ \Phi_t - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) - 0 - m_{L_{t-1}}(p_m) + v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_f) \]
\[ + \sigma^\text{max}_{L_{t-1}}(\tau) - (r - 1)/2 \cdot (v(p_m) + v(p_f)) \]
\[ \geq r \cdot v(p_m) - m_{L_{t-1}}(q) - m_{L_{t-1}}(p_m) + v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_f) \]
\[ + \sigma^\text{max}_{L_{t-1}}(\tau) - (r - 1)/2 \cdot (v(p_m) + v(p_f)) \geq 0 \]

The last inequality follows as shown in the previous case \( (p = p_m \in C_{t-1} \) and \( q \in B^\text{OPT}_{t-1} \setminus B^\text{QNL}_{t-1} ) \).

(j) \( p = p_m \in C_t \) and \( q \in S_t \setminus \{p\} \)
B.1 Case Analysis for Theorem 3.15

\( \Phi_t - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) - 0 - m_{L_{t-1}}(p_m) + v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_f) \\
+ \delta_{\max}^{\max}(\tau) - (r - 1)/2 \cdot (v(p_m) + v(p_f) + v(s_{t-1}(q)) - v(q)) \\
\geq r \cdot v(p_m) - v(q) - m_{L_{t-1}}(p_m) + v(s_{t-1}(p_m)) \\
- (r - 1)/2 \cdot (v(p_m) + v(p_f) + v(s_{t-1}(q)) - v(q)) \\
= (r + 1)/2 \cdot v(p_m) + (r - 3)/2 \cdot v(q) - m_{L_{t-1}}(p_m) + v(s_{t-1}(p_m)) \\
- (r - 1)/2 \cdot (v(p_f) + v(s_{t-1}(q))) \\
= (r + 1)/2 \cdot v(p_m) + (r - 3)/2 \cdot v(q) - (r - 1)/2 \cdot (v(p_m) + v(s_{t-1}(q))) \\
- m_{L_{t-1}}(p_m) + v(s_{t-1}(p_m)) - (r - 1)/2 \cdot v(p_f) \\
= (r - 1) \cdot v(p_m) - (r - 3)/2 \cdot v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_m) \\
- (r - 1)/2 \cdot v(p_f) \\
\geq (r - 1) \cdot (\beta - 1) \cdot v(p_f) - m_{L_{t-1}}(p_m) - (r - 1)/2 \cdot v(p_f) \\
\geq 0

(k) \quad p = p_m \notin C_t \text{ and } q \in B_t^{\text{OPT}} \setminus B_t^{\text{ONL}}

\( \Phi_t - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) - v(p_m) + m_{L_{t-1}}(p_m) + v(q) - m_{L_{t-1}}(q) \\
-m_{L_{t-1}}(p_m) + v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_f) + \delta_{\max}^{\max}(\tau) \\
- (r - 1)/2 \cdot v(p_f) \\
= (r - 1) \cdot v(p_m) - m_{L_{t-1}}(q) + v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_f) \\
+ \delta_{\max}^{\max}(\tau) - (r - 1)/2 \cdot v(p_f) \\
\geq (r - 1) \cdot v(p_m) + v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_f) - (r - 1)/2 \cdot v(p_f)

The last inequality follows as shown in the case \( p = p_m = q \).

(l) \quad p = p_m \notin C_t \text{ and } q \in B_t^{\text{ONL}} \setminus S_t

\( \Phi_t - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) - v(p_m) + m_{L_{t-1}}(p_m) - m_{L_{t-1}}(p_m) \\
v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_f) + \delta_{\max}^{\max}(\tau) - (r - 1)/2 \cdot v(p_f) \\
r \cdot v(p_m) - v(q) - v(p_m) + v(s_{t-1}(p_m)) \\
- m_{L_{t-1}}(p_f) + \delta_{\max}^{\max}(\tau) - (r - 1)/2 \cdot v(p_f) \\
\geq (r - 1) \cdot v(p_m) - m_{L_{t-1}}(q) + v(s_{t-1}(p_m)) \\
- m_{L_{t-1}}(p_f) + \delta_{\max}^{\max}(\tau) - (r - 1)/2 \cdot v(p_f) \geq 0

The last inequality follows as shown in the previous case (\( p = p_m \notin C_t \text{ and } q \in B_t^{\text{OPT}} \setminus B_t^{\text{ONL}} \)).
Appendix B — Omitted Case Analyses from Theorem 3.15 and 3.16

(m) \( p = p_m \notin C_t \) and \( q \in S_{t-1} \setminus \{ p \} \) 

