UNIVERSITY OF OSLO
Department of Informatics

Forward Error Correction in INSTANCE

Minna Kaisa Juonolainen

Cand Scient Thesis

1.2.1999

Foreword

This thesis is a part of my cand. scient. degree at the University of Oslo, Department of Informatics. My advisor has been Dr. Sc. Thomas Plagemann from the Center for Technology at Kjeller (UniK).

I wish to express my thanks to my advisor and Pål Halvorsen, who has given me invaluable guidance. I also wish to thank April T. Stilley for kindly consulting me in English grammar.

Oslo, February 1, 1999
Minna Kaisa Juonolainen

Abstract

Todays communication systems are inefficient due to redundant functions in the protocol stack. Multimedia data requires even more effective processing because of the real-time data restrictions for data transmission. The motivation in this thesis is to increase the efficiency of data communication systems by removing the redundant error correction function from the protocol stack, yet preserve the same level of reliability.

This thesis is one approach to the Intermediate Storage Node Concept (INSTANCE) project. The goal of this thesis is to study the possibility of integrating Forward Error Correction (FEC) mechanisms into Transport Protocols (TPs) and Redundant Arrays of Inexpensive Disks (RAID) data storage systems. Both RAID and TP provide error correction for reconstruction of lost data with the help of parity information. In this thesis we consider how multimedia (e.g, video, audio, and text) data behave in integrated FEC in RAID and TP.

The data distribution mechanism in RAID, called striping, is described to clarify methods used in this approach. Stripes must be adjusted in such a way that parity calculation products appropriate parity units for transmission. Another area which is addressed is how the striped data and redundant information is adjusted in order to achieve a functional combination in TP. The details we consider in this case basically concern the error coding algorithm. The algorithms for calculating parity information are the same types as in TPs. The tendency of congestion in a channel gives a guideline of how much redundant information is appropriate to store on RAID. In addition, the size of the data units for parity calculations influence disk access and data transmissions.

The stored data type indicates the size of each read from the disks. The studies show that the functionality of FEC in RAID and TP depends on the relationship between stripe units and transmission data units.

The results from literature studies show that there is not only *one* appropriate solution to integrate the FEC mechanism to RAID and TP. The methods used depend on many factors like user criteria, transmission channels, types of data, and coding algorithms. In this thesis, we consider possibilities of combining various schemes for RAID and TP to accomplish a feasible integration.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation and Background

The Internet has become a popular medium, for both work and spare time. New technologies and Internet services, such as, interactive Web-sites for bank services, bookstores, and public services have been developed. All of these new opportunities for Internet users and providers create higher demands on Quality of Services (QoS). QoS represents a set of parameters that indicate application requirements, as in multimedia, where two or more mediums are offered for users. Multimedia usually consists of continuous media like audio and video. The QoS for these types of media are for example: 1) Transport Data Units (TDUs) must come in the right order, 2) audio data must not be received earlier or later than the video, and 3) data transport must be continuous, i.e., breaks in a movie are not accepted. We can reason that it is a technical challenge to accomplish multimedia's QoS.

At the same time as new services are offered, the amount of Internet users increases rapidly. New technology has to solve the problems a large user amount brings. For example, a popular Web-site provider or File Transfer Protocol (FTP) server has to cope with many clients and high data traffic. The bottleneck for these services will be the server that cannot answer the needs of multiple clients. The bottleneck situation can worsen because of badly arranged data storage or unreliable data transmission.

Disks for storing data are remarkably larger in the status quo. However, in data storage systems, the microprocessor performance has developed faster than performance of disks. For example, large disks that are accessed frequently are not effective in seeking and reading the data from the disk. This leads to performance differences, since the quantity of disks is high, but the quality of disks cannot meet the high performance of microprocessors. In addition, large amounts of clients accessing a large disk simultaneously de-

velops bottlenecks. The solution for the performance gap is to arrange small independent disks into a large logical unit to provide faster accesses to data. One of these arrangements is called Redundant Arrays of Inexpensive Disks (RAID). The seek and access time for data is shorter in smaller disks than in large disks. The total seek time in RAID may not be less than in one large disk. However, throughput in RAID is much better because of parallel accesses to the disks. This is why RAID is an effective solution for storing large amounts of data as one unit.

Another benefit that smaller disks have is that they are not as failure prone as larger ones. The consequences of large disk failure is worse than with small a disk. This is because, larger amounts of data can be lost simultaneously from a large disk, than from a small disk. Redundant data added to the arrays of disks offers the ability to withstand the failure of a single disk. Various methods for maintaining redundant data in RAID is one of the topics in this thesis.

Since data storages have increased the amount of information, data transmissions are also increasing in size and frequency. On data transmission, the development of physical media and different services has given new challenges in maintaining data reliability. There are several methods that are used for gaining reliability in data transmission and data storage systems. One of these methods is Forward Error Correction (FEC), which will be a central topic in this thesis.

This thesis is a study in the INSTANCE project, a project that investigates, how the efficiency of servers can be increased by integrating data management systems and removing redundant functionality. The motivation in this thesis is to find methods for integrating FEC in RAID and data transmission, and to avoid the bottlenecks in data transmissions and storages. For example, more reliable data transmissions and data storage systems are achieved with FEC, which can provide more effective services for multiple clients. Data transmission that is reliable enough without retransmissions of TDUs is effective enough for real-time data. A data storage system that is capable of serving clients despite a disk crash provides an effective service for multimedia purposes, among others.

The goal is to propose a method for improving reliability of data transmission from RAID to multiple clients. Comparisons of FEC methods in RAID and in transmission medias are presented in this thesis. Methods that are appropriate and suitable for both RAID and transmission of data are investigated and evaluated. These investigations and evaluations are used as a basis for experimental implementations and implementation plans.

1.2 Problem Specification

Server can create a bottleneck in data communication systems, especially when many clients use the service that is expected to be reliable simultaneously. When problems occur in data transmission and many clients require retransmissions of lost or erroneous data, the bottleneck problem accumulates. Further, the problem is how to reduce the workload on a server, and how to serve multiple clients which use the same server without remarkably increasing the server's work.

In INSTANCE we look at several approaches of solving this problem; and in this thesis, one of these approaches is discussed. This approach is how to reduce the server's work by adding error correction to gain higher data reliability.0 Reliability of data transmission is usually achieved by implementing methods on the protocol stack at different levels. One example is retransmission of the lost data. However, our motivation is to reduce the server's workload by leaving error correcting to the client.

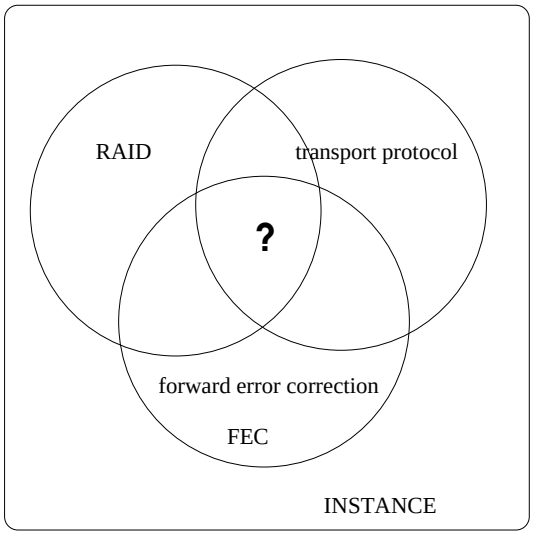

Figure 1.1: Problem specification area

Figure 1.1 contains fields of interest in relation to this thesis. The first field is FEC. FEC is the method that uses redundant data for error correcting. The second field is RAID. RAID is the data storage system that uses FEC in recreating lost data after disk crashes. FEC is also used in transmission of data to provide more efficient and reliable services. Transport Protocol is the application that is used for transmitting data from RAID to the clients.

The question mark "?" in the figure points out the goal in this thesis: how can we integrate FEC in RAID and the data transmission protocol? What do they have in common and what methods are compatible when we want to increase the efficiency of the client-server system? The main area in Figure1.1 is described as INSTANCE since it is the project that studies methods to support a higher number of clients concurrently with lower costs.

1.3 Method Specification

The basic method for this thesis is literature work. Several related works that have proposed FEC in transmission have been investigated if they have relevant information for the thesis. The mathematical background for FEC has been clarified in order to understand the FEC solutions in the related works. All relevant information is gathered and compared. Several methods proposed in the related works are evaluated and compared to our goal.

Related works that propose various solutions for RAID are also introduced. The RAID methods are evaluated in relation to INSTANCE and transmission article information. The comparison of related works is the basis for problem solving later on in this thesis.

Some experiments are executed for this thesis. The implementations of the experiments are carried out in a test environment that consists of Dynamic Configuration of Protocols (Da CaPo) [20]. Da CaPo covers multiple layers of the OSI reference model. Da Capo is used together with a simplified FEC coding protocol in this thesis. Test results of the implementation are evaluated. These results are used to explore the possibilities of accomplishing the integration of FEC in RAID and TPs.

1.4 Outline

An outline of the rest of this thesis is given in this section. Chapter 2 gives an introduction to INSTANCE. The description of INSTANCE allows a thorough introduction of the motivation in this thesis.

Chapter 3 introduces failures and errors that may occur in data transmission and data storage systems, where the reasons for the most usual errors are pointed out. In the data transmission part of this chapter, typical error correcting methods are introduced.

To be able to understand how error coding is done, basics of coding theory is given in chapter 4. This chapter also gives information which explains

why these particular methods in error correction coding are used. Chapter 4 presents some examples in theory of how it is possible to accomplish FEC in computers.

Chapter 5 focuses on describing RAID. Several related works which propose new methods to improve the properties of RAID are introduced. Relevant articles for our problem specification in relation to INSTANCE are evaluated and compared in this chapter.

Chapter 6 gives an introduction to transmission protocols that are available or proposed in literature. We describe related works and approaches that are relevant to this thesis. Chapter 5 and 6 can be read in an independent order.

Chapter 7 compares and discusses the methods and information studied in earlier chapters from 2 to 6. Ideas about how the methods proposed in this paper would work adequately are pointed out.

Chapter 8 describes a simple test environment. Experiments and test results are evaluated. The evaluations of the simple experiments point out what future work needs to be done in practice, in relation to the approach introduced in this thesis.

Chapter 9 presents conclusions and future work.

Chapter 2

Overview of INSTANCE

This Chapter introduces the basics for the Intermediate Storage Node Concept (INSTANCE) project. INSTANCE is a project at the University of Oslo, Center for Technology at Kjeller (UniK). Methods of interest in this particular project include where servers are able to support large amount of clients without considerably increasing the server-side work load and hardware resources.

2.1 A Traditional Approach to Data Communication

INSTANCE studies how to avoid bottlenecks in data communication, especially in end-to-end protocols. With end-to-end protocols, we mean the upper layers in a protocol stack. For example, in the OSI reference model, end-systems have to handle all seven layers in sending and receiving data. Sometimes protocols in the protocol stack do the same type of work for data units in their own layer. Services like buffer management and error correction is done by data management systems and communication protocols.

Figure 2.1 shows a traditional approach for data storage and transmission protocol arrangement. In this figure, the data storage system RAID, upper layers of the protocol stack, and the host-to-network interface are presented. The redundant service in this system is the error correction coding that exists both in RAID and TP. Redundant work in protocol stacks causes ineffectiveness, wasted processing time, and the production of unnecessary copies of protocol data units. These pitfalls of redundancy can accumulate when the server is accessed concurrently by many clients.

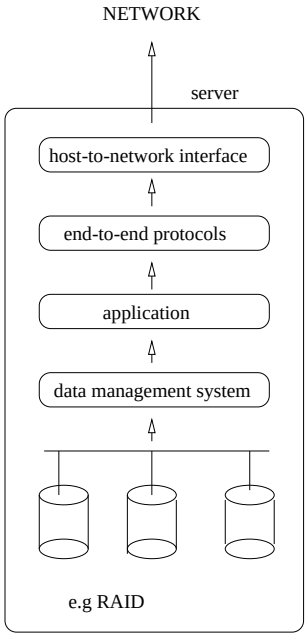

Figure 2.1: Traditional data storage to transmission arrangement

2.2 Integration of Redundancy in Protocol Stack

Layers in a protocol stack have to be optimized in a way that only one protocol does one type of job. However, protocol stack should give the same QoS and functions as before optimizing; only redundant functionality is removed. The possibilities of integrating the redundant FEC in RAID and FEC in TPs is studied in this thesis. In the course of this thesis, the best combination of protocols and methods to accomplish the integration of FEC is considered.

Figure 2.2 shows the integration method utilized in this thesis. In Figure 2.2, the oval around the layers of the protocol stack includes the data management system, application, and end-to-end protocols. It is possible to optimize services by integrating the redundant services in these three layers. The number of accesses to different layers in the transport protocol stack can be decreased with the Integrated Layer Processing (ILP) principle. ILP is an implementation technique for data manipulation functions in communication protocols. INSTANCE focuses on servers which have applied ILP principle in handling of the data management system, application and the remaining communication protocols.

Figure 2.3 describes the end-to-end environment for this thesis. Coding for restoring lost information after a disk crash is included in RAID systems.

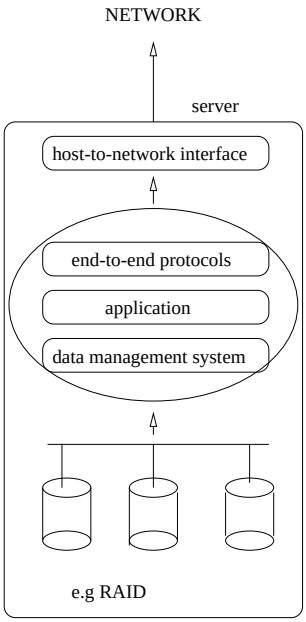

Figure 2.2: Data storage and transmission in INSTANCE

Similar types of coding techniques, called FEC, are used in data transmission for restoring lost TDUs. The coding techniques in RAID and data transmission are similar; only the target area where they are used is different. In both target areas, the coding methods are used to restore lost data. Our motivation is to find methods of integrating the error coding in the following target areas: data transmission and RAID. As shown in Figure 2.3, encoding exists before RAID on the server side, and the decoder exists on the client-side.

