Resource management in cable access networks

Resource Management in Cable Access Networks

ISBN 978-90-74445-83-2

Cover design by Bertina Senders, B-Design Grafische Vormgeving

The work described in this thesis has been carried out at the Philips Research Laboratories Eindhoven, The Netherlands, as part of the Philips Research Programme.

Resource Management in Cable Access Networks

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van
de Rector Magnificus, prof.dr.ir. C.J. van Duijn,
voor een commissie aangewezen door het College
voor Promoties in het openbaar te verdedigen op
donderdag 13 maart 2008 om 16.00 uur

door

Serverius Petrus Paulus Pronk

geboren te Vught

Dit proefschrift is goedgekeurd door de promotor:

prof.dr. E.H.L. Aarts

Copromotor:
dr.ir. J.H.M. Korst

Contents

Preface

During the first sixteen years of my career at the Philips Research Laboratories, I had never seriously considered doing a Ph.D. It was not until I was finishing a book on multimedia systems together with Jan Korst, that I realized that I might as well write a thesis.

The amount of work that I had done over the more recent years would easily lend itself for packaging part of it into a single volume. At least, that is what I thought initially. It appeared that my work was scattered over various fields, which complicated the composition of a nicely integrated thesis. I think its final title broadly covers the contained subjects, with the emphasis on broad.

Many people have somehow contributed to my development in my working environment, either directly or indirectly, and I hereby express my gratitude to them all. I do wish to mention some people in particular.

In the first place, I would like to thank Jan Korst, with whom I have worked together for more than ten years now. His creativity in finding solutions to technical problems is an important source of inspiration for me. I was glad to have him as my copromotor, and hope that our cooperation will continue for years to come.

Next, I would like to thank my promotor Emile Aarts. I will remember the meetings with Emile and Jan, which often surpassed the actual context of my Ph.D. work, with joy.

I am greatly indebted to Ludo Tolhuizen, Ronald Rietman, and Jan Korst. Chapter 2 is partly based on joint work with Ludo and Chapters 3 and 4 are joint work with Ronald and Jan, respectively. I thank Wim Verhaegh for providing me part of the introduction of Chapter 7.

I am also grateful to the many other people with whom I have worked together over the years in the context of my thesis, of which I mention in particular Carel-Jan van Driel, Peter van Grinsven, Dee Denteneer, Ewa Hekstra-Nowacka, and Wim Verhaegh.

Furthermore, I thank the management of Philips Research, and especially Fred Snijders, Maurice Groten, Reinder Haakma, Fred Boekhorst, and Willem

Jonker, for providing me the opportunity to do the research and the time to write my thesis.

Finally, I would like to thank my partner Ingrid and our children Nini and Jurre, for their support and endurance over the past years, but also because they helped me shape and live my life outside research.

March 2008 Verus Pronk

1

Introduction

Multimedia pertains to the interactive use of audio/video material, possibly enriched with text and graphics. In this context, video on demand (VOD) is one of the most demanding services because of the huge storage and bandwidth requirements as well as real-time requirements in an interactive setting. The prospect of delivering VOD with instant access, interactivity, and browsing possibilities, comparable to that offered by a conventional video cassette recorder (VCR), to the homes of millions has attracted extensive interest from academia as well as industry. In the past decades we have witnessed significant improvements in the possibilities and the ease of interaction with multimedia material. Gradually, the technical and commercial hurdles were overcome to put the user more in control of what, where, and when to enjoy. This development is progressing along different routes.

First of all, the availability of digital audio and video compression algorithms, real-time hardware implementations, and associated standards allow efficient storage and transmission of high-quality audio/video (AV) content. In addition, the recent upgrade of access networks, such as cable and telephony networks, provides broadband access to interactive services from the customer premises. As a result, large collections of AV material are becoming accessible via the Internet for on-demand viewing.

Another development is that in-home storage of AV material is greatly improving, both in terms of ease of use and storage capacity. VCRs are being replaced on a large scale by storage systems, called personal video recorders (PVRs), based on optical and magnetic or hard disks, and electronic program guides (EPGs) simplify the selection of TV programs for recording. Automatic recording based on recommender technology of interesting broadcast programs has recently been introduced in the market. Modern hard disk technology allows the recording of a considerable number of programs in parallel on a single disk, and playing back a program is possible while its recording is still in progress. The sizes of currently available hard disks allow the storage of hundreds of video files so that they are useful for maintaining a sizable collection at home, thus providing VOD in-the-small.

Current solutions for providing VOD via the Internet rely on best-effort services, meaning that there is no guaranteed level of quality, neither in terms of timely content transmission to allow uninterrupted viewing, nor in terms of (interactive) response times. The protocols used for the transmission of data over the Internet, that is, the Internet protocol (IP) and the transmission control protocol (TCP) provide guaranteed data delivery, but without real-time guarantees. Also the real-time transport protocol (RTP) does not provide any real-time guarantees.

The existing hardware overkill and the possibility to adapt compression and transmission rates to the available bandwidth alleviates this problem, but a temporarily congested network may cause service disruption, or at least deterioration, which may take the form of, e.g., hickups in the display of the video or a slow response to user actions. When the take-up of such a bandwidth-intensive service increases, these situations will become unavoidable and appropriate resource management tools such as admission control and reservation protocols as well as scheduling algorithms are required to provide a guaranteed quality of service.

The successor of the current IP/TCP protocol suite, called IP Version 6 (IPV6), does provide the means to provide a guaranteed quality of service. Furthermore, an access network like a cable access network provides a controllable environment wherein such protocols and algorithms can indeed be implemented. By placing a VOD server at a central point in the access network, it becomes possible to provide the service with the appropriate quality.

The work reported on in this thesis concerns a number of scheduling and resource management problems in the context of VOD over cable access networks. Before introducing them in Section 1.4, we first give a concise introduction to cable access networks in Section 1.1 and video on demand in Section 1.2, and sketch in Section 1.3 the major scheduling and resource man-

agement issues that play a role. We end this chapter with a short overview of the organization of this thesis in Section 1.5.

1.1 Cable access networks

During the nineties of the last century, the realization of the information superhighway dictated the need to connect the homes to backbone networks via broadband links. This problem was known as the last-mile problem. The already existing telephone and cable access networks provided the required infrastructure only partly. Hence, possible implementations, applications, and migration scenarios for these networks were surveyed; see Bisdikian, Maruyama, Seidman & Serpranos [1996], Van Driel, Van Grinsven, Pronk & Snijders [1997], and Dutta-Roy [1999].

Cable access networks, better known as CATV (community antenna television) networks, were originally designed for broadcasting analogue video signals. These networks had a tree structure where, at the root, a controller called the head-end (HE) broadcast incoming TV signals from a variety of sources over the network on downstream channels to the individual homes, amplified along the way to retain sufficient signal strength.

Over the past fifteen years, CATV networks have been upgraded to provide two-way broadband communication. This upgrading includes replacing parts of the coaxial cable near the HE by fiber-optic cable, organized in rings, extending the functionality of the HE, and installing return amplifiers in the upstream path from the homes to the HE. Access to such a hybrid fiber-coax (HFC) network at the homes requires a cable modem (CM), which separates this public network from the in-home, private networks and provides the necessary functionality for the support of these services.

To ensure interoperability between the HE and a possibly multi-vendor set of CMs, several standards have become available, of which the DOCSIS [MCNS Holdings, 1999] and DVB/DAVIC [Digital video broadcasting, 1999] standards are the two most prominent ones. A third standardization body, the IEEE 802.14 working group, was dismantled in 2000. The drafts remain accessible in their archive [IEEE 802.14 working group, 2000].

These standards describe in great detail the physical (PHY) and medium access control (MAC) layers, covering the electrical characteristics, modulation and error-correction schemes, the message formats and messaging protocols, access protocols and, for DOCSIS in particular, a multitude of quality-of-service classes for the support of more advanced services, such as constant bit-rate and real-time polling services. In these standards, there is ample free-

dom in the design and operation of a system to optimize performance, subject to channel impairments and higher-layer protocol requirements.

A HE supports a number of frequency-separated downstream channels with a bandwidth of up to 60 Mbit/s each, where 'M' stands for 2^{20}. To each downstream channel, a number of upstream channels are associated, each with a bit rate between 256 kbit/s and 20 Mbit/s. Frequency separation as well as time-division-multiple-access (TDMA) and code-division-multiple-access (CDMA) are used in the upstream direction. In TDMA, the transmission of packets must be separated in time to prevent collisions among them, and in CDMA, multiple packets may be transmitted simultaneously without colliding as long as they use different codes of encoding their packets. Frequency separation is also called FDMA. In this thesis, we assume that there is only one downstream and one upstream channel. We do not consider CDMA.

Besides transmitting the legacy analogue TV signals and, in some countries, digital TV signals, these networks are nowadays predominantly used for Internet-based services via the world-wide web. The number and variety of services are steadily increasing, including communication services such as IP-telephony, e-mailing and chatting, search engines for searching information on the web, on-line shops, information services, e.g. on-line newspapers or journals, entertainment such as on-line gaming and video-on-demand (VOD), e-commerce such as on-line booking or banking services, et cetera.

1.2 Video on demand

Among these services, VOD distinguishes itself by the combination of huge bandwidth, real-time, and interactivity requirements.

The availability of efficient digital video compression algorithms such as MPEG [LeGall, 1991; Haskel, Puri, and Netravali, 1997] enable video data to be compressed at a variable bit rate in the order of a few to tens of Mbit/s while retaining a high quality. When compared to voice, traditionally encoded at 64 Kbit/s and at less than 10 Kbit/s if compressed, or compared to MP3-encoded music, typically at rates of at most 128 Kbit/s, video is indeed characterized by huge bit rates.

To provide true video on demand, interactivity requirements must be met such as instant play, pause-resume, and jumping forward and backward, making the current practice of downloading infeasible. Instead, real-time transmission, or *streaming*, is required, where a flow of data from a server to the user's equipment must be sustained at the proper bit rate to allow uninterrupted viewing. In addition, a low response time from the server is required to support the interactivity requirements.

In this thesis, we primarily consider the setup as illustrated in Figure 1.1, where a video server is located near the HE and each user has a PVR (or a PC, etc.) with a hard disk at home. The server stores thousands of video files in compressed form on an array of hard disks. Multiple clients can simultaneously access video files in an on-demand fashion. Once selected, a video file is retrieved from the server and streamed downstream through the HFC network to the user's equipment. There, it may be decompressed and consumed immediately, or be temporarily stored in compressed form for time-shifted viewing, possible already during the recording of the file. Browsing through the offered video collection, selecting a title, and other interactivity with the server is supported via the upstream channel.

The sizes of currently available hard disks allow the storage of some hundred video files on a single disk, so that they are useful for maintaining a sizable collection at home and providing VOD in-the-small.

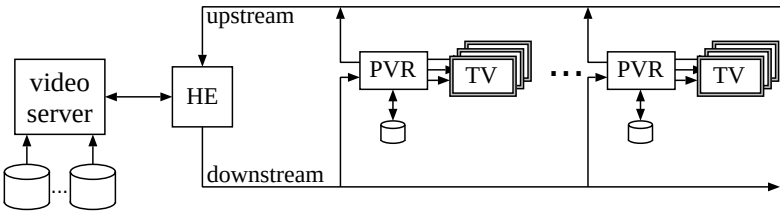


Figure 1.1. Abstract view of the system we consider.

1.3 Scheduling and resource management

Cable access networks

The basic method to transmit data on the upstream channel is by way of a request-grant procedure. If a CM has some data to transmit upstream on behalf of one of the connections it sustains, it first transmits a request to the HE. This request contains an identifier for the connection and an indication of the amount of time, say in terms of a number of slots, it requires to transmit the data. Upon reception by the HE of a request from a CM, it reserves a number of slots and informs the CM about this by transmitting a grant downstream to this CM, indicating that the HE grants exclusive access by this CM to the reserved slots.

Requests are transmitted in dedicated slots, called contention slots, wherein multiple CMs may attempt to transmit a request simultaneously. If this happens, the requests are said to collide and are all lost in the sense that the HE does not receive any of them. To resolve contention, that is, these

collisions, a contention resolution protocol is employed that governs the retransmission of collided requests. Hence, at the cost of transmitting relatively short requests in contention, the actual data is transmitted contention-free.

Depending on the standard, an alternative to transmitting a request in contention is to use piggybacking, whereby a new request is appended to the data, so that this request is also transmitted contention-free. This, of course, is only possible if the CM has at least one reserved slot at its disposal.

The area of contention-based access to shared media has been an active area of research for decades [Bertsekas & Gallager, 1992; Tanenbaum, 2003]. The type of collision resolution protocol used in a CATV network depends on the standard, and in fact, each of the standards offer several alternatives. One of the main collision resolution protocols employed is based on the well-known ALOHA protocol [Abramson, 1970; Roberts, 1975], the other main protocol is based on contention trees [Capetanakis, 1979; Tsybakov & Mikhailov, 1978; Janssen & de Jong, 2000].

A central problem for the request-grant procedure is how to divide the upstream transmission time into contention and reservation slots to optimize the delay that data incurs. Early work, specifically during standardization, primarily concentrated on extensive simulations, see, e.g., Golmie, Santillan & Su [1999], Sala [1998], Kwaaitaal [1999], Pronk & De Jong [1998], and Pronk, Hekstra-Nowacka, Tolhuizen & Denteneer [1999].

More recently, these simulation experiments have been complemented with analytical results. Palmowski, Schlegel & Boxma [2003], Denteneer [2005] and Van Leeuwaarden [2005] develop queuing-theoretic models for studying the transmission delay in the upstream channel. The latter two are dissertations and contain many useful links to related work.

For reservation-based access in the upstream direction as well as for multiplexing data for connections in the downstream direction, fair queuing algorithms, originally designed for use in switches and routers, play an important role. A fair queuing algorithm aims to guarantee for each connection its fair share of the channel, where the definition of fair share is based on a fluid-flow server that can serve all connections in parallel. These algorithms have for nearly two decades received considerable attention in the literature; see Demers, Keshav & Shenker [1989], Parekh & Gallager [1993], Golestani [1994], Zhang [1995], Bennett & Zhang [1996], Stoica, Abdel-Wahab, Jeffay, Baruah, Gehrke & Plaxton [1996], Stepping [2001], and Kunz & Stepping [2003].

Besides the basic request-grant mechanism, alternative access modes exist in the upstream direction, such as for providing services with a guaranteed

quality level. Hence, the HE must generally multiplex more than two access modes on one channel [Pronk, 2000].

A downstream channel is typically coupled to four or eight upstream channels, and CMs may switch between these upstream channels, as well as switch between downstream channels and, consequently, upstream channels. These migrations are under control of the HE, and leads to the problem of load balancing among the channels. The time-varying behavior of a CM in terms of its load on the network, both downstream and upstream, require load balancing algorithms to operate on-line.

Video on demand

From a resource management point of view, combining interactivity with AV material, which is paramount in multimedia applications and systems, is a demanding task. Storage and retrieval of AV material poses real-time constraints. Once the playout of a video file has started, real-time constraints have to be obeyed in the delivery of subsequent parts of the file to allow uninterrupted viewing by the user while keeping the required buffering of data at the user's equipment low.

In addition, interactivity requires that scheduling is carried out on-line, as we have only partial knowledge of future user requests. Some form of admission control is necessary to prevent new requests from endangering the real-time guarantees of requests already granted. In addition, for consumer applications, solutions need to be cost-effective. Hence, solutions based on hardware overkill are not very suitable.

The literature on scheduling and resource management is diverse and extensive. The combination of real-time and interactivity constraints is unique to the field of multimedia. The traditional scheduling literature in the area of operations research, such as described by Brucker [2001], Lawler, Lenstra, Rinnooy Kan & Shmoys [1993], and Pinedo [2001], does not cover this combination. It is also not covered by the real-time scheduling literature in the area of computer science, such as described by Cheng [2002], Klein, Ralya, Pollak & Obenza [1993], Liu [2000], and Liu & Layland [1973].

Gemmell, Vin, Kandlur, Rangan & Rowe [1995] provide an introduction to the field of multimedia storage and retrieval and illustrate the various issues that play a role in the design of multimedia systems. As we do not extensively cover system and implementation aspects, we refer for more information on these aspects to Bolosky, Barrera, Draves, Fitzgerald, Gibson, Jones, Levi, Myhrvold & Rashid [1996], Cabrera & Long [1991], Freedman & DeWitt [1995], Shenoy & Vin [1998], and Sincoskie [1991].

At the heart of a VOD server is a disk subsystem that stores large amounts of AV material. Allowing access to this data by multiple clients simultaneously requires disk scheduling algorithms that make efficient use of the disk subsystem.

Korst & Pronk [2005] cover multimedia systems from an algorithmic point of view, including single- as well as multi-disk storage and retrieval of both constant-bit-rate (CBR) and variable-bit-rate (VBR) video data. They also cover smoothed transmission of VBR video data through a network and near-video-on-demand strategies. In addition, they provide an extensive list of literature that is relevant in this area.

The single-disk scheduling problems associated with CBR data can be considered as a stepping stone towards the more complicated as well as practical case of handling VBR data. Therefore, additional, simplifying assumptions like synchronization among clients or assuming equal bit rates are defendable when considering CBR data, as they greatly simplify the problems. Dealing with VBR data is not only more difficult because of the variability at which data is consumed, but also because interactivity in terms of slow motion and pause make this consumption behavior inherently unpredictable. This unpredictable behavior also complicates multi-disk scheduling problems, where the additional problem of load balancing among the disks plays an important role. Aerts [2003] provides an in-depth treatment of this problem and also considers the presence of multi-zone disks.

To mitigate the problems of transmitting VBR data across a communication network, bit-rate smoothing algorithms aim to reduce the variability at which this data is transmitted, usually at the cost of additional buffering and start-up latency at the client side. For a survey on bit-rate smoothing algorithms, we refer to Feng & Rexford [1999]. Rexford & Towsley [1999] also survey several bit-rate smoothing algorithms and include the issue of multiple links in the transmission path. Al-Marri & Ghandeharizadeh [1998] provide a taxonomy of disk scheduling algorithms that includes bit-rate smoothing algorithms.

An interesting alternative to providing true VOD with full interactivity is near VOD (NVOD), which is geared towards linear viewing of a video. The approach is to broadcast a video, generally on a small number of channels, so that many clients can access this video at the same time, usually at the cost of a larger start-up latency and requiring substantial buffering capacity, such as a hard disk, in the user's equipment. Over the past decade, several strategies have been proposed to realize NVOD. We refer to Korst & Pronk [2005] and Kameda & Sun [2004] for a survey on the various strategies.

In each of the following chapters, we will provide a more detailed introduction into the specific problems addressed and provide additional literature references.

1.4 Content of this thesis

We consider a number of resource management problems: three network-related and three video-service-related problems.

Medium access control for unregistered cable modems

As for the network-related problems, we first consider the problem of establishing initial contact between a CM and the HE. This initial contact is required to obtain the operational parameters necessary for normal operation. It is governed by a contention-based access protocol, called frame-based ALOHA [Schoute, 1993; Van der Vleuten, Van Etten & Van den Boom, 1994] where multiple CMs may attempt to access the upstream channel at the same time. This may result in collisions and consequent loss of messages, so that retransmissions are required. The delay a CM may incur during this process before turning to normal operation may be significant, especially after a power outage when a large number of CMs may attempt this simultaneously, thereby affecting the perceived availability of the VOD service.

The specifics of the contention channel as well as the arrival process of CMs to establish initial contact calls for a renewed analysis of optimal frame-length control.

Medium access control using request merging

We next consider the operation of the CM during normal operation. In this mode, a CM typically issues requests to the HE on behalf of the connections it sustains in contention with other CMs, to reserve one or more time intervals for the contention-free, upstream transmission of actual data. The standards mentioned in Section 1.1 are not explicit on how to deal with simultaneous requests from multiple connections per CM. We review several possibilities to do this, including one where a number of simultaneous requests from a number of connections at a CM are combined into a single multi-request. Under some mild conditions, this alternative outperforms the others in terms of upstream transmission delay of data and results in a better response time from the server.

Fair resource sharing

The third network-related problem is concerned with the allocation of downstream bandwidth by the HE to a number of connections, each with its own

bandwidth requirements. The bandwidth in a downstream channel can be assumed constant and is to be shared in such a way that not only the bandwidth requirements are met for each connection, but also the issue of jitter plays a role. Jitter is defined as the variation in delay that individual data packets from a connection incur. This jitter has direct consequences for the required buffering of video data at the homes.

File allocation on a single disk

As for the video-service-related problems, we first look into the issue of storage on, and retrieval from disk of video data. The way in which data is retrieved from disk puts restrictions on the way in which data is stored on disk.

For many disk scheduling algorithms, storage and subsequent retrieval of data is typically done in blocks of constant size, whereby storing or retrieving one block is preferably done using only a single disk access. However, when the blocks retrieved do not align with or do not have the same size as the blocks stored, this is not generally possible, unless a specific storage strategy is employed. Contiguous storage of a file on disk solves this issue, but it suffers from the disadvantage of disk fragmentation in case the set of files, each with its own size, changes frequently.

We consider a segmented storage strategy whereby data of a file is stored contiguously in relatively large chunks, called allocation units, of a fixed size, whereby the successive allocation units of a file need not be contiguous. In addition, the contents of two successive allocation units of a file partly overlap, requiring that some data needs to be written to disk twice. We discuss the consequences of this for the well-known triple buffering disk scheduling algorithm [Biersack, Thiesse & Bernhardt, 1996].

File allocation on a multi-zone disk

We next consider the exploitation of the fact that contemporary disks can store more data on the outermost tracks than on the innermost tracks. As a result of the constant angular velocity at which these disks rotate, retrieval of data can be done faster from the outermost tracks than the innermost tracks, effectively resulting in correspondingly less resource usage. If each video file has an associated popularity, one might be tempted to store more popular files closer to the outermost track.

We consider the problem of how to store a given set of video files on disk such that a cost function, related to resource usage during retrieval, is minimized and compare it to the well-known method of track pairing by Birk [1995a]. This storage strategy is especially relevant if the set of video files is static and their respective popularities are sufficiently skewed.

On the fixed-delay pagoda schedule

The last problem we consider is related to a periodic video broadcast schedule called fixed-delay pagoda broadcast, introduced by Pâris [2001]. This is a schedule to provide NVOD, which, as the name already suggests, offers less flexibility than true VOD in terms of interactivity. In particular, it is geared towards linear viewing of a file, that is, from the start to the end in real time. One of the major benefits is that considerable bandwidth savings can be realized when many users view a particular video file at more or less the same time. Instead of requiring separate streams for each of these users, the video file is broadcast in fixed-size fragments and on a fixed number of video channels, whereby fragments near the start of the file are broadcast more frequently than fragments near the end.

Pâris uses a heuristic to optimize the number of fragments that are transmitted inside each video channel. We substantiate this so-called square root heuristic by analysis and show that the results can be slightly improved.

An important performance parameter in this context is the maximum start-up latency, which is the maximum time a user may have to wait before viewing can start after having selected a title. We prove that Pâris' schedule is asymptotically optimal in terms of this start-up latency.

1.5 Organization of this thesis

The six problems sketched above are discussed in the next six chapters. That is, in Chapters 2 to 4, we look into the three network-related problems and in Chapters 5 to 7 into the three video-service-related problems. After reading this introductory chapter, the chapters are self-contained, so that each of them can be read, independently of the other five chapters.

The last chapter, Chapter 8, contains the conclusion of this thesis and suggestions for subjects for future research.

Appendix A contains a list of the author's output, related to this thesis, consisting of internal and external publications as well as patents and patent applications.

2

Medium Access Control for Unregistered Cable Modems

This chapter concerns the start-up phase that a cable modem (CM) goes through after powering up, where it is trying to get itself registered at the head end (HE). During this phase, it has to search for appropriate channels to receive and send information and go through a series of administrative tasks, such as informing the HE of its capabilities, obtaining operational parameter settings and establishing a first connection. In particular, the CM has to establish tight synchronization with the head end (HE) and set a proper transmission-power level. This part goes by the name of *ranging*, the reason for which is explained next.

CMs are connected to the HE via a tree network of coaxial and fiber cables with amplifiers at the appropriate places in the network. Each CM has its own signal propagation delay to the HE and its own signal attenuation, resulting from its specific location in the network. For efficient operation it is required (*i*) that all CMs are tightly synchronized to allow the transmission of short bursts of information by different CMs to be performed without unnecessarily large, unused time intervals, called guard bands, between successive bursts, and (*ii*) that each CM has a specific power level at which it does its upstream

transmissions to the HE to achieve a near-constant reception power at the HE from all CMs.

A CM obtains the information required for ranging from the HE, but this information can only be obtained after a first contact has been established between the CM and the HE, initiated by the CM.

For a DVB/DAVIC-compliant system, this initial contact is achieved using a contention-based access protocol, similar to frame-based ALOHA [see Schoute, 1983; Van der Vleuten, Van Etten & Van den Boom, 1994]. In a contention-based access protocol, multiple messages may be sent simultaneously, resulting in a collision and the loss of all of these messages. A retransmission scheme is employed to ensure that the messages will eventually arrive successfully. In frame-based ALOHA, the slotted time axis is divided into variable-length, consecutive frames. An unsuccessful transmission, in an arbitrarily chosen slot in a frame, can only be repeated in the next one.

The central problem in frame-based ALOHA is the computation of the optimal frame length, which clearly depends on the number of contending CMs: a frame that is too small will result in too many collisions, whereas a frame that is too large will result in too many empty slots.

The specific context sketched above, in particular the fact that the CMs are not yet ranged, calls for a renewed analysis for optimal frame-length control, which is the main subject of this chapter.

The remainder of this chapter is organized as follows. After discussing related work in Section 2.1, we describe in Section 2.2 the access protocol considered in this chapter and discuss in more detail the differences with frame-based ALOHA. We explain the reasons for a renewed analysis for optimal frame-length control. In Section 2.3, we propose a model for the contention channel at hand, which generalizes the model commonly used in the analysis of frame-based ALOHA. The parameters in this channel model influence the determination of the optimal frame length.

In Section 2.4, we show how to compute the frame length to achieve maximal throughput, assuming that the number of contenders in this frame is known. In Section 2.5, we propose an estimate for the number of contenders in an already observed set of slots. This estimate is then used in Section 2.6 to obtain an estimate for the number of contenders in the frame that is about to start. It is noteworthy that this estimate does not make any assumptions on the arrival process of new CMs. We collect the results obtained and describe the algorithm for determining the lengths of the successive frames, thereby taking the specific context into account, in particular the non-negligible delay involved in providing feedback on transmissions.

In Section 2.7, we provide simulation results to assess the effectiveness of the estimators and their sensitivity to inaccurate channel parameters, and we make some concluding remarks in Section 2.8.

Finally, we mention that this work is partly based on work by Pronk & Tolhuizen [2000, 2001]

2.1 Related work

Since the development of the DOCSIS and DVB/DAVIC standards, ample research has been conducted on optimizing performance in these networks; see Denteneer [2005], Lin, Yin & Huang [2000], Kuo, Kumar & Kuo [2003], Liao & Ju [2004], and references therein. These investigations primarily concern 'normal' operation of cable modems (CMs), where they can send and receive data on behalf of applications running in the homes. Less emphasis has been put on performance issues during the start-up phase of a CM.

The problem we consider is also similar to the problem of controlled or stabilized ALOHA, see Bertsekas & Gallager [1992] for an introduction. In stabilized ALOHA, instead of defining a frame length, an adaptive retransmission probability for each slot is defined, based on the expected number of contenders for that slot.

Capetanakis [1979], Wolters, Van Hoof, Botte & Sierens [1997] consider a contention resolution protocol based on address splitting. By successively splitting the address range in smaller ranges and only allowing those CMs that have their address in the current range, all collisions will be resolved eventually. The large address space and a possible imbalance in the addresses of the participating CMs make the procedure sub-optimal.

Sdralia, Tzerefos & Smythe [2001] consider the ranging problem for the DOCSIS standard. This standard uses a binary exponential back-off algorithm wherein a CM repeatedly picks a random slot in a frame, initially of a predetermined length, that is doubled in length after each unsuccessful transmission until a predetermined, maximal frame length is reached. Each CM defines its own sequence of frames.

Sala, Hartman & Limb [1996] compare three different contention resolution algorithms for the ranging problem: stabilized ALOHA, which is called p-persistence in their article, binary exponential back-off, and a contention-tree algorithm. In the latter, all contenders in a slot containing a collision are allocated a number of slots, typically three, to retransmit, each contender in a randomly chosen slot, so that the group is effectively split in smaller groups. This generates a tree structure. After completion of one tree, another is started. Denteneer, Pronk, Hekstra-Nowacka & Tolhuizen [2003] describe a

more advanced method to start up new trees, effectively resulting in multiple, simultaneously active trees, which are resolved one after the other. For more information on contention trees, we refer to Tsybakov & Mikhailov [1978], and Janssen & de Jong [2000].

Hajek, Likhanov & Tsybakov [1994] consider the problem of large propagation delays in high-speed networks, where the propagation delay, and thus the feedback delay, may be in the order of hundreds of times the length of individual packets. They investigate the delay that packets may incur when using contention-based access.

2.2 Contention access in DVB/DAVIC during start-up

The contention procedure during start-up in a DVB/DAVIC-compliant HFC system can be described as follows. The time axis is divided into fixed-length slots. The HE dynamically partitions this slotted time axis into variable-length, nonoverlapping frames. A frame counts an integer number of slots and is aligned with slot boundaries. A frame starts immediately upon reception of a sign-on request message, sent by the HE, which contains the length of this frame. During a frame, each contending CM transmits a sign-on response message to the HE in a randomly chosen slot, uniformly distributed in the frame. This transmission is performed at a particular power level, that is under control of a separate procedure. In short, a CM cycles around a number of power levels.

A transmitting CM is either a newly arriving CM that entered its contention procedure during the previous frame and will transmit for the first time in the present frame, or a CM that transmitted in some earlier frame and discovered that its transmission was unsuccessful. A transmission may be unsuccessful because it was done at an improper power level and/or because it collided with a transmission by another CM. A CM discovers that its transmission was unsuccessful by using a time-out mechanism: if a CM does not receive a response from the HE of successful transmission within a maximum feedback delay T_{fb} after transmission, it considers this transmission unsuccessful. The CM then retransmits in the next available frame, designated by the reception of the first sign-on request message from the HE after this time-out. In practice, this maximum feedback delay is considerable. It is noted that a CM may have skipped various frames before transmitting again in case the frame length is short in comparison with the maximum feedback delay.

The main problem is how to determine the frame lengths online so as to make optimal use of the contention channel.

The access protocol described above is similar to frame-based ALOHA, see the papers by Schoute [1983] and Van der Vleuten, Van Etten & Van den

Boom [1994]. The major differences between the access protocol considered in this chapter and frame-based ALOHA are the following. Firstly, due to the lack of synchronization and power calibration among CMs, the contention channel, though still slotted, behaves differently in terms of successes and collisions. We will elaborate on this in Section 2.3, where we formally model the contention channel. Secondly, there is a non-negligible contention feedback delay, which complicates the retransmission process.

An important difference with the papers cited above is that the authors assume that new contenders arrive at the contention process according to a Poisson process. This assumption is not true in the current context, where the number of CMs in the network is finite and each CM goes to the contention process only once. Furthermore, a special case of interest here is to consider the situation where a large number of modems may attempt to power up nearly simultaneously, such as after a local power outage.

2.3 Modeling the contention channel

Because unregistered CMs still have to synchronize to the time axis, the slots used by unregistered CMs are made large enough, that is, three times the length of a normal slot. A transmission by a CM close to the HE will arrive (correctly received or not) at the HE at the start of a slot, whereas a transmission by a CM at a larger distance, with a maximum of 80 km according to the DVB/DAVIC standard, will arrive later during the slot. As a result, two transmissions in a single slot need not even collide, and more than one success per slot is possible. In addition, a CM does its successive transmissions at varying power levels, subject to a separate procedure. This generally influences the reception behavior by the HE as well: A transmission may go unnoticed by the HE. We model the channel as follows.

Definition 2.1. (Channel model) The reception behavior by the HE of the contents of a single slot, is described by the following parameters. For $i \geq 0$ we define

$$\begin{aligned} e_i &= \Pr(\text{the slot is perceived empty} \mid i \text{ transmissions}) \\ s_i &= \boldsymbol{E}[\text{number of successes} \mid i \text{ transmissions}] \end{aligned}$$

□

Here, the $\boldsymbol{E}$ stands for the expectation operator. It is assumed that the underlying process is stationary, that is, that these parameters are constant over time. In particular, we do not further elaborate on the procedure that CMs use for obtaining their correct power levels. It is instead assumed that this procedure

is captured in the channel model. Schoute uses a similar, but simpler model to express the presence of noise and the capturing effect in mobile packet radio networks. Note that we may assume that $s_0 = 0$ and $e_0 = 1$. Note furthermore that the case $e_0 = 1, e_i = 0$ for $i \neq 0$ and $s_1 = 1, s_i = 0$ for $i \neq 1$ corresponds to the conventional channel model.

The e_i and s_i parameters are used to determine the frame length in the following manner. The parameters s_i are used to determine the optimal frame length, given the number of contending CMs in the frame. This is covered in Section 2.4. As the number of contenders at the start of a frame is *not* known, we next give in Section 2.5 an estimate for the number of contenders in an observed set of slots, in which the e_i's are employed. Using this result, we give in Section 2.6 a new estimator for the number of contenders at the start of a frame, whereby we also take the feedback delay into account.

2.4 Determination of the optimal frame length

We consider S slots over which N contenders are independently and uniformly distributed. The expected number of successes in a given slot, denoted by $E_N(S)$, satisfies the following equation.

$$E_N(S) = \sum_{i=1}^{N} s_i \binom{N}{i} \left(\frac{1}{S}\right)^i \left(1 - \frac{1}{S}\right)^{N-i}. \tag{2.1}$$

This is easily seen by observing that, for a given slot, the number of contenders that transmit in this slot is Binomially distributed with parameters N and $\frac{1}{S}$. Given i contenders that transmitted in the given slot, the expected number of successes is given by s_i. As $s_0 = 0$, the summation starts at $i = 1$.

Note that, although the numbers of contenders among the different slots are clearly dependent, this is of no concern here, as we are only looking at a single slot.

Given the number N of contenders, we wish to determine the number of slots S such that $E_N(S)$ is maximal. We first consider S as a continuous variable in Equation 2.1.[1] By straightforward differentiation, we obtain, for $S > 1$,

[1] In this thesis, we use the colloquial meaning of the term 'equation' rather than its mathematical meaning of 'equality'. Hence, inequalities will also be referred to as equations.

$$\begin{aligned} E'_N(S) &= \sum_{i=1}^{N} s_i \binom{N}{i} \left[-\frac{i}{S^2} \left(\frac{1}{S}\right)^{i-1} \left(1-\frac{1}{S}\right)^{N-i} + \frac{N-i}{S^2} \left(\frac{1}{S}\right)^{i} \left(1-\frac{1}{S}\right)^{N-i-1} \right] \\ &= \sum_{i=1}^{N} s_i \binom{N}{i} \left(\frac{1}{S}\right)^{i+2} \left(1-\frac{1}{S}\right)^{N-i-1} \left(-iS \left(1-\frac{1}{S}\right) + N - i \right) \\ &= \sum_{i=1}^{N} s_i \binom{N}{i} \left(\frac{1}{S}\right)^{i+2} \left(1-\frac{1}{S}\right)^{N-i-1} (N - iS). \end{aligned} \tag{2.2}$$

Lemma 2.1. *Let m be the maximal index $i < N$ for which $s_i \neq 0$. Then E_N attains its maximum on $(1,\infty)$ in the interval $[N/m, N]$.*

Proof. Note that the summation in Equation 2.2 only extends to $i = m \geq 1$ and that the sign of the ith term, if not zero, equals the sign of $N - iS$. For $i \leq m$, we have that $N - iS \geq N - mS > 0$ if $S < N/m$. Hence, for $S < N/m$, it holds that $E'_N(S) > 0$, as the mth term is not zero. Conversely, for $i \geq 1$, we have that $N - iS \leq N - S < 0$ if $S > N$. Hence, for $S > N$, it holds that $E'_N(S) < 0$, as the mth term is not zero. From this, we can conclude that $E'_N(S) = 0$ implies that $S \in [N/m, N]$. □

In the special case that $m = 1$, the maximum is thus attained at $S = N$, that is, where there is one slot for each contender. Furthermore, notice that, if only $s_m > 0$, then the maximum is attained at $S = N/m$. In the latter case, a slot ideally contains exactly m contenders, requiring only N/m slots for all N contenders.

E_N can have several local maxima. As an example, suppose $N = 12$, $s_1 = 1$ and $s_5 = 1$; the other s_i's are zero. As $N = 12$ and $m = 5$, we know by the lemma above that E_N attains its global maximum on $(1,\infty)$ in the interval $[12/5, 12] = [2.4, 12]$. Figure 2.1 shows the presence of an additional, local maximum, at approx. 2.58. Restricting S to integer values yields two maxima, namely at $S = 3$ and at $S = 12$. The latter value attains the global maximum over the positive integers S.

We next give a general result on the optimal value S_{opt} for $E'_N(S)$, assuming that there is a maximal index for which $s_i > 0$.

Lemma 2.2. *Let m be such that $s_m > 0$ and $s_i = 0$ for $i > m$. Then for large values of N, the optimal frame length S_{opt} for Equation 2.1 can be written as*

$$S_{\text{opt}} \approx \beta_{\text{opt}} N, \tag{2.3}$$

where β_{opt} depends only on the s_i values.

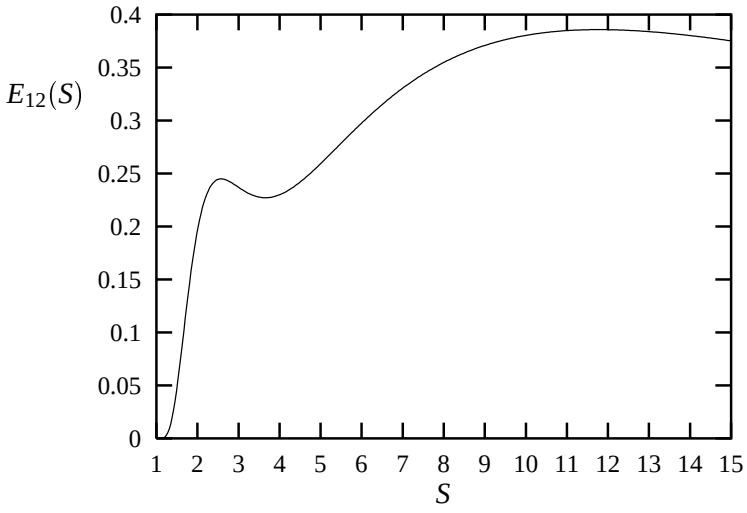


Figure 2.1. $E_{12}(S)$ with $s_1 = s_5 = 1$, the other s_i's equal to 0.

Proof. Choose $\beta > 0$ and let $S = \beta N$. It follows from Equation 2.1 that

$$\begin{aligned} \lim_{N\to\infty} E_N(\beta N) &= \lim_{N\to\infty} \sum_{i=1}^{m} s_i \frac{1}{i!} \beta^{-i} \frac{N!}{N^i (N-i)!} \left(1 - \frac{1}{\beta N}\right)^{N-i} \\ &= \sum_{i=1}^{m} s_i \frac{1}{i!} \beta^{-i} e^{-\frac{1}{\beta}}, \end{aligned}$$

which corresponds to the well-known approximation of the Binomial distribution by a Poisson distribution. Notice that this expression is independent of N, so that, by numerical optimization of the expression above, an optimal value β_{opt} can be obtained, and an optimal value S_{opt} for S then satisfies Equation 2.3. □

The optimal frame length for large values of N is thus linear in the number of contenders, the constant β_{opt} being determined solely by the channel parameters, although its value is not easily obtained in the general case.

Lemma 2.3. *For the special case that $s_0 = 0$, $s_1 > 0$, and $s_i = 0$ for $i \geq 3$, we have that, for large N, the optimal frame length S_{opt} for Equation 2.1 can be written as*

$$S_{\text{opt}} \approx \beta_q N, \tag{2.4}$$

where $\beta_q = \frac{1}{2}(1 - q + \sqrt{1+q^2})$ and $q = \frac{s_2}{s_1}$.

Proof. For this special case, we have that

$$E_N(S) = s_1 N \frac{1}{S}\left(1-\frac{1}{S}\right)^{N-1} + s_2 \frac{N(N-1)}{2}\left(\frac{1}{S}\right)^2\left(1-\frac{1}{S}\right)^{N-2}. \quad (2.5)$$

After some elementary calculus, it is found that $E_N'(S) = 0$ if and only if

$$S = \frac{[(N+1) - q(N-1)] \pm \sqrt{(N-1)^2 + q^2(N-1)^2 - 2q(N-1)}}{2}. \quad (2.6)$$

It readily follows from Equation 2.6 that E_N' has two positive roots if and only if $q(N-1) \leq 2$. Hence, for large N, E_N' has only one positive root, where $E_N(S)$ attains its global maximum. For large N, we replace $N+1$ and $N-1$ by N, so that the optimal value S_{opt} for S is easily seen to satisfy Equation 2.4. □

It can be seen that β_q is a decreasing function in q; $\beta_0 = 1$ and $\beta_q \to \frac{1}{2}$ if $q \to \infty$. This agrees with our earlier observation that E_N' has a zero in the interval $[N/m, N] = [N/2, N]$.

Lemma 2.4. *If $s_0 = 0$, $s_1 > 0$, and $s_i = 0$ for $i \geq 3$, then, for large N, the maximum capacity $E_N(S_{\text{opt}})$ satisfies*

$$E_N(S_{\text{opt}}) \approx \left(\frac{s_1}{\beta_q} + \frac{s_2}{2\beta_q^2}\right) e^{-\frac{1}{\beta_q}}. \quad (2.7)$$

where $\beta_q = \frac{1}{2}(1 - q + \sqrt{1+q^2})$ and $q = \frac{s_2}{s_1}$.

Proof. By substituting in Equation 2.5 for S the approximation of S_{opt}, given in Equation 2.4, we obtain an approximation of $E_N(S_{\text{opt}})$ as follows.

$$\begin{aligned} E_N(S_{\text{opt}}) &\approx s_1 N \frac{1}{\beta_q N}\left(1-\frac{1}{\beta_q N}\right)^{N-1} + s_2 \frac{N(N-1)}{2}\left(\frac{1}{\beta_q N}\right)^2\left(1-\frac{1}{\beta_q N}\right)^{N-2} \\ &\approx \left(\frac{s_1}{\beta_q} + \frac{s_2}{2\beta_q^2}\right) e^{-\frac{1}{\beta_q}}, \end{aligned} \quad (2.8)$$

where in the latter approximation, $N-1$ and $N-2$ have been replaced by N and the approximation $(1-1/x)^x \approx e^{-1}$ for large x has been used. □

For large N, the maximum capacity is thus nearly independent of N.

We next prove that for large N the success probability for a single contender is also nearly independent of N.

Lemma 2.5. *If $s_0 = 0$, $s_1 > 0$, and $s_i = 0$ for $i \geq 3$ and $S = S_{\text{opt}}$, then for large N, the probability p of success for a single contender satisfies*

$$p \approx \left(s_1 + \frac{s_2}{2\beta_q} \right) e^{-\frac{1}{\beta_q}}, \tag{2.9}$$

where $\beta_q = \frac{1}{2}(1 - q + \sqrt{1+q^2})$ and $q = \frac{s_2}{s_1}$.