Note that \( v(p_f) \leq \max \{ v(p_f), \delta_{L_{t-1}}^{\max}(\tau) \} \leq (\beta - 1) \cdot v(s_{t-1}(p_m)) / \beta \).

\[
\Phi_0 - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) - v(p_m) + m_{L_{t-1}}(p_m) - m_{L_{t-1}}(p_m) + v(s_{t-1}(p_m)) \\
\geq r \cdot v(p_m) - v(q) - v(p_m) + v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_f) \\
+ \delta_{L_{t-1}}^{\max}(\tau) - (r - 1)/2 \cdot (v(p_f) + v(s_{t-1}(q)) - v(q)) \\
\geq r \cdot v(p_m) - v(q) - v(p_m) + v(s_{t-1}(p_m)) \\
-(r - 1)/2 \cdot (v(p_f) + v(s_{t-1}(q)) - v(q)) \\
= (r - 1) \cdot v(p_m) + (r - 3)/2 \cdot v(q) + v(s_{t-1}(p_m)) \\
-(r - 1)/2 \cdot (v(p_f) + v(s_{t-1}(q))) \\
= (r - 1) \cdot v(p_m) + (r - 3)/2 \cdot (v(q) + (\beta - 1) \cdot v(s_{t-1}(q))) \\
+ v(s_{t-1}(p_m)) - (r - 1)/2 \cdot v(p_f) \\
\geq (r - 1) \cdot v(p_m) + (r - 3)/2 \cdot v(p_m) + \beta \cdot v(s_{t-1}(p_m)) \\
-(r - 1)/2 \cdot (v(p_f) - (\beta - 1) \cdot v(s_{t-1}(p_m))) / \beta \\
\geq (r - 1) \cdot v(p_m) + (r - 3)/2 \cdot v(p_m) \\
-(r - 1)/2 \cdot ((v(p_m) + (\beta - 1) \cdot v(s_{t-1}(p_m))) / \beta \\
-(\beta - 1) \cdot v(s_{t-1}(p_m))) / \beta \\
\geq (r - 1) \cdot v(p_m) + (r - 3)/2 \cdot v(p_m) - (r - 1)/2 \cdot v(p_m) / \beta = 0
\]

B.2 Case Analysis for Theorem 3.16

In addition to the previously mentioned facts, we know the following for our memoryless algorithm.

- If \( p = p_f \), \( \beta \cdot v(p_f) \geq v(p_m) + v(s_{t-1}(p_m)) / 2 \), due to the strategy.
- If \( p = p_m \), \( \beta \cdot v(p_f) < v(p_m) + v(s_{t-1}(p_m)) / 2 \), due to the strategy.
- For each \( q \in S_{t-1} \), \( v(p_m) + v(s_{t-1}(p_m)) / 2 \geq v(q) + v(s_{t-1}(q)) / 2 \geq v(q) \), due to the strategy.

For \( \beta := 4 \cos((\pi - \arccos(3\sqrt{3}/16)) / 3) / \sqrt{3} \), \( r := (2\beta^2 + \beta - 5) / 2 \), and \( \alpha := (\beta^2 - 3) / 2 \), the following (in)equalities hold.

- \( (r - 1) \cdot \beta - (1 + \alpha) = r - 2\alpha - 1 > 0 \)
- \( r - 2\alpha + (\alpha - 1) \cdot \beta = 0 \)
- \( (r - \alpha - 1)/2 < 1 \)
- \( (r - \alpha - 1) \cdot \beta - 1 - \alpha = 0 \)
- \( (\alpha - 1)/2 < -\alpha \)

88
B.2 Case Analysis for Theorem 3.16

- \((\alpha - 1)/2 + 1 > (r - 1)/2 > (r - 2 + \alpha)/2\)
- \((r - 2 + \alpha) \cdot \beta - \alpha = 0\)

Finally, we can distinguish the following cases to show \(\Phi_t - \Phi_{t-1} \geq 0\):

(a) \(p = pf = q\)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(p) - v(q) - 0 - m_{L_{t-1}}(pf) + \delta_{L_{t-1}}^{\max}(\tau) - \alpha \cdot v(pf) \\
\geq r \cdot v(p) - v(q) - \alpha \cdot v(pf) \\
= (r - 1 - \alpha) \cdot v(pf) \geq 0
\]