Error coding is costly, because it is time demanding and uses processing resources. The integration of error coding has many benefits. In INSTANCE, the costly error coding would be available for both the data storage system and for data transmission. This method would spare time and processing efforts since the encoded data for transmission is made readily available on disks. Should one disk from RAID fail, it has the same ability to recover from disk crashes as a RAID without connection to data transmission with FEC. This means that RAID would provide the same QoS also in INSTANCE. The next example illustrates the benefits of this approach to INSTANCE.

For example, an intermediate node is used for Video on Demand (VoD) services. Films are downloaded on RAID with the redundant information produced by an appropriate encoding method. This redundant information is used to restore data on the disks, but it is also used for FEC in transmission. Clients utilizing the intermediate node, in this case the VoD service, must

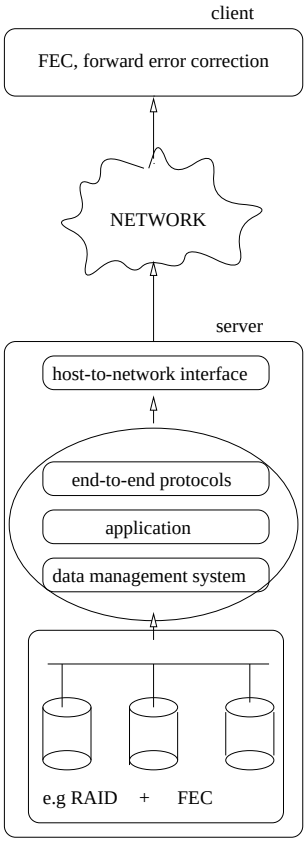

Figure 2.3: INSTANCE and forward error correction

also have the corresponding FEC system for data decoding. In the case of TDU losses in transmission, the client is capable to recreate lost TDUs during the decoding process.

The benefits in the example introduced are:

1. Many clients can utilize the same resources without redundant data encoding for transmission since the data is encoded only on RAID. This means that the data is encoded only when data is changed on disks or failure on a disk has occurred. Without integration of FEC, the data would be encoded each time it is transmitted to the clients.

2. If the data storage units, blocks are adjusted to the size of TDUs, the time for specific TDU size adjusting is spared. Error probability in these indirect operations is also avoided.

3. The number of copy operations and the number of context switches is reduced.

4. All of the benefits listed above enable several clients to access VoD services without drastically increasing hardware resources or changes in them. Also, the parallelity provided by the disk array offers the possibility for multiple clients' processes to access data effectively.

5. If TDUs are lost under data transmission, no time is used for retransmissions because of FEC. This is a good quality especially for real-time data transmissions, which do not tolerate transmission delays. This issue is discussed in detail in Chapter 3.

In order to implement FEC integration, several issues are studied. For example, what are the typical errors in RAID and data transmission and what type of error coding is appropriate for correcting them? If a common denominator is found for errors and coding in RAID and data transmission, what is the criteria for them in our work? In the course of this thesis, issues like error coding, failures, and techniques for error coding in RAID and data transmission are studied in more detail. The problems that can occur while implementing FEC integration and some methods to avoid them are discussed.

Chapter 3

Errors in Distributed Systems

An overview of possible problem areas in digital transmission, storing, and retrieval is given in this chapter. Definitions for failures and errors in distributed systems in general are also given. In the second section, we take a closer look at the failures in storing and retrieval of data. The third section describes failures and errors in data transmission. The last section briefly introduces the most typical error correcting mechanisms in data transmission.

Digital information consists of strings of bits. Bits are represented in binary format (i.e., 0's and 1's). It is inevitable that a physical device makes mistakes in reading and writing bits, even if the systems are designed for very high reliability. There is always a chance that a 1 can be interpreted as a 0, or vice versa. When a bit is mistakenly interpreted, a bit error has occurred. For example, large information systems, like distributed databases, have a great amount of data in transactions and transmission lines. Therefore, they have a larger risk of failure during interpretation. In the following section, we describe reasons for these misinterpretations.

3.1 Definitions of Failures and Errors

When a malfunction occurs in a physical device, e.g., a broken line, we call it a *failure*. The effects of such failures are often called *faults*. Faults are understood to be a logical level problem. This means that the boolean value, which is a bit in binary format, is misinterpreted. Faults become *errors* when they are at the informational level, values attached to registers[1], for example [37]. An error is a bit fail, or a bit with an unknown value in an unknown location. An error that may be located in any position in a word and is permanent in nature, is said to be a hard error, whereas *erasure* is an error with a known

[1] A special, high-speed storage area within the CPU. All data must be represented in a register before it can be processed.

location [6].

There are several reasons for malfunctions and errors. One of the factors that limits data communication performance is *noise* [32]. Noise comes from unpredictable electrical signals introduced by equipment or natural disturbances. This means that noise interferes with the intended signals [30] and generates errors. Different forms of noise are: thermal, crosstalk, intermodulation, and impulsive noise .

Thermal noise, also called white noise, is present in electrical circuits, for example, in the front end of the receiving equipment. Thermal noise is caused by thermal agitation of electrons in a conductor [45]. Usually this type of noise is not a problem. Such noise cannot be eliminated and can often be heard as background noise in radios and telephones, for example.

Crosstalk can be experienced during telephone conversations. You can probably hear another conversation during your own. Crosstalk can occur by electrical coupling between a nearby twisted pair or coax cable lines carrying multiple signals [45]. This type of noise does not cause trouble to digital data. Crosstalk is therefore not discussed closer in this paper.

Intermodulation noise produces signals at a frequency that is the sum or difference of the two original frequencies, or multiples of these frequencies. Intermodulation noise is produced when some nonlinearity in the transmitter, receiver, or intervening transmission system is present [45].

Impulsive noise occurs as short impulses or noise spikes of short duration and intersperses with short burst of errors. Impulsive noise consists of randomly occurring unwanted signals. The source of this kind of noise can be switching gear or thunderstorms, for example. Impulsive noise is not a big problem for analog data, e.g., while using a telephone, it is possible to understand the telephone conversation with small breaks caused by noise. Unlike in analog data, this type of noise is the primary cause of errors in data communication [30].

The last error source in transmission is *attenuation*. Attenuation is a reduction of strength in a signal, beam or wave during transmission. This is a typical problem for analog transmission. Digital transmission does not have this problem.

3.2 Error Scenarios during Transmission and Storage of Digital Data

Error control techniques become more important as systems are distributed and large data amounts flow between them. In this section, we describe some reasons for and consequences of the failures and errors which are pointed out below. First, we give a preview of noise connected problems within database systems. Second, we point out problems that can occur when transmitting digital data.

3.2.1 Errors in Storage of Data

In this section, we consider same possible causes of disk failure. Failures can occur during read and write operations on disks. Furthermore, aging of the magnetic media causes damages gradually. Reading disks is unlikely to cause permanent errors [13]. Therefore, we will concentrate on studying the other factors which cause disk failures, and we will refer to Figure 3.1 as an example of different failure situations. A scenario of disks arranged as an array instead of one independent disk is shown in Figure 3.1. Spheres of failure influence are marked off with dashed lines. Disk arrays will be discussed in in greater detail in Chapter 5.

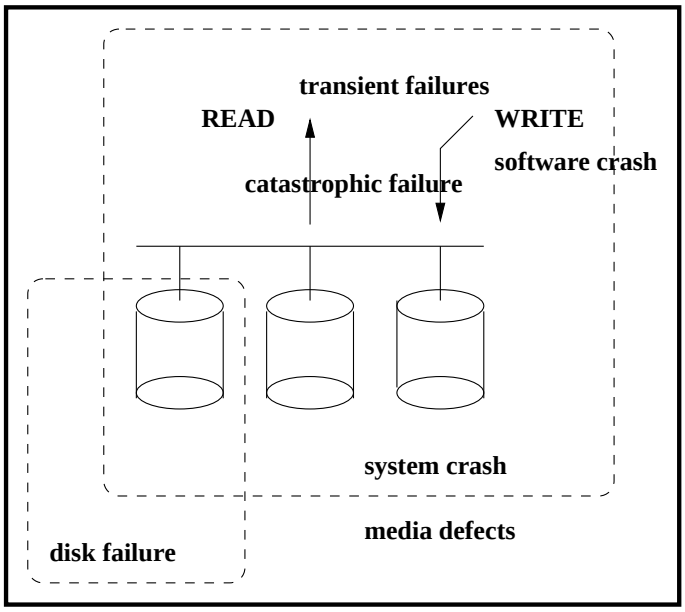

Figure 3.1: Failure possibilities during storage of data

Hellerstein [22] points out 3 primary types of failures in magnetic disk drives:

- **Transient failures** : As Figure 3.1 shows, transient failures affect read and write situations. As introduced earlier on page 13 in this chapter, a write operation can fail while interpreting bits in binary format, i.e., a 0 read as a 1 or vice versa. These are also called *soft errors*.
- **Media defects :** These failures are permanent. Hoverer, these errors are usually detected during production at the factory. Media defects stand outside of the system in Figure 3.1 because this type of failure is not usually an issue in systems that are already in use.
- **Catastrophic failures :** These failures can be head crashes or failures of the read/write or controller electronics failure. Catastrophic failures usually cause unreadable bits, i.e., erasures. A catastrophic failure is possible for one disk and in write/read situations for all of the disks in a disk array.

Hellerstein [22] concludes that future I/O systems will contain a larger number of disks. A larger number of disks has lower reliability. For example, if we have one disk that has a 0,5 possibility of getting an error, then a system of 100 disks has a 0,5*100 possibility of getting an error. Therefore, larger systems are more unreliable.

Chen [13] mention that *system crashes* refer to any event such as a power failure, operator error, hardware breakdown or software crash that can interrupt an I/O operation to a disk array. As mentioned above, Hellerstein concluded that we can say that disk arrays are more influenced by system crashes than one single disk. As a result, data redundancy[2] and error correction mechanisms become more relevant [35].

Although redundancy increases data integrity, it also leads to problem situations such as inconsistent data when a *software crash*, i.e., malfunction in software has occurred during writing to the disks. Software that interfere with I/O operations can lead to inconsistency in redundant or original data, e.g., states where redundant data is updated but the original data is not.

Environmental factors such as simultaneous power failures, can cause *correlated* disk failures. The age of the disks can cause correlated failures when the disks age simultaneously. Old and unused disks are more prone to have failures: older disks can get failures because of weariness, and new disks can have undetected transient errors that can cause failures. These failures tend

[2]With redundancy we mean redundant data that is used to reconstruct original data after failures in disk arrays. This topic is discussed closer in the chapter 4.

to happen closely together.

This chapter does not introduce error correcting mechanisms in data storages because we are basically interested in RAID and FEC. RAID uses redundant information for recovering from a disk crash. This issue is discussed closely in Chapter 5.

3.2.2 Errors in Transmission

Various transmission medias have their own typical error sources. A twisted pair is sensitive for crosstalk. Analog transmissions, like telephone conversations, suffer from *attenuation*. This means that the signal has to be amplified along its way on longer distances. Digital transmissions do not have this problem. Signals can be regenerated, and the signals are quite resistant to noise.

In wireless or satellite channels, errors will occur because of noise. Another drawback in satellite communication is attenuation by rain on links operating on high frequencies. This will become a greater issue in the future, as satellite and wireless communication becomes more widely used [1].

The next example illustrates why noise has become a problem area, especially when networks have increased their transmission speeds. On any given transmission line, the higher the transmission speed, the more error-prone the signal is. Loomis [30] illustrates this with an example:

> "If data were transmitted at the rate of 150 bps, each bit would last for about 6 μsec. For example, a noise burst of 2 μsec does not affect the data. If the transmission speed were raised to 1200 bps, each bit would last for 0.8 μsec. A 2 μsec noise burst would span then 2 1/2 bits. It is likely that one or more of the bits would be corrupted."

Fiber optics is not very sensitive to noise. In a modern fiber optic network, the usual source of errors is not bit errors on the medium, but rather overruns in the receiver or congestion in the routers [50]. Congestion is created if high speed channels exist with low speed channels. Jain [26] points out the fact that high speed channels are usually more expensive in costs than low speed channels, and are therefore shared. The higher the expense, the more sharing, and therefore, more traffic on the channel.

TDU loss is a direct consequence of network congestion and/or bit errors in the TDU header. Congestion can be understood as a temporary resource overload. If we examine Figures 3.2 and 3.3, we notice that the router in the network between the channels, i.e., transmission media, includes a buffer.

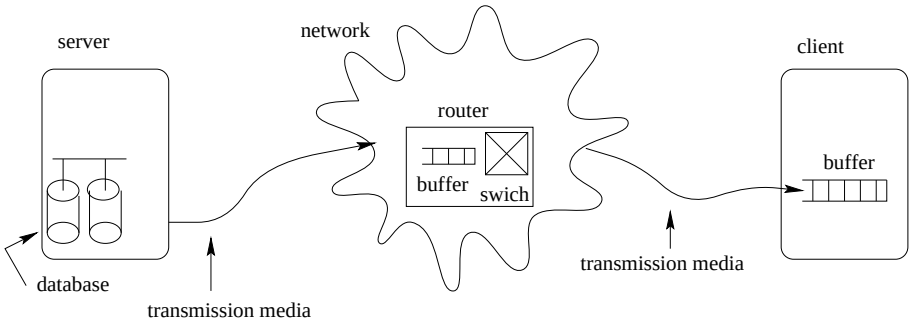

Figure 3.2: Congestion in the network

The task for a buffer is to store data from the channel where data proceeds in queue. From the buffer, data is sent to the channel forwards to the next node. If data has arrived at the receiver, then the data is removed from the buffer and transfered to the upper layers.