Proof. Clearly, the capacity of the S_{opt} slots in a frame is given by $S_{\text{opt}} E_N(S_{\text{opt}}) \approx \beta_q N E_N(S_{\text{opt}})$. The expected number of successes in this frame is also given by pN, as can be seen as follows. Let, for each contender i, the random variable X_i equal 1 if this contender is successful and 0 otherwise. Then $\boldsymbol{E}[X_i] = p$ and the expected number of successes is $\boldsymbol{E}[\sum_i X_i] = \sum_i \boldsymbol{E}[X_i] = pN$, by linearity of the expectation operator. Hence, $pN \approx \beta_q N E_N(S_{\text{opt}})$, from which the factors N can be eliminated. The result follows by using Lemma 2.4. □

Recapitulating, if we assume that N is large and that the frame lengths are chosen optimally, each contender is independently involved in a Bernoulli trial with success probability p, given by Equation 2.9, independent of the number of contenders. The expected number of attempts until success is $1/p$, although the expected *time* until success of course depends on the successive frame lengths, which are determined by and approximately linear in the number of participating contenders.

2.5 Estimating the number of contenders in a past frame

In this section we consider S slots over which an unknown number N of contenders are independently and uniformly distributed. The problem is to estimate N from S and the observed pattern of contention results, that is, slots perceived empty, collisions, and slots with a given number of successes.

As argued by Pronk & Tolhuizen [2000], it is a good idea to use only the number N^{e} of slots perceived empty. Reason is that by using the number of successes only, it is not possible to estimate N, and that by only using the number of collisions there doesn't seem to be a simple closed formula for expressing N in $\boldsymbol{E}[N^{\text{c}}]$, even in the case of a conventional channel. Furthermore, combining, for instance, both N^{s} and N^{e} to estimate N, generally leads to less accurate results. This is due to the fact that we are using two random variables instead of one.

One way to obtain an estimate for N is to do simulations. In particular, for many values of S and known numbers of contenders N, we count the number N^{e} of slots perceived empty, using a probabilistic approach based on the e_i's,

after these N contenders have each chosen a slot randomly among the S slots. This provides triples (S,N,N^{e}), from which, for each pair (S,N^{e}), a most likely value of N can be deduced. A similar approach is used by Yin & Lin [2000], who use success and collision counts to obtain an estimate for the number of contenders.

We opt for the analytical approach, partly because it saves large tables to be stored, partly because it provides more insight into the dependencies amongst the parameters, estimators, and variables.

In the same vein as Equation 2.1, the probability P^{e} that a given slot is perceived empty is readily seen to satisfy the following Binomial expression

$$P^{\mathrm{e}} = \sum_{i=0}^{N} e_i \binom{N}{i} \left(\frac{1}{S}\right)^i \left(1-\frac{1}{S}\right)^{N-i}. \tag{2.10}$$

Lemma 2.6.

$$P^{\mathrm{e}} = \frac{\boldsymbol{E}[N^{\mathrm{e}}]}{S}. \tag{2.11}$$

Proof. Let the random variable X_i be 1 if slot i is perceived empty, and 0 otherwise. Then $\boldsymbol{E}[X_i] = P^{\mathrm{e}}$ and $N^{\mathrm{e}} = \sum_{i=1}^{S} X_i$. By linearity of the expectation operator, we have that $\boldsymbol{E}[N^{\mathrm{e}}] = \boldsymbol{E}[\sum_{i=1}^{S} X_i] = \sum_{i=1}^{S} \boldsymbol{E}[X_i] = \sum_{i=1}^{S} P^{\mathrm{e}} = SP^{\mathrm{e}}$, from which Equation 2.11 follows. □

The idea of estimating N is to act as if N^{e} equals the expected number of slots perceived empty and calculating N by using Equations 2.10 and 2.11.

The following lemma states a general result on the possibility to estimate N from S, $\boldsymbol{E}[N^{\mathrm{e}}]$, and the channel parameters. We make the natural assumption that e_i is a non-increasing function of i, that is, that more contenders in a single slot do not increase the probability that it is perceived empty.

Lemma 2.7. *Let e_i be a non-increasing function of i and let m be such that $e_m > 0$ and $e_i = 0$ for $i > m$. Then, if both S and N are large, N can be estimated as*

$$N \approx S\, f_{\mathrm{e}}^{-1}\left(\frac{\boldsymbol{E}[N^{\mathrm{e}}]}{S}\right), \tag{2.12}$$

where f_{e}, defined in Equation 2.13, is an invertible function on $(0,\infty)$ that only depends on the channel parameters e_i.

Proof. Let both N and S be large and α be such that $N = \alpha S$. Then

$$\boldsymbol{E}[N^{\mathrm{e}}] \approx S \sum_{i=0}^{m} e_i \frac{N^i}{i!} \left(\frac{\alpha}{N}\right)^i (1-\frac{\alpha}{N})^{N-i} \quad \approx \quad S \sum_{i=0}^{m} \frac{e_i}{i!} \alpha^i e^{-\alpha},$$

which again corresponds to the well-known approximation of the Binomial distribution by a Poisson distribution. By defining

$$f_{\mathrm{e}}(\alpha) = \sum_{i=0}^{m} \frac{e_i}{i!} \alpha^i e^{-\alpha}, \tag{2.13}$$

we have that $\boldsymbol{E}[N^{\mathrm{e}}] \approx S\, f_{\mathrm{e}}(\alpha)$. If $f_{\mathrm{e}}(\alpha)$ is invertible on $(0,\infty)$, Equation 2.12 follows by substituting N/S for α.

What thus remains to be proved is that $f_{\mathrm{e}}(\alpha)$ is invertible on $(0,\infty)$. Differentiating $f_{\mathrm{e}}(\alpha)$ with respect to α, we obtain that

$$\begin{aligned} f'_{\mathrm{e}}(\alpha) &= \mathrm{e}^{-\alpha} \left(-\sum_{i=0}^{m} \frac{e_i}{i!} \alpha^i + \sum_{i=1}^{m} \frac{e_i}{(i-1)!} \alpha^{i-1} \right). \\ &= -\mathrm{e}^{-\alpha} \left(\sum_{i=0}^{m-1} \frac{(e_i - e_{i+1})}{i!} \alpha^i + \frac{e_m}{m!} \alpha^m \right). \end{aligned}$$

As the e_i's are non-increasing and $e_m > 0$, $f'_{\mathrm{e}}(\alpha)$ is negative for positive α. Hence, $f_{\mathrm{e}}(\alpha)$ is invertible on $(0,\infty)$, which completes the proof. □

We next consider the special case of a channel whereby $e_0 = 1$, $0 < e_1 < 1$, and $e_i = 0$ for $i \geq 2$. The assumption $e_1 > 0$ reflects the possibility that a single transmission is not necessarily detected by the HE.

Lemma 2.8. *For the special case that $e_0 = 1$, $0 < e_1 < 1$, and $e_i = 0$ for $i \geq 2$, it holds that*

$$N \approx S \frac{\ln\left(\frac{S}{\boldsymbol{E}[N^{\mathrm{e}}]}\right)}{1-e_1}. \tag{2.14}$$

provided that S is large and $e_1 N/S$ is small.

Proof. For this special case we have that

$$\boldsymbol{E}[N^{\mathrm{e}}] = S(1-\frac{1}{S})^N + e_1 N (1-\frac{1}{S})^{N-1}. \tag{2.15}$$

We next give an approximate solution to Equation 2.15. Using the fact that $\ln(1+x) = x + O(x^2),\ x \to 0$, the equality is reduced to

$$\ln(\frac{\boldsymbol{E}[N^{\mathrm{e}}]}{S}) = N\ln(1-\frac{1}{S}) + \frac{e_1 N}{S-1} + O\left(\left(\frac{e_1 N}{S-1}\right)^2\right),$$

This approximation is valid as $e_1 N/S$ is small. For large S, we approximate $S-1$ by S and $\ln(1-1/S)$ by $-1/S$. From this, the result follows. □

The results in this section primarily concern large values of S and N. For small values, the method of simulation mentioned at the beginning of this section could be used as a complementary approach.

2.6 Estimating the number of contenders in a future frame

For determining the length of a new contention frame, say contention frame i, we wish to estimate the number N_i of CMs that contend in this frame, so that the results of Section 2.4 can be applied. We propose an estimate for N_i, in which the feedback delay is taken into account as well.

Consider Figure 2.2, where the time axis is divided into successive frames by the vertical, solid lines. The maximum feedback delay T_{fb}, see Section 2.2 for an explanation, is assumed to correspond to an integer number of slots and is, in this example, considerably larger than the individual frame lengths. The length of contention frame i is denoted by S_i and is also an integer number of slots.

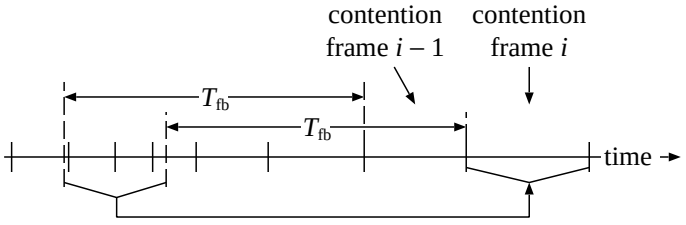


Figure 2.2. Influence of the feedback delay.

The CMs that do a retransmission in contention frame i did their previous attempt in the window delineated by the leftmost two dashed lines. This can be seen as follows. CMs that transmitted before the start of the time window and were unsuccessful already did a retransmission before contention frame i, as their feedback time-out occurred before the start of contention frame $i-1$. Conversely, those that transmitted after the end of the time window and were unsuccessful are still waiting for their time-out at the start of contention frame i, which they will consequently miss. Hence, they transmit after contention frame i. Those that transmitted in the indicated time window and were unsuc-

cessful experienced their timeout during contention frame $i-1$, so that they retransmit in contention frame i.

We call this time window related to contention frame i its *originating* frame i. Its length equals S_{i-1}, that is, the length of contention frame $i-1$. We call the time window of length T_{fb} that ends at the start of contention frame i its corresponding *feedback* frame i.

We next derive an approximation of N_i. We denote with M_i the total number of participating CMs at the start of contention frame i. This set of M_i CMs consists of three subsets, as shown in Figure 2.3. Firstly, there are newly arriving CMs. These CMs did not transmit before the start of contention frame i. Their number is denoted by n_i. Secondly, there are CMs that transmitted earlier and do a retransmission in contention frame i. These CMs last transmitted in originating frame i. Their number equals $N_i - n_i$, as these CMs together with the n_i new CMs constitute the set of N_i CMs contending in contention frame i. Finally, there are CMs that transmitted earlier and are still waiting for feedback. These CMs last transmitted in feedback frame i and their number equals $M_i - N_i$, as the total number of participating CMs equals M_i. Note that we are only counting newly or still participating CMs here, and not those that were successful in originating or feedback frame i.

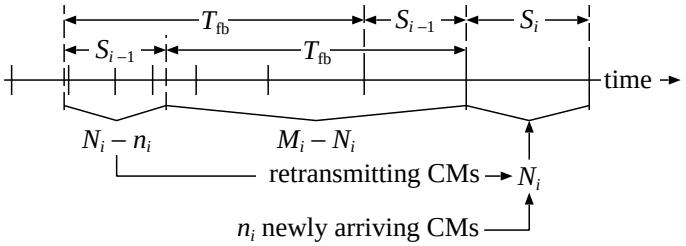


Figure 2.3. Subdivision of the number M_i of CMs over the frames.

If the transmissions of all $N_i - n_i + M_i - N_i = M_i - n_i$ retransmitting CMs are homogeneously distributed over the slots in originating and feedback frames i, then N_i can be estimated by $\hat{N}_i$, defined as

$$\hat{N}_i = (M_i - n_i)\,\frac{S_{i-1}}{S_{i-1} + T_{\text{fb}}} + n_i. \tag{2.16}$$

As n_i is unknown, we propose to ignore n_i, the number of newly arriving CMs at the start of contention frame i and to set it to 0 in Equation 2.16. To obtain an estimate for M_i, we use an *a posteriori* estimation $\hat{M}_{i-1}$ of M_{i-1}, calculated at the end of contention frame $i-1$, that is, at the start of contention frame i.

As $\hat{M}_{i-1}$ takes the n_{i-1} CMs into account that newly arrived at the start of contention frame $i-1$, the n_i CMs that newly arrive at the start of contention frame i are taken into account at the start of contention frame $i+1$. Similarly, the successful transmissions in originating frame i are taken into account at the start of contention frame $i+1$, as they did not retransmit in contention frame i and are thus not counted, and the successful transmissions in feedback frame i are taken into account in frames beyond contention frame i. Effectively, we introduce a delay of one frame to incorporate newly arriving and successful CMs.

Instead of Equation 2.16, we thus use

$$\hat{N}_i' = \hat{M}_{i-1} \frac{S_{i-1}}{S_{i-1} + T_{\text{fb}}}.$$

We next turn our attention to $\hat{M}_{i-1}$. At the start of contention frame $i-1$, each of the M_{i-1} participating CMs either transmitted during feedback frame $i-1$, or will transmit during contention frame $i-1$. Assuming again a homogeneous distribution of transmissions over the union of these two intervals, the number N_{i-1}^{e} of slots perceived empty in these intervals, which is known at the start of frame i, can be used to calculate a value for $\hat{M}_{i-1}$ as follows.

If $N_{i-1}^{\text{e}} > 0$, we use Equation 2.14, whereby $\boldsymbol{E}[N^{\text{e}}]$ is estimated by N_{i-1}^{e} and S is replaced by the total number of slots in feedback and contention frames $i-1$.

If, on the other hand, $N_{i-1}^{\text{e}} = 0$, then no good estimate can be given. However, since this situation may indicate a large value of M_{i-1} relative to the frame length, it seems safe to let $\hat{M}_{i-2}$ be at least 1 and to double the previous estimate, that is, to let $\hat{M}_{i-1} = 2\hat{M}_{i-2}$. The resulting exponential growth in $\hat{M}_{i-1}$, and thus in S_i, if N_{i-1}^{e} remains 0 for successive values of i, will eventually lead to a situation with $N_{j-1}^{\text{e}} > 0$ for some $j > i$, at which time a proper estimation can again be established.

When an estimate for the number of CMs in contention frame i has been obtained, the length for this frame can be set using Equation 2.4 and by rounding the result to the appropriate integer.

For practical purposes, however, a minimal frame length $S_{\min}$ should be imposed upon S_i to limit the required downstream messaging to indicate the start of the successive frames. This also takes care of the possibility that an estimate of S_i could be zero.

The assumption made above of homogeneously distributed transmissions is relevant especially if the frame length is in the order of the maximum feedback delay. In particular, if the minimal frame length divides the maximum feedback delay, then a sudden burst in one frame of newly arriving CMs may

'stick together' and retransmit a number of frames later, all in the same frame, and this may repeat itself. If, on the other hand, the minimal frame length does *not* divide the maximum feedback delay, CMs in one frame will eventually spread their retransmissions over various frames. We will come back to this in the next section. This divisibility may also happen for frame lengths unequal to the minimal frame length, but this is easily solved by increasing the frame length somewhat.

Collecting the results from this and the previous sections, we arrive at the algorithm shown in Figure 2.4 to determine the length of contention frame i. We assume that $s_0 = 0$, $s_1 > 0$, $s_i = 0$ for $i \geq 3$, $e_0 = 1$, $0 < e_1 < 1$, and $e_i = 0$ for $i \geq 2$, so that we can use Lemmas 2.3 and 2.8. We use β_q as defined in Lemma 2.3 and ignore initialization.

```
while true do
begin
    wait until the end of contention frame i − 1;
    N^e_{i−1} := the number of empty slots in originating and
        feedback frames i − 1;
    if N^e_{i−1} = 0 then M̂_{i−1} := 2 M̂_{i−2}
    else M̂_{i−1} := max(1, ln((S_{i−1} + T_fb)/N^e_{i−1}) / (1 − e_1));
    N̂'_i := M̂_{i−1} S_{i−1} / (S_{i−1} + T_fb);
    S_i := max(S_min, round(β_q N̂'_i));
    while S_i | T_fb do S_i := S_i + 1;
    communicate the contention frame length S_i to all CMs;
    i := i + 1
end
```

Figure 2.4. Pseudo-code for the determination of the frame length.

2.7 Simulations

For an assessment of the effectiveness of the estimators presented in this chapter, it is illustrative to show that near-optimal usage of the channel in terms of throughput is indeed achieved. To this end, we consider the following scenario, as illustrated in Table 2.1.

The unit of time is one slot. Slots are numbered from zero onwards. The values for T_{fb}, e_1, and s_1 are realistic in practice, whereas the value for s_2 depends on the topology of the access network. If we assume that the CMs are distributed homogeneously on a disk with the HE at its center and a radius of 80 km, the maximal distance according to the DVB/DAVIC standard, then s_2 can be calculated to be approximately 0.1. This results in a value of 0.5 for

parameter	*value*
T_{fb}	90 slots
$S_{\min}$	varied
e_1	0.2
s_1	0.2
s_2	0.1
burst size	1000
arrival rate	variable

Table 2.1. Parameter settings for the first experiment.

q and a value of 0.81 for β_q in Lemma 2.3. The capacity per slot, given by Equation 2.8, is approximately 0.094. In the experiments, the minimal frame length $S_{\min}$ is varied to investigate its influence on the results.

We perform three experiments: in the first two experiments, we only consider a single burst of CMs that arrive simultaneously and we investigate the decay of the remaining, participating CMs over time. In the second experiment, however, we deliberately disturb the calculated frame length by a fixed factor to evaluate the sensitivity of the algorithm to inaccurate estimations of the channel model parameters. Note that, when considering Lemmas 2.3 and 2.8, but also 2.2, it makes sense to consider a fixed factor. In the third experiment, we consider the situation in which CMs arrive according to a Poisson process and consider the delay that a CM incurs from the moment of arrival until the moment its sign-on response message is received successfully at the HE, as a function of the arrival rate and the minimal frame length.

2.7.1 Dealing with a burst of CMs

In the first experiment, a burst of 1,000 CMs arrive instantaneously at the start of the simulation, and there are no other CMs arriving after this event. We vary the minimal frame length over 4, 8, 24, and 60 slots. The results are depicted in Figure 2.5, which plots the decay in the number of participating CMs as a function of time. The solid line depicts a line with slope -0.094, corresponding to the capacity of the channel, starting at 1,000 CMs at time 0.

The first observation we make is that the slopes of all four dotted lines during steady decay correspond closely to the capacity of the channel. The use of the actual number of slots perceived empty as an estimate for the expected number of slots perceived empty apparently does not significantly influence the capacity of the channel.

The second observation is that the final phase, when only a sufficiently small number of CMs are left, shows a ‘slow finish’, that is, the decay pro-

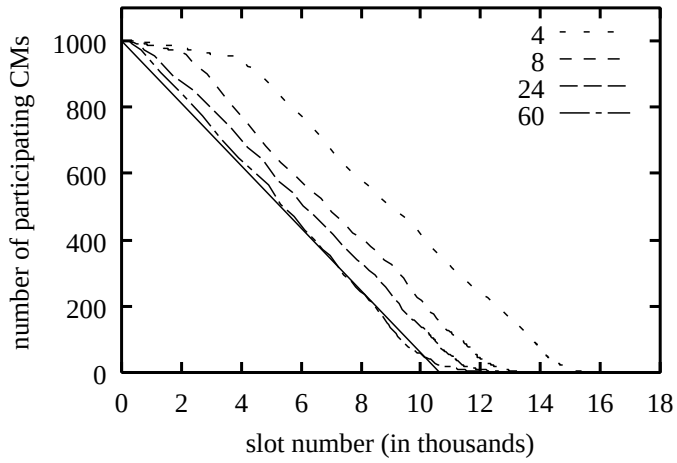


Figure 2.5. The number of participating CMs as a function of time for various values of $S_{\min}$.

gresses more slowly. This can be explained by considering that, when the minimal frame length has been reached, the channel is used less efficiently. This may have a detrimental effect on the performance of the system in terms of the delay that a CM incurs. This is further investigated in the third experiment.

The third observation is that the initial phase just after the start shows a slow start, in particular for small values of $S_{\min}$. Let us take a closer look at the process. Figure 2.6 illustrates, for the case that the minimal frame length is 8 slots, (*i*) the number of participating CMs (dashed line), (*ii*) the frame length (solid line) and (*iii*) the cardinality of each slot (dots), where the cardinality of a slot is defined as the number of CMs that transmit in this slot.

While there are slots with a high cardinality, the frame length remains minimal. When the cardinalities have become sufficiently small, the frame length increases rapidly and the steady decay phase starts.

This initial phase has to do with the simultaneous arrival of all CMs at the start of one frame and the time it takes to have them sufficiently distributed over the originating and feedback frames. If the number of empty slots is too large, resulting from a concentration of all CMs in just a few slots, relative to the total number of slots available, then the estimation of the number of participating CMs is too low. Although CMs generally do not arrive simultaneously, if the HE has not sent a sign-on request message for a while, such as after a power outage, a burst of CMs could be waiting for the next one. It is

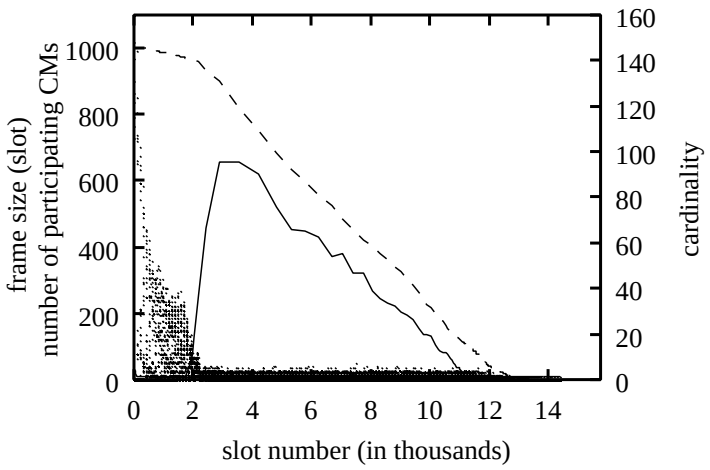


Figure 2.6. The number of participating CMs (dashed line), the frame length (solid line), and the cardinality of each slot, each as a function of time for the case that $S_{\min} = 8$.

interesting to analyze what happens. The process resembles a Galton board experiment, but with a 'wrap around'. This is explained next.

We assume that the frame length is minimal and that the minimal frame length does not divide the maximum feedback delay T_{fb}. Both assumptions are satisfied in the slow-start phase of the simulations. A CM that makes an unsuccessful transmission in frame i will transmit again, either in frame $i+k$ or in frame $i+k+1$, depending on the slot in which the transmission in frame i was done, where $k = \lceil T_{\text{fb}}/S_{\min} \rceil$. See Figure 2.7, where $k = 3$ and a transmission in frame i may occur before the start of the drawn feedback frame, in which case the CM will retransmit in frame $i+3$, or occur after this feedback frame has started, in which case the CM will retransmit in frame $i+4$. Note that the dashed line need not be positioned in the middle of a frame, so that the splitting need not be symmetric.

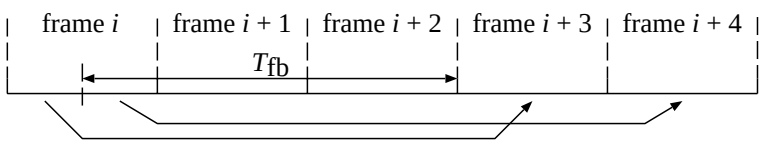


Figure 2.7. Retransmission alternatives after an unsuccessful transmission in frame i.

The same holds for frames $i+k$, $i+k+1$, $i+2k$, ..., and this can be drawn as shown in Figure 2.8. Assuming that all CMs arrive at the start of frame i, they first split into two groups, not considering the successful transmissions yet, and they retransmit in slots $i+k$ and $i+k+1$, respectively. Then the groups split again, but two groups join and retransmit in frame $i+2k+1$. The other two groups retransmit in frames $i+2k$ and $i+2k+2$, respectively. This process repeats itself and is called a Galton board experiment, until the group in frame $i+k^2$ is split. One subgroup retransmits in frame $i+k^2+k+1$, but the other subgroup retransmits in frame $i+k^2+k$, which is also occupied by another subgroup, at the other end of the board. Proceeding in this way, the board can be seen as if it is wrapped around a cylinder and the lower left part of the board is mapped onto the lower right part.

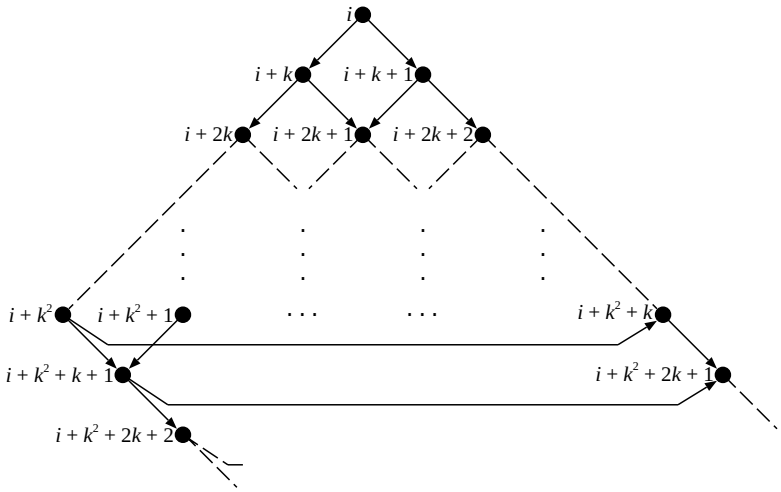


Figure 2.8. A Galton board with a wrap around

When this wrap around occurs, the process can be modeled as a circular Markov chain with $k+1$ indistinguishable states, initialized with a realization of a Binomial distribution. From the viewpoint of symmetry, the stationary distribution of this Markov chain is the homogeneous distribution.

However, two phenomena finally disturb this process. Firstly, if k is sufficiently large and the burst of CMs is not too large, the number of CMs near the edges of the board before it wraps around will be low, so that some will drop out of the process as a result of successful transmissions. Hence, the distribution of the CMs across the board will tend to remain narrower than a Binomial distribution. Secondly, at a certain moment, the number of slots perceived empty will become sufficiently small to have the frame length increase

beyond the minimal value, upon which the whole process is disturbed and the slow start is succeeded by the steady decay phase.

This is further illustrated in Figure 2.9, illustrating the slot cardinality during the initial phase in greater detail. For better visibility of the distributions, the dots are connected by lines.

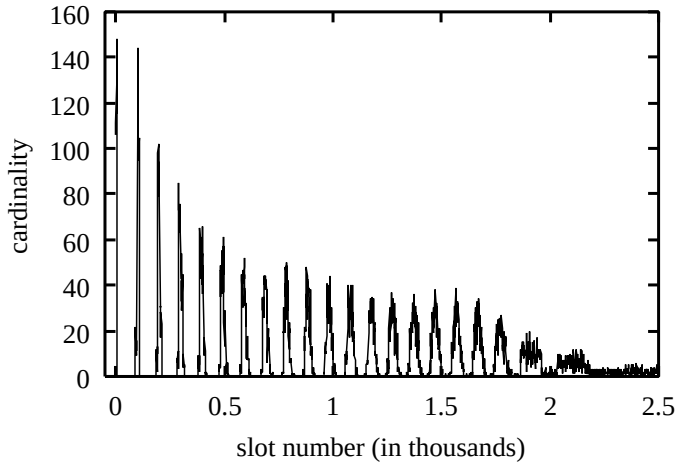


Figure 2.9. The cardinality of each slot as a function of time for the case that $S_{\min} = 8$.

It is clearly visible that the distributions become broader as time progresses. The number of empty slots between the distributions decreases, leading to the final disturbance of this process.

The analysis above explains the slow start effect, especially if the minimal period length is relatively small. In particular, if $S_{\min} = 8$, and consequently $k = 12$, only after at least $k(k+1) = 156$ frames, the above-mentioned Markov chain will start to homogenize the distribution. Before that, the number of empty slots remains relatively large and results in a minimal frame length. In the simulations, the frame length remains minimal until shortly after frame 210. By that time, already some 30 CMs have been successful in their transmission.

We make a number of final remarks. The slow-start problem also occurs if the feedback delay is negligible, but it is different. In this case, each unsuccessful CM retransmits in the next frame. The arrival of a burst of CMs in a single frame will cause zero empty slots, upon which the exponential growth in the length of the successive frames will start. This type of slow-start is also found in contention trees that are expanded in a breadth-first order: Initially,

many CMs contend in only a few slots, until the tree has become broad enough to allow successful transmissions (Denteneer & Pronk, 2001). Refining the estimation of the number of participating CMs by separately looking at the number of empty slots in contention frames, a sudden burst can be identified faster, although it does not completely solve the slow-start problem.

2.7.2 On inaccurately estimated channel model parameters

In the second experiment we investigate the sensitivity of the results to the channel model. It is conceivable that the channel parameters e_i and s_i are only available with limited accuracy, and that the channel model is not completely static, as was assumed earlier on. To this end, we consider the case that we consistently multiply the calculated frame sizes by a factor of 0.8 and 1.2, respectively, and compare the results with the original ones. We only consider the case where the minimal frame length equals 60 slots. The other cases give similar results. The burst size is again 1,000 CMs.

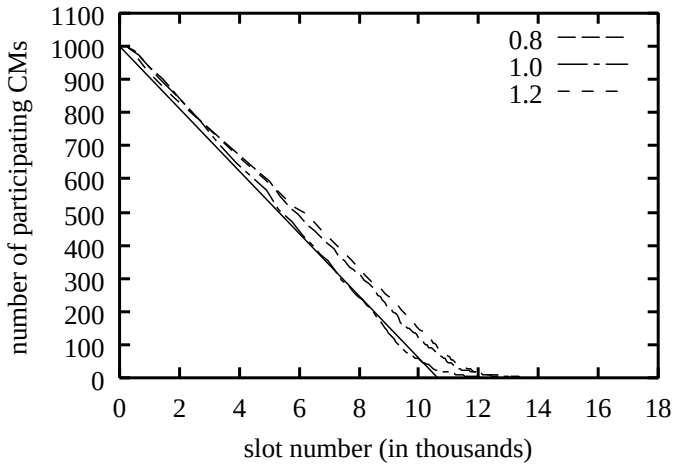


Figure 2.10. The effect of changing the frame length on the capacity of the contention channel

Figure 2.10 illustrates that the effect on the throughput of using an inaccurate channel characterization is marginal. The steady decay rates for the 0.8 and 1.2 cases are 0.089 and 0.088, respectively, approximately 6% less than the capacity of the channel. It appears that the performance of the system is not critically dependent on an accurate estimation of the channel parameters.

2.7.3 Dealing with a steady rate of arrivals

In the last experiment, we consider the situation that CMs arrive at a steady rate to investigate the delay that a CM incurs from the moment it arrives until the moment that its sign-on response message has been successfully received by the HE, as a function of the minimal frame length. We hereby neglect the propagation delay of a message to the HE, which is negligible when compared to the delay in the contention process.

We assume that CMs arrive according to a Poisson process with a given arrival rate and investigate the mean and standard deviation of the delay a CM incurs as a function of the arrival rate and for the same values of $S_{\min}$ as in the first experiment. Figure 2.11 illustrates the mean delays. The standard deviations are in the same order as the mean values.

As expected, the mean delay increases as the load increases, for each value of the minimal frame length. As, at a fixed load, the value of the minimal frame length increases, so does the mean delay for each fixed load. This was already anticipated in the analysis of the first experiment.

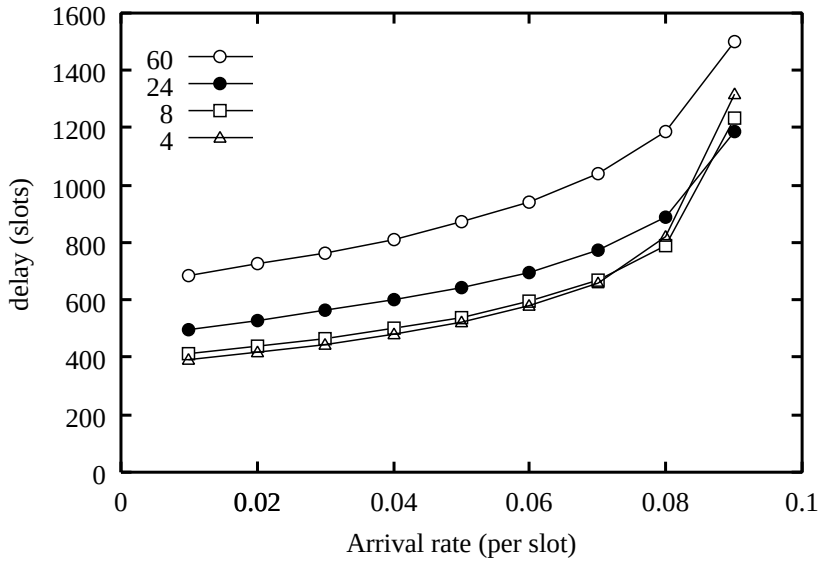


Figure 2.11. Mean delay of CMs as a function of the load for various values of the minimal frame length.

Especially at low loads, the difference between the results for a minimal frame length of four slots and 60 slots amounts to a factor of approximately 1.75, whereas at higher loads this tends to decrease and, in fact, actually vanishes.

This can be explained by observing that at higher loads, the frame lengths will more often be larger than the minimal frame lengths, so that the role of the minimal frame length vanishes. The lack of a clear convergence in the simulations is due to the fact that simulating under conditions close to capacity are difficult to perform accurately.

The choice for an appropriate minimal frame length is thus a compromise between low delays during steady-state operation and the time it takes to resolve a sudden burst. As a sudden burst would typically occur in unusual circumstances such as after a power outage, a more advanced strategy to choose an appropriate minimal frame length is conceivable.

2.8 Concluding Remarks

In this chapter, we have investigated the process a CM must go through before it can register at the HE and become operational. During this process, it must follow a contention-based access protocol, similar to frame-based ALOHA. The context gives rise to an alternative channel model and alternative traffic characteristics than are usually assumed in the literature. The main issue in this chapter was the determination of the optimal length of each frame.

To this end, an estimation procedure has been proposed, whereby the channel is monitored and the number of slots perceived as empty is used to obtain an estimation of the number of CMs currently involved in the access protocol. Based on this estimation, a frame length is calculated. Complicating factor is the existence of a non-negligible feedback delay.

Simulation results illustrate the near-optimality of the procedure and its robustness against inaccurate channel model parameters. Of particular interest is that the minimal frame length, imposed upon the frame length to limit the downstream signaling overhead, should not divide the maximum feedback delay. This minimal frame length has a significant impact on the performance of the system.

Further simulations with a more general channel model than assumed in the simulations may show the effectiveness of the estimation procedures under more general conditions. Another topic for further research is to investigate strategies to adapt the minimal frame length to the current circumstances.

3

Request Merging in Cable Networks

In this chapter, we consider cable modems (CMs) in normal operation. In this mode, a CM sustains a number of connections on behalf of the applications running in the home and transmits data to and receives data from the head end (HE) for these applications. The primary access mode for the upstream transmission of data provides best-effort service, guaranteeing data delivery, but without guarantees on timeliness. It remains nevertheless important to provide fast data transmission to, for instance, support the soft real-time requirements on response times.

In both the DVB/DAVIC and DOCSIS standards, this primary access mode is governed by a *request-grant* procedure, wherein first a request is transmitted to the HE for the transmission of the actual data. After reception of such a request, the HE issues a grant stating when the requesting CM can transmit the actual data.

The transmission of a request is done using a contention-based access protocol, where multiple CMs may send a request simultaneously, in which case these requests collide, resulting in the loss of all of them. A retransmission scheme is employed to ensure that the requests will eventually arrive successfully at the HE. As the transmission of the actual data is controlled by the HE, this can be done contention-free.

The way in which a CM deals with multiple, simultaneously active request-grant procedures on behalf of the connections it sustains is not specified in the standards mentioned. Although this leaves some freedom in implementation, the standards are currently not flexible enough to integrate the request parts of these procedures in an orderly fashion.

In this chapter we investigate the merits of merging single requests from a number of connections sustained by one CM into one *multi-request*, so that the request parts of several request-grant procedures can be integrated. This merging is such that the original requests can be retrieved from the multi-request. The rationale behind this approach is that the length of such a multi-request is significantly smaller than the sum of the lengths of the individual requests, where the length pertains to the number of bytes required to encode a request in a frame that is transmitted.

Finally, we mention that this work is based an earlier publication by Pronk & Rietman [2002].

3.1 Introduction

As each upstream channel in an HFC network is effectively a shared bus, some medium access control is necessary to efficiently utilize the channel bandwidth. One of the access modes defined in the standards is the request-grant procedure wherein requests from connections for the contention-free transmission of data are transmitted first, in contention with requests from other connections. After reception of a request, the HE issues a grant and transmits it to the requesting connection for the transmission of the actual data. The HE never issues a larger or smaller grant than requested, but in a practical situation, a grant may consist of multiple smaller grants to achieve a more fine-grained interleaving of data from several connections. For resolving the possible collisions during the contention phase among requests of different connections, a contention resolution protocol is employed, which governs when the necessary retransmissions should be done.

Hence, at the expense of transmitting the relatively short requests in contention with others, the actual data can be transmitted contention-free, generally leading to a significantly more efficient use of the shared channel than when the actual data would be sent in contention.

To allow both access modes in the same channel, the HE performs bandwidth allocation to each of the access modes. Typically, a small percentage of the bandwidth is allocated to contention-based access and the remaining bandwidth to contention-free access. Van Leeuwaarden, Denteneer & Resing [2006] analyze the problem of how to divide the bandwidth among these ac-

cess modes. Pronk [2000] describes in detail the problem of bandwidth allocation in a DVB/DAVIC-compliant system, where more than two access modes are present.

It is important to note that, during the time that a connection is trying to have its request transmitted to the HE, the size of this request yet to be transmitted or retransmitted can be updated, that is, increased to incorporate any newly arriving data for this connection that has to be transmitted upstream as well. This type of merging establishes that, for each connection, there is at most one request in the process.

The time elapsed between the generation of a request and its successful reception by the HE is called the *medium access delay* of this request. It goes without saying that the medium access delay has a direct influence on the transmission delay that the actual data incurs before it is received by the HE and, consequently, directly influences the response times for the applications that run in the homes.

A CM, however, can sustain multiple connections simultaneously, and it is the CM that does the actual transmissions on behalf of its connections. As a result, a CM must somehow deal with the problem of multiple, simultaneous request-grant procedures. How this should or may be done is not specified in the standards.

One seemingly obvious, but impractical way to do this is to add the request sizes from the different connections together and send it as one request. The problem with this approach is that, in that case, the HE only receives aggregated requests and can not distinguish anymore between the requests from the individual connections, which may lead to unfairness in the division of bandwidth to the connections among the CMs.

One possibility is to handle requests in a first-come-first-serve (FCFS) order, so that for each CM, there is at most one request being handled, while others are waiting in a queue until the request currently being handled has arrived successfully at the HE. However, as we shall see, this does not lead to a better medium access delay than when all requests are handled truly independently, as if for each connection there would be a separate CM.

In this chapter, we propose an alternative solution, called *request merging*. Request merging pertains to combining multiple requests from several connections associated to a single CM into a *multi-request*, which is then transmitted to the HE. The integration of multiple requests into one multi-request is such that the original requests can be retrieved again by the HE, so that this merging is transparent to the overall request-grant procedure. The length of a multi-request is typically significantly smaller than the sum of the lengths

of the individual requests, where the length pertains to the number of bytes required to encode a request in a frame that is transmitted.

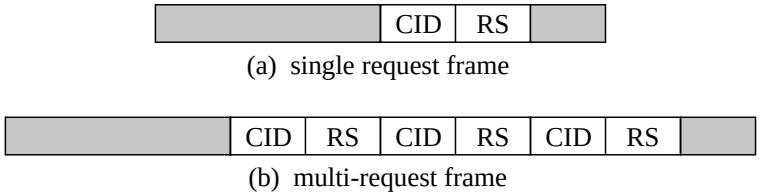
We concentrate on the contention resolution part of the request-grant procedure described to investigate the medium access delay and abstract from details such as which scheduling algorithm is used by the HE and how this scheduler is implemented. Furthermore, the use of multi-requests neither dictates nor precludes the use of any particular contention resolution protocol. We exploit this freedom by considering contention resolution based on contention trees, which is supported by DVB/DAVIC. DOCSIS uses a binary exponential back-off algorithm. Contention trees have been widely investigated in the literature [Capetanakis, 1979; Tsybakov & Mikhailov, 1978; Janssen & De Jong, 2000; Denteneer, 2005], which eases analysis. They will be explained shortly.

The remainder of this chapter is organized as follows. In Section 3.2, the multi-request is defined in more detail, and its use is explained. Section 3.3 contains an analytical treatment of the medium access delay when using requests and when using multi-requests. Then, two scenarios are compared in Section 3.4 and simulation results are given in Section 3.5. We end with some concluding remarks in Section 3.6.

3.2 Defining multi-requests

A request frame carrying a single request is typically composed of a header, and a connection ID and request size field. This header includes (*i*) physical-layer overhead such as a guard band for safely separating subsequent bursts from different CMs and a preamble that is required for demodulation of the received burst, and (*ii*) some medium-access-control-layer overhead, such as a control field for identifying the request as such and a check sequence for error detection. As a result, only a relatively small part, typically one third, of the entire request is dedicated to the connection ID and the request size. Figure 3.1a illustrates the structure of a single request frame.

Defining a multi-request frame can be done by extending the request frame with more (connection ID, request size)-pairs, resulting in a relatively modest increase in total size per added pair, see Figure 3.1b. In a practical situation, it is reasonable to assume that a multi-request can contain up to a maximum number of requests. Suppose that the length of a request frame is 1, and that a multi-request can consist of a maximum of R requests, then the length of a multi-request frame can be expressed as $1+\alpha(R-1)$, with $0 \leq \alpha < 1$. This maximum R can be made dependent on the current load, which can be estimated from the number of requests received per multi-request. The successful



(a) single request frame

(b) multi-request frame

Figure 3.1. Structure of a single-request and a multi-request frame. The grey areas are physical- and medium-access-control-layer overhead, CID stands for connection ID, and RS for request size.

reception by the HE of a multi-request is equivalent to the successful reception of all of the requests it accommodates.

Identically to single requests, the size of a request in a multi-request can be updated to incorporate new data. In addition, if for a CM, its current multi-request is not yet full, that is, it carries less requests than it can maximally contain, a request from a newly active connection can be added on the fly to the multi-request by including it in its next transmission. After all, also a multi-request may have to be transmitted repeatedly before it is successfully received by the HE.

3.3 Modeling and analysis

We start by introducing some nomenclature pertaining to the states in which a CM and a connection can be. A connection is called idle if its pending grant size, that is, the remainder of the grant it still expects to receive, corresponds to the amount of data enqueued for transmission. When idle, a connection thus does not need to transmit a request, and any data enqueued can be transmitted using the pending grant. If a connection is not idle, it is called active, in which state the amount of enqueued data exceeds the pending grant size. Upon becoming active, a connection instructs its CM to transmit a request. Upon successful reception of the request by the HE, it increases for this connection its pending grant size and the connection becomes idle again. For simplicity, we assume that the upstream signal propagation delay of a request is negligible. A CM is called idle if all of its connections are idle, and called active otherwise. CMs as well as connections thus toggle between active and idle, and we can speak of CM and connection idle and active times , both denoting the intervals during which they are continuously idle and active, respectively.

Besides the CM and connection active time, we can speak of a request-update active time. It is defined as the time that elapses between the generation of a request or update of a request and its successful reception at the HE. Although not strictly necessary in the analysis below, we include this definition out of theoretical interest.

In this section we primarily consider the connection active time, which equals the medium access delay defined earlier. The analysis uses results on the CM active time, which are presented first. After analyzing the connection active time, we elaborate a little further on the request-update active time.