(b) \(p = pf \in C_{t-1} \) and \(q \in B_{t-1}^{\OPT} \setminus B_{t-1}^{\ONL}\)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(pf) - v(q) + v(q) - m_{L_{t-1}}(pf) + \delta_{L_{t-1}}^{\max}(\tau) \\
- \alpha \cdot (v(pf) + v(pf)) \\
= (r - 2\alpha) \cdot v(pf) - m_{L_{t-1}}(pf) \geq (r - 2\alpha - 1) \cdot v(pf) \geq 0
\]

(c) \(p = pf \notin C_{t-1} \) and \(q \in B_{t-1}^{\ONL} \setminus B_{t-1}^{\OPT}\)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(pf) - v(q) - v(pf) + m_{L_{t-1}}(pf) + v(q) \\
- m_{L_{t-1}}(q) - m_{L_{t-1}}(pf) + \delta_{L_{t-1}}^{\max}(\tau) \\
= (r - 1 - \alpha) \cdot v(pf) - m_{L_{t-1}}(q) + \delta_{L_{t-1}}^{\max}(\tau) \\
\geq (r - 1 - \alpha) \cdot v(pf) \geq 0
\]

(d) \(p = pf \in C_{t-1} \) and \(q \in B_{t-1}^{\ONL} \setminus S_{t-1}\)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(pf) - v(q) - 0 - m_{L_{t-1}}(pf) + \delta_{L_{t-1}}^{\max}(\tau) \\
- \alpha \cdot (v(pf) + v(pf)) \\
\geq r \cdot v(pf) - v(q) - \alpha \cdot (v(pf) + v(pf)) = (r - 2\alpha - 1) \cdot v(pf) \geq 0
\]

(e) \(p = pf \notin C_{t-1} \) and \(q \in B_{t-1}^{\ONL} \setminus S_{t-1}\)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(pf) - v(q) - v(pf) + m_{L_{t-1}}(pf) - m_{L_{t-1}}(pf) + \delta_{L_{t-1}}^{\max}(\tau) - \alpha \cdot v(pf) \\
= r \cdot v(pf) - v(q) - \alpha \cdot v(pf) \geq 0
\]

(f) \(p = pf \) and \(q \in S_{t-1} \triangle \{p\}\)

Note that \(p \notin B_{t-1}^{\OPT} \) since \(p < q \) and \(q \in S_{t-1} \triangle B_{t-1}^{\ONL} \).

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(pf) - v(q) - 0 - m_{L_{t-1}}(pf) + \delta_{L_{t-1}}^{\max}(\tau) \\
- \alpha \cdot (2 \cdot v(pf) - v(q)) \\
\geq (r - 2\alpha) \cdot v(pf) + (\alpha - 1) \cdot v(q) \\
\geq (r - 2\alpha) \cdot v(pf) + (\alpha - 1) \cdot \beta \cdot v(pf) \\
= (r - 2\alpha + (\alpha - 1)\beta) \cdot v(pf) = 0
\]
(g) \( p = p_m = q \)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) - 0 - m_{L_t-1}(p_m) + v(s_{t-1}(p_m)) - m_{L_t-1}(p_f) + \delta_{L_t-1}^{\max}(\tau) - \alpha \cdot v(p_f)
\]

\[
\geq r \cdot v(p_m) - v(p_m) + v(s_{t-1}(p_m)) - m_{L_t-1}(p_f) - \alpha \cdot v(p_f)
\]

\[
\geq (r - 1) \cdot v(p_m) + (r - 1) \cdot v(s_{t-1}(p_m))/2 - (1 + \alpha) \cdot v(p_f)
\]

\[
\geq (r - 1) \cdot \beta \cdot v(p_f) - (1 + \alpha) \cdot v(p_f) \geq 0
\]

(h) \( p = p_m \in C_{t-1} \) and \( q \in B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}} \)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) - m_{L_t-1}(q) - m_{L_t-1}(p_m) + v(s_{t-1}(p_m))
\]

\[
- m_{L_t-1}(p_f) + \delta_{L_t-1}^{\max}(\tau) - \alpha \cdot (v(p_m) + v(p_f))
\]

\[
= r \cdot v(p_m) - m_{L_t-1}(q) - m_{L_t-1}(p_m) + v(s_{t-1}(p_m))
\]

\[
- m_{L_t-1}(p_f) + \delta_{L_t-1}^{\max}(\tau) - \alpha \cdot (v(p_m) + v(p_f))
\]

\[
\geq (r - \alpha) \cdot v(p_m) - m_{L_t-1}(p_m) + v(s_{t-1}(p_m))
\]

\[
- m_{L_t-1}(p_f) - \alpha \cdot v(p_f)
\]

\[
\geq (r - \alpha) \cdot v(p_m) - v(p_m) + v(s_{t-1}(p_m))
\]

\[
- v(p_f) - \alpha \cdot v(p_f)
\]

\[
\geq (r - \alpha - 1) \cdot (v(p_m) + v(s_{t-1}(p_m))/2)
\]

\[
- v(p_f) - \alpha \cdot v(p_f)
\]

\[
\geq (r - \alpha - 1) \cdot \beta \cdot v(p_f) - v(p_f) - \alpha \cdot v(p_f) = 0
\]