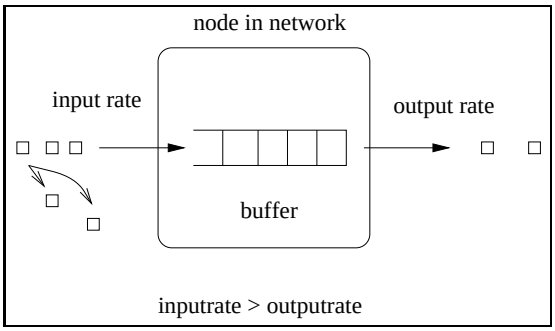

Figure 3.3: Congestion in the router

Overflow can occur when for example, two data bursts arrive at the node simultaneously and the data queue grows rapidly. When the output rate is smaller than the input rate, TDUs can be discarded when the buffer does not have enough capacity to receive data (see Figure 3.3). These discarded TDUs can increase congestion because they have to be retransmitted and are loading the network additionally. Synchronous overload occurs after an electrical power failure, for example when all of the machines reboot, and perhaps connect to a name server simultaneously. This can cause overload in the sender [49].

Figure 3.4 shows two possibilities for error distribution during transmission. The first alternative in the figure shows TDU loss due to congestion. Here, the lost TDUs are colored with grey. Consecutive TDU losses are typi-

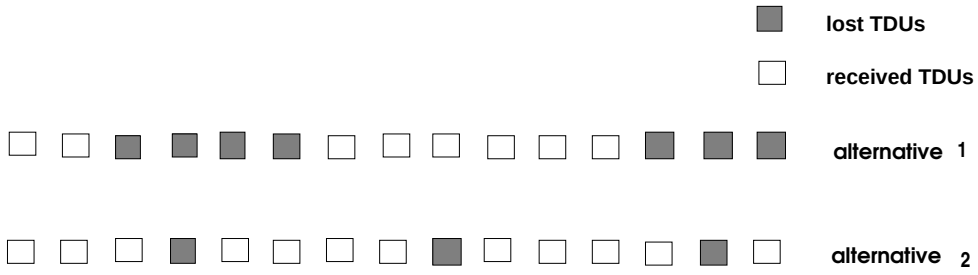

Figure 3.4: Error distribution in TDUs

cally caused by congestion. The second alternative in Figure 3.4 illustrates a single TDU loss e.g., due to a bit error on the header field. This type of error is not typical for optical fiber.

Congestion losses are the dominant form of error e.g., in ATM networks. Since large TDUs are divided into smaller units, cells, for an ATM type of network, one cell loss causes corrupted TDU. The whole TDU is therefore discarded. Alternative 2 in Figure 3.5 shows the worst possible error distribution possibility. In this alternative, all of the original TDUs are corrupted and therefore discarded. However, since congestion is the dominant cause of cell loss in ATM-type networks, alternative 1 is more likely to happen, where four consecutive cells are lost. In this case, only two original TDUs are lost. The error characteristics of ATM-type networks is an important point that is taken into account when designing error correction. This issue will be discussed in Chapter 7.

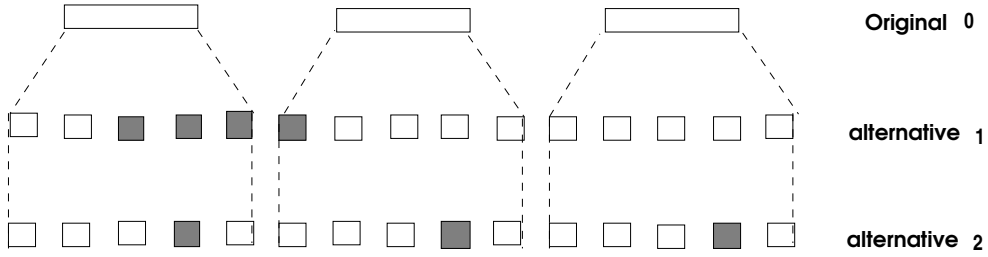

Figure 3.5: Error distribution alternatives

3.2.3 Error Correction in Transmission

Basic error detection and correcting mechanisms are introduce in this section. These mechanisms are Automatic Repeat reQuest (ARQ), Cyclic Redundancy

Check (CRC) and FEC.

- Automatic Repeat reQuest (ARQ) is an error control mechanism for data transmissions. Lost data is detected with sequence numbers and time-outs. The communication about which TDUs are received or lost is accomplished with acknowledgments. If the acknowledgment back to the sender is negative, lost TDUs are retransmitted. The two most common strategies for retransmission are:
  - Go-Back N scheme goes back to the lost TDU and restarts transmission from the lost TDU.
  - In selective repeat, only the lost packets are retransmitted.
- Cyclic Redundancy Check (CRC) is an error-detection scheme that uses redundant encoded bits (parity bits), and appends them to the digital signal. The received signal is decoded and TDUs with errors are discarded. For example, an ARQ system can be combined with CRC, if error correction is required.
- FEC also uses encoded redundant information. This information is used for reconstruction of lost TDUs. Therefore, retransmissions are not always necessary. However, the effect of the code is dependent on the algorithm that is used in coding. This issue is discussed closer in Chapter 4.

If an error has occurred during transmission, we can retransmit the original TDUs to repair data. This takes more time (i.e., at least two round trips + time-outs) and, e.g., in real-time system retransmission is not the most appropriate method to correct errors. Acknowledgments are crucial to the data traffic in satellite communication, for example. Linder [29] points out that ARQ based protocols give efficiency problems due to the long delay in the sender-receiver control loop and the high amount of data in the pipe from the sender to the receiver.

Errors usually occur as TDU loss and/or bit errors [27]. With CRC, we do not know which cell is dropped or which cell includes one bit error or more. Since CRC and ARQ are basically useful in error detection, FEC is attractive in the systems where we want to avoid retransmissions of TDUs.

Chapter 4

Foundation of Forward Error Correction

In this chapter, we introduce coding theory in areas that solve problems among data integrity. One basic method is Forward Error Correction (FEC), where redundant data is transmitted to allow reconstruction of lost data on the receiver side. First, we give an overview of basic mathematical theory behind FEC, and will focus on techniques that are used in RAID and transmission protocols.

The basics for reliable digital communication have been introduced by Claude Shannon and Richard Hamming in the late 40's. Shannon showed how faster transmission is possible with error correcting systems. Error correcting systems in data storage also helps to get the most out of the storage. Hamming discovered and implemented a single-bit error correcting code.

4.1 Short Introduction to Forward Error Correction Coding

In this section, we give an overview of coding theory by using an example of FEC in transmission of data. Error correcting code algorithms work in similar ways for disk arrays and data transmission. Instead of transmitting data, the erroneous disk is recovered inside the same system, i.e., disk array. More about how FEC is arranged in disk arrays is discussed in Chapter 5.

FEC is used by communication devices where the receiver immediately corrects errors that have been introduced to transmitted data. The transmitter computes redundant information from the original data and adds it to the data to be transmitted. There are two possibilities in transmitting redundant data: either including it to the original data or sending it in its own units in transmission. The received data is analyzed and possible errors are located with the help of redundant data. If an error has occurred the receiver tries to

recreate the lost or erroneous data. After decoding, redundant information is separated from the original data and the message is returned to its original form.

Encoding and decoding are illustrated in figure 4.1. The symbols introduced in the figure are also used later on in this chapter. This figure is illustrated according to [49].

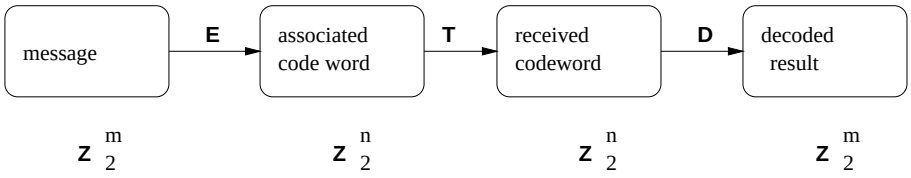

Figure 4.1: Decoding and encoding

E = encoding function where $E : W \rightarrow Z^n$
Z denotes for the group of natural numbers.
W = transmitted message
T = transmission where $n = k + r$, n is the total bits for codeword, k is information digits and r is redundant information digits.
D = decoding function to remove the redundant information
$D : Z_2^n \rightarrow Z_2^m$ let $n > m$ for $m, n \in Z^+$

This thesis focuses on transmission of digital data. Therefore, $n, k \in Z^+$. n and k are elements of positive integers belonging to the group $Z_2 = \{0, 1\}$. It is logical to have message length $m = 8$ bits, since strings of the signals, 0 or 1, have the length of one byte. m can also be a multiple of 8.

4.2 Mathematical Background for Coding Theory

Many error detecting and error correcting codes are based on certain types of arithmetic. Sets of elements which are closed under specified operations[1], like fields, give an easy way to handle sets of elements. The properties for different sets of elements are introduced next [21, 49].

- *RING* R is a nonempty set which is closed under binary operations '+' and '$\cdot$' $(R, +, \cdot)$. Finite rings are important in the theory of computer science. $(R, +)$ is an Abelian group and $(R, \cdot)$ is closed under associative operation $\cdot$.
- *FIELD* F is a commutative ring with unity, such that $\forall x \in F, x \neq z \Rightarrow x^{-1} \in F$. Therefore, a field is a group with special rules. For example,

[1] An element belongs to a set if it belongs to the same set after operations.

Z_n is a field if and only if n is prime. This leads to the observation that for any prime p, $(Z_p, +, \cdot\ )$ is a finite field. From a mathematical point of view, each bit or byte is an element in a finite field.

- *GROUP* is an nonempty set that involves only one closed binary operation $\circ$. $(G, \circ\ )$ is called a group.

Basic rules for groups are associativity, commutativity, and an element in a set always has an additive inverse and an identity element.

Examples of these rules are :

- *Associativity*: if a, b,and c belong to a set then $(a + b) + c = a + (b + c)$.
- 0 is the *identity* element if $a + 0 = 0 + a = a$.
- *The additive inverse* of a is an element that produces 0 when added to a.
- *Commutativity* is when $a + b = b + a$.

$Modulo - q$ addition forms a commutative group for all $q \geq 2$ and multiplication forms a commutative group over the nonzero elements $1, 2, \ldots p - 1$ for all prime values of p. Non prime numbers do not form a group.

We can define fields from groups. A set of elements is a Field F if and only if [41]:

1. The elements in F form a commutative group addition.
2. The nonzero elements of F form a commutative group under multiplication.
3. Multiplication distributes over addition.

The number of elements in a field is called the order of a field. In the case of error coding, we are interested only in finite fields. Finite fields have a finite number of elements. Finite fields based on modulo arithmetic are restricted to have a prime number of elements. Finite fields are also called Galois Fields GF(p). Where p stands for prime number and GF stand for Galois Field by the French mathematician Evariste Galois (1811-1832). In his work he showed how fields and polynomials are connected.

An *extension field* is a field F where F has a subgroup F'. F' is a field under inherited addition and multiplication. A *binary extension field* is a field that contains 2^m elements where each element has an m-bit binary value. It is possible to define an extension field $GF(2^m)$ for any $m \geq 2$. For example, the

binary extension field 2^3 has $2^3 = 8$ elements. These rules are useful in e.g., decoding when we remember that the subfield always divides the extension field.

The properties that are closed under certain operations are an advantage when we operate with bits or bytes. For example, we know that operating with only by adding sets together in coding session, the set stays the same. A set of bits, 0's and 1's, is always a set of 0's and 1's after coding, not decimal numbers, e.g..

4.3 Linear Block Codes

The blocks which consist of original information bits are usually called message blocks. Linear block codes operate upon blocks with fixed-length. These differ from convolutional codes that operate upon a continuous stream of information bits [41].

All linear codes compute check symbols as a linear combination of data symbols over a particular field. For binary codes, this means that, the field is integers modulo 2. Therefore, check bits are computed as the exclusive-OR of subsets of data bits.

We return to figure 4.1 that was presented earlier in this chapter. **E** and **D** are functions of an (n, m) block code. The encoder accepts k information bits from the information source and appends a set of r parity-check digits, which are derived from the information digits e.g., by using XOR calculations. The parity bit is added to every data unit. The parity bit for each unit is set so that all bytes have either an odd number or an even number of set bits. An example of odd and even parity is shown in Table 4.1.

Parity	Data	Data + Parity
Odd	1101	11010
Even	1101	11011

Table 4.1: An example of odd and even parity

A decoder operates on the received word to determine whether the information and check bits satisfy the encoding rules. XOR calculations with parity bits are the easiest way to detect and correct one error. In the next section, a method for more effective parity calculations is introduced.

4.4 Matrix Representations of Linear Block Codes

We use the definitions for encoding and decoding from Figure 4.1, and take a look at how linear block codes can be represented as parity-check matrixes. This is an easy example of coding and gives the basics for understanding other coding methods too.

Any set of basic vectors for the subspace of $GF(2)^n$ (n - bits per codeword) can be used as rows to form a matrix G called the *generator matrix*. G defines the encoding function E. Encoding function $E : Z_2^m \rightarrow Z_2^n$ is given by an $m \times n$ matrix G over Z_2. The codeword that is going to be encoded is multiplied with the generator matrix G.

A generator matrix is constructed with an *identity matrix I* and *A*, which is an $m \times (m - n)$ matrix. An identity matrix is an $m \times m$ matrix, where the field element is one (1) in every entry of the main diagonal, and the field element is zero (0) in every other matrix entry [8]. Matrix A consists of linearly independent vectors in $GF(p)^n$, each of length n. Encoding and decoding is introduced in the next two examples 4.4.1 and 4.4.2.

Example 4.4.1

Let G be a 3×6 generator matrix over Z_2. G can be described as $G = [I_3 \mid A]$, where identity matrix I_3 is a 3×3 matrix and A consists of 3 k length vectors of information bits ; [110], [011], and [101].

$$\mathbf{G} = \left[\begin{array}{ccc:ccc} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right]$$

If message w is to be send, then encoding for the message w is accomplished by $E(w)$ multiplying w with the generator matrix. $E(w) = wG$. This is done for each message in Z_2^m and the code C will be in Z_2^n. $C = E(Z_2^m) \subset Z_2^n$.