3.3.1 On the CM active time

We consider N independent CMs with C independent connections per CM. Each connection generates requests according to a Poisson process with rate λ/C, so that the (remaining) idle time of a CM is exponentially distributed with mean λ^{-1}. When an idle CM becomes active, it starts to contend with other active CMs in a blocked contention-tree process [Capetanakis, 1979]. In this process, a CM does a first transmission in a randomly chosen slot among a number of dedicated slots, collectively called the root of a tree. Then, for each slot containing a collision, a new set of slots is allocated in which only the colliding CMs in that slot do a retransmission, again in a randomly chosen slot from this set. This creates a tree structure, as shown in Figure 3.2, where each node consists of three slots.

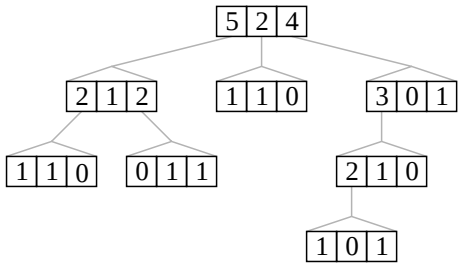


Figure 3.2. A blocked contention tree, with three slots per node. The number in each slot denotes the number of CMs that transmitted in this slot. The top node is the root. In each slot s with more than one transmission, a new node, consisting of three slots, is allocated. Each of the CMs that transmitted in slot s do a retransmission in a randomly chosen slot of this node.

The number of slots in each node of the tree is typically three, in which case the term ternary tree is used. The term blocked relates to the requirement that a CM may only enter the tree process via the root. In contrast, a non-blocked or free-access tree allows a CM to enter the process in any node of the tree.

The nodes in the tree are ordered in time, by the HE, to be multiplexed on the upstream channel, typically in breadth-first order. We assume that, upon completion of one tree, another is started, and that at any time there exists exactly one tree. Hence, when an idle CM becomes active, it first has to wait for the root of a tree.

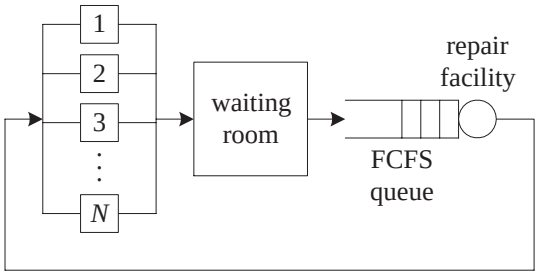


Figure 3.3. Gated machine-repair model with N machines.

This blocked contention-tree protocol can be modeled as a gated machine-repair model, as described by Boxma, Denteneer & Resing [2002]. See Figure 3.3. In this model, each of the N machines can break down, independently of one another, after an exponentially distributed working time with parameter λ. Upon a machine breaking down, it first enters a waiting room. This room is emptied into the FCFS queue as soon as the repair facility becomes idle. The order in which waiting machines enter the FCFS queue is random. If the repair facility is already idle and a machine breaks down, then this machine need not go through the waiting room and can enter the FCFS queue immediately.

The machines model CMs, and the breaking down of a machine models the corresponding CM becoming active. A machine finishing service at the repair facility corresponds to the successful reception by the HE of the (multi-)request of the CM and the CM becoming inactive again. The repair facility becoming idle corresponds to the completion of a contention tree and the emptying of the waiting room corresponds to starting up a new contention tree.

In case a blocked, ternary contention tree protocol is used for contention resolution, a reasonable model for the repair facility in this machine-repair model is a server with exponential service time with rate $\mu = \ln(3)/3$, see Mathys & Flajolet [1985], but also Janssen & De Jong [2000]. It is noted that Denteneer & Pronk [2001] analyzed contention trees and found that the server could alternatively be described as having a more or less constant service time, with a small positive offset for the first machine after the waiting room has been emptied. Using this would, however, complicate the analysis.

It is assumed that the maximum number R of single requests in a multi-request is equal to the number C of connections per CM. Clearly, $R > C$ is not very useful. The case that $R < C$ would result in a more complicated model: a CM may become active again immediately after finishing service, since, while the CM is active, more than $R-1$ additional connections may become active, which cannot all be accommodated immediately. In this case, the service at the repair facility is said to be R-limited. If $R \geq C$, it is said to be exhaustive, so that after finishing service, a CM becomes idle again. The former is much more complicated to analyze than the latter [Borst, 1996]. In Section 3.5, we do consider in the simulations the more general case that $R \leq C$.

Let the CM active time be denoted by T_{m}. The first two moments of T_{m} are (approximately) known in the regimes $N\lambda \gg \mu$ and $N\lambda \ll \mu$. In the second regime, very few connections are active, so there is little to be gained from merging. The first regime is therefore more interesting.

Lemma 3.1. (Boxma, Denteneer & Resing [2002]) *if $N\lambda \gg \mu$, then the average τ_{m} and variance σ_{m}^2 of T_{m} can be approximated by*

$$\tau_{\mathrm{m}} \approx \frac{N}{\mu} - \frac{1}{\lambda} \tag{3.1}$$

and

$$\sigma_{\mathrm{m}}^2 \approx \frac{1}{6}\left(\frac{N}{\mu} - \frac{1}{\lambda}\right)^2. \tag{3.2}$$

□

3.3.2 On the connection active time

Recall that a CM becomes active if one of the connections it sustains becomes active and that, while the CM is active, additional connections it sustains can become active, until it becomes idle again, upon which all its active connections becomes idle again as well. This is illustrated in Figure 3.4.

A convenient way to derive moments of unknown distributions is to use the Laplace-Stieltjes transform, as it provides a simple way to express the moments in terms of the derivatives of the transform.

We next derive an expression for the Laplace-Stieltjes transform of the connection active time in terms of that of the CM active time. The Laplace-Stieltjes transform [Grimmett & Stirzaker, 2001] is defined as follows.

Definition 3.1. (Laplace-Stieltjes transform) Let X be a non-negative random variable with cumulative probability density function $P(x)$. The Laplace-

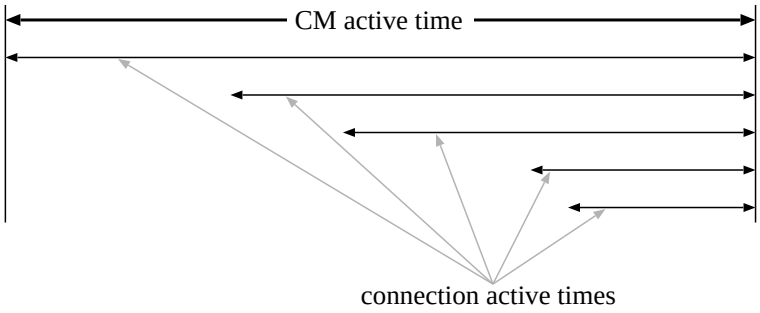


Figure 3.4. Relation between CM and the connection active times.

Stieltjes Transform $\varphi_X(u)$ of X is defined as

$$\varphi_X(u) = \boldsymbol{E}[e^{-uX}] = \int_{x=0}^{\infty} e^{-ux}\,\mathrm{d}P(x). \tag{3.3}$$

□

Lemma 3.2. *The average μ and variance σ^2 of a non-negative random variable X with Laplace-Stieltjes Transform $\varphi_X(u)$ is given by*

$$\mu = \left[-\frac{\mathrm{d}}{\mathrm{d}u}\varphi_X(u)\right]_{u=0} \quad \textit{and} \tag{3.4}$$

$$\sigma^2 = \left[\frac{\mathrm{d}^2}{\mathrm{d}u^2}\varphi_X(u) - \left(\frac{\mathrm{d}}{\mathrm{d}u}\varphi_X(u)\right)^2\right]_{u=0}. \tag{3.5}$$

Proof. This follows easily from Definition 3.1. □

Theorem 3.1. *Let the Laplace-Stieltjes transform of the* CM *active time T_{m} be denoted by $\varphi_{\mathrm{m}}(u)$. Then the Laplace-Stieltjes transform $\varphi_{\mathrm{c}}(u)$ of the connection active time T_{c} is given by*

$$\varphi_{\mathrm{c}}(u) = \frac{\varphi_{\mathrm{m}}(u) + (C-1)\left(\varphi_{\mathrm{m}}(u) - \varphi_{\mathrm{m}}(\lambda/C)\right)/(1-uC/\lambda)}{1+(C-1)\left(1-\varphi_{\mathrm{m}}(\lambda/C)\right)}. \tag{3.6}$$

Proof. The approach is that we make a summation of e^{-ut}, where t is the time that a connection is active, over all active connections during a long time period and divide this sum by the total number of active connections during this time period. In particular, we consider k consecutive CM active times.

During the ith CM active time, with $i = 1, 2, \ldots, k$, we denote the sum of e^{-ut} over all active connections during this CM active time by X_i and the number of active connections involved by Y_i. We have that

$$\varphi_c(u) = \lim_{k\to\infty} \frac{\sum_{i=1}^{k} X_i}{\sum_{i=1}^{k} Y_i}. \tag{3.7}$$

For large values of k, we can approximate the numerator and the denominator in Equation 3.7 by $k\bar{X}$ and $k\bar{Y}$, respectively, where $\bar{X}$ and $\bar{Y}$ denote the expected value of the X_i's and Y_i's, respectively. Now, k can be eliminated. Let $P_m(t)$ denote the probability that the CM active time is at most t. We next prove that

$$\bar{X} = \int_{t_m=0}^{\infty} \left[e^{-ut_m} + (C-1) \int_{t=0}^{t_m} (\lambda/C)\, e^{-\lambda t/C} e^{-u(t_m - t)}\, \mathrm{d}t \right] \mathrm{d}P_m(t_m). \tag{3.8}$$

Consider a CM active time of length t_m. The connection that makes a CM active contributes an amount e^{-ut_m} as this connection becomes idle again when the CM becomes idle again. The $C-1$ other connections independently contribute the other terms, as a connection that becomes active a time t after the CM becomes active, with $t \in (0, t_m)$, becomes idle again when the CM becomes idle again. Note that each of the other connections only contributes with a probability that is smaller than 1. This establishes Equation 3.8. In the same vein, it can be proved that

$$\bar{Y} = \int_{t_m=0}^{\infty} \left[1 + (C-1) \int_{t=0}^{t_m} (\lambda/C)\, e^{-\lambda t/C}\, \mathrm{d}t \right] \mathrm{d}P_m(t_m),$$

by counting a contribution of 1 for each connection that becomes active. The integrals over t can readily be computed and the subsequent integrations over t_m yield for $\bar{X}$ and $\bar{Y}$ the numerator and denominator, respectively, of Equation 3.6. □

Corollary 3.1. *The average active time τ_c for a connection is given by*

$$\tau_c = \frac{C\tau_m - \frac{C(C-1)}{\lambda}\left(1 - \varphi_m(\lambda/C)\right)}{1 + (C-1)\left(1 - \varphi_m(\lambda/C)\right)}, \tag{3.9}$$

where τ_m is the average CM *active time.*

Proof. Using Lemma 3.2, we have that

$$\tau_c = \left[-\frac{\mathrm{d}}{\mathrm{d}u} \varphi_c(u) \right]_{u=0}. \tag{3.10}$$

Using Theorem 3.1 and the fact that

$$\tau_{\mathrm{m}} = \left[-\frac{\mathrm{d}}{\mathrm{d}u} \varphi_{\mathrm{m}}(u) \right]_{u=0},$$

Equation 3.9 follows by straightforward differentiation of $-\varphi_{\mathrm{c}}(u)$, given in Equation 3.6, with respect to u and substituting $u = 0$. □

For calculating τ_{c}, we thus need an expression for $\varphi_{\mathrm{m}}(\lambda/C)$. Returning to the machine-repair model, Lemma 3.1 is the only information available. As an approximation, we could use the so-called *cumulant expansion* of $\varphi_{\mathrm{m}}(u)$, which is the Taylor series expansion of the logarithm of $\varphi_{\mathrm{m}}(u)$ about $u = 0$. Reason for choosing this expansion rather than the Taylor expansion of $\varphi_{\mathrm{m}}(u)$ about $u = 0$ is that the former generally requires fewer terms. The cumulant expansion of $\varphi_{\mathrm{m}}(u)$ is given by

$$\varphi_{\mathrm{m}}(u) = e^{-\tau_{\mathrm{m}} u + \frac{\sigma_{\mathrm{m}}^2}{2} u^2 + O(u^3)}.$$

However, the positive second term and the lack of knowledge of any higher-order terms in this expansion make the use of this second term doubtful: for large values of u, $\varphi_{\mathrm{m}}(u)$ should vanish. Hence, we propose to use the following approximation for $\varphi_{\mathrm{m}}(u)$.

$$\varphi_{\mathrm{m}}(u) \approx e^{-\tau_{\mathrm{m}} u}$$

Reason for choosing this approximation is also that it results in a good fit with the simulations described in the next section.

3.3.3 On the request-update active time

Using Equation 3.6, it is also possible to obtain an expression for the average request-update active time τ_{r}, although this is more of theoretical interest. It is not used in the remainder of the chapter. Once a connection becomes active because it generates a first request, updates may arrive, with rate λ/C, during its active time, similar to additional connections becoming active during their CM active time as described above. The only difference is that the process of additional connections becoming active slows down as more connections become active, whereas the process of generating request updates does not. For ease of exposition, we also consider the first request generated by a newly active connection a request update.

Theorem 3.2. *The Laplace-Stieltjes transform* $\varphi_r(u)$ *of the request-update active time* T_r *is given by*

$$\varphi_r(u) = \frac{\varphi_c(u) + \lambda(1-\varphi_c(u))/(uC)}{1+\lambda\tau_c/C}$$

Proof. The process of generating request updates can be mimicked by a large set of k 'potential' updates, each becoming 'active' at rate $\lambda/(Ck)$, and taking the limit $k \to \infty$. Hence,

$$\varphi_r(u) = \lim_{k\to\infty} \frac{\varphi_c(u) + (k-1)\,(\varphi_c(u) - \varphi_c(\lambda/(Ck)))/(1-uCk/\lambda)}{1+(k-1)\,(1-\varphi_c(\lambda/(Ck)))}.$$

The result follows by observing that $\varphi_c(\lambda/(Ck)) \to 1$ as $k \to \infty$ and that, by using the Taylor expansion about $t_c = 0$ of $e^{-\lambda t_c/(Ck)}$, it follows that $(k-1)\,(1-\varphi_c(\lambda/(Ck))) \to \lambda\tau_c/C$ as $k \to \infty$. □

By combining the above result with Equations 3.6 and 3.9 we obtain an expression for $\varphi_r(u)$ directly in terms of the average and the Laplace-Stieltjes transform of the CM active time.

Corollary 3.2.

$$\varphi_r(u) = \frac{\varphi_m(u) + \lambda(1-\varphi_m(u))/u}{1+\lambda\tau_m}. \tag{3.11}$$

□

An alternative way to derive this result is to consider all requests during the active time of a CM, sum e^{-ut} for all these requests, where t is the time that a request is "active", and divide this sum by the expected number of requests, similar to the proof of Theorem 3.1. As the requests are generated by a Poisson process with rate λ, each request, except the first one that activates the CM, is uniformly distributed over the active time. Therefore,

$$\varphi_r(u) = \frac{1}{1+\lambda\tau_m} \int_{t_m=0}^{\infty} \left[e^{-ut_m} + \sum_{n=0}^{\infty} \frac{(\lambda t_m)^n}{n!} e^{-\lambda t_m} n \int_{t=0}^{t_m} e^{-u(t_m-t)} \frac{1}{t_m}\,\mathrm{d}t \right] \mathrm{d}P_m(t_m), \tag{3.12}$$

which is equivalent to Equation 3.11, as can be shown by straightforward calculus.

Corollary 3.3. *The average request update active time* τ_r *is given by*

$$\tau_r = \frac{\tau_m + \frac{\lambda}{2}(\tau_m^2 + \sigma_m^2)}{1+\lambda\tau_m}.$$

Proof. Using Corollary 3.2, we have that

$$\frac{\mathrm{d}}{\mathrm{d}u}\varphi_{\mathrm{r}}(u) = \frac{\frac{\mathrm{d}}{\mathrm{d}u}\varphi_{\mathrm{m}}(u) - \left[\left(\frac{\mathrm{d}}{\mathrm{d}u}\varphi_{\mathrm{m}}(u)\right)\frac{\lambda}{u} + (1-\varphi_{\mathrm{m}}(u))\frac{\lambda}{u^2}\right]}{1+\lambda\tau_{\mathrm{m}}} \tag{3.13}$$

We next use the Taylor series expansion about $u = 0$ of $\varphi_{\mathrm{m}}(u)$ to rewrite the term in the square brackets. In particular, we have that

$$\varphi_{\mathrm{m}}(u) = 1 - \tau_{\mathrm{m}}u + \frac{\sigma_{\mathrm{m}}^2 + \tau_{\mathrm{m}}^2}{2}u^2 + O(u^3) \tag{3.14}$$

and, consequently,

$$\frac{\mathrm{d}}{\mathrm{d}u}\varphi_{\mathrm{m}}(u) = -\tau_{\mathrm{m}} + (\sigma_{\mathrm{m}}^2 + \tau_{\mathrm{m}}^2)u + O(u^2). \tag{3.15}$$

If we substitute these results in Equation 3.13, we can substitute $u = 0$. The result now follows by using Lemma 3.2. □

The average request-update active time can be thus be expressed in terms of the average and the variance of the CM active time and does not contain $\varphi_{\mathrm{m}}(u)$.

3.4 Comparison of two scenarios

It is interesting to compare the following two scenarios.

1. $C = 1$, $N = N_1$, $\mu = \mu_1 = \ln(3)/3$, $\lambda = \lambda_1$. In this scenario there is no merging, there are N_1 connections, one per CM, each generating requests at rate λ_1.
2. $C > 1$, $N = N_1/C$, $\mu = \mu_1/\rho$, $\lambda = C\lambda_1$. Here, the factor ρ equals $1+\alpha(C-1)$, and models the slowing down of the contention process due to the increase in length of a multi-request frame. Also in this scenario there are N_1 connections, each generating requests at rate λ_1.

The parameters were chosen such that by substituting $C = 1$ in the formulas for the second scenario, the first scenario is recovered. Note that in Scenario 1 we use N_1 CMs only to establish that all connections operate independently in terms of their request processes. This corresponds exactly with the situation where there are only N_1/C CMs, such as in Scenario 2, but where each CM sustains C independently operating connections. In Scenario 2, each CM handles the request processes of its connections by using request merging.

Using a superscript to indicate either Scenario 1 or 2, it holds that

$$\tau_{\mathrm{c}}^{(1)} = \tau_{\mathrm{m}}^{(1)} \approx \frac{N_1}{\mu_1} - \frac{1}{\lambda_1} \tag{3.16}$$

by using Lemma 3.1, and

$$\tau_c^{(2)} = \frac{C\tau_m^{(2)} - \frac{(C-1)}{\lambda_1}(1-\varphi_m(\lambda_1))}{1+(C-1)(1-\varphi_m(\lambda_1))}$$

by using Corollary 3.1. In the latter equation, again using Lemma 3.1, we have that

$$\tau_m^{(2)} \approx \frac{\rho N_1}{C\mu_1} - \frac{1}{C\lambda_1}, \tag{3.17}$$

where $\rho = 1+\alpha(C-1)$. For small and intermediate values of λ_1, we propose to approximate $\varphi_m(\lambda_1)$ by

$$\varphi_m(\lambda_1) \approx e^{-\tau_m^{(2)}\lambda_1}.$$

For large values of λ_1, $\varphi_m(\lambda_1)$ vanishes, so that under this condition, it holds that

$$\tau_c^{(2)} \approx \frac{\rho N_1}{C\mu_1} - \frac{1}{\lambda_1}.$$

Comparison with $\tau_c^{(1)}$ in Equation 3.16 shows that under this condition, the connection active time is improved if the increase in length of a multi-request frame is offset by the corresponding increase in the number of requests that fit in one multi-request, that is, if $\alpha < 1$.

3.5 Simulations

We have performed simulations to compare the average connection active time τ_c for the cases of single requests and multi-requests, using a blocked, ternary contention-tree protocol. A constant-bandwidth share of the overall channel bandwidth is used for (multi-)request frames. A round-trip delay of 0 is assumed.

parameter	*Scenario* 1	*Scenario* 2
number of CMs	500	100
number of connections C per CM	1	5
use of multi-requests	No	Yes
R	–	$1,2,3,4,5$
α	–	$\frac{1}{3}$
λ_1	varied	

Table 3.1. The two simulation scenarios.

We consider two Scenarios 1 and 2, see Table 3.1, with $N_1 = 500$ connections and $C = 1$ and 5 connections per CM, respectively, each connection generating packets according to a Poisson process with rate $\lambda_1 = 0.001, \ldots, 0.02$ in each scenario, corresponding to a total of 0.5 packets per unit time to 10 packets per unit time, respectively. Here, unit time denotes the length of a slot that can contain a single request frame. In Scenario 1, with 500 CMs, multi-requests are not used, as they are not useful with only one connection per CM, and in Scenario 2, with 100 CMs, multi-requests with up to $R = 1, 2, \ldots, 5$ requests are used. For the simulations, we thus do not restrict ourselves to the case that $R = C$ as we did in the analysis, but to the general case that $R \leq C$. The length of a multi-request frame with up to R requests is given by $1 + \alpha(R-1)$, with $\alpha = 1/3$. So, for a fixed value of R, the multi-request frame length is fixed, irrespective of how many requests it actually contains. In Scenario 2, the individual connections of a CM are served in FCFS order, in case their requests cannot be accommodated immediately in a (multi-)request frame. This, of course, only occurs in the cases that $R < C$.

The value $R = 1$ in Scenario 2 corresponds to serving connections of a single CM one by one, which differs from Scenario 1 in that in the latter, all connections operate independently of each other.

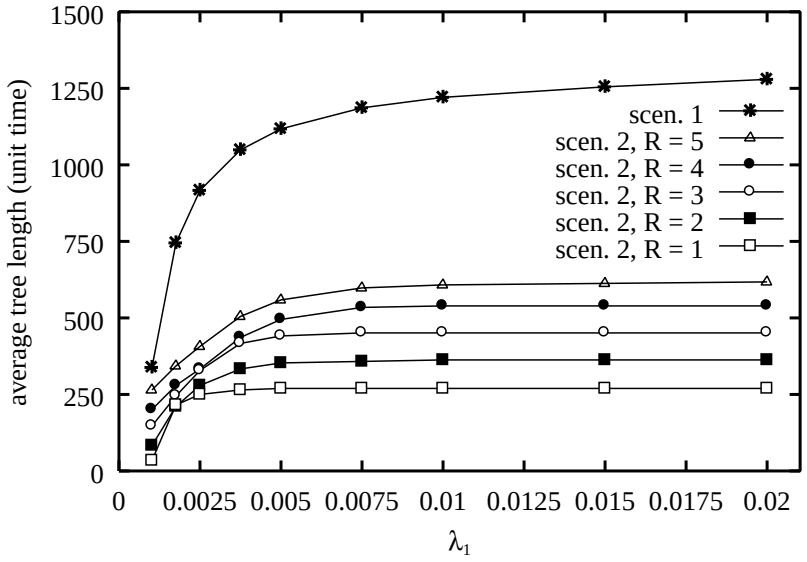


Figure 3.5. Average tree lengths for the two scenarios and the various values of R.

Figure 3.5 illustrates the average tree length, that is, the average time required to complete a single tree, as a function of the load (λ_1) for Scenario 1 and for the various values of R in Scenario 2. In Scenario 2, the average tree lengths are considerably smaller than those in Scenario 1. Despite the increase in length of multi-request frames, the number of contenders per tree in Scenario 2 is bounded by 100, whereas in Scenario 1, this bound is 500, which grossly explains the difference towards the higher loads. In Scenario 2, towards the higher loads, the larger tree lengths for increasing values of R correspond to the increase in length of multi-request frames.

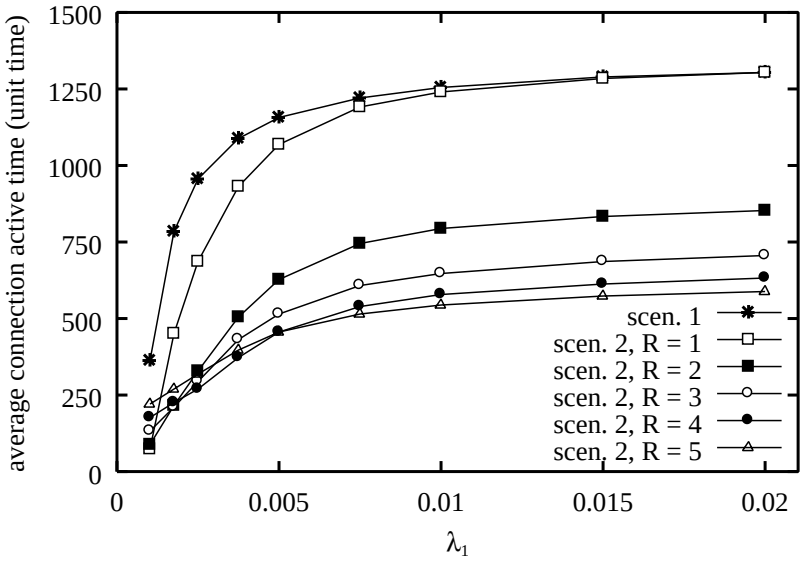


Figure 3.6. Average connection active times, based on the simulations.

More important to compare are the average connection active times. Figure 3.6 illustrates the corresponding average connection active times. For not too low loads, the average connection active times for multi-requests with $R > 1$ are significantly smaller than those with $R = 1$ and those in Scenario 1. The figure also illustrates that, for the higher loads, the average connection active time in Scenario 2 with $R = 1$, which corresponds to serving the 5 connections of an individual CM in FCFS order, approach those of Scenario 1, which corresponds to the situation wherein all 5 connections of an individual CM would operate independently and could cause collisions among themselves.

The figure also illustrates that, as the load decreases, the lower values of R lead to better results. Clearly, in the case that $R > 1$, under low loads, the multi-requests are generally not full, that is, they contain less than R single

requests, leading to an inefficient use of the channel for multi-requests. This suggests that making R dependent on the current load, starting at 1 for low loads and increasing it to 4 or 5 in this scenario towards higher loads, pays off in terms of the average connection active time, and thus in terms of the medium access delays.

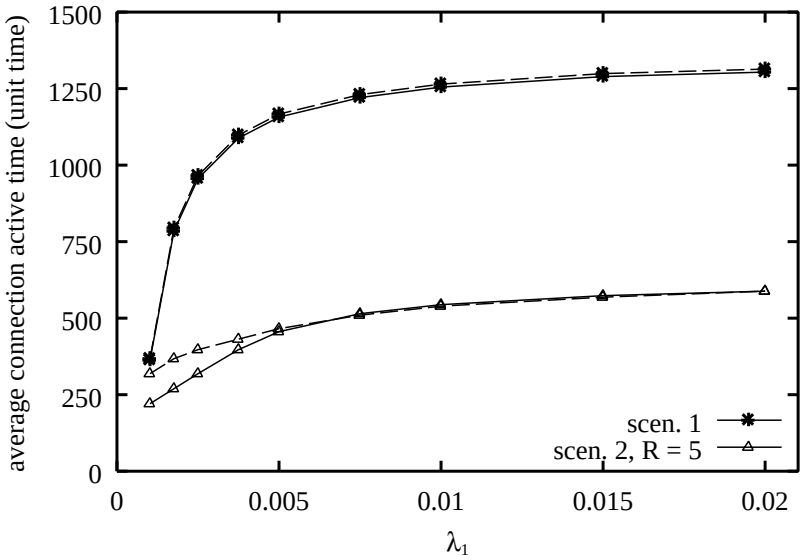


Figure 3.7. Comparison of the simulation and analytical results on the average connection active times.

In Figure 3.7, we compare some of the simulation results with the analytical results derived earlier. In Scenario 2, we only consider the case $R = 5$. The solid lines give the results from the simulations and the dashed lines give the corresponding analytical results from the two scenarios considered in the previous section.

The figure shows that, towards the higher loads, the analytical results show a good match with the simulations. Towards the lower loads, the analytical results in Scenario 2 deviate. This is presumably caused by the inaccuracy in the approximation of τ_m, given in Equation 3.1, for low values of λ_1.

3.6 Concluding remarks

In this chapter, we have introduced the notion of multi-requests in the request-grant procedure for data transmission by CMs to the HE to improve the delay in access networks. A multi-request can accommodate requests for a number

of connections sustained by a CM, in contrast to a conventional request, which can only carry a request for a single connection.

Analysis and simulation results give supporting evidence by comparing the use of multi-requests with the use of conventional requests.

The use of multi-requests is largely independent of other system operations, notably which contention resolution protocol is used. DOCSIS supports a variety of such protocols. In addition, DOCSIS also supports a polling method, whereby connections are asked explicitly whether they have any data to transmit. Their responses are transmitted collision-free. Multi-requests can also be used in this case, although actual merging may be relatively less frequent because of the absence of retransmissions.

Implementing multi-requests as described in this chapter requires the DOCSIS standard to be extended. However, by properly managing connection IDs, multi-requests could be implemented in the current draft, although strict compliance is not reached.

In a practical context, the number of connections per CM may differ among CMs, and even vary in time for individual CMs. Using multi-requests of different sizes may be an option to deal with this additional variability. This is a topic for future research.

4

Fair Resource Sharing

For providing a video-on-demand (VOD) service via an HFC network, this network should support real-time delivery of video data from the head end (HE) to the cable modems (CMs) to allow uninterrupted viewing by the clients. Simultaneously sustaining multiple, heterogeneous video streams, that is, each with its own bit-rate requirement, requires that the downstream transmission paths are carefully managed and that a scheduling algorithm ensures on-time delivery of this video data.

In this chapter, we present and analyze a scheduling algorithm that sustains a number of heterogeneous streams in a dynamic environment where streams may depart and new streams be admitted. This work is based on Pronk & Korst [2001 & 2007].

4.1 Introduction

For efficient storage and transmission of digital video data, this data is typically compressed using the MPEG (Motion Pictures Expert Group) compression standard [LeGall, 1991; Haskel, Puri, and Netravali, 1997]. Consumption by an MPEG decoder of a stream of compressed video data is characterized by a variable bit rate (VBR), resulting from the variable sizes of the successive frames. These rates may range from less than 1 Mbit/s to a peak rate that is well over 10 Mbit/s.

As sustaining this variability during the retrieval from disk and the transmission of compressed video data leads to inefficient use of bandwidth, techniques have been developed, for video servers as well as for communication networks [Korst & Pronk, 2005], to reduce this variability to such an extent that transmission of this data can be performed at a rate that does not exceed a maximum rate, which is considerably lower than the peak rate required otherwise. This is done at the expense of a relatively modest amount of additional buffering at both the server and client side and a corresponding increase in response times.

Reserving this much lower maximum rate for a stream allows significantly more streams to be sustained simultaneously than when reserving the peak rate and is much simpler to implement than reservation strategies that follow the bit-rate variability. The latter in particular suffer from the problem of multiplexing streams.

In a packet-based communication network, such as an HFC network, scheduling the non-preemptive transmission of packets for a number of streams over a transmission link is one of the main activities in such a network. Referring to the schematic in Figure 4.1, packets arriving for transmission are first put in a FIFO (first-in-first-out) queue associated to the stream to which they belong. A scheduler repeatedly chooses a single packet among all packets at the head of the queues for transmission. Although packets may be of variable length, we consider the case that all packets have unit length, such as MPEG packets.

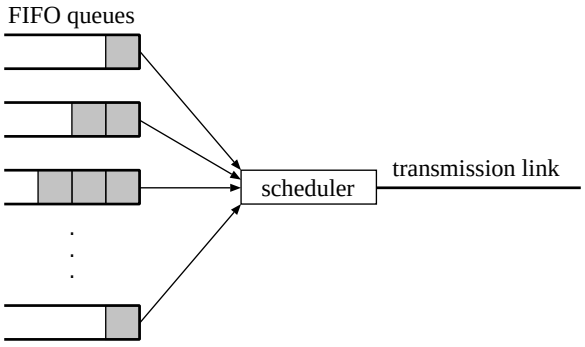


Figure 4.1. Operation of a scheduler

Besides the problem of guaranteeing the reserved bandwidth for each stream, the issue of jitter also plays a role. Jitter can loosely be defined as the difference between the ideal and the actual amount of service received thus far.

Fair queuing

In this context, *fair queuing* algorithms have for nearly two decades received considerable attention in the literature; see Demers, Keshav & Shenker [1989], Parekh & Gallager [1993], Golestani [1994], Zhang [1995], Bennett & Zhang [1996], Stoica, Abdel-Wahab, Jeffay, Baruah, Gehrke & Plaxton [1996], Suri, Vargehese & Chandranmenon [1997], Stephens, Bennett & Zhang [1999], Stepping [2001], Kunz & Stepping [2003], Zhao & Xu [2004], Valente [2004], and Karsten [2006].

A fair queuing algorithm guarantees for each backlogged stream i, that is, with packets in its queue, its so-called *fair share* of at least $R_i / \sum_{j \in B} R_j$ of the link capacity, where R_i is the share allocated to stream i and B is the set of backlogged streams, thereby also aiming to minimize for each stream its worst-case absolute jitter with respect to a so-called *virtual fluid-flow server*.

Emulating this fluid-flow server has long been considered a bottleneck for implementation of these algorithms in high-speed networks. Valente establishes a computational complexity of $\mathcal{O}(\log N)$ per packet arrival, where N is the number of streams, to emulate the server. Alternative implementations exist that only approximate the operation of the server, or make additional assumptions, leading to a lower computational complexity, but at the cost of the jitter bounds. Stephens, Bennett & Zhang, for instance, consider the special case of allowing only a limited number of different shares to obtain a complexity that is independent of the number of streams. Karsten achieves an (amortized) computational complexity of $\mathcal{O}(1)$ by rounding timestamps.

In the fair-queuing literature, the computational complexity is usually expressed per packet arrival. Although algorithms can be compared on the basis of this performance measure, considering the computational complexity in terms of the number of operations required per unit time, e.g. per slot or scheduling operation, provides a different view. In case N packets may arrive during a single slot, i.e., one for each stream, a computational complexity of $\mathcal{O}(1)$ per packet arrival, for, e.g., timestamp calculations, translates to $\mathcal{O}(N)$ operations per slot, effectively making the computational complexity per slot of such fair queuing algorithms still linear in the number of streams.

Credit-based scheduling

Credit-based algorithms form a particular class of fair queuing algorithms that use an alternative definition of jitter. This jitter is not directly related to the fluid-flow server above, but it is defined as the difference between the ideal and actual amount of service received thus far. This is explained in more detail shortly.

These credit-based algorithms explicitly maintain for each stream the jitter as a credit and use these credits for making scheduling decisions. If the absolute value of the credits remains bounded, backlogged streams are assigned the link in proportion to their allocated shares. Minimizing the worst-case absolute jitter corresponds to bounding the absolute value of the credits by an as small as possible value.

Based primarily on the use of credits rather than on the fluid-flow server, these algorithms are conceptually simpler than the above-mentioned fair-queuing algorithms.

The scheduling algorithms *weighted round robin* (WRR), by Katevenis, Sidiropoulos & Courcoubetis [1991], *deficit round robin* (DRR) by Shreedhar & Varghese [1995], and *weighted round-robin with save and borrow* (WRR-SB) by Shimonishi, Yoshida & Suzuki [1997] can be considered as forerunners of these algorithms. They operate on variable-length packets and are based on maintaining a counter for each stream to indicate the amount of service still to receive. These algorithms do not aim to minimize the worst-case absolute jitter, and it can be shown that they indeed do not attain this. Each of the algorithms has a worst-case computational complexity of $\mathcal{O}(N)$ operations per transmission, where N is the number of simultaneous streams.

Carry-over round-robin (CORR) by Saha, Mukherjee & Tripathi [1998] is a credit-based algorithm that schedules the transmission of fixed-length packets in fixed-length slots. CORR has a worst-case computational complexity of $\mathcal{O}(N)$ operations per slot. Unfortunately, one of the basic lemmas in their paper concerning the worst-case absolute jitter does not hold, as we will prove in this chapter. We will also show that, as a result of this, CORR does not minimize the worst-case absolute jitter either.

In this chapter we present a credit-based algorithm called *relaxed earliest-deadline-first* (R-EDF) for scheduling the transmission of fixed-length packets in fixed-length slots. We show that it minimizes the worst-case absolute jitter and also has a worst-case computational complexity of $\mathcal{O}(N)$ operations per slot.

R-EDF is based on an EDF algorithm that was originally introduced by Liu & Layland [1973] for preemptively scheduling a set of periodic tasks. Another important contribution of this chapter is to consider fair queuing in the context of scheduling periodic tasks.

We will also show that in a dynamic environment, where streams depart and new streams are admitted, an admission control procedure is required that, besides guaranteeing that the link is not overloaded, also ensures that the jitter bound we derive for the static case is not violated when a new stream is admitted.

The remainder of this chapter is organized as follows. We formally introduce the scheduling problem in a somewhat broader context in Section 4.2. We provide a counterexample to the above-mentioned lemma from Saha et al. in Section 4.3. In Section 4.4, we present the R-EDF algorithm and analyze its computational complexity. In Section 4.5, we prove that it indeed minimizes the worst-case absolute jitter. We discuss admission control in more detail in Section 4.6. Finally, we give some concluding remarks in Section 4.7.

4.2 Problem description

We consider the problem of fairly sharing a resource among N heterogeneous streams. Access to the resource is slotted, time-multiplexed and non-preemptive. The length of a slot is fixed and corresponds to the transmission of one fixed-length packet. We assume that a slot has unit length. Stream $i = 1, 2, \ldots, N$ is allocated a share $R_i \in \mathbb{R}^+$ of the resource capacity upon admission. It is assumed that the resource has unit capacity and that $\sum_i R_i \leq 1$. Note that this restriction necessitates the use of admission control for newly arriving streams. At any time, an admitted and not yet departed stream is either idle or busy. Upon admission, a stream is idle. A stream can become busy only at slot boundaries, and can only become idle again directly after having used a slot. This corresponds to the stream having at least one packet in its queue awaiting transmission versus having an empty queue, both at slot boundaries.

The aim is to share the resourceamong busy streams in a fair way, that is, to assign slots to busy streams in proportion to their allocated shares.

Since the resource is assigned to streams on a slot-by-slot basis, streams will generally experience some jitter. Suppose that stream i is busy during a slot. Then its ideal amount of service in this slot is R_i, whereas the amount of service it receives in this slot is either 0 or 1. If, on the other hand, stream i is idle, then its ideal amount of service is 0, and the amount of service i receives is also 0. In this way, stream i accumulates jitter over time; Starting at 0 upon admission, it either increases by an amount R_i, remains the same, or decreases by $1 - R_i$ for every slot considered. A slot may be considered more than once, for reasons explained below.

In addition to ensuring fairness, we also aim at minimizing the worst-case absolute jitter. This is achieved by maintaining a credit for each stream, which reflects the jitter accumulated thus far. A positive credit value indicates that the corresponding stream has received less than its ideal amount of service, whereas a negative credit, a debit so to speak, indicates it has received more than its ideal amount of service. For each slot considered, each busy stream i

is assigned its share of the slot *a priori* by adding R_i to its credit. Then, one of the busy streams with positive credit, if there is such a stream, is actually assigned the slot and its credit is decreased by 1. If there is no such stream, the slot is considered again in the same fashion, unless, of course, there is no busy stream. In the latter case, the slot remains unused.

Assume that consecutive slots are numbered consecutively and that stream i is continuously busy during slots $j, j+1, \ldots j+k-1$. If these slots have jointly been considered $m \geq k$ times, then its ideal amount of service thus equals mR_i.

If the absolute value of the credits remains bounded, busy streams are assigned the resource in proportion to their allocated shares. Note that this is achieved by the fact that, as long as there are busy streams, each slot is assigned to a busy stream. Minimizing the worst-case absolute jitter corresponds to bounding the absolute value of the credits by an as small as possible value.

As already mentioned, our definition of jitter differs from that based on the fluid-flow server. The major difference is in the definition of ideal amount of service. Our definition is on a slot by slot basis, whereas the definition based on the fluid-flow server is on a continuous basis, using a so-called virtual clock that mimics overall progress in this server. As a result, according to our definition, a stream may 'see' its ideal amount of service being incremented several times during a single slot, whereas in the fluid-flow server, its ideal amount of service increases continuously, proportional to the rate of the virtual clock. This rate is always at least one, relative to the actual time.

The problem can thus be stated as finding a credit-based scheduling algorithm that (*i*) guarantees for each admitted stream its fair share and (*ii*) minimizes the worst-case absolute jitter.

4.3 The carry-over round-robin algorithm

Saha, Mukherjee & Tripathi [1998] propose a solution called carry-over round-robin (CORR) to the problem described above. The link capacity is T and it is assumed that $\sum_i R_i \leq T$.

The algorithm works as follows. It operates on a cycle-by-cycle basis, that is, it repeatedly schedules a number of successive slots called a cycle. The length of each cycle is determined on-line and is bounded from above by T slots. Correspondingly, the share R_i of a stream i is expressed as slots per T slots.

Once admitted, a new stream i starts with zero credit, that is, its credit r_i is initialized to 0. At the start of each cycle, the credit r_i of each stream i is first

adjusted to $\min(n_i, r_i + R_i)$, where n_i denotes the number of packets currently in the queue associated with stream i. Then, at most T slots are assigned to a number of selected streams, with multiple slots possibly being assigned to the same stream. The number of slots actually assigned defines the length of the cycle. The streams are selected as follows. Firstly, the streams are considered in the order of non-increasing fractional part $R_i - \lfloor R_i \rfloor$ of R_i. As long as a stream i has $r_i \geq 1$ and fewer than T slots have been assigned, it is assigned a slot and its credit r_i is correspondingly lowered by 1. If all streams i with $r_i \geq 1$ have been considered in this way and still fewer than T slots have been assigned, then the set of streams is again considered, in the same order. In this second round, as long as fewer than T slots have been assigned, if a stream i has $r_i > 0$, it is assigned a slot and its credit is correspondingly lowered by 1.

In the paper considered, Lemma 2.2 states that at the start of each cycle, that is, before the credits are adjusted, it holds for each stream i that $-1 < r_i < 1$. However, this lemma is incorrect. In the next section, we give a counterexample showing that, at the start of a cycle, the credit of a stream can exceed 1. It is noted that the credit of any stream will remain less than $N-1$, since, as shown by Saha et al., $r_i > -1$ for all i and $\sum_i r_i \leq 0$. Therefore, all streams get their share in the long run.

4.3.1 Counterexample

We consider the case that $T = 1$ and an instance with 5 streams $i = 1, 2, \ldots, 5$, with shares $R_i = 0.28, 0.22, 0.18, 0.16, 0.16$, respectively. Note that the shares add up to 1, so the link capacity is fully allocated. We assume that all streams are admitted at time 0, at which cycle 1 starts, and that all streams are always and sufficiently backlogged so that their queue lengths do not influence their credits. As a result, we do not have any zero-length cycles, and each cycle contains 1 slot.

Table 4.1 illustrates the operation of CORR in terms of the credit values for each stream during the first 8 cycles, in which CORR assigns the 8 slots successively to streams $1, 2, 3, 1, 2, 3, 4, 5$. For each cycle entry in the table, the left arrows correspond to the credit adjustment step for each stream and the right arrow corresponds to the decrease by 1 of the credit of the stream that is assigned the slot.

At the start of cycle 8, the credit of stream 5 equals 1.12, which shows that the credit of a stream can indeed exceed 1 at the start of a cycle, which provides the counterexample.