(i) \( p = p_m \in C_t \) and \( q \in B_t^{\text{ONL}} \setminus S_t \)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) - 0 - m_{L_t-1}(p_m) + v(s_{t-1}(p_m)) - m_{L_t-1}(p_f)
\]

\[
+ \delta_{L_t-1}^{\max}(\tau) - \alpha \cdot (v(p_m) + v(p_f))
\]

\[
\geq r \cdot v(p_m) - m_{L_t-1}(q) - m_{L_t-1}(p_m) + v(s_{t-1}(p_m)) - m_{L_t-1}(p_f)
\]

\[
+ \delta_{L_t-1}^{\max}(\tau) - \alpha \cdot (v(p_m) + v(p_f)) \geq 0
\]

The last inequality follows as shown in the previous case \((p = p_m \in C_{t-1} \) and \( q \in B_{t-1}^{\text{OPT}} \setminus B_{t-1}^{\text{ONL}} \)).

(j) \( p = p_m \in C_t \) and \( q \in S_t \setminus \{p\} \)

90
B.2 Case Analysis for Theorem 3.16

\[ \Phi_t - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) - m_{L_{t-1}}(p_m) + m_{L_{t-1}}(p_f) + \delta_{L_{t-1}}^\max(\tau) - \alpha \cdot (v(p_m) + v(p_f)) + v(s_{t-1}(p_m)) - m_{L_{t-1}}(q) - m_{L_{t-1}}(p_f) \]

\[ \geq r \cdot v(p_m) - v(q) - m_{L_{t-1}}(p_m) + m_{L_{t-1}}(p_f) - \alpha \cdot (v(p_m) + v(p_f)) + v(s_{t-1}(q)) - v(q) \]

\[ \geq r \cdot v(p_m) - v(q) + v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_f) - \alpha \cdot (v(p_m) + v(p_f)) + v(s_{t-1}(q)) - v(q) \]

\[ = (r - \alpha) \cdot v(p_m) + (\alpha - 1) \cdot v(q) + v(s_{t-1}(p_m)) - v(p_f) \]

\[ - \alpha \cdot (v(p_f) + v(s_{t-1}(q))) \]

\[ \geq (r - \alpha) \cdot v(p_m) + (\alpha - 1) \cdot v(q) + v(s_{t-1}(q))/2 \]

\[ + v(s_{t-1}(p_m)) - v(p_f) - \alpha \cdot v(p_f) \]

\[ \geq (r - \alpha) \cdot v(p_m) + (\alpha - 1) \cdot v(p_m) + v(s_{t-1}(p_m))/2 \]

\[ + v(s_{t-1}(p_m)) - v(p_f) - \alpha \cdot v(p_f) \]

\[ = (r - 1) \cdot v(p_m) + ((\alpha - 1)/2 + 1) \cdot v(s_{t-1}(p_m)) \]

\[ - (\alpha + 1) \cdot v(p_f) \]

\[ \geq (r - 1) \cdot v(p_m) + v(s_{t-1}(p_m))/2 \]

\[ - (\alpha + 1) \cdot v(p_f) \]

\[ \geq (r - 1) \cdot \beta \cdot v(p_f) - (\alpha + 1) \cdot v(p_f) \geq 0 \]

(k) \( p = p_m \notin C_t \) and \( q \in B_t^{\text{OPT}} \setminus B_t^{\text{ONL}} \)

\[ \Phi_t - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) - m_{L_{t-1}}(p_m) + v(q) - m_{L_{t-1}}(q) \]

\[ - m_{L_{t-1}}(p_m) + v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_f) + \delta_{L_{t-1}}^\max(\tau) - \alpha \cdot v(p_f) \]

\[ = r \cdot v(p_m) - v(p_m) - m_{L_{t-1}}(p_f) + \delta_{L_{t-1}}^\max(\tau) - \alpha \cdot v(p_f) \]

\[ + \delta_{L_{t-1}}^\max(\tau) - \alpha \cdot v(p_f) \]

\[ \geq r \cdot v(p_m) - v(p_m) - m_{L_{t-1}}(q) + v(s_{t-1}(p_m)) \]

\[ - m_{L_{t-1}}(p_f) + \delta_{L_{t-1}}^\max(\tau) - \alpha \cdot v(p_f) \geq 0 \]

The last inequality follows as shown in the case \( p = p_m = q \).