The generator matrix G defines an encoding function $E_2^3 \rightarrow E_2^6$. In this case, the encoding function adds 3 parity bits to the original message. We have message $w = (010)$ which is encoded in the following way:

$$E(010) = \left[\begin{array}{ccc} 0 & 1 & 0 \end{array}\right] \left[\begin{array}{ccc:ccc} 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 & 0 & 1 \end{array}\right] = \left[\begin{array}{cccccc} 0 & 1 & 0 & 0 & 1 & 1 \end{array}\right]$$

The encoded codeword is :[010011]
End of example 4.4.1.

Example 4.4.2

In decoding, we use associated parity check-matrix H. H is an $(n-m)\times n$ matrix of the form $[A^t r \mid I_{n-m}]$. A^{tr} is A's transponent in this example and looks like this :

$$\mathbf{A^{tr}} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 1 \end{bmatrix}$$

We get H :

$$\mathbf{H} = \left[\begin{array}{ccc:ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right]$$

If we use H to multiply received codeword (r) transponent r^{tr} then we can analyze the result and make judgments. $E(w) = r$ if and only if $H \cdot r^{tr} = 0$. With other words, if result is 0, i.e., a vector of zeros, we can assume that the received message is unchanged. Other results than 0 give a syndrome. A syndrome reveals the error pattern in the received word.

In our example, we receive a codeword r, where $r = [110011]$. We calculate the syndrome of r for detecting errors. From calculations of the syndrome we get the following result:

$$\left[\begin{array}{ccc:ccc} 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \end{array}\right] \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} = [110].$$

The result is not [000]. Thus, an error has occurred under transmission. If we take a closer look at the calculation, we see that syndrome s is $s = [110]$ and it is the same as first column of H at the same time. This result gives us information about where the error is located. We can determine that r is C by changing the first component. C and r differ only in the first component. The syndrome shows us the i'th column of the matrix H. In other words, the i'th bit from codeword has changed. The next example points out the limitations parity-check matrix coding has.
End of example 4.4.2.

Example 4.4.3

Assume that at most one in every eight bits contains an error (i.e., 0's interpreted as 1's or vice versa) during transmission and as a result we get two codewords which strongly resemble one another. It is impossible to detect and correct the errors in a situation like this. If we have codewords $x = x_1, x_2 ... x_n \in Z_2^n$ and $y = y_1, y_2 ... y_n \in Z_2^n$, the distance between the two codewords x and y is the number of components where $x_i \neq y_i$, for $1 \leq i \leq n$. For example, the Hamming distance between 1011101 and 1001001 is two. *End of example 4.4.3.*

Definition 4.4.1 ***Hamming distance*** $d(x, y)$ *between two q'ary sequences* x *and* y *of length* n *is the number of places in which they differ. The minimum distance* $d*$*, is the Hamming distance of the pair of codewords with smallest Hamming distance* $(x, y, d*)$ *[49].*

When one error occurs, the codeword is Hamming distance 1 from the transmitted word. Hamming *weight* W(c) of the codeword c is equal to the number of nonzero components in the codeword.

To detect more than one error or to correct an error, we need a code with a distance of more than 3. If error occurs t-times and the distance from the received word to every other codeword is larger than t, then the decoder will correct errors properly [8]. In other words, $2t + 1 \leq d*$, where $d*$ is the minimum distance.

As explained earlier in Example 4.4.2, from $H \cdot r^{tr}$ we get a syndrome which shows that the codeword is erroneous and where the error is located. This depends on the minimum distance d^* in the code. In Example 4.4.2, the minimum distance is 3, $d^* = 3$. We are able to correct single errors, because $d^* \geq 2t + 1$.

Codes that correct all single errors, i.e., have $d^* \geq 2t + 1$, are called *Hamming codes* [8]. Hamming codes also detect all single and double bit errors.

4.5 Linear Cyclic Block Codes

Cyclic codes are easy to encode and have a well-defined algebraic structure, which has led to the development of very efficient decoding schemes. Bose-Chaudhuri-Hocquenhem (BHC) codes are a subclass of binary cyclic codes and are capable of correcting multiple errors [41].

4.5.1 Polynomial Representation of a Code

A polynomial can be represented as

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x^1 + a_0 x^0$$

where a_n is the leading coefficient if $a_n \neq 0$. $f(x)$ has degree n. Consider polynomial $a(x) = x^3 + x + 1$ which is an example of a polynomial over $GF(2)$ with degree 3. A polynomial over $GF(2)$ has coefficients where each term is either 1 or 0. The polynomial $a(x) = x^3 + x + 1$ represents the message [1011] in binary form.

Each element in $GF(2)$ is it's own inverse. This means that addition has the same effect as subtraction. It is not possible to construct a $GF(2^m)$ using $modulo - 2^m$ multiplication. This is possible with polynomial arithmetic. Polynomials with prime field coefficients require only modulo arithmetic for operating on these coefficients. These characteristics make implementing the code easier.

Definition 4.5.1 *An (n,m) linear block code C is a* ***cyclic code*** *if every cyclic shift of a codeword in C is also a codeword in C [8].*

A codeword can be represented as a polynomial:

$$C(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1}.$$

Components of the codeword are symbols in $GF(2)$. A linear code is a subset of $GF(q^n)$. To identify parity bits and information bits, we can represent the codeword like this :

$$C(x) = i_1 + i_2 x + \cdots + i_m x^{m-1} + p_1 x^m + \cdots + p_r x^{n-1},$$

where i = information bits, p = parity bits, and $r = n - m$ is the number of parity bits.

BCH codes are generated by the *generator polynomial* $g(x)$ [37]. $g(x)$ has to divide $x^n - 1$ (n = the degree of the polynomial). That is, every codeword C is a multiple of the generator polynomial $g(x)$.

A generator polynomial $g(x)$ consists of minimum polynomials α, where a minimum polynomial is an element of $GF(2^m)$ of order n. There are methods to calculate minimum polynomials by hand, but the easiest way is to use prepared tables, e.g., [41]. Minimum polynomial α is primitive, and the root is of order $n = 2^m - 1$.

Example 4.5.1

A generator polynomial has the form: $g(x) = \alpha_1(x)\alpha_3(x)\ldots\alpha_{2t-1}(x)$. Where $\alpha_1 \cdots \alpha_{2t-1}$ are minimum polynomials. m is the degree of a polynomial. For $m = 4$, $\alpha^3 + \alpha^2 + \alpha = 1110$ is in binary form and $\alpha + 1 = 0011$.
End of example 4.5.1.

A *primitive polynomial* is a *prime polynomial* over $GF(q)$ having a primitive element as a zero. Zero stands for root. If $a(r) = 0$ and $r \in GF(q)$ and $(x - r)$ is a factor of $f(x)$, then r is a root [37]. When a polynomial is not divisible by any other polynomial of degree one through $m-1$, it is said to be irreducible. Any irreducible polynomial over $GF(2)$ of degree m divides x^n+1 if $n = 2^m - 1$. If this value of n is the smallest positive integer for which $p(x)$ divides $x^n + 1$ then $p(x)$ is a primitive polynomial of degree m over $GF(2)$.

A simple encoding rule [8] for polynomials is $C(x) = i(x)g(x)$, where $i(x)$ are the information bits in polynomial form. In decoding, the received polynomial $r(x)$ is the sum of transmitted codeword polynomial $c(x)$ and the error polynomial $e(x)$. *Syndrome polynomial* $s(x)$ is calculated by dividing $r(x)$ by $g(x)$.

Linear cyclic codes can be described as a matrix form with a set of tuples like parity check codes. We can say that the generator polynomial $g(x)$ is equivalent to the generator matrix G. The parity check polynomial $h(x)$ is equivalent to the parity-check matrix H.

4.5.2 Reed-Solomon Coding

Reed-Solomon (RS) Coding is an error correction technique for data communication, storage systems, and mobile computing. As earlier discussed in Chapter 3, bursts of errors are the most common errors in satellite communication and can cause consecutive TDUs losses in computer communication. RS is suitable coding for surviving TDU losses because, unlike simple Hamming codes, RS operates on blocks of multiple bit symbols rather than single bits. With RS, we can compute multiple independent parities for the same set of TDUs.

An RS code that is able to correct t-errors has the following parameters:

1. *Block length* : $n = q^m - 1$
2. *Number of parity-check digits* : $n - m = 2t$
3. *Minimum distance* : $d^* = 2t + 1$

Since one byte is 8 bits long, it makes sense that the finite field k is of degree 8 over Z_2, so that each element of k corresponds to a single byte [14]. Typically, RS code has symbol size $m = 8$, which means that the symbols are elements of the Galois Field $GF(2^8)$. The maximum codeword length will then be $2^8 - 1 = 255$. This corresponds to 235 message bytes, 20 bytes of coding symbols, and the codeword is 255 bytes long [40].

In the next section, the work of McAuley [31] is studied. McAuley's article introduces Reed-Solomon coding in simplified form. This article is investigated here, since it gives us basic information used later on in this thesis.

4.5.3 Reed-Solomon Erasure Correcting Code

It is preferable to choose the simplest FEC code that matches to the requirements because coding uses processing time and reserves. Conventional Reed-Solomon code is computationally demanding, therefore, simpler coding methods are suitable e.g., for broadband networks. McAuley describes a simplified Reed-Solomon erasure correction coding architecture. This is a frequently used coding algorithm, and it is presented in many variations among many articles described in Chapter 6.

McAuley discusses why FEC is the best alternative for error correcting in broadband communication and points out the main reasons. He mentions three basic techniques which are the same as Carle [11] points out: 1) pure ARQ, 2) pure FEC, 3) a hybrid method that utilizes both ARQ and FEC.

One of the main motivations behind FEC is that when the existing copper circuit switched networks are replaced with broadband fiber optic networks, the failures and errors also change to different types of errors. There will be a reduction in link errors. In addition, in packet switched networks, cell losses due to congestion are going to be the dominant error type. FEC is suitable when a return channel for acknowledgments is not available or it is too slow.

McAuley compares FEC features to ARQ and concludes that in the increased communication bandwidth, ARQ is not capable of meeting all of the requirements. For example, it is difficult to manage large tables of retransmission timers. Real-time applications transmitting, e.g., over a transcontinental link, do not approve the retransmission delay. ARQ also has problems with supporting multicast when the number of receivers is large. This causes further errors due to congestion that is caused by retransmissions. These are the contentions why FEC is the best alternative for ARQ in broadband communication.

McAuley presents block codes and convolutional codes briefly. The reason why McAuley focuses on Reed-Solomon Erasure (RSE) code is that FEC hardware is computationally demanding. McAuley's proposal is capable of detecting and correcting errors with the same parity information.

RSE code creates enough parity bits to enable the receiver to correct data without retransmission. The same hardware can be used for encoding and decoding. Unlike in conventional RS, the least significant k symbols of the codeword are set equal to the k information symbols in RSE. This structure of the codeword does not change the methods of how computing in encoding and decoding is done. Compare the codeword representation in Section 4.5.1 on page 28.

Example 4.5.2 *A codeword can be represented as a polynomial*

$$C(x) = c_{n-1}x^{n-1} + cn_2x^{n-2} + \cdots + c_kx^k + i_{k-1}x^{k-1} + \cdots + i_0$$

where the information in m-bit numbers is represented as polynomials:

$$I(x) = i_{k-1}x^{k-1} + i_{k-2}x^{k-2} + \cdots + i_0$$

End of example 4.5.2.

McAuley points out seven RSE characteristics: 1) In burst correcting, RSE is capable of correcting burst up to the number of redundant symbols sent. 2) RSE can detect errors and correct erasures. The ability to do this is related to the redundant information amount sent and received. 3) The amount of redundancy and original data is possible to adapt to the current need in the channel. 4) If the channel is reliable, redundant data can be minimized. 5) Low latency in RSE is possible because decoding can begin as soon as one block with symbols without errors is received. 6) High throughput over 1 Gbit/sec in 1 micron CMOS is possible. 7) RSE is not complex to implement.

McAuley mentions that less complex hardware implementations for error correction exist, such as Fire codes or interleaved convolutional codes. However, either they require long latency or have modest burst correction capabilities.

Chapter 5

Redundant Arrays of Inexpensive Disks

Redundant Arrays of Inexpensive Disks (RAID) is a system where we can use inexpensive smaller disks arranged like an array instead of one large disk. RAID ideas are applicable to software implementations as well as hardware. In this chapter, we introduce the basic ideas of RAID and how it is built up. We give an overview of how error coding can be used with different RAID systems, and we review failure possibilities in RAID. In the last section, some articles among RAID are investigated. Related works concerning the issues of interest in this thesis are considered.

One basic idea behind RAID is to connect multiple disks and program them to operate as a single unit. As Chen [13] points out, microprocessor performance has increased more than disk transfer rates, and it is assumed it will continue with the same trend. Arranging smaller disk drives (e.g., disk drives used with personal computers and micro computers) into an array gives us the possibility of utilizing parallelism. We get higher transfer rates because of multiple I/O capabilities, higher I/O rates, increased disk capacity, and faster data access [4, 25]. Hellerstein [22] concludes that future I/O systems will contain a large number of disks. Disk arrays not only improve performance and add capacity, but also provide a level of redundancy[1] to the I/O path in the event of a drive failure [4]. This is the other basic idea in RAID, which is discussed in the next Section.

[1]The term *Redundant information* means coded data that is used for checking and reconstructing original data. The term parity check is used when coding is performed to produce parity, for example XOR. The term *check disk* is used as a name for disks where the redundant information is located.

5.1 Data Redundancy and Striping in RAID

The difference between disks arranged in parallel and RAID is that in RAID is stored redundant data. With redundant information, we understand a copy of a data element that is stored on another disk in the disk array. Redundant data can also be generated by coding error control information, e.g., by simple XOR calculation. Coded redundant data is also called parity information. As explained earlier in Chapter 5, this parity information can be used to restore original data. See closer description about parity connected with RAID below and in Figure 5.1.