For $T = 1$, alternative, more complex counterexamples can be constructed that show that credits can grow beyond 2. In addition, we note that the lemma is not only incorrect for the special case that $T = 1$. Examples with large values of T can be constructed that show credit values exceeding 21.

	cycle 1					*cycle* 2				
r_1	0.00	$\rightarrow$	0.28	$\rightarrow$	-0.72	-0.72	$\rightarrow$	-0.44		
r_2	0.00	$\rightarrow$	0.22			0.22	$\rightarrow$	0.44	$\rightarrow$	-0.56
r_3	0.00	$\rightarrow$	0.18			0.18	$\rightarrow$	0.36		
r_4	0.00	$\rightarrow$	0.16			0.16	$\rightarrow$	0.32		
r_5	0.00	$\rightarrow$	0.16			0.16	$\rightarrow$	0.32		
	cycle 3					*cycle* 4				
r_1	-0.44	$\rightarrow$	-0.16			-0.16	$\rightarrow$	0.12	$\rightarrow$	-0.88
r_2	-0.56	$\rightarrow$	-0.34			-0.34	$\rightarrow$	-0.12		
r_3	0.36	$\rightarrow$	0.54	$\rightarrow$	-0.46	-0.46	$\rightarrow$	-0.28		
r_4	0.32	$\rightarrow$	0.48			0.48	$\rightarrow$	0.64		
r_5	0.32	$\rightarrow$	0.48			0.48	$\rightarrow$	0.64		
	cycle 5					*cycle* 6				
r_1	-0.88	$\rightarrow$	-0.60			-0.60	$\rightarrow$	-0.32		
r_2	-0.12	$\rightarrow$	0.10	$\rightarrow$	-0.90	-0.90	$\rightarrow$	-0.68		
r_3	-0.28	$\rightarrow$	-0.10			-0.10	$\rightarrow$	0.08	$\rightarrow$	-0.92
r_4	0.64	$\rightarrow$	0.80			0.80	$\rightarrow$	0.96		
r_5	0.64	$\rightarrow$	0.80			0.80	$\rightarrow$	0.96		
	cycle 7					*cycle* 8				
r_1	-0.32	$\rightarrow$	-0.04			-0.04	$\rightarrow$	0.24		
r_2	-0.68	$\rightarrow$	-0.46			-0.46	$\rightarrow$	-0.24		
r_3	-0.92	$\rightarrow$	-0.74			-0.74	$\rightarrow$	-0.56		
r_4	0.96	$\rightarrow$	1.12	$\rightarrow$	0.12	0.12	$\rightarrow$	0.28		
r_5	0.96	$\rightarrow$	1.12			1.12	$\rightarrow$	1.28	$\rightarrow$	0.28

Table 4.1. Operation of CORR for the first 8 cycles. For an explanation of the table, see the running text.

Two important questions are whether there is an alternative credit-based scheduling algorithm that does satisfy the lemma and whether the lower and upper bounds on the credits are the best possible. We will show that the answer to both questions is yes.

4.4 The relaxed earliest-deadline-first algorithm

R-EDF operates on a cycle-by-cycle basis, that is, it repeatedly schedules zero or one slot, called a cycle. The length of each cycle is determined on-line.

After being admitted, a new stream i is *activated* and starts with zero credit. The issue of activation is related to the admission procedure and is further explained in Section 4.6. For the moment, it suffices to assume that some time may elapse between admission and activation. At the start of each slot, the credit r_i of each busy stream i is incremented with R_i, while the credits of idle streams retain their value. For each busy stream i, we next determine

$x_i = (1 - r_i)/R_i$. Note that a non-negative x_i denotes the time remaining after the current slot until r_i becomes 1, if r_i would increase continuously at a rate of R_i. So, if the current slot starts at time t, then $t + 1 + x_i$ can be interpreted as a, possibly passed, deadline. A stream i is *eligible* if and only if it is busy and $r_i > 0$ after the increment step. From all eligible streams, if there are any, we choose a stream i for which its x_i is minimal, assign the slot to it, decrease its credit r_i by 1, and set the cycle length to 1.

If there is no eligible stream, then there are two possibilities: either there are no busy streams at all or there are busy streams, but none of them is eligible. In the former case, the cycle length is set to 1 and the slot is left unused. In the latter case, the cycle length is set to 0 and the next cycle is started immediately. This implies that, as long as there are busy streams, each slot is assigned to a busy stream.

Lemma 4.1. *Let for each stream $i \in B$, where B is the current set of busy streams, r_i reflect its credit at the start of a cycle, that is, before the credit increment step. Then the number z of successive zero-length cycles, including the current one, is given by*

$$z = \max(0, \min_{i \in B} \lfloor -r_i/R_i \rfloor) \tag{4.1}$$

Proof. Let for busy stream i its credit r_i be non-positive at the start of a cycle before the increment by R_i. The number k_i of increments by R_i until it becomes positive is given by $k_i = \lfloor -r_i/R_i \rfloor + 1 \geq 1$, so that it is not eligible for the next $k_i - 1$ cycles, including the current one. Provided that $k = \min_{i \in B} k_i \geq 1$, the next $k - 1$ cycles, including the current one, have length zero, as none of the busy streams becomes eligible during these cycles. If $k \leq 0$, then the number of successive zero-length cycles, including the current one, equals zero, as at least one stream becomes or is eligible during the current cycle, that is, after the credit increment step. Hence, the number z of zero-length cycles satisfies Equation 4.1. □

This lemma implies that, at the start of a cycle with busy streams, the credit increment step can increment for each stream its credit by $z + 1$ times its share, thereby assuring that only zero-length cycles have been skipped and that the current cycle will have length 1.

Figure 4.2 illustrates the algorithm, assuming that the set S of streams is static and that all streams have been activated. In the code, q_i denotes the current number of packets in the queue of stream i. The q_i values are assumed to be available to the algorithm and updated automatically. Slots are numbered consecutively and s denotes the current slot number. The argmin

operator returns one index i from its domain, E in this case, for which the given expression, $(1-r_i)/R_i$ in this case, is minimal.

```
while true do
begin
    B := {i ∈ S | q_i > 0};
    if B ≠ ∅ then
    begin
        z := max(0, min_{i∈B} ⌊−r_i/R_i⌋);
        for i ∈ B do r_i := r_i + (z+1) * R_i;
        E := {i ∈ B | r_i > 0};
        i_0 := argmin_{i∈E} (1 − r_i)/R_i;
        "assign slot s to stream i_0";
        r_{i_0} := r_{i_0} − 1
    end;
    "wait until the start of the next slot";
    s := s + 1
end
```

Figure 4.2. Pseudo-code for R-EDF.

Lemma 4.2. *If the set S of N streams is static and all streams have been activated, the* R-EDF *algorithm has a worst-case computational complexity of* $\mathcal{O}(N)$ *per slot.*

Proof. The code inside the loop in Figure 4.2 assigns a single slot, possibly to no stream at all. It requires $\mathcal{O}(N)$ operations to compute B, $\mathcal{O}(|B|) = \mathcal{O}(N)$ operations to compute z, do the increment step and to compute E, and also $\mathcal{O}(|E|) = \mathcal{O}(|B|) = \mathcal{O}(N)$ to compute i_0. The remaining operations have a joint complexity of $\mathcal{O}(1)$. Hence, R-EDF has a worst-case computational complexity of $\mathcal{O}(N)$ per slot. □

The administrative task to handle newly admitted streams and stream activation and departures, which can be done at the end inside the loop, can be handled with less stringent real-time requirements, so that this can be implemented without affecting the computational complexity of R-EDF.

R-EDF is similar to the *uniform round-robin* (URR) algorithm proposed by Matsufuru and Aibara [1999]. Instead of using credits, URR is based on the operation of the virtual fluid-flow server, assuming that all streams are continuously busy. However, it operates in a more restricted context: the share for stream i is of the form w_i/R, where both w_i and R are integers, and $\sum_i w_i = R$. In URR, a schedule of length R slots is precomputed off-line and

applied repeatedly, whereby idle streams are skipped. This skipping causes URR to have a worst-case computational complexity of $\mathcal{O}(N)$ operations per slot. For a fine-grained allocation of shares, a correspondingly large value of R is required, leading to a correspondingly complex off-line computation of the schedule. Another shortcoming of URR is its inability to operate efficiently in a dynamic environment. In particular, replacing the current schedule with a new one requires the current schedule to be completed, possibly leading to a large delay before a new stream can be activated, that is, incorporated into the schedule.

Our algorithm also resembles the leap forward virtual clock (LFVC) algorithm by Suri, Varghese & Chandranmenon [1997], but with fixed-length packets. This not only establishes a strong link between fair queuing, credit-based schedulers, and scheduling periodic tasks, but also puts LFVC in this perspective.

4.5 Performance analysis of R-EDF

The remainder of this chapter primarily concerns the proof that for R-EDF, it holds for each active stream i that $-1 < r_i \leq 1 - R_i$ at the start of each cycle. This implies that $|r_i| < 1$, which means that the worst-case absolute jitter is smaller than 1 for arbitrary instances and that busy streams are assigned slots in proportion to their allocated shares.

The above, together with the following lemma implies that R-EDF is optimal among all schedulers, not only credit-based schedulers, in terms of the worst-case absolute jitter.

Lemma 4.3. *There is no scheduler that achieves a worst-case absolute jitter bound smaller than* $1-\varepsilon$ *for any* $\varepsilon > 0$ *and arbitrary instances.*

Proof. Consider an arbitrary scheduler. Let $N \in \mathbb{N}$ and consider N streams, each with a share $1/N$, that all become busy simultaneously for the first time after their activation and that remain continuously busy afterwards. Clearly, if they all become busy at slot k, then at the start of slot $k+N-1$ there will be at least one stream that has not yet been assigned a single slot by the scheduler, so that its amount of actually received service so far is 0. However, its ideal amount of received service so far is $(N-1)/N$, so that its jitter equals $1-1/N$ at that moment. For increasing N, this jitter approaches 1 arbitrarily close. □

The approach is that we generalize the results from the seminal paper by Liu & Layland [1973] who discuss the preemptive scheduling of periodic tasks on a single processor.

In their model, a periodic task i is characterized by a triple (q_i, p_i, e_i). Task i is to be executed exactly once every p_i time units. The release times for the successive executions of task i are given by $q_i + k\,p_i$, where $k \in \mathbb{N}$ and $q_i \geq 0$ is called the phase offset of task i. The release time of an execution indicates the earliest moment at which that execution may start. Once released, an execution takes e_i time units to complete. The execution must be completed before the next execution of that same periodic task is released, so that the release time of the next execution serves as the deadline of the current execution.

Liu & Layland introduced a scheduling algorithm, originally called the *deadline-driven* scheduling algorithm, but here referred to as *earliest-deadline-first* (EDF) algorithm. It works as follows. At each point in time at which an execution completes and/or a new execution is released, an execution that has the earliest deadline is scheduled immediately. In general, this will cause preemptions: if a new execution is released that has an earlier deadline than the one that is being executed at this point in time, the new execution will preempt the current one. After completion of the new execution, the current execution will be resumed immediately. It is assumed that there is no cost associated with scheduling, preemption or resumption.

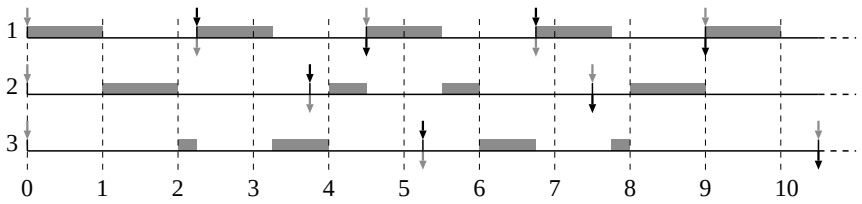


Figure 4.3. Scheduling periodic tasks. For further explanation, see the running text.

Figure 4.3 illustrates how EDF schedules periodic tasks. The figure shows three tasks called 1, 2, and 3, and three corresponding time lines on which the release times and deadlines are given for successive executions. The phase offsets of all tasks are chosen to be 0 and the execution times 1. The periods are $p_i = 2\frac{1}{4}$, $3\frac{3}{4}$, and $5\frac{1}{4}$, respectively. Successive release times are indicated by alternately grey and black arrows above each time line, and corresponding deadlines are indicated by correspondingly colored arrows below each time line. Executions are indicated by grey boxes. Multiple boxes on a time line between a release time and its deadline correspond to an execution that is preempted and resumed at least once.

At time $t = 0$, each task is immediately released (for brevity, we skip the word execution here and speak of tasks only). As task 1 has the earliest dead-

line, it is scheduled first. It is followed by task 2, after which task 3 is started. However, after execution of 25% of the latter task, it is preempted by task 1: Its deadline is before the deadline of task 3. One time unit later, task 3 is resumed again. At 75%, there is another release time, but the corresponding deadline is later, so there is no preemption. After completion of task 3, task 2 is executed for 50%, when it is preempted by task 1, et cetera.

Liu & Layland proved the following lemma.

Lemma 4.4. *Using the* EDF *algorithm, an arbitrary set of N periodic tasks* (q_i, p_i, e_i)*, for* $i = 1, 2, \ldots, N$*, can be scheduled preemptively, irrespective of their phase offsets* q_i*, such that all deadlines are met if and only if* $\sum_i e_i/p_i \leq 1$*.* □

The fact that the phase offsets do not play a role in this lemma directly leads to the following result. Suppose that the processor becomes idle at time t. This means that the next release time of each task is after t. By considering these release times as phase offsets of the corresponding tasks, time t can be considered as a new start-up time of the system, where the new phase offsets again do not play a role. Therefore, the processor time can be advanced to the earliest, new phase offset. As a result, we do not need to consider any idle time and we can concentrate on the first, so-called busy period of the processor, which starts, without loss of generality, at time 0. For convenience, we assume that the processor is idle prior to time 0.

To translate the assignment of slots to scheduling periodic tasks, we associate with stream i a periodic task with phase offset $q_i \in \mathbb{N}$ being the activation time of the stream, period $p_i = 1/R_i$, and execution time $e_i = 1$, corresponding to repeatedly assigning a single slot to stream i at a rate of $1/p_i = R_i$. Observe that $\sum_i e_i/p_i = \sum_i R_i \leq 1$.

It is noted that, if (*i*) the credit of each stream would change as a continuous function of time, (*ii*) preemptions were allowed and (*iii*) streams would never be idle or depart, R-EDF would behave exactly the same as EDF, not considering any non-determinism in breaking ties. The moment at which the credit of a task becomes 0 indicates the release time of this task, and the moment it becomes 1 indicates the corresponding deadline. See Figure 4.4 for an illustration of a credit function. For the sake of simplicity, the example does not contain any preemptions.

However, the three conditions above do not hold: streams can be idle, preemptions are not allowed, streams tend to depart after a finite time, and new streams can be admitted after a departure. Therefore, we generalize the model of periodic tasks in such a way that EDF behaves exactly the same as R-EDF,

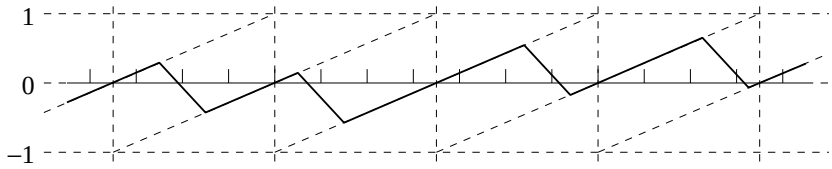


Figure 4.4. Credit function $c_i(t)$ for a task i with period p_i. The intervals where the credit function is decreasing correspond to successive executions of the task, assuming no preemptions.

again not considering non-determinism in breaking ties. We initially ignore that streams may depart after a finite time. We will consider this separately in the next section.

To model idle streams, we introduce the notion of *idling tasks*. We do this by splitting the release times and deadlines: the release time of the execution of a task is at or after the deadline of the previous execution of this task. For the first execution, the release time is at or after the phase offset. The deadline of an execution of a task with period p remains p time units after the corresponding release time. Thus, when a release time is increased, so is its corresponding deadline. This introduces so-called idle intervals for a task. The first idle interval, which may have length 0, starts right after the phase offset and all other idle intervals are between a deadline and the next release time.

Note that the lengths of idle intervals are not predetermined for any idling task, as they are based on when the corresponding stream becomes busy (again). How exactly the next release time is determined when an idle stream becomes busy is of no concern at this moment: for the idling tasks, the release times are assumed to be determined on-line. It suffices that the release time t of the execution of a task is known to EDF at time t.

An idling task can thus also be characterized as a tuple (q_i, p_i, e_i), with q_i, p_i, and e_i as before. Note that, although the phase offsets are redundant for idling tasks, they do play a role for streams: they serve to indicate the moments of activation.

Figure 4.5 illustrates how EDF schedules idling tasks. The same parameter values as in Figure 4.3 have been used for the tasks, but idle intervals have been inserted. For the sake of simplicity, the lengths of idle intervals were chosen as multiples of $\frac{1}{4}$.

At time 0, tasks 1, 2, and 3 start with an idle interval of length 0, $\frac{1}{2}$, and $\frac{3}{4}$, respectively. Until time 4, the schedule is identical to the one with

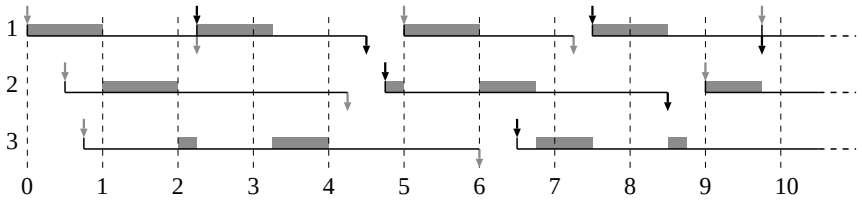


Figure 4.5. Scheduling idling tasks. For further explanation, see the running text.

periodic tasks. Then, the processor becomes idle, since the next release time of task 2 is at a future time, that is, at time $4\frac{3}{4}$. At this time, task 2 is scheduled immediately, but it is preempted by task 1 at time 5, since task 1 has an earlier deadline than the current deadline for task 2 at time $8\frac{1}{2}$. At time 6, task 2 is resumed again and it runs to completion since the deadline corresponding to the next release time at time $6\frac{1}{2}$ of task 3 is after time $8\frac{1}{2}$, et cetera.

Lemma 4.5. *Using the* EDF *algorithm, an arbitrary set of N idling tasks* (q_i, p_i, e_i)*, for* $i = 1, 2, \ldots, N$*, can be scheduled preemptively, irrespective of the phase offset and lengths of the idle intervals of each task, such that all deadlines are met if and only if* $\sum_i e_i/p_i \leq 1$.

Proof. The necessity of the condition is obvious, since all idle intervals may have length 0, in which case the tasks are periodic and we can refer to Lemma 4.4 above. Its sufficiency is proved by contradiction. Assume that $\sum_i e_i/p_i \leq 1$ and that nevertheless the idling tasks cannot be scheduled. Suppose that a first deadline violation occurs at time t_2.

Note that, just before t_2, the processor started working on tasks with deadlines at t_2. Let t_1 denote the time of the last switch before t_2 from idle or working on a task with deadline after t_2, to working on a task with deadline at or before t_2.

During the interval $[t_1, t_2]$, the processor is thus continuously busy and only works on tasks that have their deadline at or before t_2. We call these tasks *critical tasks*. All these critical tasks have their release times at or after t_1, because otherwise the processor had not been idle until t_1 or it had not been working on a task with a deadline after t_2.

We next concentrate on these critical tasks only, since the other tasks do not play a role in the interval. As the processor is continuously busy during the interval $[t_1, t_2]$, the work that has to be done in this interval is apparently too much to be completed in that interval, and causes a deadline violation. Now, by moving, for all critical tasks, the first release times at or after t_1 back

in time to t_1 and subsequently deleting all later idle intervals for these tasks, the amount of work that has to be done in the interval $[t_1, t_2]$ does not decrease. Consequently, the deadline violation will still be at t_2, or possibly earlier. This contradicts the fact that, starting at t_1, the now periodic, critical tasks can be scheduled without missing deadlines, again by Lemma 4.4. Hence, the idling tasks can be scheduled. □

We mention that Liu [2000] also considers idling tasks, but states that Lemma 4.5 follows straightforwardly from Lemma 4.4. The added flexibility in release times, however, justifies a separate proof.

We next consider how to relax the model with idling tasks in such a way that preemptions are avoided. Clearly, if the execution times are 1 and all release times are at integer points in time, then preemptions do not occur: release times of new executions always coincide with the completion of an execution or occur at integer points in time when the processor was already idle. Note that it now makes sense to talk of slots in the context of tasks as well.

We relax the idling-task model by rounding each release time downwards to the nearest integer, whereas the corresponding deadlines remain unchanged. In other words, if an execution is released at time t in the model with idling tasks, it is now released at time $\lfloor t \rfloor$, called relaxed release time, whereas the corresponding deadline for this execution remains at $t + p$, where p is the period of the task. Note that the relaxed release time of the execution of a task may precede the deadline of the previous execution of the same task, but always by an amount less than one slot.

We define a *relaxed idling task* as an idling task, but with relaxed release times. We use the term *original* release time of an execution of a relaxed idling task to denote the release time of the corresponding execution of the corresponding idling task. As before, EDF must know each relaxed release time at or before the moment that it occurs. How exactly the next relaxed release time is determined when an idle stream becomes busy is again of no concern at this moment, but we will return to this issue shortly.

Figure 4.6 illustrates how EDF schedules relaxed idling tasks. The same parameters as in Figures 4.3 and 4.5 are used, and we use idle intervals corresponding to those in Figure 4.5. The schedule is non-preemptive, as expected.

Lemma 4.6. *Using the* EDF *algorithm, an arbitrary set of N relaxed idling tasks (q_i, p_i, e_i), with $e_i = 1$, for $i = 1, 2, \ldots, N$, can be scheduled non-preemptively, irrespective of the phase offset and the lengths of the idle intervals of each task, such that all deadlines are met if and only if $\sum_i e_i / p_i \leq 1$.*

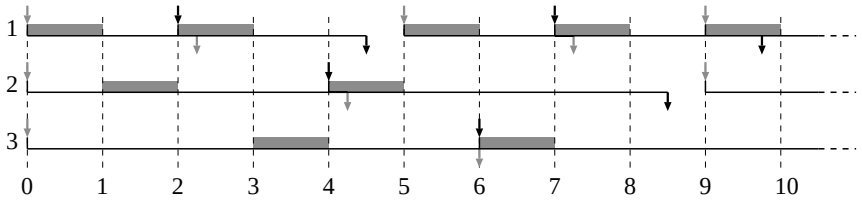


Figure 4.6. Scheduling relaxed idling tasks. For further explanation, see the running text.

Proof. The necessity of the condition is obvious, since the inequality states that the fraction of time the processor may necessarily be busy should not exceed 1.

Its sufficiency is shown as follows. First notice that EDF produces a non-preemptive schedule, as explained above. Similar to the proof of Lemma 4.5, we can construct an interval $[t_1, t_2]$ during which the processor is continuously busy and only works on tasks with relaxed release times at or after t_1 and deadlines at or before t_2, whereas a first deadline violation occurs at t_2. Note that at time $\lfloor t_2 \rfloor$, a task was scheduled that subsequently suffered a deadline violation at t_2.

Now, we move all relaxed release times, starting at t_1, back to their original values and, if necessary, increase t_1 to align with the earliest, original release time. This generally makes the schedule preemptive. Again, the amount of work that has to be done in the possibly shorter interval $[t_1, t_2]$ does not decrease. Note that moving relaxed release times that correspond to deadlines after t_2 does not decrease the amount of work either, since no work was carried out for those tasks in the original interval $[t_1, t_2]$ anyway. Thus, the deadline violation remains. Since, starting at t_1, the tasks are no longer relaxed, but just idling tasks, they can be scheduled by Lemma 4.5. This is a contradiction. Hence, the relaxed idling tasks can be scheduled. □

It is easy to see that increasing the credits at integer points in time, rather than as a continuous function, has exactly the same effect as rounding the release times down to integer points in time, so that for an original release time t, the expression $\lfloor t \rfloor \leq t < \lfloor t \rfloor + 1$ corresponds to $r_i \leq 0 \wedge r_i + R_i > 0$. In other words, the moment at which a busy stream becomes eligible coincides with the release time of the associated relaxed idling task. The term 'relaxed' in R-EDF derives from this correspondence. See Figure 4.7 for an illustration of a relaxed idling task with an original release time at t. The credit values of the associated stream are given by the step function. Each time the stream is

assigned a slot is illustrated by a dashed step up with the size of its share and a step down of size 1. The relaxed release time $\lfloor t \rfloor$ coincides with the moment that the credit becomes positive.

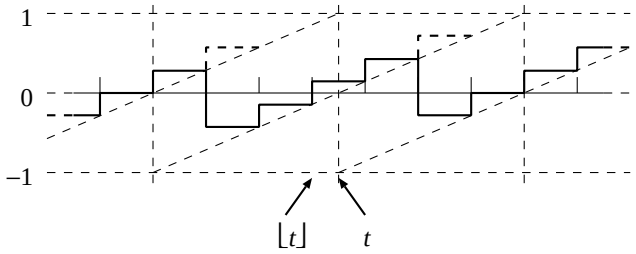


Figure 4.7. A relaxed idling task with original release time at t and relaxed release time at $\lfloor t \rfloor$, where the credit of the associated stream becomes positive.

Now recall that for EDF we assume that release times are known in time. For R-EDF to operate as EDF, a corresponding condition should hold. In particular, the credit of an idle stream should be at most 0, so that, when this stream becomes busy, say at time t, it sets a relaxed release time at or after t. This settles the issue of how the next relaxed release time is determined when an idle stream becomes busy. We can now prove the following theorem.

Theorem 4.1. *For* R-EDF, *it holds for each stream i that $-1 < r_i \leq 1 - R_i$ at the start of each cycle, provided that $\sum_i r_i \leq 1$.*

Proof. Since only eligible streams, that is, streams with positive credit after the credit increment step, are assigned slots, the credit of all streams will remain larger than -1. For the remainder of the proof, we associate a relaxed idling task to each stream as described above. For each $j \geq 0$, we define $B(j)$ and $I(j)$ as the set of busy and idle streams, respectively, at the start of slot j. These sets incorporate the newly busy and idle streams, respectively.

Let $r_i(j)$ denote the credit of stream i at the start of slot j, that is, before the increment step, and if $i \in B(j)$, let $x_i(j)$ denote the corresponding x-value, that is, $x_i(j) = (1 - r_i(j) - R_i)/R_i$. Define $\mathcal{P}(j)$ as $\forall_i [i \in B(j) \Rightarrow r_i(j) \leq 1 - R_i]$ and $\mathcal{Q}(j)$ as $\forall_i [i \in I(j) \Rightarrow r_i(j) \leq 0]$. We prove that

$$\forall_{j \geq 0} [\mathcal{P}(j) \wedge \mathcal{Q}(j)],$$

which implies that $r_i \leq 1 - R_i$ for each stream i. We prove the above expression by induction. Assume that $\mathcal{P}(j) \wedge \mathcal{Q}(j)$ holds for $0 \leq j < j_0$. This holds trivially for $j_0 = 0$. Let $j_0 \geq 0$. We prove that $\mathcal{P}(j_0) \wedge \mathcal{Q}(j_0)$ holds, starting with $\mathcal{Q}(j_0)$.

Let $i \in I(j_0)$. There are two cases to consider. If, on the one hand, i has been idle since its activation, its credit is still at its initialization value 0, so that $r_i(j_0) = 0 \leq 0$. If, on the other hand, i has already been busy, then it has been assigned a last slot, say slot $k < j_0$ after which i immediately became and remained idle until the start of slot j_0. Since $k < j_0$ and $i \in B(k)$, it holds by hypothesis that $r_i(k) \leq 1 - R_i$. But then $r_i(k+1) = r_i(k) + R_i - 1 \leq 0$. Since idle streams retain their credit, it holds that $r_i(j_0) = r_i(k+1) \leq 0$. So, in any case $r_i(j_0) \leq 0$.

We have now established that $\mathcal{Q}(j)$ holds for $0 \leq j \leq j_0$. Therefore, by analogy with the associated relaxed idling tasks, there are no deadline violations during slot j_0. Therefore, for each stream $i \in B(j_0)$, it holds that $x_i(j_0) \geq 0$, which is equivalent to $r_i(j_0) \leq 1 - R_i$. This proves that $\mathcal{P}(j_0)$ also holds. □

Corollary 4.1. R-EDF *is optimal in terms of minimizing the worst-case absolute jitter.*

Proof. This follows directly from Lemma 4.3 and Theorem 4.1. □

4.6 Admission/activation control

In a dynamic environment, streams are admitted and depart again after some time. However, the departure of a stream is not covered in the model with relaxed idling task. If streams would only be admitted and would never depart, the phase offset of a relaxed idling task would serve as the activation time of the associated stream. In the presence of stream departures in addition to admissions, the question arises of when new streams can be activated after another stream has departed.

In the extreme case that, on a fully loaded resource, a stream departs right after it has been assigned a slot and is replaced immediately by another stream with the same share, the generally negative credit of the departed stream is replaced by a zero credit for the new stream, resulting in a so-called *credit jump*. It stands to reason that credit jumps may result in deadline violations. As an example, consider three streams with shares 0.25, 0.25, and 0.5, respectively, all starting at time 0. The stream with share 0.5 is serviced in and departs after the first slot and is replaced immediately by a new stream with the same share. This new stream is serviced in and departs after the second slot and is similarly replaced. The sum of the two credit jumps equals 1 and, at the start of the fourth slot, there are two streams whose deadlines are at the end of this slot. This causes a deadline violation if neither of these streams departs. It can be shown that, even if the time between a stream departure and the activation of another stream takes one slot, deadline violations cannot generally be prevented.

Suppose that a stream departs. After this departure, the remaining shares add up to strictly less than 1. As a result, there will be zero-length cycles. Now, note that the start of the first zero-length cycle or empty slot after the departure of the stream can be considered as a new start-up of all remaining and all new streams, each with their own phase offset. The already existing streams have a non-negative phase offset corresponding to their non-positive credit at that moment. For the newly admitted streams the phase offset is determined by their activation time, which can be chosen as the current time. As from that moment, it is again guaranteed that deadline violations will not occur, provided, of course, that the shares add up to at most 1 at all times.

In other words, after a stream departure, there is a so-called *dead time*, during which no new streams can be activated without compromising isolation between the streams, that is, risking deadline violations. After this dead time, newly admitted streams can again be activated safely. The URR algorithm discussed in Section 4.4 suffers from this problem if the precomputed schedule would be interrupted. The actual credit values or similar data should be available to ensure a smooth transition to a new schedule.

4.7 Concluding remarks

In this chapter, we have presented a linear, optimal algorithm for fairly sharing a resource among streams in a dynamic environment. The optimality is stated in terms of jitter.

In a similar approach to that of Saha, Mukherjee, and Tripathi [1998], it is possible to consider scheduling cycles with a length of at most T slots. This will generally lead to worse jitter, since the exact position of the w_i slots assigned to each stream i is no longer determined. It is noted, however, that this can be partly repaired by stating this much smaller problem as follows. $T' = \sum_i w_i \leq T$ slots have to be divided over a number $M \leq T$ of streams, whereby each of these streams i should obtain $w_i > 0$ slots. This can be done using URR. Although the worst-case computational complexity will still be $\mathcal{O}(N)$ operations per slot, the average case computational complexity may become much better.

To support isolation among streams, a dead time after a departure is necessary, during which no other streams are allowed to be activated after being admitted. It stands to reason that this dead time plays a similar role in other fair queuing algorithms. This is considered a topic for further research. Whether a dead time until the next zero-length cycle or empty slot is also necessary, or a shorter dead time suffices, is a subject for further research. To this end, the paper by Stoica, Abdel-Wahab, Jeffay, Baruah, Gehrke & Plaxton [1996] may provide a useful starting point.

An interesting question that warrants further research is how credit-based scheduling algorithms relate to fair queuing algorithms as mentioned in the introduction that involve this virtual fluid-flow server. The analyses by Suri, Vargehese & Chandranmenon [1997], Saha, Mukherjee & Tripathi [1998], and Pronk & Korst [2002] provide promising results.

The commonalities between R-EDF and LFVC suggest that the given, straightforward implementation of R-EDF can be optimized, and even significantly improved by additional rounding, the latter at the cost of a small deterioration of the jitter bounds. It also stands to reason that R-EDF can be generalized to deal with variable-length packets, in the same vein as LFVC, but in the context of scheduling periodic tasks. We consider these as subjects for further research.

5

Storage and Retrieval of Variable-Bit-Rate Video Streams

Providing real-time guarantees for video streams not only requires proper bandwidth management and scheduling algorithms for the transmission of video data, of which we saw an example in the previous section, it also requires that the bandwidth of the disk or disks in a video server is treated likewise. In this and the next section, we focus on the two alternative file allocation techniques to store video data on a single disk such that subsequent retrieval of this data results in efficient use of this disk. We in particular consider the problem that a block should be readable from disk using a single disk access.

Emphasis in this section is on segmented allocation, where the disk space is partitioned into relatively large, constant-size, contiguous chunks, called allocation units. A video file uses an integer number of allocation units to store its data. Emphasis in the next section is on contiguous allocation of video files on a multi-zone disk.

5.1 Introduction

The playout of a video file from disk requires that a continuous stream of data is fed into a decoder for subsequent display on a monitor or TV. This stream is

generally consumed at a variable bit rate (VBR), depending on the sizes of the individual frames. In contrast, a video file stored on disk is typically fetched in blocks of constant size. To bridge this gap between the way in which data is retrieved from disk and the way in which it has to be fed into a decoder, or, more generally, leaves the server, a buffer is employed where the blocks from disk are temporarily stored and from where a separate process consumes the data at the appropriate rate. This process is sometimes also referred to as a stream. The same holds for writing data to disk, be it that the buffer is filled by a stream of data, for instance from a network, and emptied in a block-wise fashion.

To ensure a hiccup-free display of the video file it is required that buffer under- and overflow should be prevented at all times. This requires a disk scheduling algorithm that monitors and controls the fill level of the buffer.

The rate at which data can be retrieved from disk is generally significantly larger than the rate required for the playout of a single video file, so that disk scheduling algorithms are typically designed to sustain multiple streams simultaneously. The major performance characteristics of a scheduling algorithm are the maximum number of streams it can sustain simultaneously, the required buffer sizes, and the start-up latency. The latter is defined as the time that elapses between the arrival of a request to start up a new stream and the moment consumption from the corresponding buffer for this stream can start.

An important observation is that the block size used for a stream depends on the scheduler employed, the server-specific settings, as well as on the bit-rate characteristics of the specific video file at hand. We return to this issue shortly.

For efficient use of the disk, it is required that a block can be fetched from disk with a single disk access, which consists of a seek operation of the disk arm to the appropriate track, a rotational latency of at most one rotation to let the data to be read move under the disk head, and a read operation. This read operation may be interrupted several times to perform a track of cylinder switch in case the block is stored on several successive tracks or cylinders, respectively. The time required for a switch is typically shorter than that of a seek operation and the maximal rotational latency. For this reason, we say that a block is stored contiguously on disk if, while reading it, only track or cylinder switches are necessary.

Establishing that individual blocks of a video file are stored contiguously on disk, to allow retrieval of any of these blocks to be performed with a single disk access, poses additional constraints on the way in which the video file is written to disk. Two extreme approaches are that (*i*) individual blocks are stored contiguously, while successive blocks of a file may be positioned

arbitrarily on the disk and (*ii*) entire files are stored contiguously on disk. The latter approach is suitable if the set of files on disk does not change often. If files of different sizes are repeatedly added and removed over time, then the total amount of free space available for additional files gets fragmented into many relatively small parts. These parts remain unused if they are too small to incorporate a complete file. Furthermore, recording of a video file of unknown size, for instance a live recording, may lead to problems as well. In the next chapter, we consider contiguous storage of video files on disk.

The former approach of only storing blocks contiguously on disk not only assumes that the block size used for retrieval is known beforehand, but also that the individual blocks are known. The latter is problematic if reading a file may start at an arbitrary position in the file, which we shall refer to a non-aligned access. In addition, as already observed, writing a single block of a specified size to disk with a single disk access poses constraints on the bit-rate characteristics of the stream that delivers the data, which may be impractical.

An often used alternative is to use *segmented allocation*, where the complete disk space that is used for video data is partitioned into equal-sized allocation units. A video file uses an integer number of allocation units to store its data contiguously within an allocation unit, but the successive allocation units need not be contiguous. There exist various segmented allocation strategies [Korst & Pronk, 2005] that mitigate some or all of the problems encountered above.

We concentrate on the strategy called *segmented allocation with redundancy* by Lawerman [1995]. Assume that a playback stream requires blocks of size at most b from this file and that the size u of an allocation unit, or allocation size for short, satisfies $u \geq 2b$. Starting at the beginning of the file, the first allocation unit is completely filled with data from this file. In each subsequent allocation unit, the first b bits are obtained by duplicating the last b bits of the previous allocation unit. The remainder of this allocation unit is filled with subsequent data from the file. This proceeds until the file has been completely stored and part of the last allocation unit may be left unused.

This allocation strategy leads to adding redundancy in the storage of the file on disk, which we call *overlap*. Figure 5.1 gives an example, where the block of size b at the end of each allocation unit is also written at the beginning of the subsequent allocation unit.

It is easily seen that any block of size at most b from this file can be retrieved from disk with only one access: if the starting position of the block is at a distance less than b from the end of an allocation unit, then it can be retrieved in a single access from the next allocation unit. The latter is guaranteed because $u \geq 2b$.

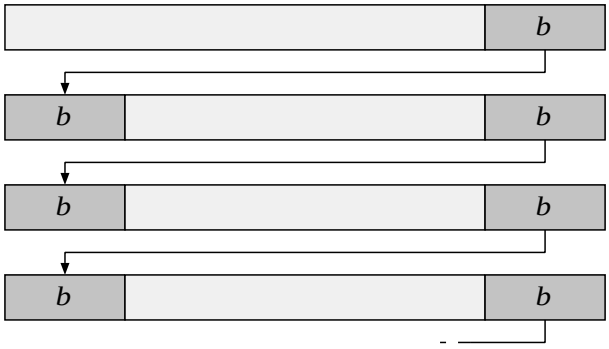


Figure 5.1. The first four allocation units of a file. The last b bits at the end of each allocation unit are duplicated at the beginning of the subsequent one.

When the size of allocation units are chosen small with respect to the sizes of the video files stored on disk and large, relative to the overlap size b, the amount of disk waste, as a result of this duplication as well as of unused space at the end of the last allocation unit, is only marginal.

The problem we consider in this chapter is how to implement this file allocation strategy. In particular, we discuss how to adapt the well-known *triple buffering* disk scheduling algorithm [Biersack, Thiesse & Bernhardt, 1996; Korst & Pronk, 2005] such that it can handle both record and playback streams.

The remainder of this chapter is organized as follows. We discuss related work in Section 5.2. In Section 5.3 we present a model for both video servers and streams. Then, in Section 5.4 we discuss the triple buffering algorithm for playback streams, and serves as a stepping stone towards Section 5.5, where we deal with the inclusion of record streams into the algorithm. We end with some concluding remarks in Section 5.6.

5.2 Related work

In this chapter, we mainly concentrate on the allocation of a single file on a single disk, and we ignore any relation between the positions on disk of the allocation units belonging to one file or belonging to different files. This leaves ample freedom for additional optimizations besides accessing a block with a single disk access. One of them is discussed in the next chapter, where the multi-zone character of a disk is exploited.

Where additional constraints apply, the allocation of files on disk may be further tuned towards the application. Vin & Rangan [1993] consider con-

strained storage of high-bit-rate video files. To realize efficient retrieval of data from one file, successive blocks are stored sufficiently close together, using the notion of scattering. By making use of interleaving, the gaps between these blocks are filled by other files. For retrieval, they consider a round-robin-like algorithm and discuss several variations to provide deterministic or statistical guarantees.

Using sweeps instead of individual disk accesses to lower the access time can be taken a step further by constraining the allocation of files such that a sweep is constrained to a particular region of the disk space. See, for instance, Chang & Garcia-Molina [1996] and Ghandeharizadeh, Kim & Shahabi [1995]. This can be exploited in the case that streams remain properly synchronized in their consumption behavior.

Both Özden, Rastogi & Silberschatz [1996] and Srivastava, Kumar & Singru [1997] consider interleaving constant-size blocks of a single file, assuming that the disk only has to serve streams accessing the same file. Multiple streams are realized in this way, where the consumption by each pair of streams is separated by a constant time offset. In addition, multiple blocks can be read using one disk access. Tsao, Huang, Lin, Liou & Huang [1997] apply the same principle, but using optical disks.

For the storage of audio on CD-ROM, we refer to Yu, Sun, Bitton, Yang, Bruno & Tullis [1989], Wells, Yang & Yu [1991], Korst & Pronk [1996], and Subrahmanian [1998]. In these papers, the issue is how to interleave a number of audio files.

For storage of files on multiple disks, additional criteria play a role, most notably the problem of how to distribute the files over the disks such that the load balance among the disks is optimized. We refer to Korst & Pronk [2005] and references therein for an in-depth treatment of this and related problems.

A good choice for the allocation size obviously also depends on the average size of the files that are expected to be present on the disk. If the disk only stores relatively large video files, then the allocation size can be chosen quite large. If it only contains small files, then the allocation size will have to be chosen correspondingly. When the disk contains both large and small files, it may be a good idea to partition the disk, where for each part a different allocation size is chosen. A disadvantage is that, in that case, a fixed fraction of the disk is reserved for each type of file. Alternatively, one can use a hierarchical organization of allocation units. A large allocation unit can be partitioned into smaller allocation units whenever required for the storage of small files. This is done in the Buddy system, as discussed by Burton [1976], Koch [1987], and Knuth [1969].

5.3 Modeling the server

Figure 5.2 illustrates the basic architecture of a video server and displays its main components. These are a magnetic disk or disk drive, an internal bus, FIFO (first-in-first-out) buffers, which are typically implemented using solid-state memory, a disk scheduling algorithm, and an interface to the outside world, consisting of streams and requests.

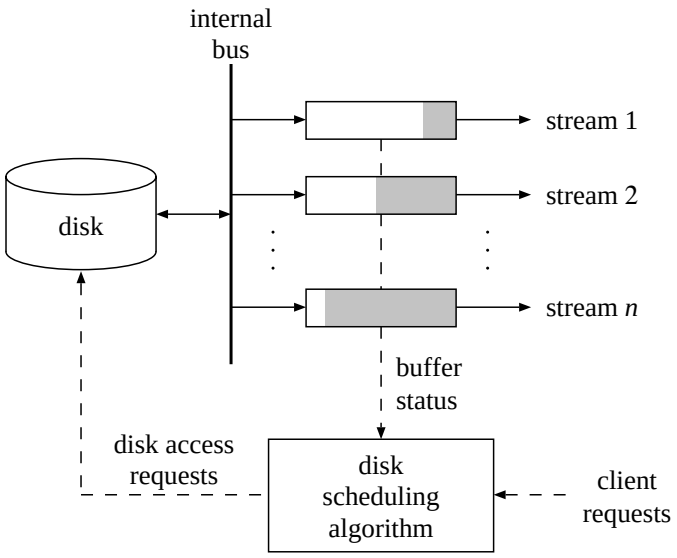


Figure 5.2. Basic architecture of a video server.

In the figure, the data paths are drawn by solid lines. They are used to transfer data between the disk and the buffers via the internal bus and between the buffers and the outside world. The dashed lines indicate the control paths and are used for control functions such as handling client requests and controlling the disk.

As the disk in a video server is shared by multiple streams simultaneously, blocks of data, or blocks for short, are repeatedly fetched from disk and transferred to the appropriate buffers. These data transfers should be scheduled in such a way that the buffers do not underflow or overflow. In this way, it is guaranteed that each stream can consume the data from its buffer without undue interruptions.

Providing real-time guarantees for the delivery of video data presupposes requirements for the various components of the video server. We concentrate on providing deterministic guarantees as opposed to only statistical guarantees. To provide deterministic guarantees, it is necessary to make worst-case

assumptions. We next discuss each of the components in more detail and state the corresponding assumptions.