(l) \( p = p_m \notin C_t \) and \( q \in B_t^{\text{ONL}} \setminus S_t \)

\[ \Phi_t - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) - m_{L_{t-1}}(p_m) + m_{L_{t-1}}(p_f) + \delta_{L_{t-1}}^\max(\tau) - \alpha \cdot v(p_f) \]

\[ + v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_f) + \delta_{L_{t-1}}^\max(\tau) - \alpha \cdot v(p_f) \]

\[ = r \cdot v(p_m) - v(p_m) - v(q) - m_{L_{t-1}}(p_f) + \delta_{L_{t-1}}^\max(\tau) - \alpha \cdot v(p_f) \]

\[ - m_{L_{t-1}}(p_f) + \delta_{L_{t-1}}^\max(\tau) - \alpha \cdot v(p_f) \geq 0 \]

The last inequality follows as shown in the previous case \( p = p_m \notin C_t \) and \( q \in B_t^{\text{OPT}} \setminus B_t^{\text{ONL}} \).
Appendix B — Omitted Case Analyses from Theorem 3.15 and 3.16

(m) \( p = p_m \not\in C_t \) and \( q \in S_t \setminus \{p\} \)

\[
\Phi_t - \Phi_{t-1} \geq r \cdot v(p_m) - v(q) - v(p_m) + m_{L_{t-1}}(p_m) - m_{L_{t-1}}(p_m) + v(s_{t-1}(p_m))
- m_{L_{t-1}}(p_f) + \delta_{L_{t-1}}^{\text{max}}(\tau) - \alpha \cdot (v(p_f) + v(s_{t-1}(q)) - v(q))
\]
\[
= r \cdot v(p_m) - v(q) - v(p_m) + v(s_{t-1}(p_m)) - m_{L_{t-1}}(p_f)
+ \delta_{L_{t-1}}^{\text{max}}(\tau) - \alpha \cdot (v(p_f) + v(s_{t-1}(q)) - v(q))
\]
\[
\geq r \cdot v(p_m) - v(q) - v(p_m) + v(s_{t-1}(p_m))
- \alpha \cdot (v(p_f) + v(s_{t-1}(q)) - v(q))
\]
\[
= (r - 1) \cdot v(p_m) + (\alpha - 1) \cdot v(q) + v(s_{t-1}(p_m))
- \alpha \cdot (v(p_f) + v(s_{t-1}(q)) - v(q))
\]
\[
\geq (r - 1) \cdot v(p_m) + (\alpha - 1) \cdot (v(q) + v(s_{t-1}(q))/2)
+ v(s_{t-1}(p_m)) - \alpha \cdot v(p_f)
\]
\[
\geq (r - 1) \cdot v(p_m) + (\alpha - 1) \cdot (v(p_m) + v(s_{t-1}(p_m))/2)
+ v(s_{t-1}(p_m)) - \alpha \cdot v(p_f)
\]
\[
= (r - 2 + \alpha) \cdot v(p_m)
+ ((\alpha - 1)/2 + 1) \cdot v(s_{t-1}(p_m)) - \alpha \cdot v(p_f)
\]
\[
\geq (r - 2 + \alpha) \cdot v(p_m)
+ (r - 2 + \alpha) \cdot v(s_{t-1}(p_m))/2 - \alpha \cdot v(p_f)
\]
\[
\geq (r - 2 + \alpha) \cdot \beta \cdot v(p_f) - \alpha \cdot v(p_f) = 0
\]
Bibliography


Bibliography


Name        Matthias Englert
Geburtsort  Warendorf, Deutschland

Beruflicher Werdegang
Seit 2/2005  Wissenschaftlicher Mitarbeiter
der Nachwuchsgruppe *Flexible Online-Algorithmen*
im Rahmen des Aktionsplans Informatik der DFG
Fachbereich Mathematik, Informatik und Naturwissenschaften
RWTH Aachen

Studium
10/1999 – 1/2005  Studium der Informatik mit Nebenfach Mathematik
                 Universität Dortmund
Abschluss: Diplom-Informatiker

Schulbildung
1986 – 1999  Freie Waldorfschule Münster
Abschluss: Allgemeine Hochschulreife