Striping, in which data is sequentially arranged over disks, can be utilized in ordinary arrangement of parallel disks. In RAID, redundant information can also be striped. RAID can be arranged in different ways where striping and coding varies. This gives further divergence in performance and reliability. Therefore, RAID has been classified in levels ranging from 0 to 6, in order to distinguish between the different organizations. The terminology of RAID is explained with the help of Figure 5.1 where RAID level 5 is used as an example. All levels are explained in more detail later in this chapter.

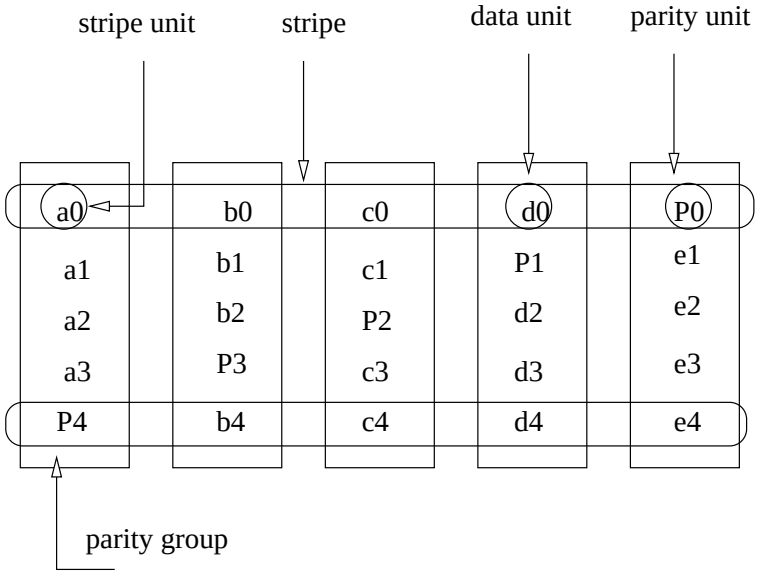

Figure 5.1: Terminology

Data Striping: Real-Time Encyclopedia [38] defines data striping as

> "Segmentation of logically sequential data, such as a single file, so that segments can be written to multiple physical devices (usually disk drives) in a round-robin (algorithm in which

processes are activated in a fixed cyclic order) fashion. This technique is useful if the processor is capable of reading or writing data faster than a single disk can supply or accept it. While data is being transferred from the first disk, the second disk can locate the next segment."

Striping means simply interleaving a body of data across two or more disk drives. With only two drives, the first "chunk" of data is written to drive 1, the second chunk is written to drive 2, the third chunk is back on drive 1, and so on.

Stripe: If we have 5 disks, for example, then a stripe is a set of 5 stripe units which all have the same physical address on each disk.

Stripe unit: A unit of data which represents a group of logically contiguous blocks that are placed consecutively on a single disk before placing blocks on a different disk [28].

Parity and Data units: A parity unit is one stripe that includes only parity information. A data unit includes only original information.

Parity group: a set of parity units and data units. Parity is computed from the data units in the same group. A *parity stripe* can be understood as a minimum set of data and parity stripes covering a given parity group [28]. Coding theory for redundant data and parity has been explained earlier in Chapter 5. Hamming coding and Reed-Solomon are examples of codes used for calculating redundant data in RAID

Use of striping and storing of redundant data results in better data availability and the possibility of recovering from disk crashes. Parity check codes can be used to improve integrity of data in RAID systems. This can be done by calculating parity of data information in one stripe and adding calculated parity information to a parity disk. Adding parity data to the disk array makes it possible to recover easier and faster from disk failure, since missing data can be reconstructed using parity information.

As an example, we can think of video-on-demand where each movie is spread out over multiple drives. Disk management software places movies on the magnetic disks and handles disk requests for the output stream [49]. Striping allows multiple I/O requests to be served in parallel because the information is distributed across all disks. If one of the disks fails, it is possible to reconstruct the movie segments on this disk with the help of parity information.

5.2 Various RAID Organizations

In order to operate the various methods in the coding and striping, RAID has been arranged into different levels ranging from level 0 to 6. The different RAID levels are introduced in this section. An overview of the RAID levels is given in Tables 5.1 and 5.2. These tables have been collected with the help the following references: [2, 4, 25].

Level	Striping and FEC	Performance	Cost	+	-
0. Data Striping	Block Interleaving, **no redundancy**	high	low	• High performance , • No cost penalty • No parity calculation overhead is involved • Very simple design • Easy to implement	• No Fault Tolerance. • Significantly reduced data availability • The failure of just one drive will result in all data in an array being lost
1 . Transparent Disk Mirroring / Shadowing / Duplexing	Duplicate of the data, **no redundancy**	medium/ high	high	• Excellent data availability • Higher read performance than a single disk	• Expensive - requires twice the desired disk space • Moderately slower write performance
2 . Striping and Shadowing	Stripes data to a group of disks using a bit stripe. A **Hamming code** symbol for each data stripe is stored on the check disk.	high	high	• Excellent data availability • High performance • “On the fly” data error-correction	• Requires twice the desired disk space • no significant advantages over RAID-3 architecture. • requires very high transfer rate requirement to justify
Level	**Striping and FEC**	**Performance**	**Cost**	**+**	**-**
3 . Bit Interleaved Data Striping with Parity Checking	A striped parallel array where data is distributed by bit or byte. ***One drive*** *in the array provides data protection by storing a parity check* byte for each data stripe.	medium	medium	• Good data availability • High performance for transfer rate intensive applications • Cost effective - only 1 extra disk is required for parity • Disk failure has a low impact on throughput	• Can satisfy only 1 I/O request at a time • Poor small, random I/O performance • Transaction rate equal to that of a single disk drive at best (if spindles are synchronized)
4 . Block Interleaved Data Striping with Parity Checking, Independent Data disks with shared Parity disk	Parity is interleaved at the sector or transfer level. Is i*dentical to RAID-3 except that large stripes are used, so that records can be read from any individual drive* in the array apart from the parity drive.	medium	medium	• Good data availability • High performance for read operations • Cost effective - only 1 extra disk is required for parity • Low ratio of ECC (Parity) disks to data disks means high efficiency	• Poor write performance • Poor small, random I/O performance • Difficult and inefficient data rebuild in the event of disk failure

Table 5.1: RAID levels 1 - 4

Level	Striping and FEC	Performance	Cost	+	-
5 . Block Interleaved Data Striping with Distributed Parity. *Independent Data disks with distributed parity blocks*	Combines the throughput of block interleaved data striping of RAID-0 with the parity reconstruction mechanism of RAID-3 without requiring an extra parity drive. *Both parity and data are striped across a set of disks.*	medium	medium	• Good data availability • High performance in request rate intensive applications • Cost effective - only 1 extra disk is required • allows multiple concurrentread /write operations for improved data throughput • Low ratio of ECC (Parity) disks to data disks means high efficiency	• Poor write performance • No performance gain in data transfer rate intensive applications • Difficult to rebuild in the event of a disk failure • Most Complex controller design
6 . P&Q redundancy *Independent Data disks with two independent distributed parity schemes*	The same method as in level 5 but with ***double parity*** , **Reed-Solomon** Data is striped on a block level across a set of drives,	medium	high	• An extremely high data fault tolerance • Can sustain two simultaneous drive failures	• Very bad write performance • Requires N+2 drives to implement, because of second parity

Table 5.2: RAID levels 5 and 6

5.2.1 Zero Level RAID

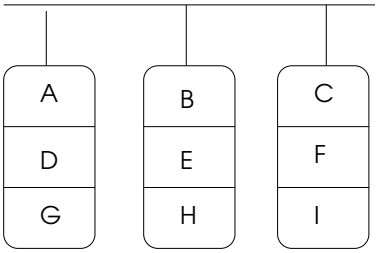

Figure 5.2: RAID Level 0

Raid level 0 is an array of non-redundant arrangement with *block-level striping*. This means that data is interleaved across disks in blocks of arbitrary size rather than in bits [13]. As Figure 5.2 presents, a data ($Data = A + B + \cdots + H + I$) is broken down to smaller blocks and each block ($A + B + \cdots + H + I$) is written to a separate disk drive [2]. With this solution, we can reach the best write performance; but without redundancy, any disk failure will lead to data loss. This solution is not actually a true RAID because it is not fault tolerant and has no parity calculation.

5.2.2 First Level RAID

At RAID level 1, we have one check disk for every data disk. This is an easy, but expensive, way to arrange redundancy. All disks are duplicated as shown

in Figure 5.3. This is why RAID level 1 is also called *mirroring*. Mirroring can be used e.g., in places like database systems where transaction rate is not important but availability is [13, 25]. In mirroring, a single disk controller is used, and when one disk partition fails, information from the other disks can be used to continue the I/O operations.

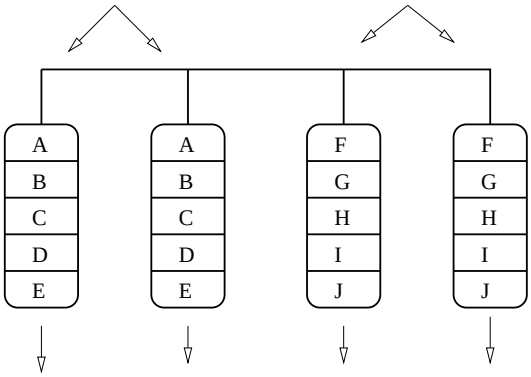

Figure 5.3: RAID Level 1

In mirroring, when data is written, a duplicate of the data is made and written to a check disk automatically. The nature of this arrangement allows two reads or one write at the same time. Another method for RAID level 1 is duplexing. In duplexing, separate disk controllers are used, and when one disk or controller fails, data from the other system can be used continually [4]. This level is designed to protect against a single disk failure, not multiple ones. If both the original and the copy fails, the data on these disks can be lost.

5.2.3 Second Level RAID

The basic interest in second level RAID is to reduce overall cost. This RAID level is also known as striping and shadowing. This RAID level is sometimes referred to as RAID-0+1. One method used here for calculating parity information is Hamming coding. The parity information is stored to check disks [13]. Data is striped to groups or disks using byte striping [4].

At this level each data bit is written into data disks and each Hamming coded word is recorded to the check disk [2]. This can be viewed from Figure 5.4, where $Data = A_1 + A_2 + \cdots + E_2 + E_3$ designates data, and redundant information located on the check disks is designated with $eecA + \cdots + eecE$. As shown in the figure, each write and read operation spans all drives. A single check disk can detect one error, but if we want to correct errors we

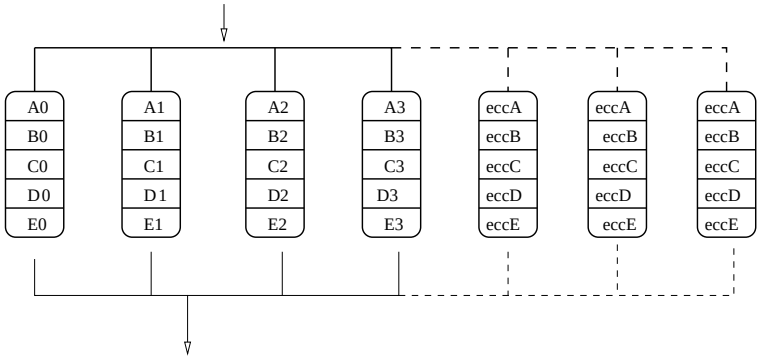

Figure 5.4: RAID Level 2

have to have enough uncorrupted disks to locate the disk with error. Thus, this level is rarely used [25].

5.2.4 Third Level RAID

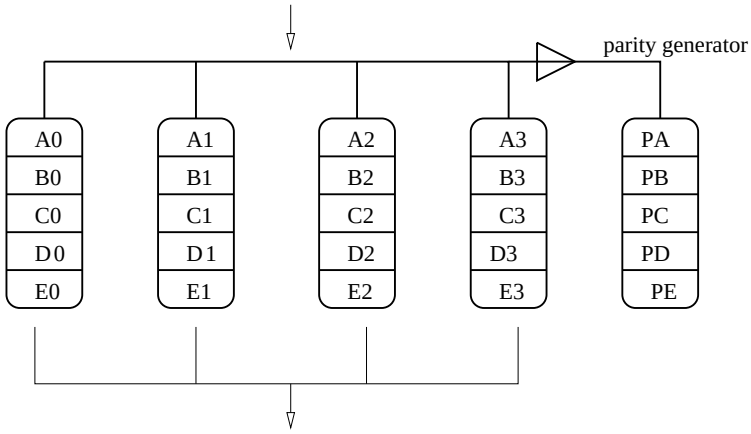

Figure 5.5: RAID Level 3

Third level RAID interleaves data in a bit- or byte-wise order over disks [13]. To tolerate single disk failures, one parity disk is added. As Figure 5.5 illustrates, every write has to access all of the data disks and the parity disk [13]. To maintain data integrity with each write, a unique parity check is calculated and written to the parity disk. The bottleneck of third level RAID is the parity on a single drive [25].

Disk controllers can identify in which disk an error has occurred. In the case of a read, parity information is checked. A disk array manager should

run a parity check daily and for example, after an abnormal shutdown, to ensure that data and parity information are correct and to guarantee data integrity.

Advantages at this level are high write and read data transfer rates [2, 4]. When one disk fails, data can be reconstructed while the system is functioning by reading parity and reading/writing to the other disk drives, as long as the parity drive does not fail. If drive 1 fails, for example, we use drives 2, 3, 4 and the parity drive to restore drive 1. Therefore, availability of data is good.

RAID level 3 is suited for large data objects but does not work well for transaction processing systems [25]. RAID level 3 can satisfy only one I/O request at a time, and parallel accesses therefore decrease data transfer time for long sequential records [4].