The disk scheduling algorithm

The disk scheduling algorithm operates on a cycle-by-cycle basis. In each cycle, the algorithm serves a number of streams by fetching one or more blocks for these streams. The length of each cycle is bounded from above by a constant, called the period length. This period length may be as short as a worst-case time to perform a single disk access, but may also be an upper bound on the time required by the disk to perform multiple disk accesses in a single batch, where the number of accesses is bounded from above.

Definition 5.1. (Safeness) A disk scheduling algorithm is called *safe* if it prevents buffer underflow and buffer overflow at all times. □

Streams

A stream, once admitted service, can be in one of two states: waiting or consuming. Initially, a stream is waiting. When sufficient data has been fetched from disk and stored in its buffer, the stream becomes consuming and can start to consume data from its buffer for an indefinite period of time. The time an admitted stream spends in the waiting state is called the start-up latency and the moment it becomes consuming is denoted by t^{start}. Although in practice the consumption of data from a buffer is in discrete units, we assume a fluid-flow model for this consumption, mainly for ease of presentation. The difference from a discrete model is small when it comes down to bit-rate and buffer requirements.

We assume that streams consume data at a variable bit rate (VBR). To keep the model relatively simple, we define the bit rate characterizing a stream as follows.

Definition 5.2. (Stream) A stream i is characterized by a (maximum) bit rate r_i. This bit rate is an upper bound on the average rate at which stream i is allowed to consume data from its buffer during any time interval of length equal to the period length of the scheduling algorithm. So, if the period length is p, stream i consumes at most an amount $p\,r_i$ of data from its buffer during any interval of length p. □

Note that the momentary consumption rate of a stream may be considerably higher than r_i, provided that this is sufficiently compensated for in the short term. It also follows from the definition that during any time interval at most p, an amount of data at most $p\,r_i$ is consumed. Although, formally, r_i is generally a function of p, we will omit the argument p.

For a prerecorded video file f, the sizes of all individual video frames are known in advance. Suppose that this file is played out at a constant frame rate φ, taking a time t_f to complete. Let $c_i(t)$ denote the instantaneous rate at time t at which an associated stream i consumes data from its buffer, for instance at a piecewise constant rate corresponding to one frame per frame time. If the period length is p, then the bit rate for this stream can be chosen equal to

$$r_i(p) = \max_{t \in [0,\, t_f - p]} \frac{1}{p} \int_t^{t+p} c_i(t)\, dt.$$

This expression is called the empirical envelope; see Knightly, Wrege, Liebeherr & Zhang [1995]. Hence, instead of choosing the bit rate equal to the peak rate, which is given by $r_i(1/\varphi)$, a usually much smaller rate can be chosen. See also Dengler, Bernhardt & Biersack [1996] for a more extensive treatment.

For a given MPEG video file, Figure 5.3 gives $r_i(p)$ as a function of the period length p expressed in number of frames. As we can see, the bit rate reduces rapidly as p increases from 1 to, say, 10 frames. The small peaks, for instance at $p = 4$ and 7, are caused by the regular structure by which the three types of frames are interleaved. The structure used is the repetition of the pattern IBBPBBPBBPBB, which explains the peaks at 4 and 7.

Beyond these 10 frames, the bit rate reduces only slowly, and a relatively large gap remains, even for a period length of 100 frames, between $r_i(p)$ and the mean bit rate, given by $r_i(t_f)$ and indicated in the figure by the horizontal line at approximately 4 Mbit/s. This gap is caused by the presence of long, complex scenes that require a high bit rate.

The disk

For an extensive treatment of disk-drive modeling, we refer to Ruemmler & Wilkes [1994] and Korst & Pronk [2005].

A disk can be used efficiently by releasing disk access requests in batches consisting of multiple requests. By appropriately reordering these individual requests, a significant reduction in the worst-case seek time for the entire batch can be obtained as compared to handling the requests one by one. This reordering is such that, as the heads move in one direction across the surface, it handles the requests in the order in which it encounters them. This operation is called a *sweep*.

The time required for a sweep depends on the number of accesses, the sizes of the individual blocks, and the time required for seeks and rotational latencies. We call the latter the *access time* and the time required for reading the blocks the *read time*.

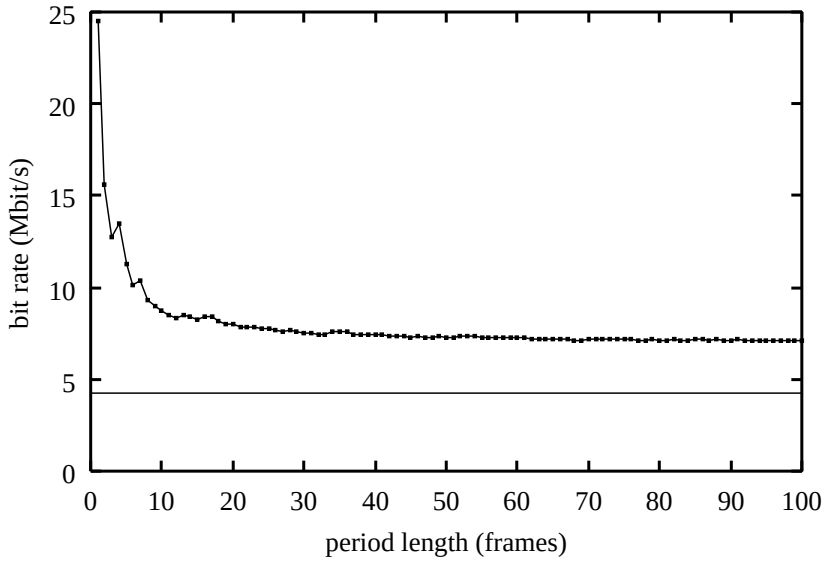


Figure 5.3. Example of $r_i(p)$ as a function of p, expressed in number of frames, for a specific MPEG movie.

Definition 5.3. (Disk) The worst-case behavior of a disk is defined by the transfer rate r and access time function a, where r is the guaranteed rate at which an arbitrary, contiguous block can be read from disk and a is a function of the number n of accesses and provides an upper bound on the total access time required to fetch n blocks from disk. We assume that these bounds are tight and can be attained simultaneously. □

Based on this definition, executing a batch of n access requests for blocks of size $b_1, b_2, \ldots, b_n$, respectively, is guaranteed to complete in a time t that is tightly bounded from above as

$$t \leq a(n) + \frac{\sum_{i=1}^{n} b_i}{r} \tag{5.1}$$

In practice, the completion time is generally smaller, as r is only a lower bound and a is only an upper bound.

We additionally make the following three assumptions. We assume that sweeps are non-preemptive, that is, once a sweep has started it is not interrupted and runs to completion. Although preemption is in principle possible, it complicates the model and the analyses. Also single disk accesses are not preempted.

We assume that the requests in a batch can be handled in arbitrary order, which is necessary to implement sweeps. In other words, if two requests have to be handled in a specific order, then we assume that it is the responsibility of the file system or the application at hand not to have these requests released in the same batch. As an example, if two disk requests concern overlapping physical locations of the disk and at least one of them is a write request, then they cannot be made available to the disk scheduling algorithm to be scheduled in the same sweep. We assume that these situations are avoided.

A block that is read from disk during a given sweep may arrive in the buffer at any point in time during the sweep. It may arrive immediately at the beginning or only when the sweep has completed. To determine whether buffer underflow can occur for a stream, we assume conservatively that a block that is read during a sweep is only available for consumption when the sweep has completed. Analogously, to determine whether buffer overflow can occur for a stream, we assume conservatively that a block that is read from disk during a sweep is already available for consumption at the beginning of this sweep.

Internal bus and buffers

The internal bus allows the blocks retrieved from disk to be transferred to the appropriate buffers. We abstract from any actual implementations of this bus and assume that the bus bandwidth is sufficient under all conditions and that the transfer delay is negligible.

For each stream, a buffer is used to decouple the transfer of data from disk and its subsequent consumption. The size of a buffer is generally expressed in the number of maximum-size blocks it can contain and depends on the disk scheduling algorithm used in the video server. The maximum block size, in turn, is determined by the requirements of the stream, such as the bit rate, as well as by server-specific settings. We assume that blocks can be of any size, rather than being restricted to discrete units such as bits or sectors. This is done mainly for ease of presentation.

5.4 Triple buffering algorithm

We assume that we are given n VBR streams, numbered $1, 2, \ldots, n,$, whereby for each stream i its bit rate is denoted by r_i. We furthermore have a disk that is characterized by a transfer rate r and access time function a. A period length p is given, which dimensions the server, in particular the buffers. To each stream i, a buffer is associated that can hold three blocks of size $b_i = p\,r_i$, which is an upper bound on the amount of data stream i consumes during any interval of length at most p.

The triple buffering algorithm (TB) operates on a cycle-by-cycle basis. In each cycle, it serves a number of streams by fetching one block for these streams. Stream i is served during a cycle if and only if it has sufficient room in its buffer at the start of this cycle to store a block of size b_i, and, if served, one block of this size is fetched for this stream during this cycle. Upon completion of a cycle, the next cycle is started immediately. It is assumed that the processing time required to determine which streams should be served during a cycle is negligible. In case there are no streams to be served during a cycle, there is an idle cycle of positive length at most p. Idle cycles serve to wait until there are streams with sufficient room in their buffer. A stream i becomes consuming, that is, can start consuming data from its buffer, at the end of the cycle in which the first block for this stream has been fetched and placed in its buffer. This time is denoted by t_i^{start}.

Theorem 5.1. *Given a disk with transfer rate r and access time function a,* TB *safely sustains a set of n* VBR *streams with bit rates $r_1, r_2, \ldots, r_n$, where $\sum_{i=1}^{n} r_i < r$, if and only if the period length p satisfies*

$$p \geq a(n)\,\frac{r}{r - \sum_{i=1}^{n} r_i}. \tag{5.2}$$

Proof. The necessity of Equation 5.2 is proved by contradiction as follows. Assume that the equation does not hold. We next prove that buffer underflow may occur for some stream. The invalidity of Equation 5.2 implies that

$$p < a(n) + \frac{\sum_{i=1}^{n} p\, r_i}{r} = a(n) + \frac{\sum_{i=1}^{n} b_i}{r}. \tag{5.3}$$

Note that the right-hand side of Equation 5.3 gives the length of a worst-case cycle. As this length is larger than p, a stream i may consume more than one block during such a worst-case cycle. Now, if the cycle starting at t_i^{start} is of worst-case duration, which may indeed happen, then buffer underflow for stream i may occur during this cycle: at the start of the cycle, stream i has one block in its buffer. Since the next block may arrive in its buffer at the end of the cycle, stream i may already have suffered from buffer underflow before this block arrives.

We next demonstrate the sufficiency of Equation 5.2. Buffer overflow will never occur, because a block for stream i will be fetched during a cycle only if there is already room for it in its buffer at the start of this cycle. What thus remains is to prove that buffer underflow never occurs either. First note that Equation 5.2 implies that the length of a cycle is bounded from above by p, which also holds for idle cycles. As a result, each stream i will consume at most one block during any cycle.

Let i be an arbitrary stream. Assume that cycles are successively numbered and that t_i^{start} coincides with the start of cycle j. Let $f_i(k)$ be defined as the buffer filling of stream i at the start of cycle k, for $k \geq j$. We next prove by induction on k that $f_i(k) \geq b_i$ for all $k \geq j$. For $k = j$, this clearly holds, because during cycle $j-1$ the first block (of size b_i) for stream i has been fetched, and stream i has not yet consumed any data from its buffer until the start of cycle j. Now assume that $k > j$ and that $f_i(l) \geq b_i$ for all l with $j \leq l < k$. We must prove that $f_i(k) \geq b_i$. We consider two cases: either $f_i(k-1) > 2b_i$ or $f_i(k-1) \leq 2b_i$. In case $f_i(k-1) > 2b_i$, no block for stream i has been fetched during cycle $k-1$. However, since during cycle $k-1$, stream i consumes at most one block from its buffer, it holds that $f_i(k) \geq f_i(k-1) - b_i > 2b_i - b_i = b_i$. In case $f_i(k-1) \leq 2b_i$, a block has been fetched for stream i during cycle $k-1$. As the induction hypothesis implies that $f_i(k-1) \geq b_i$ and stream i consumes at most one block from its buffer, it holds that $f_i(k) \geq f_i(k-1) - b_i + b_i = f_i(k-1) \geq b_i$. So, in either case, it holds that $f_i(k) \geq b_i$.

Since at the start of each cycle $k \geq j$, the buffer for stream i contains sufficient data to survive this cycle, buffer underflow will never occur. This completes the proof. □

The worst-case start-up latency for a new stream is $2p$. This occurs when a request for a new video stream arrives at the server just after the start of a cycle, so that the first block for this stream is fetched in the next cycle, at the end of which it becomes consuming. This may take an amount $2p$ of time. The average-case start-up latency of one and a half cycle depends on the lengths of the successive cycles, which is determined dynamically.

To provide some insight into the dynamic operation of TB, we assume that the disk generally performs better than is worst-case accounted for and that the streams occasionally require less data than is accounted for. The latter is typical for VBR video data, but a stream may also consume less data because the video application is in slow-motion or pause mode.

As a cycle is generally shorter than is worst-case accounted for, the next cycle will start early. As a result of this, generally less data needs to be fetched for each of the streams, leading to yet a shorter next cycle, et cetera. This may cause a significant reduction in the average cycle length and can be so extreme that the term *cycle-length implosion* is appropriate to describe the effect. Korst & Pronk [2005] analyze this effect and show in a practical setting, that a significant amount of time is spent on idle cycles, and that only very few, that is, mostly only one or two, streams are served per cycle for the remaining cycles. As a result, the average start-up latency is generally significantly smaller than $2p$.

Having a buffer size of three blocks is clearly sufficient, as shown above. Korst & Pronk [2005] discuss under what circumstances this size is also necessary and when a smaller buffer may suffice.

5.5 Dealing with record streams

The server depicted in Figure 5.2 can easily be generalized to incorporate record streams. Instead of consuming data from its buffer, a record stream writes data into its buffer. We assume that a record stream i is also characterized by its bit rate r_i, so that, during any interval of length p, an amount of data at most $b_i = p\,r_i$ is written into its buffer. And instead of fetching blocks from disk, blocks are written to disk. In the figure, changing the direction of the two solid arrows associated to a buffer illustrates this generalization.

We next analyze in detail how writing a file to disk with overlap should be implemented, using a generalization of the triple buffering algorithm. For reference, we call this generalization TBR, which stands for TB with record streams.

We assume an allocation unit of size u and an overlap of size $b > 0$, with $u \geq 2\,b$. With a period length p, this overlap then allows the file to be retrieved from disk non-aligned at any rate at most b/p.

Although a record stream i can be thought of as requesting the bit rate r_i that characterizes itself, the server should take into account that per unit of time it should write more data to disk than the stream will supply. This naturally leads to allocating to this record stream a larger bit rate than r_i to ensure safeness. We make this explicit by distinguishing *requested bit rates* and *allocated bit rates*, based on the parameters u, b, and p.

Definition 5.4. (Allocated bit rate) Given a requested bit rate r_i for a record stream i, we define the corresponding allocated bit rate r_i' as

$$r_i' = \frac{u}{\left\lfloor \frac{u-b}{p\,r_i} \right\rfloor p} \tag{5.4}$$

□

It is easily shown that $r_i' > r_i$, so that, if a stream i is characterized by its requested bit rate r_i, it is also characterized by its allocated bit rate r_i'. We next give the rationale behind Equation 5.4. The rate r_i' that has to be allocated to stream i should satisfy

$$r_i' \geq \frac{u}{u-b}\,r_i \tag{5.5}$$

in order to prevent buffer overflow in the long run. The reason for this is that for each amount $u - b$ of data written to the buffer, an amount u should be written to disk; see Figure 5.1.

The associated block size $b_i' = p\,r_i'$ should divide u to ensure that an allocation unit is completely filled with an integer number of blocks. Hence, there should be an integer l_i' such that $u = l_i'\,p\,r_i'$. Combining this with Equation 5.5 yields that

$$l_i' \leq \frac{u-b}{p\,r_i}.$$

Choosing l_i' as large as possible results in a minimal block size and thus a minimal value of r_i'. We choose

$$l_i' = \left\lfloor \frac{u-b}{p\,r_i} \right\rfloor, \tag{5.6}$$

which, combined with $u = l_i'\,p\,r_i'$, yields Equation 5.4.

TBR thus allocates to each record stream a bit rate that is larger than requested, resulting in correspondingly larger blocks. Analogous to TB, TBR only writes a block of size b_i' for a record stream i to disk during a cycle if it is already available in the corresponding buffer at the start of this cycle.

We next describe how provisions for writing parts of the data twice to disk are incorporated. We assume that the buffer for record stream i has size $3b_i' + b$, that is, corresponding to the original buffer size of $3b_i'$ used for a stream that is allocated a bit rate r_i', augmented with the overlap size b.

Assume that the buffer of stream i is circular, as illustrated in Figure 5.4. The inward-pointing arrow indicates the position of the write pointer, which is the position where the stream writes its next data to the buffer. The outward-pointing arrow indicates the position of the read pointer, which is the position where the next data is read from the buffer to be written to disk. The write pointer only proceeds in a clockwise direction. The read pointer also proceeds in a clockwise direction, but is occasionally placed back in counter-clockwise direction. The dark-shaded area indicates the data already written to the buffer, but not yet read from the buffer. The light-shaded area indicates either void data or data that has already been read from the buffer, but not yet been overwritten by the stream. For clarity, if we talk about the amount of data in a buffer, we mean the amount between the read and the write pointer, indicated by the dark-shaded area. In the case that the write pointer would pass the read pointer, we assume that it does not pass the read pointer, but that data is not written to the buffer and is lost instead.

The idea is that, when the last b bit positions of an allocation unit have been filled with data, the read pointer is placed back in counter-clockwise

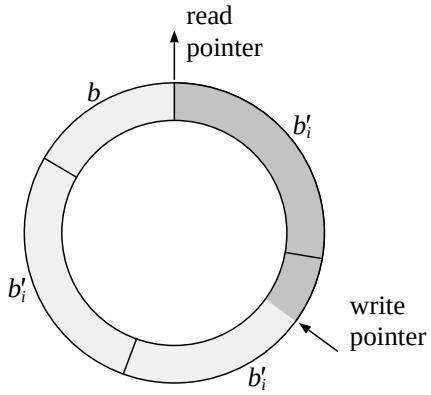


Figure 5.4. Circular buffer arrangement for a record stream i.

direction by an amount of b bits, so that writing in the next allocation unit starts by rewriting these last b bits to disk, as intended. More specifically, at the end of each cycle during which the last block of an allocation unit is written to disk, the read pointer is placed back b bits, before the next cycle is scheduled. The main concern is that the write pointer should not interfere with the read pointer, that is, the write pointer should not already have passed the point where the read pointer is repositioned. Note that the data to be written to disk again has already been written to disk earlier, so that repositioning the read pointer does not cause void data to be written to disk.

As the buffer of a record stream is initially empty, it can start writing data to its buffer at the beginning of the cycle following the one during which it has been admitted.

This completes the description of TBR and we can now state the following theorem.

Theorem 5.2. *Given is a disk with transfer rate r and access time function a, an allocation size u, overlap b and period length p. Given also are n* VBR *streams with requested bit rates $r_1, r_2, \ldots, r_n$, of which the first k are record streams and the remaining $n-k$ are playback streams. Let r'_i denote the bit rate that is allocated to record stream i, given by Equation 5.4, and assume that $\sum_{i=1}^{k} r'_i + \sum_{i=k+1}^{n} r_i < r$. Then* TBR *safely sustains these streams if and only if the period length p satisfies*

$$p \geq a(n)\,\frac{r}{r - \sum_{i=1}^{k} r'_i - \sum_{i=k+1}^{n} r_i}. \tag{5.7}$$

□

Note the correspondence of Equation 5.7 with Equation 5.2 in Theorem 5.1. As before, Equation 5.7 is equivalent to saying that the length of any cycle is bounded from above by p, so that for playback streams it can be proved, virtually identically to the proof of Theorem 5.1, that buffer under- or overflow will never occur. We can thus concentrate on the record streams and only consider the possibility of buffer overflow, as buffer underflow is trivially avoided.

The theorem will be proved in a number of steps by stating two lemmas. We first introduce some notation. Assume that the allocation units that are successively filled with data from record stream i are numbered from 1 onwards and that cycles are also successively numbered. Let the amount of data in the buffer of record stream i at the start of cycle j be denoted by $b_i'(j)$ and let the cycle during which the last block of allocation unit l is written to disk be denoted by j_l. For notational convenience, let j_0+1 denote the cycle during which record stream i can start writing data to its buffer.

Lemma 5.1. *For record stream i the following results hold.*

(a) For each $l \geq 1$, it holds that $b_i'(j_l) \leq 2b_i'$.

(b) The write pointer never interferes with the read pointer.

(c) For each $j = j_0+1, j_0+2, \ldots, j_1$, we have $b_i'(j) \leq 2b_i'$.

(d) For each $l \geq 1$ it holds that $b_i'(j_l+1) \leq 2b_i' + b$.

Proof. The proof mainly concerns part (*a*), while (*b*)–(*d*) are proved on the fly. For $l = 1$, the inequality holds, as can be seen as follows. During cycles $j_0+1, j_0+2, \ldots, j_1$, allocation unit 1 is being filled with data. Note that these may include cycles during which no data is being written to disk for record stream i. During these first $j_1 - j_0$ cycles, record stream i behaves as an 'ordinary' stream, similar to a playback stream, characterized by r_i', except for the repositioning of the read pointer at the end of the last cycle. Following a reasoning similar to that given in the proof of Theorem 5.1 for playback streams, we establish that $b_i'(j) \leq 2b_i'$ for each $j = j_0+1, j_0+2, \ldots, j_1$, which proves (*c*). In particular, we have that $b_i'(j_1) \leq 2b_i'$.

We now proceed by induction. Suppose that $b_i'(j_{l-1}) \leq 2b_i'$ for some $l > 1$. We next prove that $b_i'(j_l) \leq 2b_i'$. Firstly, during cycle j_{l-1}, an amount of at most b_i is written to the buffer, an amount b_i' of data is read from the buffer, and the amount of data in the buffer is increased instantly by an amount b at the end of this cycle because of the repositioning of the read pointer. As $b_i \leq b_i'$, it turns out that the write pointer does not interfere with the read pointer: At the moment that the read pointer is about to be repositioned, the amount of data in the buffer is at most $b_i'(j_{l-1}) + b_i - b_i' \leq 2b_i'$, so that the read pointer can

indeed be placed back b bits without interference. As $l-1 \geq 1$ and arbitrary, this proves (*b*). Furthermore, it holds that

$$\begin{aligned} b'_i(j_{l-1}+1) &\leq b'_i(j_{l-1}) + b_i - b'_i + b \\ &\leq b'_i + b_i + b \\ &\leq 2\,b'_i + b, \end{aligned}$$

which proves (*d*).

Returning to (*a*), we next consider two cases: either (1) during all remaining cycles $j_{l-1}+1, j_{l-1}+2, \ldots, j_l - 1$ a block for record stream i is written to disk, or (2) there is a cycle, say k, with $j_{l-1} < k < j_l$, during which no block is written to disk for record stream i.

In case (1), we have that $j_l - j_{l-1} = l'_i$, where l'_i is given by Equation 5.6. During the remaining $l'_i - 1$ cycles, an amount of data at most $(l'_i - 1)\,b_i$ is written to the buffer, whereas exactly an amount $u - b'_i$ is read from the buffer. Thus,

$$\begin{aligned} b'_i(j_l) &\leq b'_i(j_{l-1}+1) + (l'_i - 1)\,b_i - (u - b'_i) \\ &\leq b'_i + b_i + b + (l'_i - 1)\,b_i - (u - b'_i) \\ &\leq 2\,b'_i + l'_i\,b_i + b - u \\ &= 2\,b'_i + \left\lfloor \frac{u-b}{b_i} \right\rfloor b_i + b - u \\ &\leq 2\,b'_i. \end{aligned}$$

Alternatively, in case (2), it holds that $b'_i(k) < b'_i$, as no data is written to disk for record stream i during cycle k. Again, following a reasoning similar to that given in the proof of Theorem 5.1, we establish that $b'_i(j) \leq 2\,b'_i$ for each $j = k+1, k+2, \ldots, j_l$, as record stream i again behaves as an ordinary stream during these cycles. So in particular we have that $b_i(j_l) \leq 2\,b'_i$. This also completes the proof of part (*a*). □

Lemma 5.2. *For each $j \geq j_0 + 1$ it holds that $b'_i(j) \leq 2\,b'_i + b$.*

Proof. For $j = j_0+1, j_0+2, \ldots, j_1$, the above follows from Lemma 5.1(*c*). We next proceed by proving the above result for each $l \geq 1$ and for each $j = j_l + 1, j_l + 2, \ldots, j_{l+1}$ by induction on j.

Let $l \geq 1$. Lemma 5.1(*d*) states that $b'_i(j_l+1) \leq 2\,b'_i + b$. Now suppose that $b'_i(j-1) \leq 2\,b'_i + b$ for some j with $j_l + 1 < j \leq j_{l+1}$. We have to prove that $b'_i(j) \leq 2\,b'_i + b$. First note that the read pointer is not repositioned at the end of cycle $j-1$. During cycle $j-1$, an amount of data at most b_i is written to the buffer. In case $b'_i(j-1) < b'_i$, we certainly have that $b'_i(j) < b'_i + b_i \leq 2\,b'_i + b$. In case $b'_i(j-1) \geq b'_i$, an amount $b'_i \geq b_i$ of data is written to disk as well, so

that $b'_i(j) \leq b'_i(j-1) + b_i - b'_i \leq b'_i(j-1) \leq 2b'_i + b$. So, in any case, it holds that $b'_i(j) \leq 2b'_i + b$. This completes the proof. □

We are now ready to prove Theorem 5.2.

Proof of Theorem 5.2 Lemma 5.2 ensures that at the start of each cycle there is sufficient room in the buffer to store an additional amount $b_i \leq b'_i$ of data, whereas Lemma 5.1(b) ensures that repositioning the read pointer does not cause any problems, so that buffer overflow is guaranteed never to occur. □

When considering in more detail the structure of the proofs above, note that, although Lemma 5.1(a) is not used directly in either the proof of Theorem 5.2 or that of Lemma 5.2, it is used to prove Lemma 5.1(d), which in turn is used in the proof of Lemma 5.2.

5.6 Concluding remarks

We end this chapter by providing some remarks on possible refinements and optimizations that are possible in this context.

Although sufficient, a buffer size of $3b'_i + b$ is not necessary. By a more careful analysis of the required buffer size, it can be shown that a smaller buffer of size $2b_i + \max(b'_i, b_i + b)$ is enough to prevent overflow for a record stream i.

For each record stream, the same overlap b was used, but it goes without saying that different values of b can be chosen for the different streams, corresponding to the rates at which the recorded files are typically played back.

We have illustrated how TB can be generalized to deal with record streams as well. Alternative disk scheduling algorithms exist that can similarly be generalized. Korst & Pronk [2005] discuss two alternative algorithms, namely the variable-block double buffering (VDB) and the dual sweep algorithm (DS).

VDB bears great resemblance to TB, the main differences being that for each stream i, blocks of variable size are fetched from disk and a buffer for two blocks suffices. Generalizing VDB to incorporate record streams complicates the writing process a little further. As the blocks that are successively written to disk are generally of variable size, it cannot be guaranteed that the allocation unit is completely filled with an integer number of such variable-size blocks. One way to fix this is the following. If the next block to be written to disk does not fit in the current allocation unit, the read pointer is placed back and writing the next allocation unit starts by rewriting the last b bits written in the current allocation unit instead of writing the next block. The remaining, unused part of the current allocation unit is at most one block of size b'_i. This

increases the bit rate to be allocated, and hence the block size b_i'. In that case, Equation 5.5 should be replaced by

$$r_i' \geq \frac{u - b_i'}{u - b_i' - b} r_i, \tag{5.8}$$

where $b_i' = p r_i'$. This yields a quadratic inequality for r_i', which, together with the requirement that the denominator in Equation 5.8 is positive, yields a minimal value for r_i', provided that r_i is not too large. We do not further elaborate on this.

DS, like TB, only fetches fixed-size blocks for a stream and only requires a buffer for two blocks, be it that the sizes of the blocks are larger than for TB. We conjecture that DS allows for a generalization, similar to that of TB, to incorporate record streams.

6

Resource-Based File Allocation on a Multi-Zone Disk

In the preceding chapter, we have used a fairly simple model of a disk consisting only of a transfer rate r and an access-time function a. They are used to bound the actual transfer times of blocks and the associated overhead incurred by head movements and rotational latencies, respectively.

In practice, the rate at which data is transferred from disk depends on where this data is stored on disk. Assuming that data is stored at a constant density, the size of a track is proportional to its distance to the spindle of the disk. As the disk rotates at a constant angular velocity, the transfer rate increases as the heads move from the inner toward the outer tracks. The ideal situation of a *constant-density* disk is approximated by *multi-zone* disks, where the cylinders are grouped into a number of zones. A zone consists of a number of adjacent cylinders, each of which has a constant track size, so that the transfer rate is constant within a zone.

In the previous chapter we used the transfer rate that can be sustained in the innermost zone, to provide guarantees on the transfer time of a block, irrespective of the location of this block on disk. As such, we did not exploit the higher rates that can be achieved in the other zones, which can be a factor of two higher.

In this chapter, we present and compare three alternative approaches that alleviate this shortcoming. Two of them are based on *track pairing*, originally introduced by Birk [1995a]. Track pairing allows the average transfer rate of the disk to be used instead of the conservative disk rate. This transfer rate can be guaranteed, irrespective of which files are being requested. In the third approach files are stored on disk contiguously and in a particular order so as to minimize overall disk resource usage. This approach is useful if information is available on the popularity of individual files, something that cannot be exploited by track pairing. We will show that, under several simplifying assumptions, this contiguous storage can outperform track pairing if there is sufficient skew in popularity.

For ease of exposition, we make a number of simplifying assumptions on the disk model. We assume that it has only one head and that the number of tracks is even. We also assume that the time required to switch from one track to an adjacent track is negligible.

6.1 Track pairing

To improve the efficiency of reading data from disk, Birk [1995a] introduced a storage strategy called *track pairing*, where tracks are combined two-by-two, starting at the two extreme tracks of the disk. If there are c tracks, numbered $1, 2, \ldots, c$, starting at the outer track, then tracks 1 and c, 2 and $c-1$, $\ldots$, t and $c+1-t$, $\ldots$, are combined to form $c/2$ track pairs. For a constant-density disk, each track pair has the same size. For a multi-zone disk, where the tracks in one zone have equal size, the size of a track pair may vary somewhat across the disk, depending on the zones or zone in which the two tracks are positioned. The minimal size of any of the track pairs can then be used as a lower bound.

Track pairing improves upon an earlier method by Heltzer, Menon & Mitoma [1993] in terms of buffering requirements. The latter considers logical tracks, where each logical track consists of one track of each of the zones. The idea is to emulate a fixed track size and to read only whole logical tracks.

Returning to track pairing, blocks are stored on disk as follows. The transfer rate of the disk at a position x is denoted by $r(x)$. A block of size b is stored using one or more track pairs. In the case of one track pair, say tracks t_1 and $t_2 = c+1-t_1$, the block is subdivided into two sub-blocks 1 and 2 of size b_1 and b_2, respectively, such that

$$\frac{b_1}{b_2} = \frac{r(t_1)}{r(t_2)}. \tag{6.1}$$

Sub-block 1 is stored on track t_1 and sub-block 2 is stored on track t_2. It immediately follows that the reading times of these sub-blocks are equal, that is, $b_1/r(t_1) = b_2/r(t_2)$. Figure 6.1 illustrates this subdivision.

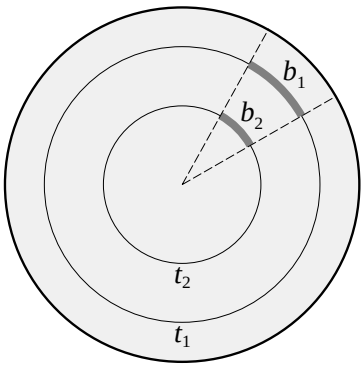


Figure 6.1. Illustration of how a block of size $b = b_1 + b_2$ is stored on track pair t_1 and t_2.

The transfer rate realized for this block is given by

$$\begin{aligned} \frac{b}{\frac{b_1}{r(t_1)} + \frac{b_2}{r(t_2)}} &= \frac{b_1\,(1 + \frac{r(t_2)}{r(t_1)})}{\frac{2b_1}{r(t_1)}} \\ &= \frac{r(t_1) + r(t_2)}{2}, \end{aligned} \tag{6.2}$$

which is the average of the two transfer rates. For a constant-density disk, this is a constant, independent of the track pair. This leads to the following definition.

Definition 6.1. For a constant-density disk, its *average transfer rate* r_{avg} is defined as

$$r_{\text{avg}} = \frac{r_{\text{min}} + r_{\text{max}}}{2},$$

where r_{min} and r_{max} denote the transfer rates at the inner and outer tracks of the disk, respectively. □

The average transfer rate of a constant-density disk that uses track pairing can be used to provide guarantees on the transfer time of data. For a multi-zone disk, the minimal value attained in Equation 6.2 as (t_1, t_2) ranges over the track pairs can be used for this purpose. Note that transfer rates can be guaranteed irrespective of the popularity distribution over the files.

Another consequence of subdividing a block as given by Equation 6.1 is that, as blocks are stored on a track pair, each of its tracks fills up at the same speed in terms of their relative filling. Storing a block on more than one track pair is thus a straightforward extension of the above.

Consequences for scheduling. By storing a block on disk in the above way, it requires two accesses instead of one to retrieve the block from disk. This increases the access-time overhead when compared to storing the block contiguously, and diminishes the advantage of reading the block at the average transfer rate.

An alternative to this approach is the following. For simplicity, we restrict ourselves to constant-block-size (CBS) scheduling algorithms like the triple buffering algorithm discussed in the previous chapter. These scheduling algorithms require that at most one block is retrieved from disk during any cycle. Consider a stream i with block size b_i, and let the blocks to be successively retrieved from disk for this stream be numbered $1, 2, 3, \ldots$. Instead of storing these blocks individually on disk as described above, successive pairs $(2i-1, 2i)$ of blocks are stored as a large block and a small block, whereby the large block contains block $2i-1$ and part of block $2i$ and the small block contains the remaining part of block $2i$. Assuming that only one track pair is required for storing the pair of blocks, the large block is stored on the outer of the two tracks and the small block on the inner track. It is then sufficient to retrieve at most one large or one small block for stream i from disk during any cycle, provided that the buffer size is sufficiently increased and reading starts with an odd block. This is because data is then only read ahead of time. In other words, in the same access that an odd block is retrieved, part of the next even block is also retrieved. The next access for stream i causes the remaining part of the even block to be retrieved. The amount of additional buffer space required is given by the maximum amount of data that is prefetched. This occurs when reading a sub-block for the stream that is closest to the outermost track of the disk.

The requirement that reading should start with an odd block has a detrimental effect on the start-up latency. If the first block requested happens to be an even block, which may happen if starting anywhere in the file is allowed, then two accesses are necessary to retrieve this block.

For reference, we call the two approaches outlined above *double-access track pairing* (DTP) and *single-access track pairing* (STP), respectively. Both approaches are illustrated in Figure 6.2.

Assuming again a constant-density disk, for DTP, the time required to read a block of size b_i for stream i is fixed and can be written as b_i/r_{avg}. For STP,

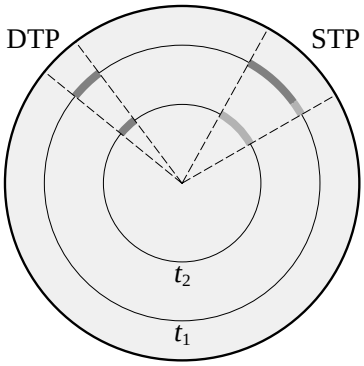


Figure 6.2. Illustration of DTP, where reading a block requires two accesses, and STP, where reading a block requires only one access, provided that, for the light-grey block, the part on track t_1 has been read during the access where the dark-grey block was read.

the time to read a large or a small block is given by the same expression. This again shows that the average transfer rate of the disk can be used instead of the transfer rate in the innermost zone. For multi-zone disks, the situation is similar.

6.2 Resource-based file allocation

The condition for safeness for the disk scheduling algorithm discussed in the previous chapter makes use of the fact that the transfer rate is guaranteed, providing an upper bound on the time to read a block from disk. These conditions can be relaxed if we take into account the location of the files on disk. In particular, consider TB, for which the safeness condition, stated also in Theorem 5.1, is given by

$$p \geq a(n)\,\frac{r}{r-\sum_{i=1}^{n} r_i}, \tag{6.3}$$

where p is the period length, a is the access-time function, r is the transfer rate, n is the number of streams, and r_i the bit rate of stream i, for $i = 1, 2, \ldots, n$. If we associate a transfer rate $\hat{r}_i$ with each stream i, indicating the minimal rate at which the file requested by stream i can be retrieved from disk, then Equation 6.3 can be relaxed to

$$p \geq a(n)\,\frac{1}{1-\sum_{i=1}^{n} r_i/\hat{r}_i}. \tag{6.4}$$

The transfer rate $\hat{r}_i$ equals $r(x_i)$, where x_i denotes the position of a block, from the file requested by stream i, that is stored closest to the inner track of the disk. As $\hat{r}_i \geq r$, and generally even $\hat{r}_i > r$, the right-hand side in this equation will generally be smaller than that in Equation 6.3, leading to a smaller minimal period length, or to possibly more simultaneous streams if the period length is not changed.

In this section, we take a closer look at the problem of how to store a number of files on disk to take maximum advantage of this idea. For reference, we call this resource-based file allocation. We do this while taking the popularity of files into account. We will see that, although it is tempting to store the popular files closer to the outermost track of the disk than the less popular ones [Ghandeharizadeh, Ierardi, Kim & Zimmermann, 1996], the situation is more complicated than this.

In the remainder of this section, we formally define the offline problem of storing a number of files contiguously on a multi-zone disk and prove that it is NP-hard. We propose a heuristic algorithm and analyze its performance by analysis as well as simulation. In Section 6.3 we consider a special case and analytically compare RFA with DTP and STP. Then, in Section 6.4, we provide simulation results pertaining to this comparison. We provide related work in Section 6.5. Finally, we end with some concluding remarks in Section 6.6.

6.2.1 Problem definition

Informally, the problem we consider is how to store m video files on disk such that disk resource usage is optimized. To this end, each file i, with $i = 1, 2, \ldots, m$, is characterized by its bit rate r_i, duration d_i, popularity γ_i, and size s_i. The bit rate is as defined in the previous chapter. The duration d_i typically expresses the linear playout time of the entire file, but could alternatively describe the average playout time of an associated stream in case, for example, jumping and replaying are performed often. The popularity γ_i is given as the fraction of the total number of stream requests per unit time that request the file, and the size s_i gives the size of the file. Note that for a VBR file it generally holds that $r_i > s_i/d_i$, although this is not used in the remainder of this chapter.

When a file i has been stored on disk, there is a position x_i at which the transfer rate of data from this file is minimal. We associate a transfer rate $\hat{r}_i = r(x_i)$ with this file. This rate corresponds to the transfer rate that is attained when reading data from this file that is stored closest to the inner track of the disk.

For a stream, the actual transfer rate that can be guaranteed may increase over time, such as when the requested file is viewed linearly and reading the

file from disk is performed from its innermost location outward. Furthermore, in such a situation, the required bit rate may likewise be decreased as the file is being viewed. We do not take these complicating issues into account, but instead allow increased flexibility in viewing a file.

When a stream request for file i is admitted, we have to allocate to this stream a bit rate of (at least) r_i, which corresponds to a fraction $r_i/\hat{r}_i$ of the time available for reading, for an average of d_i time units. The value $r_i/\hat{r}_i$ can be considered as the momentary disk load for this stream. Allocating less than $r_i/\hat{r}_i$ may eventually lead to a buffer underflow, as the stream may consume data at a rate of r_i for an indefinite amount of time. This leads to an allocation of a total of $r_i d_i/\hat{r}_i$ of disk resource to this stream. The expected amount of disk resource to be allocated to an arbitrary stream is thus given by

$$\sum_{i=1}^{m} \frac{r_i d_i}{\hat{r}_i} \gamma_i. \tag{6.5}$$

The unit of this expression is time, and reflects the total expected amount of reserved transfer time on the disk for an arbitrary stream.

The problem we consider is to find a file allocation strategy that minimizes Equation 6.5. It is easily seen that we only have to consider file allocation strategies where each file is stored contiguously on disk, as interleaving files cannot increase the individual transfer rates, and that storing the files should proceed from the outer track inward, without gaps between the files.

Equation 6.5 allows for an alternative interpretation. Consider an arbitrary moment in time, and assume that admission control ensures that the popularity of individual files is reflected in the distribution of the streams over the files.[1]

For a randomly chosen stream at that moment, the probability that it is associated with file i is proportional to $d_i \gamma_i$, and $r_i/\hat{r}_i$ gives its momentary disk load, given that it is associated with this file. Equation 6.5 thus gives the expected, momentary disk load for a randomly chosen stream at an arbitrary moment in time. As a result, minimizing the value of Equation 6.5 also implies that the expected number of simultaneous streams is maximized.

To simplify the appearance of Equation 6.5, we introduce for each file i a weight $w_i = r_i d_i \gamma_i$. We are now ready to formally state the problem.

Problem 6.1. [Resource-based file allocation problem (RFA)]. Given are a disk of size $s > 0$ with a non-increasing transfer rate function $r > 0$ on $\{1, 2, \ldots, s\}$ and m files numbered $1, 2, \ldots, m$, each file i being characterized

[1] This need not be the case if certain files are favored over others for economic or other reasons, or if files with smaller bit rates are favored over others during admission control.

by a weight $w_i > 0$ and size $s_i > 0$, with $\sum_{i=1}^{m} s_i = s$. Find an ordering π of these files on disk, such that the cost $c(\pi)$ is minimal, where $c(\pi)$ is defined as

$$c(\pi) = \sum_{i=1}^{m} \frac{w_{\pi(i)}}{r(\sum_{j=1}^{i} s_{\pi(j)})}. \tag{6.6}$$

□

Note that the function r is sufficiently general to cover multi-zone disks. The ordering π lists the order in which the files are stored on disk: file $\pi(1)$ is stored first, starting at the outermost track, followed by file $\pi(2)$, et cetera. The transfer rate associated with file $\pi(i)$ is the transfer rate at the position on disk where the last bit of file $\pi(i)$ is stored, which is given by $r(\sum_{j=1}^{i} s_{\pi(j)})$.

We next prove that RFA is NP-hard. NP-hardness implies that no polynomial-time algorithm exists that solves each instance of RFA to optimality, unless P=NP. We refer to Garey & Johnson [1979] for more background on the theory of computational complexity.

Theorem 6.1. RFA *is* NP*-hard.*

Proof. An optimization problem is NP-hard if its decision variant is NP-complete. The decision variant RFA-D of RFA is defined as follows. Given an instance I and a cost k, is there an ordering π such that $c_I(\pi) \leq k$?

RFA-D is clearly in NP: given an instance I, a cost k, and an ordering π, it can be checked in polynomial time whether $c_I(\pi) \leq k$. We next present a reduction from PARTITION, a well-known NP-complete problem [Garey & Johnson, 1979]. PARTITION is defined as follows. Given a set $U = \{1, 2, \ldots, m\}$ of items, each item i being characterized by a positive, integer size a_i, is there a subset V of U such that $\sum_{i \in V} a_i = \sum_{i \in U \setminus V} a_i$?