5.2.5 Fourth Level RAID

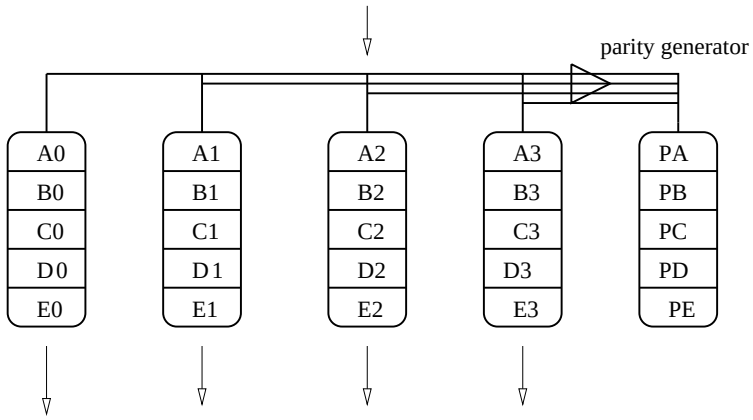

Figure 5.6: RAID Level 4

The fourth RAID level has the same main idea as level 3, but instead of striping data in bits or bytes, data is interleaved in blocks. When using large stripe units, it is possible to read records from any drive in the array apart from the parity drive. This means that independent reads are possible. As Figure 5.6 presents, it is possible to have high read transfer rates, but writes are slow. In Figure 5.6, data is denoted : $Data = A_o + A_1 + \cdots + E_1 + E_2 + E_3$ and parity for A data blocks $= PA$. Parity for B, C, D and E is marked in the same way as parity for A. A high read rate is feasible because multiple read requests can be served parallel [4]. I/O requests need only to reference the drive where the required data is stored.

Write operations at the fourth level are slow because the parity drive has to be updated whenever data is written or modified. All other disks have to be read for parity calculations also. Therefore, rebuilding of the data to one parity disk is quite inefficient. This is the bottleneck of the RAID level 4 [2, 4, 13]. In the next section, we introduce the fifth RAID level, which has a better solution for write updates.

5.2.6 Fifth Level RAID

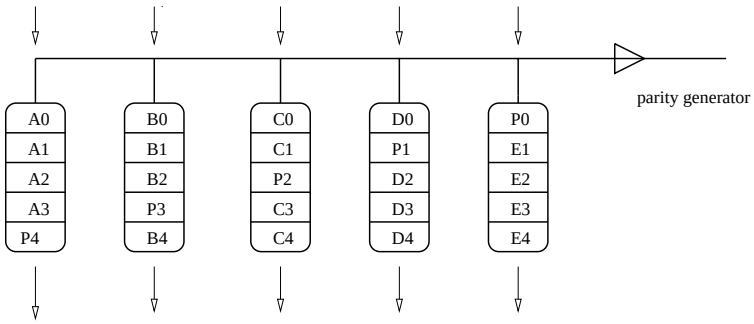

Figure 5.7: RAID Level 5

The fifth RAID level, also called the Block-Interleaved Distributed-Parity, cuts down the bottleneck problem mentioned in RAID level 4, because this level does not have only one disk dedicated to parity. Parity information and data are uniformly distributed over all disks and data blocks are much larger than the average I/O size [4].

Parity is marked with an *P* in Figure 5.7(compare with the Figure 5.6). The new parity for write is derived as: $P_{new} = P_{old} + D_{old} + D_{new}$. This means that one update requires four disk accesses. This manner of updating is similar to the fourth level. The bottleneck of this level is the updating of parity information when many small writes occur. As Figure illustrates, parity distributing is *left-symmetric*, i.e., parity is distributed symmetrically over disks from the lower left to the upper right corner. In this parity arrangement, each parity block is accessed separately. The advantage is the reduced possibility of disk conflict while serving large requests [13].

The fifth RAID level provides a high level of fault tolerance because each parity block contains information to reconstruct the information in the other blocks of data. Parity data is never stored on the same drive as the data it protects. This means that concurrent read and write operations can be performed. RAID level 5 can survive even if one disk fails, but performance

falls below an acceptable level. The major performance weakness at this level is therefore all of the small writes which must take place every time parity information is updated [4, 13].

5.2.7 Sixth level RAID

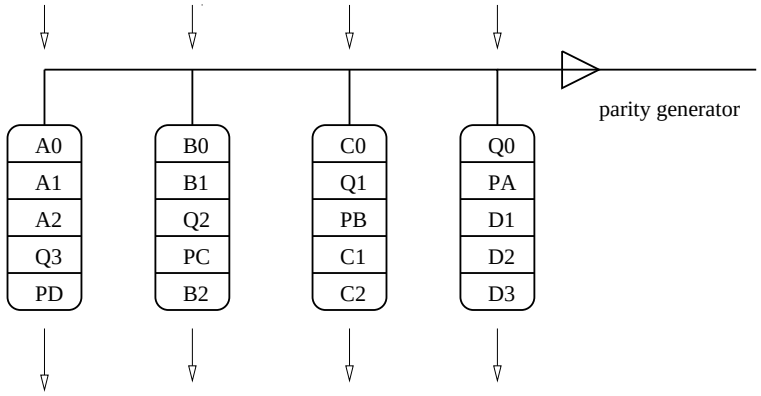

Figure 5.8: RAID Level 6

The sixth RAID level is also called P&Q Redundancy. P and Q denote error redundant information which is doubled compared to the fifth level. P is parity information calculated *vertically* over one disk, and Q is parity information calculated *horizontally* over all disks. As shown in Figure 5.8, $Q0+\cdots+Q3$ is Q redundancy, and $PA+\cdots+PD$ is P redundancy. This means that the sixth RAID level uses a second distributed parity scheme. None of the parities are located in the disk they protect. Because of this structure, the sixth RAID level can sustain performance through multiple simultaneous drive failures [2]. P&Q redundancy resembles the fifth RAID level, and they operate in a quite similar manner, but the difference is in the amount of disk access required when parity information must be updated for both P&Q [13].

The sixth RAID level uses Reed-Solomon codes to achieve better recovery from disk failures [13]. This implementation uses 2^4 bit symbols to provide double-erasure correction for up to 13 data disks [19].

5.3 Error Coding in RAID

As discussed earlier in Chapter 3, the typical sources of failure for disks are system crashes and individual disk crashes due to software errors or electricity problems, for example. Chen [13] describes how bit-interleaved disk arrays are affected less by system crashes than block-interleaved disk arrays.

In bit-interleaved disk arrays, the inconsistent parity can only affect the data that is currently being written. In a block interleaved disk array, an interrupted write operation can affect the parity of other data blocks in a stripe that is not written. System crashes can be similar to disk failures in the way that modified parity stripes can be lost during a system crash. System crashes can occur more frequently than disk failures. Recovery from a system crash requires regeneration of the parity sectors that are known to be inconsistent.

Failures and errors do not always occur individually and by one at a time. There is a possibility of having a disk crash and a system crash simultaneously. Disk failures at the same time as a system crash can cause an uncorrectable situation: a situation similar to two disk failures. To illustrate, P&Q information is invalidated by a system crash. Therefore, parity protected disk arrays are not necessarily better than conventional disks. However, P&Q redundancy is useful against multiple correlated, i.e., simultaneous disk failures. Double disk failure or one disk failure together with bit errors are also possible. In addition, it is not possible to avoid system crashes without increasing performance expenses. Non-volatile storage for parity information is one possibility which can be utilized to avoid loss of parity information.

The most common methods of coding in RAID are Hamming coding and N+1 coding. Hamming coding is explained in Chapter 4.4, and N+1 parity coding is described by Gibson [19] as follows:

> "Each codeword contains one bit from each of N data disks and one bit from a single redundant data, or parity disk. The value of the bit from the redundant data disk is equal to the exclusive OR of bits. Therefore if a disk is erased a single identified bit is lost from each codeword. The value of the bit is the value necessary to make the parity of the data bits equal to the value stored in the parity disk."

Which error coding we should choose depends on our goals. We might want to have a solution which always has all data available and uses more disk space, or we might want a solution which uses more complicated coding, less place, and hence more time for error correction. If we use parity coding, it becomes less costly than for example mirroring where more disks have to be bought to ensure data integrity.

5.4 Related Work to RAID

In this Section, we investigate proposed RAID methods in related work and evaluate whether or not they can be utilized in INSTANCE. A summary of

each article is presented in Table 5.3. The number of each article investigation refers to the number in this table.

The following papers have concentrated on improving the typical problem areas for RAID e.g., bottleneck of parity disks or parity updating. The related work about RAID propose new methods to improve RAID level 5 or to make new techniques similar to level 5 as article number two does [47]. In article number four [9], a method is proposed to decrease redundancy, yet still be able to recover from two disk failures.

1. Hot mirroring : A method of hiding parity update penalty and degradation during rebuilds for RAID 5

Mooi et al. [33], has combined two RAID levels for different needs in performance and storage efficiency.

In this scheme the mirrored storage space is characterized by high performance efficiency and low storage efficiency. The RAID level 5 region, is characterized by low performance and high storage efficiency. Data blocks migrate from and are mirrored to RAID level 5 and vice versa. In other words, data is not constantly stored in one RAID level. This is done because access frequency varies from time to time.

Comments: In concluding remarks for this article, we can point out that the method introduced is not inside the range of the type of solution we are looking for in this thesis. This is because mirroring without parity information is not suitable for our error correcting goals in this thesis. However, the basic idea of hot mirroring could be used with a slightly modified scheme. For example, for different types of error scenarios and type of data we could choose the RAID level with parity that is most appropriate. In another words, more error prone transmissions and sensitive data are stored to the RAID level that has better reliability or other features like FEC block size. However, the data should constantly be stored in one type of RAID level. This is because the client that receives the data has to be aware which coding algorithm and FEC block size is in question at the time.

2. Parity Logging Overcoming the Small Write Problem in Redundant Disk Arrays

Stodolsky et al. [47], has a solution, parity logging, to the small write problem for RAID level 5. This article focuses on one special problem area. Parity logging is proposed to resolved the small write bottleneck. In this method the small updates that produce only small amounts of new parity are “logged”, or stored until enough parity information is gathered for larger updates.

Comments: The article does not have relevance to INSTANCE if we consider the FEC aspects in this thesis. The article does not propose any methods e.g., for striping that could be utilized in INSTANCE. However parity logging itself could be utilized as a method for better performance in RAID only.

3. Parity Declustering for Continuous Operation in Redundant disk Arrays

Holland et al. [23], examines a parity-based redundancy scheme called parity declustering. The method introduced by Holland et al. focuses on how to keep RAID on-line during disk crash and reconstruction, and having minimal impact on system performance. An airline reservation system is one example of a system that has to be on-line continuously. The goal he has is to minimize reconstructing time and minimize the affects of reconstructing. An example of such an affect is a load imbalance during one disk crash when all other disks besides the crashed disk are accessed.

In parity declustering the number of parity stripes, i.e., a logical array, is arranged into a larger physical array. In a physical array one stripe is larger than a stripe in logical array. With this arrangement, one stripe can include several stripe units (data or parity units), from a logical array. The benefit from this arrangement is that not every disk is involved in the reconstruction of a parity stripe. Uninvolved disks stay idle during this time.

The parity declustering algorithm is suitable only for systems that have to stay on-line. For other systems, the conventional method is more effective.

Comments: The idea of having RAID functioning on-line during disk failure is also interesting in INSTANCE. Since we intend to store multimedia data, the continuity of data availability for transmission is important. The minimizing of recovery time from disk crash also has a benefit in INSTANCE. The algorithm for striping does not have an impact to the FEC requirements on transmission. The redundant data is available in the disks as in conventional RAID.

4. Fast, On-Line Failure Recovery in Redundant Disk Arrays

Holland et al. [24], introduces a disk oriented reconstruction algorithm in his study. This method is investigated in two environments: RAID level 5 and declustered parity. Declustered parity was first introduced by Holland et al. [23].

The reconstruction process is associated with with one disk instead of

stripes. This means that as many reconstruction processes can be started as there are disks. Unlike parity stripe oriented reconstruction algorithm, in a disk oriented reconstruction algorithm, more than one stripe is buffered for the reconstruction algorithm. This requires more buffer place than the conventional parity stripe algorithm.

Comments: The method proposed by Holland et al. does not differ in the requirements we have with our study in this thesis. Since this method is an improvement on the parity declustering discussed, we come to the same conclusions in the previous article investigation.

5. EVENODD: An Optimal Scheme for Tolerating Double Disk Failures in RAID Architectures

Blaum et al. [9], introduced a coding method for RAID that tolerates double disk failures.

The EVENODD algorithm can be implemented in software or in hardware depending on the application. EVENODD can be used in other applications where there it is necessary to correct two erased symbols with low complexity. The scheme Blaum et al., used is implemented on RAID level 4. However, it is possible to distribute parity to all disks, like in RAID level 5, to avoid the write bottleneck that is typical for fourth level. He points out that EVENODD can be implemented on standard RAID-5 controllers without any hardware changes.

EVENODD is capable of correcting two disk failures, therefore it needs two disks for redundant information. The calculations for parity are done horizontally and diagonally with simple XOR calculations. This calculation scheme has influence from the sixth level RAID introduced earlier in this chapter. The example in the article is introduced with seven and five information disks plus two parity disks. However, Blaum et al. points out that the procedure also works for disks with arbitrary capacity.

Blaum et al., has compared EVENODD to traditional parity based schemes and Reed-Solomon based schemes. The advantages of EVENODD are : 1) It is possible to implement with standard parity hardware. 2) It is available in e.g., fifth RAID level. He points out that RS schemes require special hardware to support a finite field type of computations.

Comments: This article is a good example of how reliability can be increased without complicating the parity computation methods. However, in this thesis, we have to consider how this method could be utilized in this thesis. The parity calculation, whether it is XOR or RS, is not the first thing

we have to address: Determining which stripes are involved in parity calculations is even more important.

In our approach to INSTANCE, it is important that the parity data is also easily accessed. As earlier explained in Chapter 4, the FEC block that is transmitted over the channels has some requirements. Because of the receiver side decoder, it is required that the coding environment does not change. In other words, the receiver side needs to know the size of FEC block in the encoder, and have the same coding method. This means that RAID cannot change methods on the fly, even if it would create a more efficient storage system and error recovery possibilities. The striping problematic is discussed closely in Section 7.1.1.

6. Combined RAID 5 and Mirroring for Cost-Optimal Fault Tolerant Video Servers

Biersack et al. [7], has proposed combining RAID level 5 and mirroring to create an effective data storage system for multimedia purposes.