Let $a_1, a_2, \ldots, a_m$ denote an arbitrary instance of PARTITION, and let $U = \{1, 2, \ldots, m\}$. We assume a disk of size $s = \sum_{i=1}^{m} a_i$, containing two zones 1 and 2 both of size $s/2$ with transfer rates $\hat{r}_1$ and $\hat{r}_2$, with $\hat{r}_1 > \hat{r}_2$. We consider m files, each file i being characterized by $w_i = s_i = a_i$, and we define $k = s/(2\hat{r}_1) + s/(2\hat{r}_2)$. We next prove that we have a yes-answer to the instance of PARTITION if and only if we have a yes-answer to this instance I of RFA-D.

Assume that there is a $V \subseteq U$ such that $\sum_{i \in V} a_i = \sum_{i \in U \setminus V} a_i = s/2$. Then storing the files in V in the outer zone and the others in the inner zone results in a cost equal to $\sum_{i \in V} a_i/\hat{r}_1 + \sum_{i \in U \setminus V} a_i/\hat{r}_2 = s/(2\hat{r}_1) + s/(2\hat{r}_2) = k$, so that this instance of RFA has a yes-answer. This is illustrated in Figure 6.3a, where the files in V are stored in the dark-grey area and the files in $U \setminus V$ in the light-grey area of equal size.

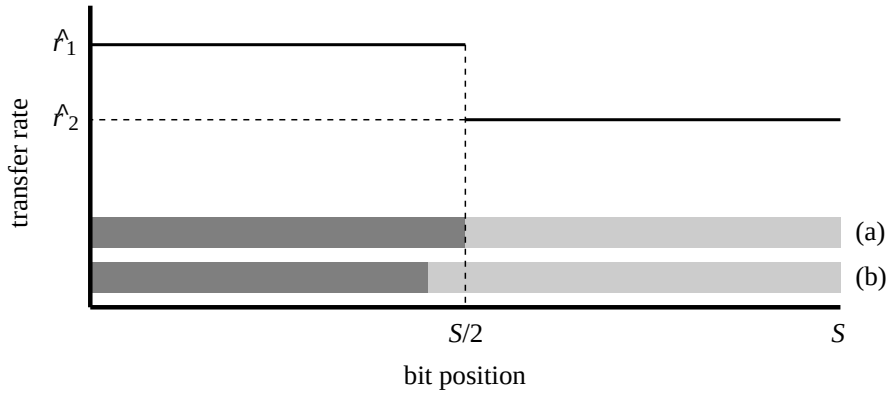


Figure 6.3. Illustration of the two cases. For explanation, see the running text.

Conversely, assume that there is an ordering π such that $c_I(\pi) \leq k$, that is

$$\sum_{i \in U} \frac{a_{\pi(i)}}{r(\sum_{j=1}^{i} a_{\pi(j)})} \leq \frac{s}{2\hat{r}_1} + \frac{s}{2\hat{r}_2}. \tag{6.7}$$

Note that the left-hand side of this equation can alternatively be written as $\sum_{i \in V} a_i/\hat{r}_1 + \sum_{i \in U \setminus V} a_i/\hat{r}_2$, with $\sum_{i \in V} a_i \leq s/2$ for some $V \subseteq U$. This is because there are only two zones, with transfer rates $\hat{r}_1$ and $\hat{r}_2$, respectively, and the total size of all files with associated transfer rate $\hat{r}_1$ is at most the size of the corresponding zone, which is $s/2$. Elementary calculus now shows that, as $\hat{r}_1 > \hat{r}_2$, Equation 6.7 can only hold if $\sum_{i \in V} a_i \geq s/2$, so that $\sum_{i \in V} a_i = s/2$ holds. In other words, the instance of PARTITION has a yes-answer. Therefore, RFA-D is NP-complete and hence RFA is NP-hard.

Figure 6.3b illustrates the equivalent case that there is no equal split. All files in the dark-grey area have an associated transfer rate of $\hat{r}_1$, whereas all files in the larger, light-grey area have an associated transfer rate of $\hat{r}_2$, resulting in a total cost that is easily seen to be larger than k. □

It can be shown that RFA is NP-hard in the strong sense by applying a similar reduction from 3-PARTITION. The latter problem is known to be NP-hard in the strong sense [Garey & Johnson, 1979]. As this reduction from a given case of 3-PARTITION with $3m$ elements leads to a disk with m zones, we can conclude that, if we add the restriction to RFA that the disk contains only two zones, the problem is not NP-hard in the strong sense anymore, but still NP-hard.

Theorem 6.2. RFA *with equal file sizes is in* P*, and an optimal solution is found by storing the files in order of non-increasing weight from the outermost to the innermost track of the disk.*

Proof. In the case that all files have equal size $\tilde{s}$, then Equation 6.6 simplifies to

$$c(\pi) = \sum_{i=1}^{m} \frac{w_{\pi(i)}}{r(i\tilde{s})}.$$

In this equation, besides having a fixed set of numerators, the set of denominators is now also fixed. This means that, by interchanging the positions of two files, only their weights switch places, whereas all denominators remain identical. Suppose we have two numerators w_1 and w_2 and two denominators r_1 and r_2, with $r_1 \geq r_2$. It readily follows that, if $w_1 \geq w_2$, then

$$\frac{w_1}{r_1} + \frac{w_2}{r_2} \leq \frac{w_2}{r_1} + \frac{w_1}{r_2},$$

so that storing the files in order of non-increasing weight from the outermost to the innermost track of the disk yields an optimal solution. □

6.2.2 A heuristic algorithm

As RFA is (strongly) NP-hard, algorithms that solve this problem to optimality can only deal with relatively small instances. For larger instances, it is necessary to consider heuristic algorithms that produce good results, although not necessarily optimal. Surprisingly enough, applying a simple sorting algorithm on the m files, which takes $O(m \log m)$ time, yields remarkably good results for practical cases. By storing files in order of non-increasing weight-size ratio on disk, results are generally obtained that are either optimal or close to optimal. For reference, we call this heuristic the *largest ratio first* algorithm (LRF).

Note that, if all files have the same size, LRF effectively stores the files in order of non-increasing weight. In this case, LRF yields an optimal solution by Theorem 6.2.

If each file i is streamed at a constant bit rate r_i and only once from beginning to end, then for each file i, it holds that $s_i = r_i d_i$ and, consequently, $w_i/s_i = \gamma_i$. The ordering suggested above then boils down to storing files in order of non-increasing popularity, corresponding to the approach by Ghandeharizadeh, Ierardi, Kim & Zimmermann [1996].

We next investigate the worst-case performance ratio of LRF, where we show that LRF is one of the worst algorithms. We proceed by illustrating why this algorithm performs remarkably well in practice and substantiate this further by simulation results.

Worst-case performance ratio. The general form of the problem statement, in particular the fact that the transfer rate r of the disk need only be non-increasing, leads to the following result.

Theorem 6.3. *The worst-case performance ratio of* LRF *is* ∞.

Proof. We prove this result by, for a fixed, but arbitrary value of k, providing an instance and comparing the solution found by LRF for this instance with an alternative solution. The ratio of their costs then provides a lower bound on the worst-case performance ratio.

We fix a value of k and consider a multi-zone disk with n zones numbered $1,2,\ldots,n$. Each zone i, except zone n, has size $z_i = z$, where z is a constant, and zone n has size $z_n = n$. The transfer rate r_i in zone i, for $i = 1,2,\ldots,n$, is given by $r_i = r_1/k^{i-1}$, where r_1 is a constant. We have $n+1$ files, numbered $1,2,\ldots,n+1$. File 1 has size $s_1 = 1$ and weight $w_1 = n+1$ and file i, for $i = 2,3,\ldots,n$ has size $s_i = z_{i-1} = z$ and weight $w_i = (n+2-i)\,s_i = (n+2-i)\,z$. Finally, file $n+1$ has size $s_{n+1} = z_n - 1 = n-1$ and weight $w_{n+1} = n-1$.

The file sizes s_i sum up to the disk size, which is the sum of the zone sizes z_i. Indeed, $\sum_{i=1}^{n+1} s_i = 1 + \sum_{i=2}^{n} z + n - 1 = (n-1)\,z + n$, which equals $\sum_{i=1}^{n} z_i = (n-1)\,z + n$.

As the weight-over-size ratios of each file i is $n+2-i$, LRF stores these files on disk in the order $1,2,\ldots,n+1$, starting at the outer track, as illustrated in Figure 6.4.

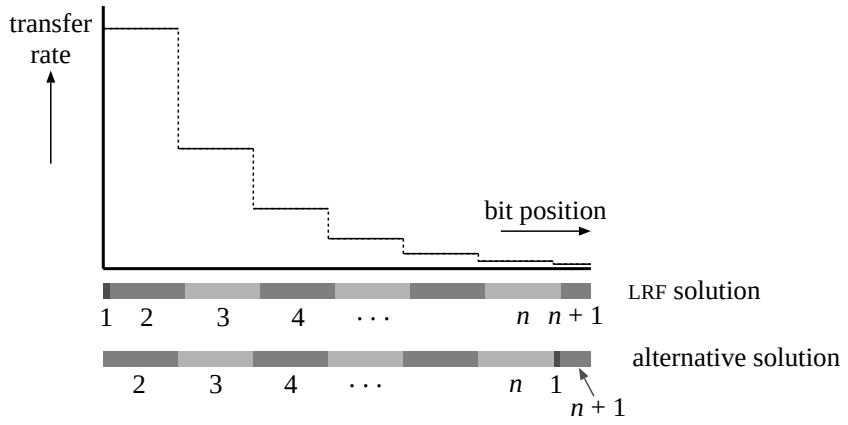


Figure 6.4. Illustration of how LRF stores the $n+1$ files on disk.

We compare the cost c_{LRF} of the LRF solution with the cost c_{ALT} of the alternative solution, whereby file 1 is repositioned between files n and $n+1$ and

files 2 to n shifted outward, as illustrated in the same figure. We have that

$$\begin{aligned} c_{\text{LRF}} &= \sum_{i=1}^{n} \frac{w_i}{r_i} + \frac{w_{n+1}}{r_n} \\ &= \frac{n+1}{r_1} + \sum_{i=2}^{n} \frac{(n+2-i)z}{r_1/k^{i-1}} + \frac{n-1}{r_1/k^{n-1}} \\ &= \frac{1}{r_1}\left(n+1+(n-1)k^{n-1} + z\sum_{i=2}^{n}(n+2-i)k^{i-1}\right) \\ &= \frac{1}{r_1}\left(n+1+(n-1)k^{n-1} + zkq\right), \end{aligned}$$

where q is defined as

$$q = \sum_{i=2}^{n}(n+2-i)k^{i-2}. \tag{6.8}$$

Similarly, we have that

$$\begin{aligned} c_{\text{ALT}} &= \frac{w_1}{r_n} + \sum_{i=2}^{n} \frac{w_i}{r_{i-1}} + \frac{w_{n+1}}{r_n} \\ &= \frac{n+1}{r_1/k^{n-1}} + \sum_{i=2}^{n} \frac{(n+2-i)z}{r_1/k^{i-2}} + \frac{n-1}{r_1/k^{n-1}} \\ &= \frac{1}{r_1}\left(2nk^{n-1} + zq\right), \end{aligned}$$

with q as defined in Equation 6.8. As, for fixed k, r_1, and n, it holds that

$$\lim_{z\to\infty} \frac{c_{\text{LRF}}}{c_{\text{ALT}}} = k,$$

we can choose z large enough to establish that

$$\frac{c_{\text{LRF}}}{c_{\text{ALT}}} \geq k/2.$$

As k was arbitrary, we have proved that the worst-case performance ratio of LRF is ∞. □

A few remarks on this proof are in place. Firstly, it can be shown that the alternative solution is the optimal solution, provided that z is sufficiently large, using a similar reasoning as used in the proof of Theorem 6.2. Secondly, it is remarkable that in the instance, all except two files have the same size, whereas Theorem 6.2 states that, if *all* files have the same size, the problem is in P. Evidently, LRF can have significant difficulty with even "almost easy" instances.

Thirdly, switching the positions of adjacent files does not improve upon the result. This can be seen by adding a file $n+2$ of size 1 and weight $n+0.5$ and reducing the size of file $n+1$ by 1. Files 1 and $n+2$ will be stored by LRF at the outermost positions. Switching files $n+2$ and 2 does not decrease the cost, as the transfer rate associated to file 2 does not change, whereas the transfer rate associated to file $n+2$ decreases. Furthermore, although moving file $n+1$ outward by switching adjacent files until it is next to file $n+2$ *does* lower the initial cost of LRF, its effect, that is, the term $(n-1)k^{n-1}$ is replaced by the term $n-1$, is marginal and does not influence the overall result. Finally, switching two adjacent files with the same size does not improve upon the result either, as argued earlier. Consequently, no strategy based on switching adjacent files improves upon the result.

Fourthly, the disk parameters used are not realistic in practice. On a multi-zone disk, one would expect the transfer rate to approach that of a constant-density disk, where the transfer rate decreases linearly from the outermost track to the innermost one, resulting in a rate that decreases as a square-root function of the bit position, and not as a negative-exponential function.

The following theorem provides more practical results, where we express the worst-case performance ratio in terms of some disk characteristics.

Theorem 6.4. *The worst-case performance ratio of* LRF *is* $1/\rho$*, where* ρ *denote the ratio of the minimum and maximum transfer rate of the disk.*

Proof. We first prove that $1/\rho$ is an upper bound for *any* algorithm that computes an ordering for the files. This then holds in particular for LRF. Consider Equation 6.6. The cost of any ordering can be bounded from above by substituting the minimum transfer rate $r_{\min}$ for the denominators, leading to an upper bound of $\sum_{i=1}^{n} w_i/r_{\min}$. Conversely, the cost of an optimal ordering can be bounded from below by substituting the maximum transfer rate $r_{\max}$ for the denominators, leading to a lower bound of $\sum_{i=1}^{n} w_i/r_{\max}$. The ratio of these two bounds equals $1/\rho$.

We next construct an instance that approaches this bound; see Figure 6.5. The disk of size s has two zones, the outer is of size $s-1$ and the inner is of size 1. We have two files, file 1 has size $s-1$ and weight w_1, file 2 has size 1 and weight w_2. We assume that $w_1 > w_2 > w_1/(s-1)$, so that LRF stores file 2 on the outer track. The associated cost c_{LRF} is given by

$$c_{\text{LRF}} = \frac{w_2}{r_{\max}} + \frac{w_1}{r_{\min}}.$$

The alternative solution has cost c_{ALT}, given by

$$c_{\text{ALT}} = \frac{w_1}{r_{\max}} + \frac{w_2}{r_{\min}}.$$

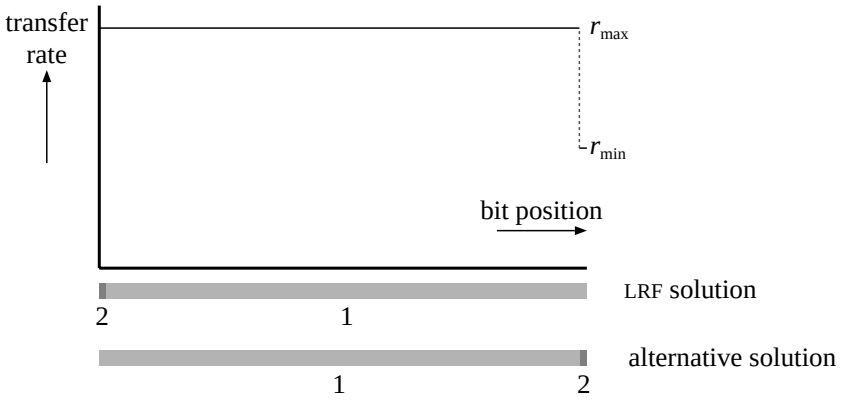


Figure 6.5. Illustration of a worst-case instance.

As $w_1 > w_2$, it holds that $c_{\text{ALT}} < c_{\text{LRF}}$, as is easily verified, so that the alternative solution is the optimum. We now have that

$$\frac{c_{\text{LRF}}}{c_{\text{ALT}}} = \frac{\frac{w_2}{r_{\max}} + \frac{w_1}{r_{\min}}}{\frac{w_1}{r_{\max}} + \frac{w_2}{r_{\min}}} = \frac{w + \frac{1}{\rho}}{1 + \frac{w}{\rho}},$$

with $w = w_2/w_1$. We choose $w = (1+\varepsilon)/(s-1)$ for some small $\varepsilon > 0$ and conclude that

$$\frac{c_{\text{LRF}}}{c_{\text{ALT}}} \to 1/\rho \quad \text{as} \quad s \to \infty.$$

□

It thus turns out that, in terms of the worst-case performance ratio, LRF is one of the worst algorithms. In practice, however, the algorithm works remarkably well, as we will illustrate below.

Performance analysis. To illustrate why the sorting algorithm works so well, we split each file i into s_i sub-files of unit size, where sub-file j of i is characterized by a weight $w_{ij} = w_i/s_i$, for each $j = 1, 2, \ldots, s_i$.

The splitting operation results in a total of $\sum_{i=1}^{m} s_i$ sub-files of unit size, whereby all sub-files of one file have equal weight. On account of Theorem 6.2, storing these sub-files in order of non-increasing weight from the outermost to the innermost track of the disk yields an optimal ordering. In particular, as all sub-files of one file have the same weight, these sub-files may be stored contiguously without violating the property of optimality. As such, Equation 6.6 is minimized for this collection of sub-files.

Now, let us only look at contiguous orderings of the sub-files where all those associated with a single file are stored contiguously. For each such ordering of sub-files, each sub-file j of file i has a transfer rate $\hat{r}_{ij}$, based on its location on disk, and the cost of this ordering is given by

$$\sum_{i=1}^{m} \sum_{j=1}^{s_i} \frac{w_{ij}}{\hat{r}_{ij}} = \sum_{i=1}^{m} \frac{w_i}{s_i} \sum_{j=1}^{s_i} \frac{1}{\hat{r}_{ij}}. \tag{6.9}$$

Now, observe that $(\sum_{j=1}^{s_i} 1/\hat{r}_{ij})/s_i$ denotes the average of the reciprocal values of the bit rates associated with the sub-files of file i. In case s_i is not too large and the transfer rate function descends relatively smoothly, the individual bit rates in this average will be relatively close to each other, as all sub-files of file i are stored contiguously, so that replacing this average by $\max_{j=1}^{s_i} 1/\hat{r}_{ij}$ will result in a cost function that only differs marginally from Equation 6.9. Optimizing Equation 6.9 then also yields a good ordering for this approximating cost function. But then, as this maximum denotes the reciprocal value of the bit rate associated with the sub-file of i that is stored closest to the innermost track of the disk surface, it also denotes the reciprocal value of the bit rate associated with the entire file in the original problem, so that this approximating cost function is the cost function of the original problem.

Summarizing, storing files in order of non-increasing ratio of their weights and sizes from the outermost to the innermost track of the disk may lead to close to optimal orderings, especially when the individual file sizes are not too large and the transfer rate function descends relatively smoothly.

These conditions are satisfied when a large number of files of comparable size has to be stored on a disk with relatively many, approximately equal-size zones in terms of the number of tracks and where the track size of the successive zones decrease relatively smoothly from the outermost to the innermost zone. The latter properties are typical for contemporary disks.

Simulations. To support these findings for relatively small numbers of files, we conduct a set of simulations on 1,000 small instances using a constant-density version of a contemporary disk; see Korst & Pronk [2005], assuming that only a single surface is used for video files. The constant-density version of this disk is characterized as follows; see also Figure 6.6. The transfer rate at the innermost track of each zone of the original disk is equal to the transfer rate of the corresponding track on the constant-density disk.

This constant-density disk has a maximum transfer rate $r_{\max}$ of 467 Mbit/s, a minimum transfer rate $r_{\min}$ of 221 Mbit/s, and a size s of 172 Gbit. For each instance, we choose the number m of files randomly in the range $\{5, 6, \ldots, 10\}$, each file being characterized by a randomly chosen weight in the range [0,1)

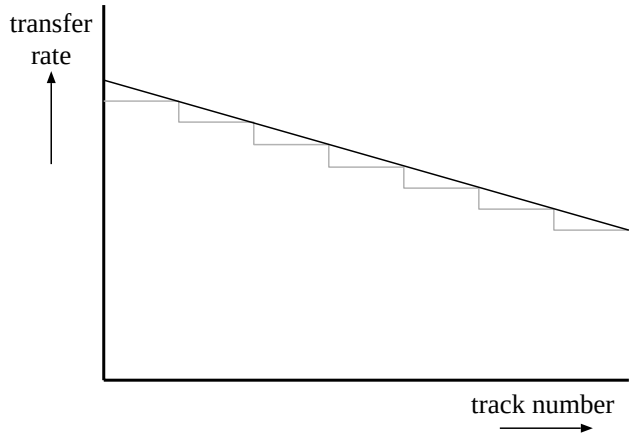


Figure 6.6. The transfer rate function of the original disk (gray step function) and the constant-density version (black line).

and a randomly chosen integer size in the range $[0, z)$, where z is chosen as $4s/m$ to allow a relatively large variation in file sizes. For the generation of each instance, repeatedly m file sizes are chosen until the sum of the sizes is at most the disk size. The small number of files allows optimal orderings to be computed.

The results for LRF are as follows. In approximately 70% of the instances, an optimal ordering is found. For the remaining 30%, the cost of each ordering is at most 3% larger than the cost of an optimal ordering. To give an indication of the relevance of finding good or optimal orderings, we also generate for each instance a worst ordering with maximal cost. The costs of the worst orderings are 35% larger than the optimal costs on average, and the cost of the worst ordering observed is 88% larger. Note that any ordering can be at most $r_{\max}/r_{\min} - 1 = 111\%$ larger than the optimal cost, as all transfer rates are between the minimum and maximum transfer rates. It thus turns out that LRF performs quite well, even for small numbers of large and small files.

We next conduct the same simulations as described above, but instead of using a constant-density disk, we use the original multi-zone disk. The results are as follows. In approximately 30% of the instances, an optimal ordering is found. For the remaining 70%, the cost of each ordering is at most 6% larger than the cost of an optimal ordering, but in only 2% of the instances, the cost is more than 2% larger. For these simulations, the costs of the worst orderings found are 37% larger than the optimal costs on average, and the cost of the worst ordering observed is also 88% larger.

An ordering found by the sorting algorithm can, of course, be used as an initial ordering by other algorithms that attempt to improve upon this ordering. As an example, consider the operation of scanning an initial ordering once from the beginning to the end, where the positions of two adjacent files are switched in case this results in an improvement. Applying this $O(m)$ algorithm to the orderings found by LRF improves upon the results in the sense that, for the constant-density disk, in 98% instead of 70% of all instances an optimal ordering is found. For the multi-zone disk, the improvement is that in 68% instead of 30% of all instances an optimal ordering is found.

6.3 Analysis of a special case

In this section and the next, we study a special case in more detail. In particular, we compare RFA with DTP and STP. We start with an analytical comparison of the maximum number of streams that can be sustained simultaneously, under various simplifying assumptions, most notably that the files only differ in their popularity. We initially ignore the issue of start-up latency, but include this in a second analysis. Section 6.4 provides simulation results, supporting the results of this section while relaxing some of the assumptions.

6.3.1 Without constraints on the start-up latency

We consider m files, numbered $1,2,\ldots,m$, of equal duration $\tilde{d}$, equal size $\tilde{s}$, and equal bit rate $\tilde{r}$, each file i being characterized by its popularity γ_i, with $\sum_{j=1}^{m}\gamma_j = 1$ and $\gamma_j \geq \gamma_{j+1}$ for $j < m$. The files are thus numbered in order of non-increasing popularity. We use a single, constant-density disk with a minimal transfer rate of $r_{\min}$ and a maximal transfer rate of $r_{\max}$. We assume that the disk is exactly large enough to contain all files, that is, we assume that its size s equals $m\tilde{s}$.

Referring again to the remark at the end of Section 6.2.1, as all files are of equal size $\tilde{s}$, they are already sorted appropriately, and they should be stored in order of increasing file number from the outermost to the innermost track of the disk to minimize the cost. Note that LRF also results in this ordering.

The bits on the disk are numbered from 1 onwards, starting at the outermost track. File i thus occupies bits $(i-1)\tilde{s}+1,(i-1)\tilde{s}+2,\ldots,i\tilde{s}$. It can be shown that at bit position $i\tilde{s}$, the transfer rate is accurately described by

$$r(i) = r_{\max}\sqrt{1-(1-\rho^2)\frac{i}{m}}, \tag{6.10}$$

where $\rho = r_{\min}/r_{\max}$. This rate is based on the assumption that the successive bits on disk are organized as a spiral, rather than as a number of concentric cir-

cles. The difference with the actual transfer rate can be shown to be negligibly small.

For file i, the minimum rate used to read data from disk is thus given by $r(i)$. The average transfer rate is given by $r_{\text{avg}} = (r_{\text{min}} + r_{\text{max}})/2$, as shown in Section 6.1.

Now, assume that a large number n of streams simultaneously exist, and that a part $\gamma_i n$ of them is associated with file i. For simplicity, we ignore the fact that these fractions need not be integer-valued. Each of the $\gamma_i n$ streams associated with file i only uses at most a fraction $\tilde{r}/r(i)$ of the time available for reading, the latter of which can be arbitrarily close to 1 per unit time. So, it should hold that

$$\sum_{i=1}^{m} \gamma_i n \frac{\tilde{r}}{r(i)} < 1,$$

or, equivalently,

$$n < \frac{1}{\tilde{r} \sum_{i=1}^{m} \frac{\gamma_i}{r(i)}}. \tag{6.11}$$

The right-hand side of this inequality is maximized if the sum in the denominator is minimized, which nicely illustrates the relevance of RFA.

To substantiate this result further, we assume that the popularity of files is given by Zipf's law, which is often used as a popularity distribution [Breslau, Cao, Fan, Phillips & Shenker, 1999; Griwodz, Bär & Wolf, 1997]. Zipf's law states that, for each file i,

$$\gamma_i = \frac{\frac{1}{i}}{\sum_{j=1}^{m} \frac{1}{j}}. \tag{6.12}$$

Note that the denominator is just a normalization constant. In practice, Zipf's law is generally not followed exactly, and in Section 6.4 we provide simulation results where Zipf's law is followed only statistically.

By substituting Equations 6.10 and 6.12 in Equation 6.11, we can express n as a function of the known parameters.

Numerical results. We use $m = 10$ files, each with a bit rate $\tilde{r} = 7$ Mbit/s and size $\tilde{s} = 17.2$ Gbit, $r_{\text{min}} = 221$ Mbit/s, and $r_{\text{max}} = 467$ Mbit/s, so that $\rho = 0.473$ and $r_{\text{avg}} = 344$ Mbit/s.

When using either DTP or STP, a maximum of $\lfloor 344/7 \rfloor = 49$ streams can be sustained simultaneously. Using RFA and assuming Zipf's law, evaluation of Equation 6.11 yields a maximum of 54 simultaneous streams, an increase of slightly over 10% when compared to both variants of track pairing.

In the case that all files are equally popular, that is, $\gamma_i = 1/m$ for all i, RFA performs worse than STP. This is because it can be shown that $\sum_{i=1}^{m} 1/r(i)$ is bounded from below by m/r_{avg}. However, this bound becomes tighter as m increases.

For completeness, we mention that, when using the minimum transfer rate of 221 Mbit/s, only 31 streams can be sustained simultaneously, and the absolute maximum number of simultaneously active streams is given by $\lfloor r_{\max}/\tilde{r} \rfloor = 77$. The latter is possible if all streams only read data at the outermost track of the disk.

6.3.2 With constraints on the start-up latency

In practice, the transfer rate of the disk is not fully exploited, in order to keep the start-up latencies and buffer requirements at an acceptable level. Referring to Equation 6.4, allowing $\sum_{i=1}^{n} r_i/\hat{r}_i$ to approach 1 leads to very large period lengths, which in turn leads to correspondingly large blocks, buffers, and start-up latencies. The latter was already observed in the previous chapter. In this section, we investigate how this influences the maximum number of simultaneously sustainable streams for DTP, STP, and RFA.

We assume that the n streams are served using the triple buffering algorithm discussed in Chapter 5 with period length p so that the block size for each stream equals $p\tilde{r}$. For the access time function a we use the following affine function, where for n disk accesses the access time is given by

$$a(n) = \alpha + \beta n, \tag{6.13}$$

for some constants α and β. For details, we again refer to Korst & Pronk [2005].

The worst-case start-up latency for a new stream is $2\,p$. We now concisely reconsider the safeness condition in Equation 6.3 for DTP, STP, and RFA. Clearly, the period length should be large enough to read one block from disk for each of the n streams. For DTP, reading a block results in a transfer time of $p\tilde{r}/r_{\text{avg}}$, whereas the access time is $a(2n)$, since each block requires two accesses. It should thus hold that

$$p \geq a(2n) + n\frac{p\tilde{r}}{r_{\text{avg}}}.$$

Using Equation 6.13, this can be rewritten as

$$n \leq \frac{p-\alpha}{2\beta + \frac{p\tilde{r}}{r_{\text{avg}}}}. \tag{6.14}$$

For STP, reading a large or small block also results in a transfer time of $p\tilde{r}/r_{\text{avg}}$, whereas the switching overhead is only $a(n)$, so that it should hold that

$$p \geq a(n) + n\frac{p\tilde{r}}{r_{\text{avg}}}.$$

Using Equation 6.13, we obtain

$$n \leq \frac{p-\alpha}{\beta + \frac{p\tilde{r}}{r_{\text{avg}}}}. \tag{6.15}$$

When comparing Equations 6.14 and 6.15, it follows that, for any value of p, STP performs at least as well as DTP in terms of the maximum number of simultaneously sustainable streams, at the cost of a small increase in buffer size.

For RFA, each of the $\gamma_i n$ streams associated with file i requires a read time of at most $p\tilde{r}/r(i)$ for each block. Consequently, for RFA, it should hold that

$$p \geq a(n) + \sum_{i=1}^{m} \gamma_i n \frac{p\tilde{r}}{r(i)},$$

which can likewise be rewritten as

$$n \leq \frac{p-\alpha}{\beta + p\tilde{r}\sum_{i=1}^{m}\frac{\gamma_i}{r(i)}}. \tag{6.16}$$

Numerical results. We next compare Equations 6.14 – 6.16 for several values of p and assuming Zipf's law. For the parameters of the switching overhead in Equation 6.13, we use $\alpha = 27$ ms and $\beta = 17.3$ ms, derived from the parameters of the disk used. The other parameters are as before. The results are shown in Table 6.1. For each value of p, it lists the maximum values of n satisfying Equations 6.14 – 6.16. Figure 6.7 shows the comparison graphically.

As α and β vanish in each of the three equations for growing values of p, the curves for DTP and STP approach 49, and the curve for RFA approaches 54, when p becomes sufficiently large.

The results show that RFA outperforms both DTP and STP, the absolute differences between STP and RFA generally becoming more pronounced as the period length increases.

Alternatively, to be able to simultaneously sustain, say, 35 streams, DTP leads to a worst-case start-up latency of 9 s, STP 4.5 s, and for RFA with Zipf's law, between 3.5 and 4 s.

p	DTP	STP	RFA	p	DTP	STP	RFA
0.25	5	9	10	2.75	30	37	40
0.50	10	17	17	3.00	31	37	41
0.75	14	22	23	3.25	31	38	41
1.00	17	25	27	3.50	32	39	42
1.25	20	28	30	3.75	33	39	43
1.50	22	30	32	4.00	34	40	43
1.75	24	32	34	4.25	34	40	44
2.00	26	34	36	4.50	35	41	44
2.25	27	35	37	4.75	35	41	45
2.50	28	36	39	∞	49	49	54

Table 6.1. Comparison between DTP, STP, and RFA of the maximum number of simultaneously sustainable streams, assuming Zipf's law for RFA.

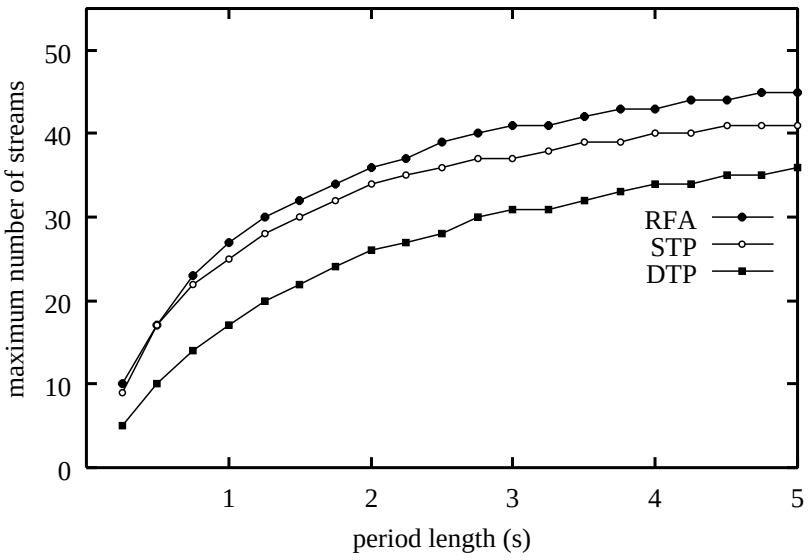


Figure 6.7. Graphical comparison between DTP, STP, and RFA of the maximum number of simultaneously sustainable streams, assuming Zipf's law for RFA.

Recall that for STP, reading should start with an odd block. This is to ensure that data is only read ahead of time. In case it should be possible to start reading at any block, then the worst-case start-up latency is given by $3\,p$. This may occur when the first block to read is an even block, which is split in two

parts and stored at two locations on disk. This requires two cycles to read the entire block instead of one. Hence, to achieve comparable worst-case start-up latencies as with DTP and RFA, a smaller period length should be used. For example, to achieve a worst-case start-up latency of 6 s, p should be chosen equal to 3 for DTP and RFA and equal to 2 for STP. The maximum number of simultaneously sustainable streams in this case is then 31 for DTP, 34 for STP, and 41 for RFA. In the next section, we do not consider this issue and assume that reading starts at an odd block.

6.4 Simulations

The results in Section 6.3.2 are obtained while assuming an ideal division of the streams among the files, that is, file i has $\gamma_i n$ associated streams. In practice, deviations will generally exist, most notably because $\gamma_i n$ need not be integer, but also because Zipf's law is only followed statistically.

To obtain insight into the actual number of simultaneously sustainable streams using RFA and assuming Zipf's law, we conduct 10^6 independent simulation runs for each of the worst-case start-up latencies of 2, 5, and 10 s, corresponding to period lengths of 1, 2.5, and 5 s, respectively. The parameters are as before. In each run, we start with zero streams, successively generate additional streams according to Zipf's law, and stop just before the disk becomes overloaded by adding a stream with largest disk resource requirements, in this case with minimal transfer rate. This admission control criterion expresses fairness among the files in the sense that Zipf's law is indeed followed statistically.

Figure 6.8 illustrates for the three values of the start-up latency the three normalized frequency histograms of the maximum number of simultaneously sustainable streams. The dashed lines directly to the left of each histogram indicate the corresponding values for STP. It shows that STP is outperformed by RFA with high probability for each of the three cases.

6.5 Related work

Birk [1995b] considers track pairing in the broader context of various load balancing techniques, for single disks as well as multiple disks. He also briefly considers tertiary storage such as tapes.

Ghandeharizadeh, Ierardi, Kim & Zimmermann [1996] also consider the problem of storing files on a single multi-zone disk. The cost function they minimize is the expected time to read an entire file from disk, taking into account that a file may cross zone boundaries. The authors store files contiguously from the outermost to the innermost track of the disk in order of

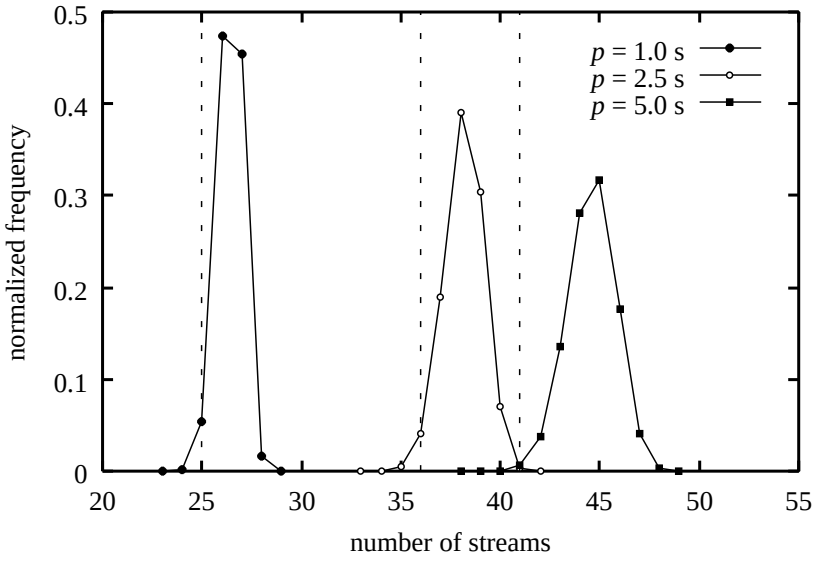


Figure 6.8. Normalized frequency histograms for period lengths 1, 2.5, and 5 s, assuming Zipf's law and the corresponding bounds for STP. For further explanation, see the text.

non-increasing popularity and prove that this is indeed optimal for this cost function. For each file, they thus consider the total amount of resource, that is, disk reading time, used by a stream when it reads the entire file once. In a practical context, this measure is not necessarily representative of the amount of resource required or reserved to provide real-time guarantees to individual streams, although it can be used as a lower bound. Actually attaining or approaching this lower bound generally leads to complex admission control, (online) renegotiation, and disk scheduling algorithms, as resource requirements may vary over time. This holds in particular for VBR files, but also for CBR files that cross a zone boundary. Allocating an explicit bit rate to a stream for its entire lifetime to provide real-time guarantees greatly simplifies these tasks, but results in the cost function given by Equation 6.6.

Tse & Leung [2000] consider a constant-density disk and analyze an allocation strategy wherein the files with higher bit rates are stored closer to the outermost track of the disk. The authors also consider non-real-time data, which is stored closer to the innermost track.

Tong, Huang & Liu [1998] discuss two strategies for reorganizing the zones on disk, called free-π and fixed-π. Both strategies define logical zones,

less in number than the physical zones, and aim at maximizing the average bandwidth achieved when scanning the disk once.

Michiels [1999] and Michiels & Korst [2001] propose an alternative approach to exploiting the multi-zone properties of a disk. The authors choose a fixed transfer time for all blocks that is smaller than the transfer time of a block stored in the innermost zone. As the actual transfer times of the individual blocks during a cycle may be smaller or larger than this fixed transfer time, additional buffering, expressed in number of blocks, is required to prefetch a sufficient amount of data to prevent buffer underflow. As a result of using this fixed transfer time, the period length, and thus the block size, can generally be chosen smaller when compared to using the guaranteed transfer rate. This effectively decreases the required buffer size. Distributing the successive blocks of a single file appropriately over the zones allows the fixed transfer time to be chosen to correspond closely to the average transfer rate of the disk.

In this chapter, we have mainly concentrated on the effective utilization of the variable transfer rate of a multi-zone disk, and put less emphasis on access-time overhead. The primary reason is that it is difficult to develop a worst-case model that improves upon the access-time function used in this chapter. We next provide some references where the access-time overhead is taken into account.

Ghandeharizadeh, Kim, Shahabi & Zimmermann [1996] introduce two allocation strategies: FIXB and VARB. In FIXB, a file is subdivided into fixed-size blocks and assigned in a round-robin fashion to the zones of the disk. In VARB, the blocks have variable size, depending on the bandwidth of the zone it is allocated to. Huang & Tsao [1997] propose to partition a disk in logical zones, each of which has the same number of tracks. A file is divided into variable-length blocks, of which the size corresponds to a fixed number of whole tracks, and the successive blocks are stored in successive logical zones in a zig-zag fashion. Both Ghandeharizadeh et al. and Huang and Tsao use the read-ahead established by reading near the outermost zones to compensate for the insufficient amount of data read near the innermost zones. By assuming that all streams are synchronized such that they all require a block in the same zone, a reduction in the access-time overhead is realized.

Tewari, King, Kandlur & Dias [1996] allocate CBR and CTL VBR blocks on disk, assuming that blocks are retrieved independently and randomly according to a popularity distribution on the blocks. The authors minimize the mean response time to fetch a block from disk. Starting at the innermost and outermost cylinder, blocks are allocated to the disk in order of non-decreasing popularity, converging toward a ‘hottest’ cylinder, where the most popular blocks are allocated.

Kim, Lho & Chung [1997] define the cylinder containing the middle sector of the disk as the hottest cylinder and allocate blocks toward the innermost and outermost cylinder, starting at the hottest cylinder, in order of non-increasing popularity. The authors compare this with the allocation strategy whereby the files are stored in order of non-increasing popularity from the outermost cylinder inward.

Triantafillou, Christodoulakis & Georgiadis [2000] calculate the optimal placement of blocks on disk such that accessing any number of these blocks incurs minimal cost, given an access probability for each block.

Triantafillou, Christodoulakis & Georgiadis [2002] provide an analytical model to assess the performance of disk devices, including multi-zone disks, under random workloads.

Wang, Tsao, Chang, Chen, Ho & Ko [1997] and Tsao, Chen & Sun [2001] propose to calculate a block size for each zone, based on the file to store, such that the read time for any block for this file is constant. The problem addressed is where to store the successive blocks on disk, assuming a fixed number of blocks per zone for the file.

Kang & Yeom [1999] propose a file allocation strategy called nearly constant transfer time whereby, for each file, a nearly constant transfer time is realized for each block stored. The authors employ a CTL approach for defining block sizes. As part of their file allocation strategy, they use the popularity of the individual files.

Kang & Yeom [2003] include multi-rate smoothing of VBR video with prefetching for storing files on a single multi-zone disk.

Several papers consider generalizations to multiple disks, such as Kim, Lim, Kim & Chung [1997], Lho & Chung [1998], Huang & Tsao [1999], and Chen & Wu [2003]. Park, Kim & Chung [1999] consider the heterogeneous case where the multi-zone disks need not be identical. Aerts [2003] focuses on the redundant storage of multiple files on multiple disks, where the disks may have a multi-zone character.

6.6 Concluding remarks

In this chapter, we have considered the offline problem of storing a given set of files on a multi-zone disk such that expected disk resource usage is minimized. One of the parameters describing the files is the popularity of each file. We have shown that this problem in NP-hard in the strong sense, but proposed a heuristic that performs remarkably well in practice. On the down side, the worst-case performance ratio is the worst obtainable with any storage algorithm for this problem.

Comparison with the well-known technique of track pairing shows that resource-based storage of files can outperform track pairing, provided that the popularity of the files is sufficiently skewed with respect to a uniform distribution.

The proposed heuristic algorithm suggests how to apply the results in an online setting, where files are occasionally added and deleted. This algorithm sorts the files, based on a simple function of the parameters that describe each file. Hence, by subdividing the disk into a number of *virtual zones* and associating to the successive virtual zones, from the outer to the inner zone, a range of relevant function values in decreasing order, files can be stored in a virtual zone with associated function value that is closest to the value of the function, when applied to the file to be stored. This storage need not be contiguous and can be based on a segmented storage strategy, e.g., the storage strategy with overlap presented in Chapter 5, within each virtual zone.

It stands to reason that, as the disk fills up or popularities of files change, file migration and deletion strategies will be necessary. This more dynamic problem is considered a subject for further research.

7

On the Fixed-Delay Pagoda Broadcast Schedule

Providing *true* video on demand requires a dedicated stream or channel for each user to enable full control by the user over this stream, such as pause/resume or skipping parts of the video. Large-scale introduction of this service is still hindered by resource limitations and cost aspects, as resource requirements, in the server as well as in the cable access network, increase linearly with the number of users.

During the past decade, research efforts in this area have led to alternative solutions that provide less flexibility, but are more cost effective. These solutions collectively go by the name of *near* video on demand (NVOD), and differ from their counterpart in that the emphasis is on linear viewing of a video rather than on full interactivity. The availability of local storage in the form of a hard disk at the user's home is generally assumed, so that significant parts of the selected video can, if necessary, be stored before they are consumed.