RAID level 5 and mirroring are combined in this solution. The new RAID can be used for multimedia purposes. The reason for this arrangement is that the RAID level 5 is not effective enough during the disk crash for a video server. The services are reduced during the disk crash because of reconstructing processes. The reconstructing processes reduce the availability of data.

To avoid a reduction in service level, Biersack et al., has proposed two classes of video objects: hot and cold. For example, hot video objects are stored mirrored and cold video objects are stored on RAID level 5 style. The logic behind this arrangement is that popular movies belong to the hot class, and less popular movies are in the cold class.

The hot class movies are made easily available since the direct copy can be sent to the client in case of a disk crash. The clients that access movies in the cold class, must wait while reconstruction coding takes. However, the less popular movies are not accessed as often and the process of reconstructing does not include as many clients as in the hot class accesses.

Comments: The proposal to combine mirroring with the RAID level 5 is an appropriate solution for video servers. However, in our approach, we need to send redundant information in the transmission also. In mirroring, only a copy of the original data is stored. Therefore, mirroring does not provide parity information. For our purpose, this kind of arrangement is not suitable because we need the parity information in transmission FEC.

NR	FEC, Algorithm	Problemarea	RAID level	Basic Idea	Conclusion
1	• FEC not explained	A method of hiding parity update penalty and degradation during rebuilds for RAID 5	• mirrored • level 5	Storage space is partitioned into two regions: • high performance and low storage efficiency. • RAID 5 region - low performance, highstorage efficiency	Hot block clustering in hot mirroring achieves higher performance than conventional RAID 5 arrays.
2	• XOR with some modifications, does not consider more powerful codes	To reduce the cost of small writes.	Mirroring, floating storage and level 5	Journaling techniques to reduce the cost of small writes. Comparing mirroring, floating storage and level5 with parity logging	Better performance than level 5.
3	• disk-oriented reconstruction • parallel stripe-oriented • XOR	Fast, On-Line Failure Recovery.	• declustered parity • raid level 5	Disk oriented reconstruction instead of stripe oriented.	When utilizing arrays excess disk bandwidth they get improvement in failure recovery time with a small degradation in user response time during failure recovery.
NR	**FEC, Algorithm**	**Problemarea**	**RAID level**	**Basic Idea**	**Conclusion**
4	• parity declustering	How to get higher user throughput during recovery and/or shorter recover time.	• new scheme compared to level 5 and mirroring	Declustered parity organization for balancing cost against data reliability and performance during failure recovery. Reduces the additional load during reconstruction of disk.	Proposal improves standard parity organization.
5	• EVENODD • XOR	How to tolerate double disk failures without a lot of redundancy .	• extension of level4 • can be done as level 5	Simple parity scheme. (x2. because of two parity disks) Parity on the two last disks. Capable of correcting two erasures.	Advantage is that we need only parity hardware.
6	• combined parity based (or level5 i.e RS) with mirroring	How to make a video server tolerant against all single disk failures.	• Level 5 + Mirroring	When combining level 5 and mirroring how to achieve 100% service availability when operating with a failed disk with lowest possible cost	Proposals for how to gain I/O bandwidth during failed disk operation is given.

Table 5.3: Related work to RAID, articles 1 - 6

The next two articles introduce other issues that include some information related to our work:

Coding techniques for handling failures in large disk arrays

Gibson et al. [22] has discussed how parity calculations are done in one dimension and in two dimensions. A typical two-dimensional method is introduced in RAID level 6, where P&Q redundancy is calculated in a two-dimensional way. Two-dimensional parity is correcting two erasures. Erasure is used in this paper, because his failure model is erasure. Since disk con-

trollers identify the crashed disk, it is said to be erased.

Codes introduced by Gibson et al. are able to protect against catastrophic disk failures e.g., head crashes or controller electronics failures. However, the codes presented do not protect against data loss due to failure of power, cabling, memory or processor.

Comments: This article introduces the idea behind the dimensions in parity calculations and how they can be used. However, we cannot utilize the benefits from two-dimensional parity in INSTANCE. Reasons for this are discussed in Chapter 7.

A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems

Plank [36] presents RS coding in RAID-like systems. With RAID-like systems, Plank means both network file systems and fast distributed checkpoint systems as well as RAID. Planks also includes an introduction to the RS algorithm and some good examples of it in practise in his paper. He also briefly introduces some other coding methods, like one and two-dimensional parity.

Comments: This tutorial is easy to understand and gives a simplified explanation of how RS coding can be implemented. This paper gives basic information directed towards the systems programmer. However, the reader has to have some basic knowledge of coding theory.

5.5 Comparison of Related Work

A brief overview of the issues that were discussed in the last Section is given here. Most of the related studies in RAID focus on reducing the affects of the bottlenecks that are specific to each RAID level. These types of studies do not usually have direct relevance to our work in this thesis. Some articles create new methods by combining different RAID levels, like [7] and [33]. As discussed earlier, these combined RAID systems are not appropriate in our approach if they do not include parity information. These are the reasons why these studies seldom have useful methods that can be utilized in this thesis.

In INSTANCE, we must have a clear possibility to read the parity information from the disks. This creates some restrictions to striping and to the stripe units. Parity information has to be suitable for transmission protocols, and the availability of parity information must not be prohibited. Some adjustable parity calculations like adaptive RAID methods, as in [33], are not

preferable due to transmission of parity information. Only methods such as parity declustering, that do not interfere with requirements in in this thesis, can be considered. These issues are elaborated in Chapter 7.

Usually, the related studies focus on the RAID levels that have the possibility to be improved the most. Because of the typical bottlenecks with parity disk, RAID levels two and four are not better solutions than RAID level 5. Because RAID levels one and three can be viewed as subclasses of RAID level 5, it is natural that the studies concentrate on RAID level 5. This is a fact which is also pointed out later in Chapter 7.

Chapter 6

Overview of Error Correction in Communication Protocols

In this chapter, an overview of the possibilities to detect and correct errors in data transmission are given. In the first section of this chapter, we introduce various communication protocols and how they manage failures and error situations. In the last section, an overview of error coding methods is given by investigating related works among FEC in transmission. Each approach is evaluated with respect to its capability to detect and correct errors.

6.1 Transmission Protocols and Error Correction

In computer networks, protocols are used to establish and maintain connections, or control connectionless traffic between applications. These applications can be located in the same network or distributed over several locations in different networks. Among these protocols are Transmission Control Protocol (TCP), User Datagram Protocol (UDP), Asynchronous Transfer Mode - Adaptation Layer (ATM - AAL), eXpress Transport Protocol (XTP), and Transmission Protocol++ (TP++).

The demand for error correction is ruled by the QoS level needed in each case where data is to transmitted. For example, consider transmitting of real-time data: the transmission does not have to be fully reliable. Real-time data, e.g., audio and video, only needs to get to the receiver in the right order and without delays. Protocols that detect errors can be adequate for real-time data transmission. As discussed earlier in Chapter 3, a telephone conversation is understandable even with small breaks caused by errors.

Protocols which are able to detect errors but cannot correct them are usually used for data transmissions that do not require guaranteed delivery. One of these protocols is **UDP** which is a connectionless transport protocol. UDP

delivers data in less demanding applications and is used for data that does not need highly guaranteed delivery because it provides very few error recovery services. Data reliability is defined by the protocol, not by the application. UDP can be used to distribute audio or video streams to several receivers simultaneously, but does not offer any error control other than checksum tests.

Checksums are computed for each frame. The simplest way is to use XOR calculation to create parity bits. The checksum is calculated at the receiver again. Incorrect parity reveals the corrupted data. In UDP, a packet, that fails the checksum test is discarded with no retransmissions.

Some protocols provide better data integrity by requesting retransmission of corrupted data. For example, XTP uses checksums and has the ability to require retransmission if needed. **XTP** is a high performance transport protocol designed to meet the needs of distributed, real time, and multimedia systems in both unicast and multicast environments [5]. XTP has the ability to use burst and rate control dynamically. It can control the size of data to be sent and how much data is sent at a time.

Error control, flow control, and rate control are each configured to the needs of communication in XTP. XTP supports three types of error control: 1) the fully reliable mode, which is used in applications like file transmission, 2) UDP like unacknowledged services, and 3) fast negative acknowledgments which also uses a fastnack option that identifies an out-of-sequence delivery immediately. Without the use of time-outs, missing data is retransmitted immediately. XTP has another option when carrying digital voice and video. This option is called noerror mode. This mode suspends the normal retransmission scheme. The first packet is sent with a flag that instructs the receiver to always report that no data has been lost. In XTP, a user is allowed to define whether or not a transmission is acknowledged, and lost TDUs are retransmitted. Choices for retransmission in XTP are Go-Back N and selective retransmission (see Section 3.2.3).

In XTP, detection of errors is accomplished by sequence numbers, length field and checksums. XTP packets carry two checksums. XTP uses the two checksums over the XTP packet contents to verify the integrity of the data received through the network. Erroneous packets are discarded and retransmitted.

Retransmissions are also used in **TCP**. Like XTP, TCP has both Go-Back N algorithm for providing high data integrity. TCP is designed to be fully reliable and to adapt to various properties of the Internet. TCP provides error free data transmission. This is achieved by using packet retransmission which use the Go-Back N algorithm. This is why TCP is not capable of the

unrestricted flow that is needed in, for example, real-time systems. TCP uses sliding window protocols, timers and checksums to provide reliability. For example, congestion control is regulated by manipulating the window size.

Compared to TCP, XTP has many more possibilities for transmission of data, but neither XTP nor TCP provide FEC for quick error recovery to gain data integrity. At the present, it is more realistic to assume that the physical transmission of data is basically reliable. This is why Strayer et al. [48], point out that it is not very actual to have FEC system available in XTP. However, XTP's noerror mode is not adequate in all situations when transmitting real-time data, especially if the channel has a tendency to become congested due to low bandwidth or a high traffic rate. Protocols which can provide FEC are of interest because of their relevance for INSTANCE. For example, TP++ and AAL protocols have been examined with various FEC models.

Like XTP, ATM was designed to be suitable for fiber optic networks, which are highly reliable. Therefore, error control is left to the higher layers [49], e.g., for Adaptation Layer (**AAL**). Four protocols in the AAL layer AAL1, AAL2, AAL3/4 and AAL5, are made to support various services for requirements. For example, services supported by AAL 5 are: reliable, unreliable, unicast and multicast. AAL 5 is capable of detecting lost, wrongly inserted or missing cells with CRC. In addition to CRC, AAL has ARQ to guarantee data integrity.

So far, none of the discussed transport protocols in this chapter have FEC as a standard. The next protocol introduced, **TP++** [18], also has FEC as an error correction method. TP++ is a transport protocol for multimedia applications that operate across heterogeneous high-speed networks. TP++ is capable of carrying voice and video, small transactions in distributed systems and larger amounts of data also. TP++ is designed to handle data loss caused by congestion and packet disorder caused by multipath routing [18].

The FEC code used in TP++ is RSE code [31], which was introduced earlier in Section 4.5.3. TP++ uses FEC for error correction and ARQ for retransmission of lost data. The error detection code is calculated with the help of polynomials using the parity principle. The minimum Hamming distance is four, and the code is capable of detecting all errors that change less than four bits [18]. In the next section, research and papers that combine FEC coding with different transmission protocols are investigated.

6.2 Forward Error Coding in Transmission

The error correcting methods for transmission were briefly introduced in Section 3.2.3. A closer examination of FEC coding was given in Chapter 4. Instead of going back into detail concerning FEC correcting methods in transmission, we investigate papers which describe FEC in action with transmission protocols.

These articles study e.g., the flexibility of methods and possibilities to combine FEC with other methods in transmission. The articles investigated in this section are summarized in Tables 6.1 and 6.2. These tables give a review to the main topics: in which environment and what problem areas are solved with FEC.

6.2.1 Related Work about FEC in Transmission

The study of related work in this section are focused on: 1) environments, 2) problem areas, 3) problem solving, 4) FEC coding methods, and 5) conclusions that are described by authors. Whether or not the methods or design goals proposed in the articles can be utilized in our approach are also discussed.

1. Reliable Broadband Communication Using a Burst Erasure Correcting Code

McAuley discusses why the best alternative for error correcting in broadband communication is FEC and introduces the RSE coding method. This work was investigated earlier in Section 4.5.3: therefore, only comments are introduced here.

Comments: This article describes basic information about the error correcting possibilities in current transport protocols. The focus rests on ATM type networks. McAuley [31] compares ARQ to FEC and points out the major benefits that FEC has. Further, McAuley specifies the recommended characteristics in RSE and FEC on broadband channels. Since RSE coding technique is widely used in other articles, we consider how it suits our purpose. There are basically not any major hindrances preventing the integration of RSE in RAID and TP. However, we have to consider some details like stripe size compared to cell size; and how these details can be realized in implementation. Further discussion about this issue is presented in Section 7.2.1.

2. Performance Evaluation of Forward Error Correction in an ATM Environment

Biersack [6] has studied the performance of FEC in an ATM environment. In this paper, he first gives an explanation of ATM and explains which the typical errors exist. These are bit and switching errors and cell losses. Switching errors occur due to corrupted cell headers. Biersack points out that cell losses due to congestion are more common than the other two types.

Because of the disadvantages ARQ has (e.g., not applicable for low latency sensitive traffic), Biersack has studied FEC as an alternative for ARQ. According to Biersack, the use of FEC presents two effects in data transmission. The first effect is unavoidable redundant information, also called overcode. The second effect is an overall load due to increased overcode, which increases the cell loss rate. These two antagonist effects are often a starting point for further discussion and testing in many other studies. For example, how much redundancy it is possible to add to the transmission before it causes congestion and FEC becomes meaningless.

The algorithm used in Biersack's experiments is RSE. Each decoded cell includes a sequence number to be able to identify which cells are missing. A characteristic of RSE is that original cells are not modified and are transmitted regardless of redundant cells. Therefore, decoding delay is minimal if all original cells are received. Biersack studied various traffic scenarios where video (with a variable bit rate) or burst sources (bulk data or transactions) are transferred either homogeneously or heterogeneously, i.e., video and bulk data together. The heterogeneous scenario includes 24 video sources and eight burst sources.