By restricting user control in this way, it is possible to have users share streams, thereby significantly reducing resource requirements. Video data is broadcast, or multicast to a group of users, rather than transmitted to individual users.

The various strategies that have been proposed in the literature can broadly be subdivided into two categories, namely *client-centered* and *data-centered*. In the first category, the broadcasting of a video is mainly governed by the users requesting access to this video. Depending on the requests, the server broadcasts the video trying to serve as many requests as possible with an acceptable start-up latency. The start-up latency is defined as the time that elapses between the moment that a user issues a request for a video and the moment that the user can start watching this video. Data-centered NVOD techniques, on the other hand, use fixed broadcast schedules, independent of user requests, also aiming at minimizing the maximum start-up latency. A survey paper on the latter category is given by Kameda & Sun [2004].

In this chapter, we focus on data-centered NVOD, and in particular on the *fixed-delay pagoda broadcast* (FDPB) schedule, proposed by Pâris [2001]. FDPB is a fragmented broadcast schedule, where a video file is split up into multiple, equal-size fragments and fragments near the start of the video are broadcast more frequently than fragments near the end of the video. Playout of the video is assumed to be at a constant bit rate (CBR), so that each fragment also has the same duration. For broadcasting the fragments, a number of channels are available, each having a bandwidth equal to the bit rate of the video at hand. These channels are slotted and synchronized at slot boundaries, where the duration of a slot equals the transmission time or, equivalently, play-out time of a fragment. FDPB describes when and on which channel each of the fragments is broadcast.

The idea of fragmented broadcasting already dates back to a patent filing by DeBey [1989], who proposes the following broadcast schedule for a given video file. Assume that the video file is partitioned into n fragments, numbered $1, 2, \ldots, n$. Fragment k is broadcast strictly periodically at slot s if and only if $s \bmod k = 0$, where slots are numbered from 0 onwards. Fragment k is thus broadcast at slots $0, k, 2k, \ldots$.

After requesting the video, a user only needs to wait until the start of the next slot, upon which it starts receiving and can start viewing fragment 1, as it is broadcast every slot. At the start of the next slot, either fragment 2 has already been received during the first slot, or it will be received during this slot, as fragment 2 is broadcast every other slot. Hence, during this next slot, the user can view fragment 2. Using a similar reasoning for fragments $3, \ldots, n$, it can be shown that the user receives all fragments in time to view the video without interruption, provided that he does not start watching the video until the start of the first slot after his request. The maximum start-up latency is thus one slot, which corresponds to a fraction of $1/n$ of the total duration of the video.

It goes without saying that for this broadcast schedule, the reception behavior at the user is quite bursty, ranging from 1 fragment at slot 1, to n fragments at slot 0, thus occasionally requiring n channels. This can be alleviated somewhat by appropriately shifting some periodic schedules.

The above schedule can be improved considerably [Hollmann & Holzscherer, 1991] by not broadcasting each fragment k strictly periodically, that is, exactly once every k slots, but instead broadcasting it *at least once* every k slots. This does not increase the maximum start-up latency, but allows the burstiness to be minimized, at the cost of broadcasting fragments possibly more often than strictly necessary. We formalize this as the *Broadcast Schedule Composition* problem (BSC).

Problem 7.1. [Broadcast Schedule Composition Problem (BSC)] Given are a constant-bit-rate video file f and m channels, each with a bandwidth equal to the bit rate of the file. Compose a broadcast schedule, while maximizing the number n of equally sized fragments in which f can be split up such that (*i*) the fragments are transmitted in the m available channels and such that (*ii*) each fragment k is broadcast at least once every k slots. □

Figure 7.1, taken from Korst & Pronk [2005], gives an example of a periodic broadcast schedule with a period of 12 and for $m = 3$. Since the schedule contains fragments 1 to 9, that is $n = 9$, the maximum start-up latency is $1/9$, again expressed as a fraction of the duration of the video. Van Kreij [2001] proves that there are no periodic fragmented broadcast schedules for $m = 3$ channels that subdivide the video into more than 9 fragments.

channel	time →											
1	1	1	1	1	1	1	1	1	1	1	1	1
2	4	2	3	2	4	2	6	2	3	2	5	2
3	6	5	7	8	9	3	5	7	4	8	9	3

Figure 7.1. A periodic broadcast schedule with a period of 12 and $m = 3$ to broadcast fragments 1 to 9.

This problem formulation can be generalized [Hollmann & Holzscherer, 1991; Pâris, 2001] by introducing an *offset* o and requiring that each fragment k is broadcast at least once every $o + k$ slots. In this case, the start-up latency is bounded from below by o slots and from above by $o + 1$ slots. During the o complete slots that a user is waiting, data is already received and stored locally. As we shall see, this offset allows n to be chosen considerably larger than when no offset is used.

Problem 7.2. [Broadcast Schedule Composition with Offset Problem (BSCO)] Given are a constant-bit-rate video file f, m channels, each with a bandwidth equal to the bit rate of the file, and an offset o. Compose a broadcast schedule, while maximizing the number n of equally sized fragments in which f can be split up such that (*i*) the fragments are transmitted in the m available channels and such that (*ii*) each fragment k is broadcast at least once every $o+k$ slots. □

The FDPB schedule provides a broadcast schedule for any offset o, although it generally does not optimize the number of fragments. The term *fixed delay* in FDPB pertains to the fact that the start-up latency is approximately constant, that is, approximately o slots.

In this chapter, we present two new results on FDPB. The first result pertains to the empirically found square-root heuristic used in the construction of the broadcast schedule. This heuristic aims to maximize the number of fragments that fit in a particular channel. We substantiate this heuristic and show that it can be slightly improved. The second result is that the schedule is asymptotically optimal in terms of the maximum start-up latency, by choosing the offset large enough.

The remainder of this chapter is organized as follows. We describe FDPB in detail in Section 7.1. Then, in Section 7.2, we discuss the square-root heuristic, and in Section 7.3 we prove that FDPB is asymptotically optimal in terms of the maximum start-up latency.

7.1 The fixed-delay pagoda broadcast schedule

The following description of FDPB is primarily taken from Verhaegh, Rietman & Korst [2004], slightly modified, and is included to make the chapter self-contained.

We are given a video file that, for normal playout, requires a constant bit rate, and we are given m broadcast channels, numbered $1,2,\ldots,m$, each having a bandwidth equal to the bit rate of the video file. The video file is split up into n equally sized fragments, numbered $1,2,\ldots,n$. The channels are slotted and synchronized on slot boundaries, where each slot corresponds to the time required for the playout, and thus the transmission, of one fragment.

In addition to using an offset as introduced above, we assume that at most $r \leq m$ channels may be tapped simultaneously, where tapping a channel is performed for a channel-dependent, prescribed number of consecutive slots. Initially, channels $1,2,\ldots,r$ are tapped. Upon completion of tapping channel i, channel $i+r$ starts being tapped, provided that $i+r \leq m$.

We next describe the general structure of the broadcast channels. We denote the first slot at which the user starts tapping channels by slot 0. We denote the start of tapping channel i by s_i, and the number of consecutive slots that is tapped in this channel by t_i. In order not to exceed the maximum number r of channels that may be tapped in parallel, tapping channel $i = r+1, r+2, \ldots, m$ is started after the tapping channel $i-r$ has ended. Hence, it holds that

$$s_i = \begin{cases} 0 & \text{for } i = 1, 2, \ldots, r \\ s_{i-r} + t_{i-r} & \text{for } i = r+1, r+2, \ldots, m. \end{cases} \tag{7.1}$$

In channel i a consecutive series of fragments is transmitted, the lowest fragment number being given by l_i and the number of fragments by n_i. We thus have that

$$l_i = \begin{cases} 1 & \text{for } i = 1 \\ l_{i-1} + n_{i-1} & \text{for } i > 1. \end{cases} \tag{7.2}$$

Of course, we have that $n_i \leq t_i$, as we can put at most t_i fragments in channel i, but in general, strict inequality will hold. This is because these fragments are each generally broadcast more than once in these t_i slots.

Figure 7.2 globally illustrates the mapping of the fragments of a designated part of the video file onto a channel. The sub-channels in this figure are explained shortly.

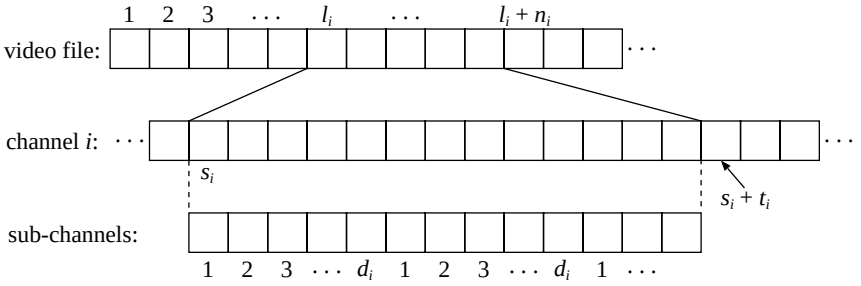


Figure 7.2. Global mapping of a number of consecutive fragments of the video file onto a channel i. The actual mapping is governed by a subdivision of this channel into a number of sub-channels.

In order to receive each fragment in time, fragment k should be transmitted at least once every $o+k$ slots. However, if fragment k is transmitted in channel i, which starts being tapped in slot s_i, then fragment k should be broadcast in this channel at least once every $o+k-s_i$ slots. Ideally, this period is exactly met for each fragment k, but it may be smaller. The problem we face here is that for a number of consecutive values of k, these periods should be approximated from below as well as possible.

The structure within a channel i in the fixed-delay pagoda schedule is as follows. Firstly, channel i is divided into a number d_i of *sub-channels*, numbered $1, 2, \ldots, d_i$, where d_i is defined as

$$d_i = \left\lfloor \sqrt{o + l_i - s_i} \right\rceil, \tag{7.3}$$

that is, the square root of the optimal period of the lowest fragment number l_i in channel i, rounded to the nearest integer. This is the square-root heuristic mentioned earlier that was initially proposed by Pâris. In Section 7.2 we investigate this in more detail. Each of these sub-channels gets a fraction $1/d_i$ of the slots in a round-robin fashion.

Each fragment k is given a fixed period within the sub-channel in which it is placed. Now, if a fragment k is given a period p_k within a sub-channel of channel i, it is broadcast in channel i with a period of $p_k d_i$. Hence, as we must have that $p_k d_i \leq o + k - s_i$, this means that

$$p_k \leq \left\lfloor \frac{o + k - s_i}{d_i} \right\rfloor. \tag{7.4}$$

By taking equal periods for all fragments within each sub-channel, we can trivially avoid collisions. So, if l_{ij} is the lowest fragment number in sub-channel j of channel i, this means that we choose a period

$$p_{ij} = \left\lfloor \frac{o + l_{ij} - s_i}{d_i} \right\rfloor \tag{7.5}$$

for sub-channel j of channel i, and hence we can transmit $n_{ij} = p_{ij}$ fragments (fragments $l_{ij}, \ldots, l_{ij} + n_{ij} - 1$) in this sub-channel. The fragment number l_{ij} is given by

$$l_{ij} = \begin{cases} l_i & \text{for } j = 1 \\ l_{i,j-1} + n_{i,j-1} & \text{for } j > 1. \end{cases} \tag{7.6}$$

The total number n_i of fragments transmitted in channel i is then given by

$$n_i = \sum_{j=1}^{d_i} n_{ij}. \tag{7.7}$$

Finally, we need an expression for t_i, the number of consecutive slots that have to be tapped in channel i. It must be tapped long enough to receive all fragments at least once. As l_{ij} is increasing in j, so is n_{ij}, which implies that sub-channel d_i of channel i must be tapped for the longest time, which amounts to $t_i = d_i n_{i,d_i}$.

Note that the description above only considers frequencies at which fragments are broadcast and numbers of slots that must be tapped in each channel to ensure that all fragments are received at least once. This not only leaves some freedom in composing an actual schedule, it also means that any schedule that satisfies the structure outlined above will do. In other words, given an actual schedule, a user may start tapping (the first r) channels at the start of any slot. This completes the description of FDPB.

Figure 7.3 illustrates the composition of part of a schedule for $m = 5$ channels, of which at most $r = 2$ channels may be tapped simultaneously and with offset $o = 0$. For channel $i = 5$, tapping starts at slot $s_5 = s_3 + t_3 = 1 + 4 = 5$. The first fragment tapped is $l_5 = l_4 + n_4 = 7 + 5 = 12$. The number of sub-channels in channel 5 is $d_5 = \lceil \sqrt{l_5 - s_5} \rceil = \lceil \sqrt{12 - 5} \rceil = 3$. In sub-channel 1, the lowest fragment number is $l_{5,1} = l_5 = 12$, so that this sub-channel contains $n_{5,1} = p_{5,1} = \lfloor (12-5)/3 \rfloor = 2$ fragments. For sub-channel 2, we thus have $l_{5,2} = 14$ and it contains $\lfloor (14-5)/3 \rfloor = 3$ fragments. For the last sub-channel we have $l_{5,3} = 17$, so that this sub-channel contains $n_{5,3} = \lfloor (17-5)/3 \rfloor = 4$ fragments, giving a total of $n_5 = 9$ fragments in channel 5. The number of slots that channel 5 has to be tapped is $t_5 = 3 \cdot 4 = 12$.

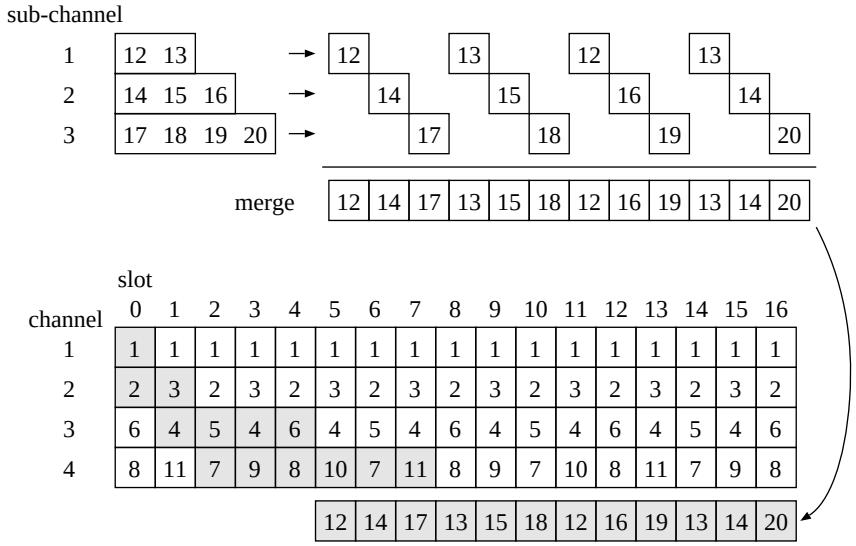


Figure 7.3. Composition of an example schedule. For further explanation, see the running text.

Table 7.1 illustrates the number $n(m,r)$ of fragments that can be realized using m channels, of which at most r channels may be tapped in parallel, using an

offset $o = 0$. Each of the series converges to a power series, with bases of about 1.75, 2.42, 2.62, and $e \approx 2.72$, for $r = 2$, 3, 4, and m, respectively.

m	$r = 2$	$r = 3$	$r = 4$	$r = m$
1	1	1	1	1
2	3	3	3	3
3	6	8	8	8
4	11	17	20	20
5	20	39	47	50
6	38	86	113	124
7	68	198	276	316
8	122	467	692	822
9	221	1102	1770	2176
10	397	2632	4547	5818
11	708	6308	11800	15646
12	1244	15192	30748	42259
13	2195	36672	80273	114420
14	3862	88710	210027	310284
15	6757	214792	549998	842209

Table 7.1. The values for $n(m,r)$ for different values of m and r, and an offset $o = 0$. The last column corresponds to having no limit on the number of channels that may be tapped in parallel.

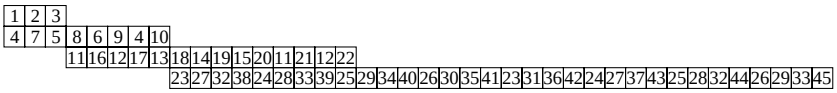
The maximum start-up latency as fraction of the total duration of the video is given by $(o+1)/n(m,r) = 1/n(m,r)$, since we have $o = 0$. Hence, for example, using $m = 10$ channels, of which at most $r = 3$ channels may be tapped simultaneously, the maximum start-up latency is $1/2632$, which amounts to less than 3 seconds for a 2-hour video. When using an offset of $o = 1$, this reduces to 1 second.

7.2 On the square-root heuristic

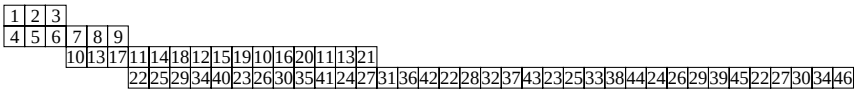
From the description of FDPB in the preceding section, it follows that the number n_i of fragments that are transmitted in channel i is a function of the number d_i of sub-channels in this channel, which can be freely chosen. In this section we focus on the problem of finding the optimal value d_i^{opt} for d_i that maximizes n_i. In particular, we substantiate by analysis the square-root heuristic, but also provide an improvement upon it.

Optimizing the number of fragments n_i in each channel i separately does not necessarily lead to a global optimum n for the total number of fragments in all channels. As an example, consider the case $(m,o,r) = (4,2,2)$. In each

channel $i = 1,2,3,4$, the optimal values d_i^{opt} are 1, 2, 2, and 4, respectively, and the corresponding numbers n_i of fragments is 3, 7, 12, and 23, leading to a total of 45 fragments. The global optimum, however, is 46 fragments, which is achieved by, for instance, using 1, 1, 3, and 5 sub-channels and 3, 6, 12, and 25 fragments in each of the channels, respectively. Figure 7.4 illustrates the schedules for each of these cases.



(a)



(b)

Figure 7.4. Two schedules for the case $(m,o,r) = (4,2,2)$. Part (a) illustrates the schedule using locally optimal values d_i^{opt}, part (b) illustrates a globally optimal schedule.

We next investigate how to express n_i directly in terms of d_i by aggregating the results of the procedural steps that involve the sub-channels.

Theorem 7.1. *The total number n_i of fragments in channel i is bounded from below as*

$$n_i \quad \geq \quad (z_i - d_i) \cdot \left(\left(1 + \frac{1}{d_i} \right)^{d_i} - 1 \right), \tag{7.8}$$

where $z_i = o + l_i - s_i$.

Proof. By using Equations 7.5 and 7.6 and that $n_{ij} = p_{ij}$, we have

$$n_{ij} \quad \geq \quad \frac{o + l_{ij} - s_i}{d_i} - 1 \quad = \quad \frac{z_i - d_i + \sum_{k=1}^{j-1} n_{ik}}{d_i}. \tag{7.9}$$

Note that for $j = 1$, the summation on the right-hand side is empty. We next define m_{ij} by the following recurrence relation:

$$m_{ij} \quad = \quad \frac{z_i - d_i + \sum_{k=1}^{j-1} m_{ik}}{d_i} \tag{7.10}$$

and first prove by induction on j that

$$n_{ij} \geq m_{ij}. \tag{7.11}$$

For $j = 1$, we have that $n_{i1} \geq (z_i - d_i)/d_i = m_{i1}$. Suppose that Equation 7.11 holds for $j = 1, 2, \ldots, j_0$. We next prove that $n_{i,j_0+1} \geq m_{i,j_0+1}$. It holds that

$$\begin{aligned} n_{i,j_0+1} &\geq (z_i - d_i + \sum_{k=1}^{j_0} n_{ik})/d_i \\ &\geq (z_i - d_i + \sum_{k=1}^{j_0} m_{ik})/d_i \\ &= m_{i,j_0+1}, \end{aligned} \tag{7.12}$$

whereby the second inequality follows from the induction hypothesis. This establishes Equation 7.11. By defining

$$m_i = \sum_{j=1}^{d_i} m_{ij},$$

we thus have, by Equations 7.7 and 7.11, that $n_i \geq m_i$. Proving that

$$m_i = (z_i - d_i) \cdot \left(\left(1 + \frac{1}{d_i} \right)^{d_i} - 1 \right)$$

then completes the proof. To this end, we will prove, again by induction on j, that

$$m_{ij} = \frac{z_i - d_i}{d_i} \left(1 + \frac{1}{d_i} \right)^{j-1}. \tag{7.13}$$

For $j = 1$, this follows directly from Equation 7.10. Suppose that Equation 7.13 holds for $j = 1, 2, \ldots, j_0$. We next prove that it also holds for $j = j_0 + 1$. We derive

$$\begin{aligned} m_{i,j_0+1} &= \frac{z_i - d_i + \sum_{k=1}^{j_0} m_{ik}}{d_i} \\ &= \frac{z_i - d_i + \sum_{k=1}^{j_0} \frac{z_i - d_i}{d_i} \left(1 + \frac{1}{d_i} \right)^{k-1}}{d_i} \\ &= \frac{z_i - d_i}{d_i} \left(1 + \frac{1}{d_i} \sum_{k=1}^{j_0} \left(1 + \frac{1}{d_i} \right)^{k-1} \right) \\ &= \frac{z_i - d_i}{d_i} \left(1 + \frac{1}{d_i} \frac{1 - \left(1 + \frac{1}{d_i} \right)^{j_0}}{1 - \left(1 + \frac{1}{d_i} \right)} \right) \end{aligned}$$

$$= \frac{z_i - d_i}{d_i}\left(1+\frac{1}{d_i}\right)^{j_0},$$

whereby on the second line we used the induction hypothesis and on the fourth line the identity

$$\sum_{i=0}^{n-1} a^i = \frac{1-a^n}{1-a}.$$

Using Equation 7.13, we now have that

$$\begin{aligned} m_i &= \sum_{j=1}^{d_i} m_{ij} \\ &= \frac{z_i - d_i}{d_i}\sum_{j=1}^{d_i}\left(1+\frac{1}{d_i}\right)^{j-1} \\ &= \frac{z_i - d_i}{d_i}\frac{1-\left(1+\frac{1}{d_i}\right)^{d_i}}{1-\left(1+\frac{1}{d_i}\right)} \\ &= (z_i - d_i)\left(\left(1+\frac{1}{d_i}\right)^{d_i} - 1\right), \end{aligned}$$

which completes the proof. □

The above theorem bounds n_i from below directly in terms of the free parameter d_i. It stands to reason that a value for d_i that maximizes the right-hand side of Equation 7.8 is a good candidate value. Hence, the next step is to maximize the right-hand of Equation 7.8.

Theorem 7.2. *For large values of z, the function* $f_z(x)$*, defined by*

$$f_z(x) = (z-x)\left(\left(1+\frac{1}{x}\right)^x - 1\right)$$

attains its maximum value at

$$x \approx \sqrt{\frac{ez}{2e-2}} \approx 0.89\sqrt{z}. \tag{7.14}$$

Proof. By straightforward differentiation with respect to x of $f_z(x)$, we obtain that

$$f_z'(x) = 1 + \left(1+\frac{1}{x}\right)^x\left[(z-x)\log\left(1+\frac{1}{x}\right) - \frac{z+1}{x+1}\right]. \tag{7.15}$$

By equating $f'_z(x)$ to zero and rearranging terms, we obtain that

$$z\left(\frac{1}{1+x}-\log\left(1+\frac{1}{x}\right)\right) \quad = \quad \frac{1}{\left(1+\frac{1}{x}\right)^x}-\frac{1}{1+x}-x\log\left(1+\frac{1}{x}\right).$$

From this equation we can see that, as for $x \geq 1$, the right-hand side is bounded, the left-hand side must be bounded as well. This means that, for increasing z, the other factor on the left-hand side decreases to 0. The latter, however, is only possible when x increases to infinity.

This establishes that, for large values of z, the positive roots of $f'_z(x)$ are large as well. To simplify Equation 7.15, we use the following approximation.

$$\left(1+\frac{1}{x}\right)^x \quad \approx \quad e.$$

Next, as we expect x to be in the order of $\sqrt{z}$, we should use the following approximation.

$$\log\left(1+\frac{1}{x}\right) \quad \approx \quad \frac{1}{x}-\frac{1}{2x^2}.$$

This simplifies the equation $f'_z(x)=0$ to

$$1+e\left((z-x)\left(\frac{1}{x}-\frac{1}{2x^2}\right)-\frac{z+1}{x+1}\right) \quad = \quad 0.$$

By straightforward calculus, we can rewrite this formula as

$$(2-2e)+\frac{2-3e}{x}+\frac{e(z+1)}{x^2}-\frac{ez}{x^3}=0$$

As the term $2-2e$ is a negative constant, and for large z, and hence large x, the terms $(2-3e)/x$ and e/x^2 vanish, we can discard these two terms. Furthermore, the term ez/x^3 vanishes, relative to the term ez/x^2. When considering the already simplified equation, this implies that the latter term is bounded and, consequently, the former vanishes in absolute terms. These observations simplify the equation to

$$\frac{ez}{x^2} \quad = \quad 2e-2,$$

which has as single positive root the value as given in Equation 7.14. As $f'_z(x)<0$ as $x\to\infty$, $f_z(x)$ attains a maximum value at this root. This completes the proof. □

Applying the results to the current context, where $z=z_i=o+l_i-s_i$, we can conclude that, for large values of $o+l_i-s_i$, the right-hand side of Equation 7.8

is maximized for $d_i \approx 0.89\sqrt{o+l_i-s_i}$. This resembles the square-root heuristic, given in Equation 7.3, save a constant.

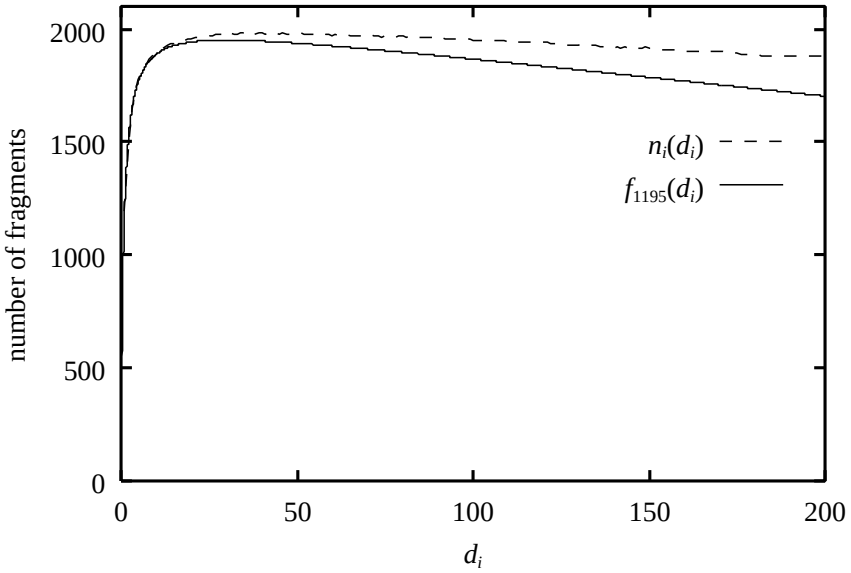


Figure 7.5. Comparison of $f_{1195}(d_i)$ with n_i as a function of d_i.

Figure 7.5 compares $f_{1195}(d_i)$ with $n_i(d_i)$ as a function of d_i. Clearly, the slope of the asymptote of $f_{1195}(d_i)$ for $d_i \to \infty$ is $1-e$, which follows easily from the definition of $f_z(x)$. The values of n_i, however, seem to approach an asymptote with slope $(1-e)/2$. This can be understood when, instead of using Equation 7.9, we use

$$n_{ij} \quad \approx \quad \frac{o+l_{ij}-s_i}{d_i} - \frac{1}{2},$$

where we have replaced the lower bound by an approximation. Assuming equality in the above equation directly leads to the function $f_{z+x/2}(x)$, whose maximum value is attained at $x \approx \sqrt{ez/(e-1)} \approx 1.26\sqrt{z}$ for large values of z. The function $f_{1195+d_i/2}(d_i)$, not shown in Figure 7.5, practically coincides with $n_i(d_i)$ for the range of values for d_i shown in the figure.

Searching for an optimal value for small values of $o+l_i-s_i$, say at most 1000, and using the 1.26-square-root heuristic for larger values, improves upon the results of using the original square-root heuristic. Table 7.2 provides a comparison with Pâris' results for $m = 15$, $o = 0$, and various values of r. The second column, headed '$\sqrt{}$', repeats Pâris' results, which are also given in the last row of Table 7.1. The third column, headed '$1.26\sqrt{}$', gives

our results. The numbers between brackets give the relative deviation in percent from the local optima given in the fourth column. These were obtained by maximizing d_i for each channel separately. Our results thus improve upon those by Pâris and approach the local optima.

r	$\sqrt{\ }$	$1.26\sqrt{\ }$	local optimum
2	6757 (16)	8027 (0.4)	8058
3	214792 (10)	237481 (0.6)	238841
4	549998 (6)	583461 (0.3)	584993
m	842209 (5)	885801 (0.1)	887124

Table 7.2. Comparison of our results with Pâris' results for $m = 15$ channels, offset $o = 0$, and various values of r, the number of channels that can be tapped in parallel. For further explanation, see the running text.

Verhaegh, Rietman & Korst [2004] suggest another improvement to Pâris' schedule by refining the way in which channels are tapped. Where, according to Equation 7.1, channel i starts being tapped when tapping in channel $i-r$ has ended, Verhaegh et al. suggest to start tapping additional sub-channels when the tapping of the appropriate, earlier sub-channels has ended. This results in a further increase in the number of fragments in which a video file can be split, and thus in a further decrease in the maximum start-up latency. Their schedule does require that switching among sub-channels in different channels can be carried out fast enough. Their approach can also benefit from a better choice of d_i.

7.3 On the asymptotic optimality of FDPB

In this section, we will prove that FDPB is asymptotically optimal in terms of the maximum start-up latency by choosing the offset large enough. We start with a result from Hollmann & Holzscherer [1991] on a lower bound on the maximum start-up latency of any fragmented broadcast schedule, for which we supply the proof to provide some insight into this lower bound.

Theorem 7.3. *Given is a fragmented broadcast schedule that uses m channels and an offset o. Then the maximum start-up latency $w_{\max}$ can be bounded from below by*

$$w_{\max} \geq \frac{1}{e^m - 1}. \tag{7.16}$$

Proof. In order to broadcast fragment i at least once every $o+i$ slots, we need at least a fraction of $1/(o+i)$ of a channel. In order to broadcast n fragments with an offset o, we need at least $\sum_{i=1}^{n} 1/(o+i) = \sum_{i=o+1}^{o+n} 1/i$ channels. Hence, it is required that

$$m \geq \sum_{i=o+1}^{o+n} 1/i.$$

Since $1/x$ is decreasing for $x > 0$, we have

$$1/i > \int_{i}^{i+1} \frac{1}{x}\,\mathrm{d}x.$$

Consequently,

$$\begin{aligned} m &> \int_{o+1}^{o+n+1} \frac{1}{x}\,\mathrm{d}x \\ &= \ln(o+n+1) - \ln(o+1) \\ &= \ln(1+n/(o+1)). \end{aligned}$$

Since the maximum waiting time $w_{\max}$ is given by $(o+1)/n$, we have

$$m > \ln(1+1/w_{\max}).$$

Since e^x is monotonically increasing in x, this implies that

$$e^m > 1+1/w_{\max}.$$

Rewriting this expression implies the required result. □

Korst & Pronk [2005] provide, with some hand waving, a proof that FDPB is asymptotically optimal. Here, we provide a rigorous proof of correctness of this theorem.

Theorem 7.4. *For the fixed-delay pagoda broadcast schedule, using m channels that may all be tapped in parallel, we have*

$$\lim_{o\to\infty} w_{\max} = \frac{1}{e^m - 1}.$$

Proof. Since we assume that all channels can be tapped simultaneously, we have $s_i = 0$ for all $i = 1, 2, \ldots, m$. Using Theorem 7.1 and Equation 7.3, we can bound n_i from below by

$$n_i \geq (z_i - \lceil\sqrt{z_i}\rceil) \cdot \left(\left(1 + \frac{1}{\lceil\sqrt{z_i}\rceil} \right)^{\lceil\sqrt{z_i}\rceil} - 1 \right), \tag{7.17}$$

with $z_i = o + l_i \to \infty$ as $o \to \infty$ for all $i = 1, 2, \ldots, m$. As the right factor approaches $e - 1$ and the left factor can be written as $z_i(1 - y_i)$, with $y_i \downarrow 0$ as $o \to \infty$, we can simplify Equation 7.17 to

$$n_i \geq (o + l_i)(e_i - 1), \tag{7.18}$$

where $e_i \uparrow e$ as $o \to \infty$. We next prove by induction on i that

$$l_i \geq (o+1) \prod_{k=1}^{i-1} e_k - o. \tag{7.19}$$

for $i = 1$, the statement clearly holds, as $l_1 = 1$. Suppose the statement holds for $i = i_0 \geq 1$. We next prove that the statement also holds for $i = i_0 + 1$. Using Equations 7.2 and 7.18, we have that

$$\begin{aligned} l_{i_0+1} &= l_{i_0} + n_{i_0} \\ &\geq l_{i_0} + (o + l_{i_0})(e_{i_0} - 1) \\ &= o(e_{i_0} - 1) + e_{i_0} l_{i_0} \\ &\geq o(e_{i_0} - 1) + e_{i_0} ((o+1) \prod_{k=1}^{i_0 - 1} e_k - o) \\ &= (o+1) \prod_{k=1}^{i_0} e_k - o, \end{aligned}$$

whereby the second inequality follows from the induction hypothesis. This establishes Equation 7.19. The total number of fragments in the m channels is given by $l_{m+1} - 1$, which is thus bounded from below by $(o+1)(\prod_{k=1}^{m} e_k - 1)$.

As the maximum start-up latency is given by $w_{\max} = (o+1)/(l_{m+1} - 1)$, we can conclude that

$$w_{\max} \leq \frac{1}{\prod_{k=1}^{m} e_k - 1}. \tag{7.20}$$

Since $e_k \uparrow e$ as $o \to \infty$ for all $k = 1, 2, \ldots, m$, the right-hand size of this inequality approaches $1/(e^m - 1)$ from above. Combining this with Theorem 7.3 then gives the desired result. □

When considering the proof above, we can conclude that the exact choice of d_i does not matter. As long as $d_i/(o + l_i) \to 0$ and $d_i \to \infty$, we have that $y_i \downarrow 0$ and

$e_i \uparrow e$, respectively, as $o \to \infty$. Hence, even choosing $d_i = \lceil \log(o + l_i) \rceil$ would result in an asymptotically optimal schedule. When considering the relative simplicity of FDPB, it is surprising that it is asymptotically optimal, in view of the proof of Theorem 7.3.

Note furthermore that the asymptotic optimality of FDPB implies that allowing more freedom in the construction of the schedule, for instance by not limiting the placement of blocks to only one channel, cannot improve upon this result. Also attempting to use globally optimal numbers of sub-channels does not result in any improvement.

8

Conclusion

Providing a video-on-demand service over a cable access network creates various challenges for the management of resources, the foremost important ones being the real-time delivery of large amounts of video data to the homes and realizing acceptable response times to support interactivity. In this context, we have discussed six problems on a variety of subjects. Two of these problems are related to the transmission of data from the cable modems at the homes to a central node in the network, called the head end. These problems relate to the response times that should be kept to a minimum. The four other problems concern the storage of video data on disk and the transmission of video data, or other real-time data, over these networks. Here, the real-time requirements for the delivery of data as well as the volume of the data calls for optimal use of the scarce resources, namely disk bandwidth and the bandwidth of the communication channel from the head end to the homes.

The first problem, addressed in Chapter 2, concerns a medium-access-control protocol, as defined in one of the relevant standards on cable access networks. This contention-based protocol is used by a cable modem just after powering up and serves to establish a first contact between the cable modem and the head end. Although the protocol resembles the well-known protocol named frame-based ALOHA, it differs from the latter in several respects. Firstly, there is a non-negligible feedback delay. This complicates the deter-

mination of the frame length. Secondly, the channel model, which describes the reception behavior by the head end of packets sent by the cable modems, does not obey the commonly-used rules. For example, two packets sent simultaneously may both be received correctly, and in case only one packet is sent, it may be received correctly, it may be received as if a collision occurred, or it may not be received at all. These differences call for a renewed analysis of optimal frame-length control.

We define a generalized channel model to reflect these differences and study the effect of this channel model on the control parameters that govern the protocol. The main findings are the following. Firstly, where the conventional protocol sets the length of the current frame in terms of the number of slots equal to the expected number N of cable modems that will transmit in this frame, using the generalized model and for large N, this number changes to βN, where the value of β depends only on the channel model parameters. Secondly, the estimation of the expected number of transmitting cable modems in the current frame, which we base on the fraction of empty slots seen in the previous frame, is characterized by a similar, albeit somewhat more complex adjustment. We show that, under some mild channel conditions and assuming that both the number of cable modems as well as the length of the previous frame are large, instead of requiring an additional constant, the logarithmic function used for the conventional channel model should be replaced by a more complex function, again dependent only on the channel parameters. Thirdly, the feedback delay impacts the minimal frame length that should be imposed to limit the signaling overhead from the head end to the cable modems: the feedback delay should not be an integer multiple of the minimal frame length.

A special case of interest is the arrival of a burst of cable modems that enter the protocol simultaneously. This event may happen after a local power outage, when multiple cable modems power up nearly simultaneously. Upon such an event, the behavior of the protocol resembles that of a Galton-board experiment, in that a high concentration of cable modems in a relatively short frame should be distributed over multiple short frames in order to observe empty slots, necessary to obtain an estimate of the number of cable modems present.

We provide simulations to support our findings and illustrate that the results are relatively insensitive to inaccurately estimated channel model parameters.

A subject for further research is to investigate how a burst of simultaneously arriving cable modems can be detected early. A possible approach is to take a closer look at the distribution of the empty slots in the previous frame.

In Chapter 3, we consider cable modems in normal operation, each of which sustains a number of connections. When a cable modem enters a request-grant procedure to transmit data to the head end on behalf of a first connection, it starts by transmitting a packet containing a request in contention with other cable modems. Requests for other connections should be temporarily queued until the request for the first connection is successfully received by the head end. An important performance characteristic is the delay that a request incurs before it is successfully received by the head end. This delay may thus include a queuing delay.

We propose to introduce multi-requests wherein multiple, say R, requests at a single modem can be merged and transmitted simultaneously as a single packet, also in contention, and analyze the reduction in request delay. This merging is such that the packet size is fixed, irrespective of the actual number of requests it contains, and such that the head end can retrieve the original requests from a multi-request. The overhead that is incurred when a multi-request is not completely used, that is, it still contains less than R requests, is offset against the possibility to add new requests to such a multi-request. After all, a multi-request may have to be transmitted repeatedly before it is successfully received by the head end, providing opportunities to add requests before a retransmission is done.

We use results on the gated machine-repair model that can be used for modeling the multi-request delay and extend the analysis to obtain results on the delay of the individual requests. In particular, we obtain an approximation of the mean delay of an individual request under high loads, and show that the expected request delay decreases only if the the size of a packet containing a multi-request with up to R requests is shorter than R packets, each containing only a single request. Under the latter assumption, simulation results indicate that, under low loads, the request delay increases, as could have been expected, but under higher loads, the request delay decreases as R increases. Under these loads, the results also show a good match with the analytical results.

In the next chapter, Chapter 4, we consider the problem of fairly sharing a resource, such as a transmission link, among a number of heterogeneous users. Each user is characterized by its reserved share of the resource, and the shares add up to at most the capacity of this resource. Besides guaranteeing, for each user, its share of the resource, the aim is to minimize the jitter that users incur in receiving their share.

We first report on an error in a publication, wherein it is stated that the scheduling algorithm the authors propose bounds the worst-case absolute jitter by a non-trivial bound, by providing a simple counterexample. We then

propose a scheduling algorithm called relaxed-earliest-deadline-first (R-EDF), of which we prove that it does obey this bound. We furthermore prove that this bound cannot be improved upon, implying that R-EDF is optimal in this respect. The algorithm is a non-preemptive version of the well-known earliest-deadline-first algorithm for preemptively scheduling a set of periodic tasks. The computational complexity of the proposed algorithm is linear in the number of users.

We also discuss its operation in a dynamic environment, wherein users may be admitted service and may depart again. One of the major findings is that, after a departure, there should be a so-called dead time, during which no admissions are allowed. We provide an upper bound on the length of this dead time, namely the time until the scheduler becomes idle for the first time after a departure, but we conjecture that this can be improved. We consider this a subject for further research.

Chapter 5 concerns the storage on and retrieval from disk of a variable-bit-rate video stream. To offer flexibility in the way that video data is retrieved from disk in terms of, e.g., bit rate or starting position, special provisions are necessary while storing the data on disk. In particular, we assume that relatively small, fixed-size parts of the video data to be stored, should be stored twice.

We integrate this requirement into an existing disk scheduling algorithm, called triple buffering, and prove that, under suitable assumptions, the resulting algorithm is safe. By safeness we mean that it prevents the buffers, used for temporarily storing incoming data before it is written to disk or outgoing data before it is transmitted, from under- and overflowing. The required increase in buffer size is at most the size of a single part that should be written twice.

A subject for further research is how to integrate the aforementioned requirement into an alternative disk scheduling algorithm, called dual sweep [Korst & Pronk, 2005], which compares favorably with triple buffering in terms of the required buffer sizes.

In Chapter 6 we consider the problem of storing a given set of video files on a multi-zone disk. Each file is characterized by a set of parameters such as its size, its bit rate required for streaming the file into a network, and its popularity. The aim is to minimize the expected resource usage, where the resource is the disk load. Minimizing this implies that more streams or streams with a higher bit rate can be provided service simultaneously.

We show that this problem is NP-complete in the strong sense. In the special case that all files have the same size, we show that the problem is in P. We provide a heuristic solution with a computational complexity of $\mathcal{O}(m \log m)$,

where m is the number of files to be stored. We show that the worst-case performance ratio is given by the ratio of the maximum and minimum bit rate of the disk. Although this implies that the proposed algorithm is one of the worst in terms of its performance ratio, we argue that it will work quite well in practice if the file sizes are not too large. We further substantiate this by simulations.

We compare the approach followed with the well-known technique of track pairing and show, by analysis as well as by simulation, that our approach outperforms track pairing if the popularity distribution of the files is sufficiently skewed with respect to a uniform distribution.

The heuristic solution suggests how to apply the results in an online setting, where files are occasionally added and deleted and provide an outline of how this may be done.

Our last subject, discussed in Chapter 7, is on near-video-on-demand and concerns a well-known video broadcasting algorithm, called fixed-delay pagoda broadcasting. In this algorithm, a constant-bit-rate video file is subdivided into a number n of fixed-size fragments. Each fragment is periodically broadcast at its own frequency on one of a fixed number of channels, each of the same bit rate of the video file at hand, where the frequencies generally become smaller as the fragments occur later in the file. The idea behind this approach is that, once a user starts viewing a video from the beginning to the end, he will need fragments near the start of the video sooner than those near the end. The aim is to maximize n, as this provides a measure for the waiting time before the user can start watching the video.

We solve two still open issues. The first relates to the so-called square-root heuristic that is used to determine the number of sub-channels within a channel. We substantiate this heuristic and indicate that it can be slightly improved. The second concerns the optimality of the broadcast schedule. In particular, we show that the schedule is asymptotically optimal in terms of the maximum waiting time. The asymptotic behavior is defined as the limit to infinity of the period with which the first fragment is broadcast.

A

Related output

Books and Book contributions

KORST, J., AND V. PRONK [2005]. *Multimedia Storage and Retrieval: An Algorithmic Approach*, Wiley & Sons, Ltd.

PRONK, V. [2003]. Storage of VBR video content on a multi-zone recording disk based on resource usage, *Algorithms in Ambient Intelligence (Eds. W. Verhaegh, E. Aarts, and J. Korst)*, Kluwer.

External Publications

AARTS, E., E. DEN BOEF, J. KORST, V. PRONK, W. VERHAEGH, AND C. WÜST [2002]. Adaptive scheduling and resource management in ambient intelligence, *PT Embedded Systems Research Dossier*, 12−15.

DENTENEER, T.J.J., AND V. PRONK [1998]. Traffic models for telecommunication, *Proceedings of the 13th COMPSTAT Conference of the International Association for Statistical Computing*, IASC'98, Bristol, England, August 24−28.

DENTENEER, T.J.J., AND V. PRONK [1999]. WWW traffic modelling for HFC networks, *Proceedings of the 48th Annual NCTA Convention and International Exposium*, CABLE'99, Chicago, Illinois, June 13−16, 145−152.

DENTENEER, T.J.J., AND V. PRONK [2001]. On the number of contenders in a contention tree, *Proceedings of the 14th ITC Specialists Seminar on Access Networks and Systems*, Girona, Spain, April 25−27, 105−112.

DENTENEER, T.J.J., V. PRONK, J.M. GRIFFITHS, AND L.G. CUTHBERT [2000]. Impact of the resource needed for renegotiating ATM rates, *Computing Networks 34*, 211−225.

DRIEL, C.-J.L. VAN, P.A.M. VAN GRINSVEN, V. PRONK, AND W.A.M. SNIJDERS [1997]. The (r)evolution of access networks for the information super-highway, *IEEE Communications Magazine 35*, 2−10.

DRIEL, C.-J.L. VAN, P.A.M. VAN GRINSVEN, W.A.M. SNIJDERS, AND V. PRONK [1997]. Drivers of the evolution of the broadband networks, *Proceedings of the European Telecommunications Congress*, FITCE'97, Thessaloniki Greece, September 22−27, 15−20.

GRINSVEN, P.A.M. VAN, S.-B. COLAK, A. JANSEN VAN DOORN, V. PRONK, F. SNIJDERS [2001]. Reconfigurable active nodes on HFC/CATV networks with hybrid fiber-wireless ad-hoc links of mesh topology, *Proceedings of the SCTE'01*.

HEKSTRA-NOWACKA, E.B., V. PRONK, L. TOLHUIZEN, AND T.J.J. DENTENEER [1999]. Bandwidth allocation in HFC networks *Proceedings of the Philips Workshop on Scheduling and Resource Management*, SCHARM'99, Eindhoven, The Netherlands, December 8−9, 129−137.

KORST, J., AND V. PRONK [1999]. Bit-rate smoothing algorithms for prerecorded VBR video, *Proceedings of the Philips Workshop on Scheduling and Resource Management*, SCHARM'99, Eindhoven, The Netherlands, December 8−9, 29−38.

KORST, J., V. PRONK, E.H.L. AARTS, AND F. LAMERIKX [1995]. Periodic scheduling in a multimedia server, *Proceedings of the INRIA/IEEE Symposium on Emerging Technologies and Factory Automation*, ETFA'95, Paris, October 10−13, Vol. 1, 205−216.

KORST, J., V. PRONK, AND P. COUMANS [1997]. Disk scheduling for variable-rate data streams, *Proceedings of the European Workshop on Interactive Distributed Multimedia Systems and Telecommunication Services*, IDMS'97, Darmstadt, September 10−12, Lecture Notes in Computer Science, LNCS 1309, 119−132.

KORST, J., V. PRONK, P. COUMANS, G. VAN DOREN, AND E. AARTS [1998]. Comparing disk scheduling algorithms for VBR data streams, *Computer Communications 21*, 1328−1343.

PRONK, V. [2002]. Disk storage of VBR video content based on resource usage, *Proceedings of the 1st Philips Symposium on Intelligent Algorithms*, SOIA'02, Eindhoven, The Netherlands, December 11−12, 245−257.

PRONK, V., P.A.M. VAN GRINSVEN, AND C.-J.L. VAN DRIEL [1998]. A performance analysis of the bit-map access protocol for shared-medium networks, *Proceedings of the International Zurich Seminar*, IZS'98, Zurich, Switzerland, February 17−19, 69−73.

PRONK, V., AND J. KORST [2001a]. Comment on "Carry-Over Round Robin: A Simple Cell Scheduling Mechanism for ATM Networks", *IEEE/ACM Transactions on Networking 9:3*, 373−374.

PRONK, V., AND J. KORST [2001b]. Scheduling ATM cells using the R-EDF algorithm, *Proceedings of the Philips Workshop on Scheduling and Resource Management*, SCHARM'01, Eindhoven, The Netherlands, June 28−29, 187−196.

PRONK, V., AND J. KORST [2007]. Fair resource sharing using the R-EDF scheduling algorithm, *accepted for publication in Springer Real-Time Systems Journal*.

PRONK, V., AND R. RIETMAN [2002]. Medium access control using request merging, *Proceedings of the 1st Philips Symposium on Intelligent Algorithms*, SOIA'02, Eindhoven, The Netherlands, December 11−12, 133−142.

PRONK, V., AND L. TOLHUIZEN [2001]. Medium access control for unregistered cable modems, *Proceedings of the Philips Workshop on Scheduling and Resource Management*, SCHARM'01, Eindhoven, The Netherlands, June 28−29, 169−178.

SINHA, A.N., T.J.J. DENTENEER, AND V. PRONK [1996]. Traffic Contract: key to orderly use, provisioning, and charging of ATM services, *Proceedings of the IEE Colloquium on Charging for ATM*, London, England, November 12, 8/1−8/6.

TOLHUIZEN, L., V. PRONK, E.B. HEKSTRA-NOWACKA, AND T.J.J. DENTENEER [1999]. Scheduling for HFC networks, *Proceedings of the Philips Workshop on Scheduling and Resource Management*, SCHARM'99, Eindhoven, The Netherlands, December 8−9, 139−146.

Internal Publications

HEKSTRA-NOWACKA, E.B., AND V. PRONK [1999]. Piggybacking in DVB/DAVIC-compliant HFC networks, *Philips Technical Note Nr. 394/99*.

PRONK, V. [2000]. Upstream channel bandwidth allocation in a DVB-compliant HFC network, *Philips Technical Note 2000/429*.

PRONK, V., T.J.J. DENTENEER, J. GRIFFITHS, L. CUTHBERT, D. BOTVICHM AND J. KARLSSON [1997]. Source modelling and its applications for ATM networks, *Philips Report Nr. 7012*

PRONK, V., E.B. HEKSTRA-NOWACKA, L. TOLHUIZEN, AND T.J.J. DENTENEER [1999]. Description and performance analysis of the DVB/DAVIC MAC protocol for HFC networks, *Philips Report 7105*.

PRONK, V., AND M.J.M. DE JONG [1998]. Multi-standard simulation platform for hybrid fiber/coax networks; I standards survey, IIU basic architecture, *Philips Technical Note 179/98*.

PRONK, V., AND J. KORST [2002]. On fair queuing and scheduling periodic tasks, *Philips Technical Note 2002/218*.

PRONK, V., AND L. TOLHUIZEN [2000]. Ranging in DVB/DAVIC II, *Philips Technical Note 2000/317*.

PRONK, V., L. TOLHUIZEN, AND P.A.M. VAN GRINSVEN [2000]. Optimal ranging in DVB/DAVIC, *Philips Technical Note Nr. TN 2000/113*.

SINHA, A.N., T.J.J. DENTENEER, V. PRONK, AND H.G.J. THEUNIS [1996]. Video source modelling for ATM networks, *Philips Report Nr. 6943*.

TOLHUIZEN, L., AND V. PRONK [2001]. Downstream signalling in an HFC network: on the Reservation Grant Message in DVB/DAVIC, *Philips Technical Note Nr. TN 2001/174*.

Patents and patent applications

DENTENEER, T.J.J., S.P.P. PRONK, E.B. HEKSTRA-NOWACKA, L.M.G.M. TOLHUIZEN [2001]. Method of and system for transmitting a plurality of messages, *U.S. Patent 7,251,251*, granted July 31, 2007.

KORST, J.H.M., AND S.P.P. PRONK [1998]. Method of and system for interleaving real-time files with different periods, *U.S. Patent 5,848,437*, granted December 8, 1998.

KORST, J.H.M., AND S.P.P. PRONK [1999]. Method and system for reading data for a number of users, *U.S. Patent 5,950,015*, granted September 7, 1999.

KORST, J.H.M., E. LAWERMAN, S.P.P. PRONK, AND G. VAN DOREN [2000]. Method and system for supplying streams of data having identical maximum consumption rate in a storage medium, *U.S. Patent 6,138,221*, granted October 24, 2000.

PRONK, S.P.P. [2003]. Method for storing programs on a disk, *Patent application*, submitted January 22, 2003.

PRONK, S.P.P., AND R. RIETMAN [2002]. Medium access control using request merging, *Patent application*, submitted May 2, 2002.

PRONK, S.P.P., AND L.M.G.M. TOLHUIZEN [2006]. Contention resolution protocol, *U.S. Patent 7,009,993*, granted March 7, 2006.

PRONK, S.P.P., L.M.G.M. TOLHUIZEN, AND E.B. HEKSTRA-NOWACKA [2005]. Communication network having minimized roundtrip contention delay, *U.S. Patent 6,967,968*, granted November 22, 2005.

SINHA, A.N., T.J.J. DENTENEER, S.P.P. PRONK, AND H.G.J. THEUNIS [1999]. Encoded digital video transmission system, *U.S. Patent 5,990,945*, granted November 23, 1999.

Bibliography

AERTS, J. [2003]. *Random redundant storage for video on demand*, Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

ABRAMSON, N. [1970]. The ALOHA system – another alternative for computer communications, *Proceedings of the Fall Joint Computer Conference 37*, AFIPS'70, Montvale, NJ, 281–285.

AL-MARRI, J., AND S. GHANDEHARIZADEH [1998]. An evaluation of alternative disk scheduling techniques in support of variable bit rate continuous data, *Proceedings of the 6th International Conference on Extending Database Technology*, EDBT'98, Valencia, Spain, March 23–27, 231–245.

BENNETT, J.C.R., AND H. ZHANG [1996]. WF^2Q: worst-case fair weighted fair queuing, *Proceedings of the 15th Annual Joint Conference of the IEEE Computer and Communications Societies*, IEEE INFOCOM'96, March 24–28, San Francisco, California, 120–128.

BERTSEKAS, D.P., AND R.G. GALLAGER [1992]. *Data Networks*, Prentice Hall.

BIERSACK, E., F. THIESSE, AND C. BERNHARDT [1996]. Constant data length retrieval for video servers with VBR streams, *Proceedings of the 3rd IEEE International Conference on Multimedia Computing and Systems* ICMCS'96, Hiroshima, Japan, June 17–21, 151–155.

BIRK, Y. [1995a]. Track pairing: A novel data layout for VoD servers with multi-zone recording disks, *Proceedings of the 2nd IEEE International Conference on Multimedia Computing and Systems*, ICMCS'95, Washington, DC, May 15–18, 248–255.

BIRK, Y. [1995b]. Deterministic load-balancing schemes for disk-based video-on-demand storage servers, *Proceedings of the 14th IEEE Symposium on Mass Storage Systems*, MSS'95, Monterey, CA, September 11–14, 17–25.

BISDIKIAN, C., K. MARUYAMA, D. SEIDMAN, AND D. SERPRANOS [1996]. Cable access beyond the hype: on residential broadband data services over HFC networks, *IEEE Communications Magazine 34*, 128–135.

BOLOSKY, W.J., J.S. BARRERA, R.P. DRAVES, R.P. FITZGERALD, G.A. GIBSON, M.B. JONES, S.P. LEVI, N.P. MYHRVOLD, R.F. RASHID [1996]. The Tiger video file

server, *Proceedings of the 6th International Workshop on Network and Operating System Support for Digital Audio and Video*, NOSSDAV'96, Zushi, Japan, April 23−26, 212−223.

BORST, S.C. [1996]. *Polling Systems*, CWI Tracts, Amsterdam.

BOXMA, O., T.J.J. DENTENEER, AND J. RESING [2002]. Some models for contention resolution in cable networks, *Networking 2002 Workshops, Lecture Notes in Computer Science 2345*, 117−128, Springer, 2002.

BRESLAU, L., P. CAO, L. FAN, G. PHILLIPS, AND S. SHENKER [1999]. Web caching and Zipf-like distributions: Evidence and implications, *Proceedings of the 18th Annual Joint Conference IEEE Computer and Communications Societies*, IEEE INFOCOM'99, New York, NY, March 21−25, 126−134.

BRUCKER, P. [2001]. *Scheduling Algorithms*, 3rd edition, Springer.

BURTON, W. [1976]. A buddy system variation for disk storage allocation, *Communications of the ACM 19:7*, 416−417.

CABLE TELEVISION LABORATORIES, INC. [2001]. Data-over-cable service interface specifications (DOCSIS), Radio frequency interface specification, SP-RFIv2.0−W02−011024.

CABRERA, L., AND D.D.E. LONG [1991]. Swift: Using distributed disk striping to provide high I/O data rates, *Computing Systems 4:4*, 405−436.

CAPETANAKIS, J.I. [1979]. Tree algorithms for packet broadcast channels, *IEEE Transactions on Information Theory 25:5*, 505−515.

CHANG, E., AND H. GARCIA-MOLINA [1996]. Reducing initial latency in a multimedia storage system, *Proceedings of the International Workshop on Multimedia Database Management Systems*, IW-MMDBMS'96, Blue Mountain Lake, NY, August 14−16, 2−11.

CHEN, M.-J., AND C.-C. WU [2003]. A zone-based data placement and retrieval scheme for video-on-demand applications regardless of video popularity, *IEICE Transactions on Communications E86−B:10*, 3094−3102.

CHENG, A.M.K. [2002]. *Real-Time Systems: Scheduling, Analysis, and Verification*, Wiley.

DEBEY, H.C. [1989]. Program transmission optimisation, *United States Patent 5,421,031*.

DEMERS, A., S. KESHAV, AND S. SHENKER [1989]. Analysis and simulation of a fair queuing algorithm, *Proceedings of the ACM Symposium on Communications Architectures & Protocols*, ACM SIGCOMM'89, September 19−22, Austin, Texas, 1−12.

DENGLER, J., C. BERNHARDT, AND E. BIERSACK [1996]. Deterministic admission control strategies in video servers with variable bit rate streams,

Proceedings of the 3rd International Workshop on Interactive Distributed Multimedia Systems and Services, IDMS'96, Berlin, March 4−6, 245−264.

DENTENEER, T.J.J., V. PRONK, E.B. HEKSTRA-NOWACKA, L.M.G.M. TOLHUIZEN [2003]. Method of and system for transmitting a plurality of messages, *United States Patent application 20030091060*.

DENTENEER, T.J.J. [2005]. *Data transfer in cable networks: Delay models for multiaccess with contention trees*, Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

DENTENEER, T.J.J., AND V. PRONK [2001]. On the number of contenders in a contention tree, *Proceedings of the 14th ITC Specialists Seminar on Access Networks and Systems*, Girona, Spain, April 25−27, 105−112.

DIGITAL VIDEO BROADCASTING (DVB) [1998]. DVB interaction channel for cable TV distribution systems (CATV), *ETS 300 800*.

VAN DRIEL, C.-J.L., P.A.M. VAN GRINSVEN, V. PRONK, AND W.A.M. SNIJDERS [1997]. The (r)evolution of access networks for the information super-highway, *IEEE Communications Magazine 35*, 2−10.

DUTTA-ROY, A. [1999]. Cable, it's not just for TV, *IEEE Spectrum 35*, 53−59.

FENG, W.-C., AND J. REXFORD [1999]. Performance evaluation of smoothing algorithms for transmitting prerecorded variable-bit-rate video, *IEEE Transactions on Multimedia 1:3*, 302−313.

FREEDMAN, C.S., AND D.J. DEWITT [1995]. The SPIFFI scalable video-on-demand system, *Proceedings of the ACM SIGMOD International Conference on Management of Data*, SIGMOD'95, San Jose, CA, May 22−25, *SIGMOD Record 24:2*, 352−363.

GAREY, M.R., AND D.S. JOHNSON [1979]. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman.

GEMMELL, D.J., H.M. VIN, D.D. KANDLUR, P.V. RANGAN, AND L.A. ROWE [1995]. Multimedia storage servers: A tutorial, *IEEE Computer 28:5*, 40−49.

GHANDEHARIZADEH, S., D.J. IERARDI, D. KIM, AND R. ZIMMERMANN [1996]. Placement of data in multi-zone disk drives, *Proceedings of the 2nd International Baltic Workshop on Databases and Information Systems*, BalticDB'96, Tallin, Estonia.

GHANDEHARIZADEH, S., S.H. KIM, AND C. SHAHABI [1995]. On configuring a single disk continuous media server, *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling o f Computer Systems*, SIGMETRICS'95, Ottawa, Canada, May 15−19, *Performance Evaluation Review 23:1*, 37−46.

GHANDEHARIZADEH, S., S.H. KIM, C. SHAHABI, AND R. ZIMMERMANN [1996]. Placement of continuous media in multi-zone disks, in: S. Chung (Ed.), *Multimedia Information Storage and Management*, Chapter 2, 23−59, Kluwer.

GOLESTANI, S.J. [1994]. A self-clocked fair queuing scheme for broadband applications, *Proceedings of the 13th Annual Joint Conference of the IEEE Computer and Communications Societies*, IEEE INFOCOM'94, June 12−16, 1994, Toronto, Ontario, Canada, 636−646.

GOLMIE, N., Y. SANTILLAN, AND D.H. SU [1999]. A review of contention resolution algorithms for the IEEE 802.14 networks, *IEEE Communications Surveys 2*, 2−12.

GRIMMETT, G.R., AND D.R. STIRZAKER [2001]. *Probability and Random Processes, third edition*, Oxford University Press, Oxford.

GRIWODZ, C., M. BÄR, AND L.C. WOLF [1997]. Long-term movie popularity models in video-on-demand systems: Or the life of an on-demand movie, *Proceedings of the 5th ACM International Conference on Multimedia*, MM'97, Seattle, WA, November 11−13, 349−357

HAJEK, B., N.B. LIKHANOV, AND B.S. TSYBAKOV [1994]. On the delay in a multiple-access system with large propagation delay, *IEEE Transactions on Information Theory 40:4*, 1158−1166.

HASKEL, B.G., A. PURI, AND A.N. NETRAVALI [1997]. *Digital Video: An Introduction to MPEG-2*, Digital multimedia standards series, Chapman & Hall.

HEKSTRA-NOWACKA, E.B. [2000]. Fixed Rate Scheduling – Implementation, *Philips Research Report 7149*.

HELTZER, S.R., J.M. MENON, AND M.F. MITOMA [1993]. Logical data tracks extending among a plurality of zones of physical tracks of one or more disk devices, *United States Patent 5,202,799*.

HOLLMANN, H.D.L., AND C.D. HOLZSCHERER [1991]. Information on demand with short access times: Preliminary investigations, *Philips Research Technical Note 109/91*.

HUANG, Y.-M., AND S.-L. TSAO [1997]. An efficient data placement and retrieval scheme of zoned disks to support interactive playout for video servers, *IEEE Transactions on Consumer Electronics 43:1*, 69−79.

HUANG, Y.-M., AND S.-L. TSAO [1999]. An efficient data layout scheme for multi-disks continuous media servers, *Multimedia Tools and Applications 9:2*, 147−166.

IEEE WORKING GROUP [2000]. http://www.media.mit.edu/physics/pedagogy/fab/fab_2002/help_pages/networking_resources/Protocols/home.knology.net/ieee80214/index.html

JANSSEN, A.J.E.M., AND M.J.M. DE JONG [2000]. Analysis of contention tree algorithms, *IEEE Transactions on Information Theory 46:6*, 2163−2172.

KAMEDA, T., AND R. SUN [2004]. A survey of VOD broadcasting schemes, *to be published*.

KANG, J., AND H.Y. YEOM [1999]. Placement of VBR video data on MZR disks, *Proceedings of the 9th International Workshop on Network and Operating System Support for Digital Audio and Video*, NOSSDAV'99, Basking Ridge, NJ, June 23−25, 231−236.

KANG, S., AND H.Y. YEOM [2003]. Storing continuous media objects to multi-zone recording disks using multirate smoothing technique, *IEEE Transactions on Multimedia 5:3*, 473−482.

KARSTEN, M. [2006]. SI-WF^{2}Q: WF^{2}Q approximation with small constant execution overhead, *Proceedings of the 25th Annual Joint Conference of the IEEE Computer and Communications Societies*, IEEE INFOCOM'06, April 23−29, Barcelona, Spain.

KATEVENIS, M., S. SIDIROPOULOS, S., AND C. COURCOUBETIS [1991]. Weighted round-robin cell multiplexing in a general-purpose ATM switch chip, *IEEE Journal on Selected Areas in Communication 9:8*, 1265−1279.

KIM, J.-W., Y.-U LHO, AND K.-D. CHUNG [1997]. An effective video block placement scheme on VOD server based on multi-zone recording disks, *Proceedings of the 4th IEEE International Conference on Multimedia Computing and Systems*, ICMCS'97, Ottawa, Canada, June 3−6, 29−36.

KIM, J.-W., H.-R. LIM, Y.-J. KIM, AND K.-D. CHUNG [1997]. A data placement strategy on MZR for VoD servers, *Proceedings of the International Conference on Parallel and Distributed Systems*, ICPADS'97, Seoul, South Korea, December 11−13, 506−513.

KLEIN, M.H., T. RALYA, B. POLLAK, AND R. OBENZA [1993]. *Practitioner's Handbook for Real-Time Analysis: Guide to Rate Monotonic Analysis for Real-Time Systems*, Kluwer.

KNIGHTLY, E.W., D.E. WREGE, J. LIEBEHERR, AND H. ZHANG [1995]. Fundamental limits and trade-offs of providing deterministic guarantees to VBR video traffic, *Proceedings of the ACM Sigmetrics Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS'95, Ottawa, Canada, May 15−19, *Performance Evaluation Review 23:1*, 98−107.

KNUTH, D.E. [1969]. *The Art of Computer Programming. Volume I, Fundamental Algorithms*, 442−445, Addison Wesley.

KOCH, P.D.L. [1987]. Disk file allocation based on the Buddy system, *ACM Transactions on Computer Systems 4:5*, 352−370.

KORST, J., AND V. PRONK [1996]. Storing continuous-media data on a compact disc, *Multimedia Systems 4:4*, 187−196.

KORST, J., AND V. PRONK [2005]. *Multimedia Storage and Retrieval: An Algorithmic Approach*, Wiley & Sons, Ltd.

VAN KREIJ, A.J. [2001]. *Near video on demand broadcasting strategies*, M.Sc. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

KUNZ, A., AND C. STEPPING [2003]. Overview and implementation of scheduling algorithms for wireless environments, *Proceedings of the 5th European Personal Mobile Communications Conference*, EPMCC'03, April 22−25, Glasgow, England, 441−446.

KUO, W.-K., S. KUMAR, AND C.-C.J. KUO [2003]. Improved priority access, bandwidth allocation and traffic scheduling for DOCSIS cable networks, *IEEE Transactions on Broadcasting 49:4*, 371−382.

KWAAITAAL, J.J.B. [1999]. *A Multi-standard simulation platform for hybrid fiber/coax networks*, Graduate Report, Eindhoven University of Technology, Eindhoven, The Netherlands.

LAWERMAN, E. [1995]. System with data repetition between logically sucessive clusters, *United States Patent 5,890,168*.

LAWLER, E.L., J.K. LENSTRA, A.H.G. RINNOOY KAN, AND D.B. SHMOYS [1993]. Sequencing and scheduling: Algorithms and complexity, in: S.C. Graves, A.H.G. Rinnooy Kan, and P. Zipkin (Eds.), *Handbooks in Operations Research and Management Science*, Volume 4: Logistics of Production and Inventory, 445−522, North-Holland.

VAN LEEUWAARDEN, J. [2005]. *Queueing Models for Cable Access Networks*, Ph.D. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

VAN LEEUWAARDEN, J., T.J.J. DENTENEER, AND J. RESING [2006]. A discrete-time queueing model with periodically scheduled arrival and departure slots, *Performance Analysis 63*, 278−294.

LEGALL, D.J. [1991]. MPEG: A video compression standard for multimedia applications, *Communications of the ACM 34:4*, 46−58.

LHO, Y.-U., AND K.-D. CHUNG [1998]. Performance analysis and evaluation of allocating subbanded video data blocks on MZR disk arrays, *Proceedings of the Advanced Simulation Technologies Conference*, ASTC'98, Boston, MA, April 5−9, 335−340.

LIAO, W., AND H.-J. JU [2004]. Adaptive slot allocation in DOCSIS-based CATV networks, *IEEE Transactions on Multimedia 6:3*, 479–488.

LIN, Y.-D., W.-M. YIN, AND C.-Y. HUANG [2000]. An investigation into HFC MAC protocols: Mechanisms, implementation, and research issues, *IEEE Communications Surveys and Tutorials 3:3*.

LIU, C.L., AND J.W. LAYLAND [1973]. Scheduling algorithms for multiprogramming in a hard-real-time environment, *Journal of the ACM 20:1*, 46–61.

LIU, J.W.S. [2000]. *Real-Time Systems*, Prentice Hall.

MATHYS, P., AND P. FLAJOLET [1985]. Q-ary collision resolution algorithms in random-access systems with free or blocked channel access, *IEEE Transaction on Information Theory 31:2*, 217–243.

MATSUFURU, N., AND R. AIBARA [1999]. Efficient fair queuing for ATM networks using uniform round robin, *Proceedings of the 18th Annual Joint Conference of the IEEE Computer and Communications Societies*, IEEE INFOCOM'99, March 21–25, New York, New Jersey, 389–397.

MCNS HOLDINGS [1999]. DOCSIS: Data-over-cable service interface specification, *Public Report SP-RFIv2.0–W02–011024*.

MICHIELS, W. [1999]. *Block placement on multi-zone disks*, M.Sc. Thesis, Eindhoven University of Technology, Eindhoven, The Netherlands.

MICHIELS, W., AND J. KORST [2001]. Min-max subsequence problems in multi-zone disk recording, *Journal of Scheduling 4:5*, 271–283.

ÖZDEN, B., R. RASTOGI, AND A. SILBERSCHATZ [1996]. On the design of a low-cost video-on-demand storage system, *ACM Multimedia Systems 4*, 40–54.

PALMOWSKI, Z., S. SCHLEGEL, AND O. BOXMA [2003]. A tandem queue with a gate mechanism, *Queueing Systems 43*, 349–364.

PAREKH, A.K., AND R.G. GALLAGER [1993]. A generalized processor sharing approach to flow control in integrated services networks: The single-node case, *IEEE/ACM Transactions on Networking 1:3*, 344–357.

PÂRIS, J.-F. [2001]. A fixed-delay broadcasting protocol for video-on-demand, *Proceedings of the 10th International Conference on Computer Communications and Networks*, ICCCN'01, Scottsdale, AZ, October 15–17, 418–423.

PARK, Y.-S., J.-W. KIM, AND K.-D. CHUNG [1999]. A continuous media placement using B-ZBSR on heterogeneous MZR disk array, *Proceedings of the International Workshop on Parallel Processing*, ICPP'99 Workshop, Wakamatsu, Japan, September 21–24, 482–487.

PINEDO, M. [2001]. *Scheduling: Theory, Algorithms, and Systems*, 2nd edition, Prentice Hall.

PRONK, V. [2000]. Upstream channel bandwidth allocation in a DVB-compliant HFC network, *Philips Research Technical Note 2000/429*.

PRONK, V. [2002]. Disk storage of VBR video content based on resource usage, *Proceedings of the 1st Philips Symposium on Intelligent Algorithms*, SOIA'02, Eindhoven, The Netherlands, December 11−12, 245−257.

PRONK, V. [2003]. Storage of VBR video content on a multi-zone recording disk based on resource usage, *Algorithms in Ambient Intelligence (Eds. W. Verhaegh, E. Aarts, and J. Korst)*, Kluwer, 2003.

PRONK, V., E.B. HEKSTRA-NOWACKA, L.M.G. TOLHUIZEN, AND T.J.J. DENTENEER [1999]. Description and performance analysis of the DVB/DAVIC MAC protocol for HFC networks, *Philips Research Report 7105*.

PRONK, V., AND M.J.M. DE JONG [1998]. Multi-standard simulation platform for hybrid fiber/coax networks; I standards survey, IIU basic architecture, *Philips Research Technical Note 179/98*.

PRONK, V., AND J. KORST [2001]. Comment on "Carry-Over Round Robin: A Simple Cell Scheduling Mechanism for ATM Networks", *IEEE/ACM Transactions on Networking 9:3*, 373−374.

PRONK, V., AND J. KORST [2007]. Fair resource sharing using the R-EDF scheduling algorithm, *accepted for publication in Springer Real-Time Systems Journal*.

PRONK, V., AND J. KORST [2002]. On fair queuing and scheduling periodic tasks, *Philips Research Technical Note 2002/218*.

PRONK, V., AND R. RIETMAN [2002]. Medium access control using request merging, *Proceedings of the 1st Philips Symposium on Intelligent Algorithms*, SOIA'02, Eindhoven, The Netherlands, December 11−12, 133−142.

PRONK, V., AND L. TOLHUIZEN [2000]. Ranging in DVB/DAVIC II, *Philips Research Technical Note 2000/317*.

PRONK, V., AND L. TOLHUIZEN [2001]. Medium access control for unregistered cable modems, *Proceedings of the Philips Workshop on Scheduling and Resource Management*, SCHARM'01, Eindhoven, The Netherlands, June 28−29, 169−178.

REXFORD, J., AND D.F. TOWSLEY [1999]. Smoothing variable-bit-rate video in an internetwork, *IEEE/ACM Transactions on Networking 7:2*, 202−215.

ROBERTS, L.G. [1975]. ALOHA packet system with and without slots and capture, *ACM SIGCOMM Computer Communication Review 5:2*, 28−42.

RUEMMLER, C., AND J. WILKES [1994]. An introduction to disk drive modeling, *IEEE Computer 27:3*, 17–29.

SAHA, D., S. MUKHERJEE, AND S.K. TRIPATHI [1998]. Carry-over round robin: A simple cell scheduling mechanism for ATM networks, *IEEE/ACM Transactions on Networking 6:6*, 779–796.

SALA, D. [1998]. *Design and Evaluation of MAC Protocols for Hybrid Fiber/Coaxial Systems*, Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA.

SALA, D., D. HARTMAN, AND J.O. LIMB [1996]. Comparison of algorithms for station registration on power-up in an HFC network, *IEEE 802.14 Meeting*, Doc. nr. 96–012, Boulder, Colorado, January 1996.

SCHOUTE, F.C. [1983]. Dynamic frame length ALOHA, *IEEE Transactions on Communications 31:4*, 565–568.

SDRALIA, V., P. TZEREFOS, AND C. SMYTHE [2001]. Recovery analysis of the DOCSIS protocol after service disruption, *IEEE Transactions on Broadcasting, 47:4*, 377–385.

SHENOY, P.J., AND H.M. VIN [1998]. Cello: A disk scheduling framework for next generation operating systems, *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, SIGMETRICS'98, Madison, WI, June 22–26, *Performance Evaluation Review 26:1*, 44–55.

SHIMONISHI, H., M. YOSHIDA, AND H. SUZUKI [1997]. Improvement of weighted round robin cell scheduler, *Technical Report of IEICE CQ96–57 (1997–02)*, 25–32.

SHREEDHAR, M., AND G. VARGHESE [1995]. Efficient fair queuing using deficit round robin, *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ACM SIGCOMM'95, August 28–September 1, Cambridge, Massachusetts, 231–242.

SINCOSKIE, W.D. [1991]. System architecture for a large scale video on demand service, *Computer Networks and ISDN Systems 22*, 155–162.

SRIVASTAVA, A., A. KUMAR, AND A. SINGRU [1997]. Design and analysis of a video-on-demand server, *ACM Multimedia Systems 5:4*, 238–254.

STEPPING, C. [2001]. Wireless scheduling approaches and practical implementation issues, *Proceedings of the Personal Wireless Conference*, PWC'01, August 8–10, Lappeenranta, Finland, 39–57.

STOICA, I., H. ABDEL-WAHAB, K. JEFFAY, S.K. BARUAH, J.E. GEHRKE, AND C.G. PLAXTON [1996]. A proportional share resource allocation algorithm for real-time, time-shared systems, *Proceedings of the 17th IEEE Real-*

Time Systems Symposium, RTSS'96, December 4–6, Washington, Colorado, 288–299.

SUBRAHMANIAN, V.S. [1998]. *Principles of Multimedia Database Systems*, Morgan Kaufman, San Francisco.

SURI, S., G. VARGHESE, AND G. CHANDRANMENON [1997]. Leap forward virtual clock: a new fair queuing scheme with guaranteed delays and throughput fairness, *Proceedings of the 16th Annual Conference of the IEEE Computer and Communications Societies*, IEEE INFOCOM'97, April 7–11, Kobe, Japan. Extended version: Washington University of St. Louis, Technical Report #96-10.

TANENBAUM, A.S. [2003]. *Computer Networks, Fourth Edition*, Prentice Hall.

TEWARI, R., R. KING, D. KANDLUR, D.M. DIAS [1996]. Placement of multimedia blocks on zoned disks, *Proceedings of the SPIE Multimedia Computing and Networking*, San Jose, CA, January 29, *SPIE Proceedings 2667*, 360–367.

TONG, S.-R., Y.-F. HUANG, AND J.C.L. LIU [1998]. Study of disk zoning for video servers, *Proceedings of the IEEE International Conference on Multimedia Computing and Systems*, ICMCS'98, Austin, TX, June 28–July 1, 86–95.

TRIANTAFILLOU, P., S. CHRISTODOULAKIS, AND C.A. GEORGIADIS [2000]. Optimal data placement on disks: A comprehensive solution for different technologies, *IEEE Transactions on Knowledge and Data Engineering 12:2*, 324–330.

TRIANTAFILLOU, P., S. CHRISTODOULAKIS, AND C.A. GEORGIADIS [2002]. A comprehensive analytical performance model for disk devices under random workloads, *IEEE Transactions on Knowledge and Data Engineering 14:1*, 140–155.

TSAO, S.-L, M.C. CHEN, AND Y. SUN [2001]. Placement of VBR video on zoned disks for real-time playback, *IEICE Transactions on Information and Systems E84–D:12*, 1767–1781.

TSAO, S.-L., Y.-M. HUANG, C.-C. LIN, S.-C. LIOU, AND C.-W. HUANG [1997]. A novel data placement scheme on optical disks for near-VOD servers, *Proceedings of the 4th International Workshop on Interactive Distributed Multimedia Systems and Telecommunication services*, IDMS'97, Darmstadt, Germany, September 10–12, 133–142.

TSE, P.K.C., C.H.C. LEUNG [2000]. Improving multimedia systems performance using constant-density recording disks, *Multimedia Systems 8*, 47–56.

TSYBAKOV, B.S., AND V.A. MIKHAILOV [1978]. Free synchronous packet access in a broadcast channel with feedback, *Problems in Information Transmission 14*, 259–280.

VALENTE, P. [2004]. Exact GPS simulation with logarithmic complexity, and its application to an optimally fair scheduler, *Proceedings of the Annual Conference of the Special Interest Group on Data Communication*, SIGCOMM'04, August 30–September 3, Portland, Oregon, 269–280.

VERHAEGH, W.F.J., R. RIETMAN, AND J. KORST [2004]. Near video-on-demand with limited bandwidth and distributed servers, in: W. Verhaegh, E. Aarts, and J. Korst (Eds.), *Algorithms in Ambient Intelligence*, Kluwer.

VIN, H.M., AND P.V. RANGAN [1993]. Designing a multi-user hdtv storage server, *IEEE Journal on Selected Areas in Communication 11:1*. 153–164.

VAN DER VLEUTEN, R.J., W.C. VAN ETTEN, AND H.P.A. VAN DEN BOOM [1994]. Optimal controlled ALOHA for two-way data communication in a cable television network, *IEEE Transactions on Communications 42:7*, 2453–2459.

WANG, Y.-C., S.-L. TSAO, R.-Y. CHANG, M.C. CHEN, J.-M. HO, AND M.-T. KO [1997]. A fast data placement scheme for video server with zoned disks, *Proceedings of the SPIE Multimedia Storage and Archiving Systems II*, MSAS'97, Dallas, TX, November 3, *SPIE Proceedings 3229*, 92–102.

WELLS, J., Q. YANG, AND C. YU [1991]. Placement of audio data on optical disks, *Proceedings of the 1st International Conference on Multimedia Information Systems*, MIS'91, Jurong, Singapore, January, 123–134.

WOLTERS, R., H. VAN HOOF, C. BOTTE, AND C. SIERENS [1997]. Initialisation protocol for a burst-mode transport HFC system with delay determination by power distribution measurement, *Proceedings of SPIE 3233*, 353–360.

YIN, W.-M., AND Y.-D. LIN [2000]. Statistically optimized minislot allocation for initial and collision resolution in hybrid fiber coaxial networks, *IEEE Journal on Selected Areas in Communications 18:9*, 1764–1773.

YU, C., W. SUN, D. BITTON, Q. YANG, R. BRUNO, AND J. TULLIS [1989]. Efficient placement of audio data on optical disks for real-time applications, *Communications of the ACM 32:7*, 862–871.

ZHANG, H. [1995]. Service disciplines for guaranteed performance service in packet-switching networks, *Proceedings of the IEEE 83:10*, 1374–1396.

ZHAO, Q, AND J. XU [2004]. On the computational complexity of maintaining GPS clock in packet scheduling, *Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies*, IEEE INFOCOM'04, March 7–11, Hong Kong.

Author Index

Subject Index

Samenvatting

Efficiënte opslag en transmissie van data in kabelnetwerken

Een kabelnetwerk is tegenwoordig niet meer alleen een medium waarover analoge TV-signalen vanuit een centraal punt, *kopstation* genaamd, naar de aangesloten huizen worden gestuurd. Sinds enkele jaren is het mogelijk om thuis data digitaal te versturen en te ontvangen. Deze data gaat via een kabelmodem thuis en het kopstation, dat in verbinding staat met andere netwerken. Op deze wijze zijn kabelnetwerken onderdeel geworden van het wereldwijde Internet en kunnen computers thuis hier mee verbonden worden.

Door aan zo'n kopstation een digitaal videosysteem met duizenden films te koppelen, ontstaat er de mogelijkheid een *video-op-verzoek* dienst aan te bieden: Via de computer of zelfs de TV thuis kunnen films worden besteld en direct bekeken, of worden opgeslagen in de computer.

Om dit te bewerkstelligen is meer nodig dan alleen een netwerk: Voor de transmissie van video data dient er zorg voor te worden gedragen dat deze zonder hinderende interrupties kan geschieden, omdat dergelijke gebeurtenissen door de gebruiker direct te zien zijn in de vorm van een stilstaand of zwart beeld. Verder is ook de reactiesnelheid van het systeem van belang voor het ondersteunen van operaties door de gebruiker, zoals het bestellen van een film, maar ook het vooruit- of terugspoelen, pauzeren, enzovoorts.

Binnen deze context beschrijven en analyseren we in dit proefschrift zes problemen. Vier daarvan houden verband met de transmissie van data over het kabelnetwerk en de overige twee houden verband met het opslaan van video data op een harde schijf.

In twee van de vier problemen uit de eerste categorie analyseren we de vertraging die data ondervindt wanneer die vanuit een modem wordt gestuurd naar het kopstation. Deze vertraging bepaalt met name de reactiesnelheid van

het systeem. Karakteristiek voor dataverkeer in deze richting is dat pakketten van verschillende modems tegelijkertijd mogen worden verstuurd en daardoor verloren gaan. Met name de vereiste hertransmissies zorgen voor vertraging. Meer concreet beschouwen we een variant op het bekende ALOHA protocol, waarbij we uitgaan van een kanaalmodel dat afwijkt van het conventionele model. Het afwijkende model is van toepassing wanneer een modem een eerste contact probeert te leggen met het kopstation na te zijn opgestart. Met name na een stroomuitval, wanneer een groot aantal modems tegelijkertijd opnieuw opstart, kunnen de vertragingen aanzienlijk zijn. Daarnaast beschouwen we modems tijdens normale operatie en analyseren wij de verbetering in vertraging wanneer pakketten die vanuit één modem moeten worden verstuurd, worden verpakt in een groter pakket. In beide studies worden wiskundige resultaten vergeleken met simulaties die reële situaties nabootsen.

In de andere twee van de vier problemen richten wij ons op de transmissie van video data in de andere richting, namelijk van het kopstation naar de modems. Hierbij spelen stringente tijdsrestricties een voorname rol, zoals hierboven reeds is beschreven. Meer specifiek presenteren we een planningsalgoritme dat pakketten voor een aantal gebruikers op een kanaal zodanig na elkaar verstuurt dat de variatie in de vertraging die de verschillende pakketten ondervinden, minimaal is. Op deze wijze wordt zo goed mogelijk een continue stroom van data gerealiseerd die van belang is voor het probleemloos kunnen bekijken van een film. Daarnaast analyseren we een bestaand algoritme om een film via een aantal kanalen periodiek naar de aangesloten huizen te versturen. In dit geval ligt de nadruk op de wachttijd die een gebruiker ondervindt na het bestellen van een film. In deze analyse onderbouwen we een in het algoritme gebruikte heuristiek en brengen hierin verdere verbeteringen aan. Daarnaast bewijzen we dat het algoritme asymptotisch optimaal is, iets dat reeds langer werd aangenomen, maar nooit rigoreus bewezen was.

Bij de laatste twee problemen, die verband houden met het opslaan van video data op een harde schijf, analyseren we hoe deze data zodanig kan worden opgeslagen dat die er nadien efficient van kan worden teruggelezen. In het ene probleem beschouwen we een bestaand planningsalgoritme om pakketten van verschillende videostromen naar een harde schijf te schrijven en passen dit aan om ervoor te zorgen dat het teruglezen van de stroom met bijvoorbeeld een andere pakketgrootte mogelijk wordt zonder daarbij de schijf onnodig te belasten. In het andere probleem analyseren we hoe we effectief gebruik kunnen maken van het gegeven dat data aan de buitenkant van de schijf sneller gelezen kan worden dan aan de binnenkant. We bewijzen dat het probleem

van het zo efficient mogelijk opslaan van een gegeven aantal video files NP-lastig is en presenteren een eenvoudige heuristiek die, hoewel voor bijzondere instanties een bewijsbaar slechte prestatie levert, in de praktijk in het algemeen goede prestaties levert. Hierbij maken we met name gebruik van het verschil in populariteit van de verschillende films.

Biography

Verus Pronk was born in Vught, The Netherlands, on April 28, 1963. After graduating from Maurick College, Vught, The Netherlands, in 1981, he went to the Eindhoven University of Technology, Eindhoven, The Netherlands. As part of his curriculum, which specialized in computer science, he did an apprenticeship on low-level hardware design at the California Institute of Technology in Pasadena, California. He graduated from the University with honors in the field of mathematics in 1988. His graduation work concerned the derivation of algorithms in the area of string matching and was performed under the supervision of M. Rem and F.E.J. Kruseman-Aretz.

Since 1988, Verus has been working at the Philips Research Laboratories in Eindhoven, where he has worked in various areas, such as fault-tolerant computing, multimedia systems prototyping, scheduling, communication networks, and machine learning. Together with Jan Korst, he co-authored the book *Multimedia Storage and Retrieval: An Algorithmic Approach*, published by Wiley & Sons, Chichester. He wrote a contribution to the book *Algorithms in Ambient Intelligence*, published by Kluwer Academic Publishers, Dordrecht. He has co-authored more than 27 conference and journal papers and holds seven U.S. patents.