Biersack points out the performance of several scenarios and concludes that the homogeneous scenarios were not effective with FEC. This is due to findings from the video transmission scenario: the increased value in load due to FEC also increases the block loss rate. When there were many FEC sources the load was also increased. This causes an increase in the block loss rate and gain becomes low or even negative.

In bulk data transmission, the high levels of overcode cause significant cell loss rates and makes FEC ineffective. With other words, the cell loss rate depends on the burst rate. This scenario has the same type of behavior in gain as the video transmission scenario. Biersack recommends an ARQ based loss recovery scheme for the retransmission of the lost messages. However, Biersack found that FEC is effective for heterogeneous traffic scenarios and that gain from the FEC sources varies with the load, the amount of over-code and the relative number of sources using FEC. The cell loss rate and block

loss rate were moderate in the heterogeneous scenario. When the overcode amount was high or the load was low, the block loss rate was lowest. However, increasing the amount of overcode beyond 20% from the original data did not give better results.

Comments: Biersack presents basic facts and information about errors in ATM type networks. The comparison of ARQ and FEC is also valuable when designing TPs. Biersack has given useful scenario findings such as results that give the limit for redundant code in transmission. These results can help design the FEC code for RAID and TP. We have to address the problems in choosing the parity information ratio to original data. This is because, we have to avoid congestion and simultaneously increase TDUs in the transmission.

3. Forward Error Correction Control on AAL5 : FEC-SSCS

Kanai et al. [27], present a novel FEC model for ATM AAL called FEC-Service Specific Converge Sublayer (FEC-SSCS). They introduce a FEC scheme that uses variable size packets and can easily modify the redundant data length, even during session.

The article compares TCP/IP packets to ATM cells and concludes that an ATM type network is a kind of packet network. If TDU from TCP/IP networks is sent over an ATM type network, it is divided into smaller units: cells. Therefore, one single cell loss can cause a retransmission of the whole TDU. Kanai et al., point out, how serious a problem this is for high speed data transmission services with small latency requirements. Especially in multicast services, a packet error due to bit errors will linearly increase with the number of receivers.

Kanai et al. mention other issues that can degrade performance in data transmission. These are unexpected congestion and poor data transmission performance for packet flow with a large delay and bandwidth product. Unexpected congestion happens when end-stations and switch nodes misbehave or when traffic fluctuates. The motivation for the study in this paper is to reduce the cell loss and the loss of larger TDUs in TCP/IP type networks at the same time.

Kanai et al. also discuss where FEC should be located in the protocol stack. The advantage of having FEC at the AAL level comes from the error correcting capability in bit errors and cell loss. Modification of upper levels is therefore unnecessary. If the FEC scheme was at the physical level, only bit errors would be corrected, which would not be adequate. However, they point out that this scheme could be applied at the application or physical level.

Kanai et al. describe sender and receiver behavior. They do not discuss the FEC algorithm closely. However, they do mention that the error detection is done with CRC. If the detected error is located on redundant TDUs, no further FEC calculations are made. Therefore, this method spares processing time and effort because no unnecessary FEC calculations are performed.

In the performance evaluation, Kanai et al. found out that even if FEC does cause some increased latency, the results are better with FEC than without. Especially if a larger amount of redundant cells were used, they got better latency characteristics even at large cell loss ratio. The efficiency was also dependent upon the parity information ratio in the FEC scheme. However, the network cell loss pattern gives a guideline for choosing the most appropriate amount of parity information.

Comments: This article presents a good example of the dilemma which can occur also in our approach. The dilemma is: how should the packet size be adjusted to fit the transmission data units? In our approach the question is: how should one adjust the stripe size to suit the TDU size in transmission? For example, the original TDU size can be equal to one stripe unit in RAID. If large data units are used in RAID, we have to divide them into smaller units for transmission. Therefore, this dilemma is comparable to the dilemma introduced by Kanai et al. This issue will be discussed in greater detail in Chapter 7. Another useful method proposed in this article is the use of CRC to recognize lost cells. No processing time for FEC is used if all of the cells are received.

4. A Cell Loss Recovery Method Using FEC in ATM Networks

Ohta and Kitami [34] propose a new cell loss recovery method to be applied to Virtual Paths (VP) of ATM networks. The problem area they preview is cell loss due to congestion. They focus on consecutive cell discards. Ohta and Kitami discuss the cell discard process and compare cell loss methods applied to Virtual Circuits (VC) and VPs. They have proposed a method to avoid discarding consecutive cells due to buffer overflow. They present two approaches in cell loss recovery methods: End-to-End and Node-to-Node.

In the End-to-End approach, i.e., from terminal equipment to terminal equipment, cell loss detection is accomplished using sequence number checks. An FEC decoding/encoding delay is large with this method and requires an installing processing circuit for each terminal. This is why Ohta and Kitami introduce the Node-to-Node approach in their paper, which is described below.

Effective cell loss detection is important in the Node-to-Node approach, because lost cell position information transmission is required. This approach reduces the average decoding/encoding delay due to high speed processing and allows VCs to share the FEC equipment. Because of the advantages of the Node-to-Node approach, Ohta and Kitami introduce an efficient cell loss detection and recovery method for VPs.

The calculation for FEC coding is based on a matrix where all cells belonging to a specific VP are grouped. A single parity check code is applied with erasure correction which minimizes encoding/decoding delays. A Cell Loss Detection (CLD) cell is used to detect cell loss. CLD also has its own parity cell. CLD cells and parity cells have the same VPI and length as data cells. Original data remains unchanged.

Ohta and Kitami give a detailed description of the reconstruction of the lost cells. Information fields that identify different cells, Cell Recognition Patterns (CRP), are used in the coding of the cells. In original data regenerating, modulo 2 calculation is used. If the result is 0, then no cells are missing. Otherwise, the result is the value of the discarded cell.

The evaluation that Ohta and Kitami describe, is given for cell loss rate improvement, reductions of bit error influence, and coding/decoding delay. One of the interesting observations is that the encoding delay time is small because cells do not have to be buffered. Only CRP's and parity cells are stored. They also noticed that large matrices were preferable if VP occupied a large portion of its assigned transmission line capacity. In addition, they point out how cell loss could be reduced with the methods for bit correction mentioned above.

Comments: This article is not directly relevant to this thesis. This is mainly because FEC in the VP's only is not adequate. We need an FEC that is possible to attach to the terminal equipment, i.e., RAID and receiver, and not only to the VP. In addition, this article points out the problem how adding redundant information to the transmission increases the risk of congestion and thereby the loss of consecutive cells.

5. Adaptive Error Correction for ATM Communications using Reed-Solomon Codes

Almulheim et al. [1], has experimented with Reed-Solomon coding to correct errors within ATM type networks.

Almuheim et al. point out that ARQ is not suitable for satellite links where retransmissions are too expensive. They also refer to another study that ar-

gues for the advantages of FEC with multicast [31]. However, they remind how conventional FEC [31, 34] can cause added processing overhead in ATM communication. This is why they introduce a new FEC technique that is based on RS codes.

Almuheim et al. have focused on RS coding that can adapt to the channel capacity and characteristics. This method can adapt to the requested QoS level and adapt to the level of congestion and noise. In the adaptive RS code they have three parameters instead of the conventional two. $RS(n,k,l)$ where n is original data + redundant data size. k refers to the redundant data size and l is the measure of information symbols dropped from the original $RS(n,k)$. The reasoning behind this idea is to send only $k-l$ cells. This method has two two benefits. The first is increased protection against errors, because the n to k ratio is relatively small. The second benefit is the decreased congestion problem when less information is sent. Also, the end-to-end delay associated with transmitting a block is reduced.

Almulheim et al. introduce the adaptive FEC. The algorithm in this code is RS code which is similar to McAuleys[31] code. The performance analysis is done by comparing the adaptive FEC to conventional FEC. One of the performance measures was end-to-end delay encountered by a block of data. The observation they made was that the conventional FEC had a stable end-to-end delay, while the larger blocks of data in adaptive FEC had an effect delay. The smaller the data block is, the larger l becomes. Therefore, end-to-end delay decreases and correcting capability increases. The worst delay for an adaptive FEC was equivalent to the delay with conventional FEC. The conclusion is that the versatile adaptive FEC can decrease the effect of congestion and increase channel utilization.

Comments: In our approach to FEC, we use the parity information that is encoded on the disks. Almulheim et al. propose an FEC method that can adapt to the channel conditions. This is a good feature since we can reduce the parity information send on the channel. Hoverer, the question is: is it possible to encode the required amount of data in the RAID system in advance? Date encoded into RAID in advance would need complicated coding, since the parity information would be updated whenever the channel capacity changes. We can reason that this solution would cause a great deal of coding effort in RAID. Therefore, RAID in INSTANCE prohibits the kind of adaptivity that is described in this article.

6. Block Loss Reduction in ATM Networks

Srinivasan et al. [44] describe the metric block loss rate in real-time applications and high-level protocols e.g., IP that use ATM as a transport mecha-

nism. Srinivasan et al. point out why single cell loss from a consecutive ATM block may cause the loss of the entire block. Srinivasan et al. have studied two approaches to the block loss problem. The first method is to set high priority the types of messages that get through first. The motivation is to limit cell loss for high priority customers at the expense of low priority customers. The second method is to use FEC. The code utilized in this paper is the same RS burst erasure code as McAuley [31] presented.

Srinivasan et al., introduce a hybrid block loss reduction scheme that utilizes a selective discard mechanism, which is a system for discretionary selection of the cells in the queue. Srinivasan propose a solution that combines both of the approaches, priority and FEC. The cells are market with high or low priority before they are sent to the channel. High priority means that the session does not use FEC. High priority cells arriving to the full buffer can push out cells with lower priority. High priority cells are lost only if the buffer is filled with only high priority cells.

To avoid the problems in high traffic loads, Srinivasan et al. introduce a buffer management policy called Adaptive Pushout (ADP). This policy accounts for correlations in cell loss behavior. A low priority cell is dropped from the full buffer if it is still possible to gather one FEC block. This method works with principle which tells how many cells can be lost and yet code all original data in one block with the help of redundant cells. The subsequent cells from a block are discarded if too many cells are already lost, i.e., one FEC block is lost.

According to Srinivasan et al., the best result in simulations were with 30% redundant information when all sessions used FEC. This is because the cell is lost only in unavoidable situations. They also observed that when the number of sessions using FEC is larger, the cell losses do not occur consecutively, but are spread out. This is exactly the improvement to the problem they focused on in this paper.

Comments: The method Srinivasan et al. describes is useful for transmission with an extra quality guarantee if we want to have the choice to send data with or without redundant information. Hoverer, the method ADP, proposed by Srinivasan et al., is not relevant to the methods we are interested in RAID and transmission. As earlier mentioned, an adaptive FEC coding method is not the most appropriate choice for us.

7. Providing Support for Data Transfers in a New Networking Environment

Schatzmayr et al. [42] has pointed out the benefits of flexibility when adapting different QoSs in networking environment. They introduce a new transport protocol: RAPID. RAPID is capable of adapting to different user needs in data transport and runtime environment facilities.

The reasoning behind the new transport protocol RAPID is the development of physical mediums, large bandwidths and low noise levels. In addition, the emergence of multimedia applications with different requirements than before is giving new challenges to transport protocols. Specifically, the diversity of multimedia data characteristics requires adaptivity from the transport protocol. These requirements can be e.g., desired error rate or timing constraints that are limiting retransmissions.

Schatzmayr et al. point out the typical problem areas in transmission that are also discussed earlier in this chapter. If the network's reliability degree must be improved, four FEC algorithms can be introduced in the protocol. These are algorithms from simple XOR type calculations to more complicated RS coding. The idea with many different algorithm choices is that the simplest algorithms are less bandwidth consuming schemes. The FEC algorithm is processed in one processor. FEC and data transfer protocols can be processed independently.

Comments: RAPID is able to adapt its mechanisms to different requirements. We could choose one of the FEC mechanisms and use it in our approach. However, it would be against the principles behind RAPID to tie up the transport protocol to follow one profile. The basic idea of RAPID is the adaptivity and the freedom to change the requirement profile. It is better to have only one simple transmission protocol with FEC first, to see how the integration of FEC with RAID work. Then it is reasonable to study whether the transport protocol can be designed to be more flexible like RAPID is.

8. Adaptive Error Correction to Support Heterogeneous Multicast Groups

Dresler et al. [15] has proposed a method to combine IP with FEC. Because TCP does not support real-time transmissions and multicast, they have conjuncted IP with UDP. Since UDP and IP do not have error correction, FEC is included.

For this purpose they propose Adaptive Layered FEC. Different streams are sent for different receiver groups in this method. For example, one data stream with FEC and another data stream without FEC. The receiver can de-

cide which receiver groups to join. Receivers can also change group if needed due to channel conditions or FEC requirements.

The redundant information is computed from the user data in IP packets. The most suitable coding for this purpose is XOR or RSE. It is possible to adjust the amount of redundancy with actual code in RSE, which is a more advanced code. Dresler et al. propose an RS(255,251) code where four redundant packets are computed (see Section 4.5.2). They also point out that it is possible to shorten the code class by adding virtual zero packets to the user data before encoding takes place. By virtual packets they mean zero packets that are not sent. The receiver reinserts zero packets before encoding. This method has an advantage for transmissions that do not tolerate much of an encoding/decoding delay such as audio and video transmissions.

Comments: Adaptive layered FEC is suitable for systems that can tolerate some packet loss but not retransmission delay, like audio and video transfers. Dresler et al., also propose a methods for less powerful receivers where local retransmissions are used. This method is not considered in this thesis and is thus, not described here. Hoverer, the idea of a receiver selecting the an adequate group for the transmission type with or without FEC could be utilized in INSTANCE. RAID systems can contain different types of data like audio, video or text. It is not necessary to have full reliability for all types of data in this case. Some receivers can afford longer delays or some packet losses and can choose the appropriate receiver group. How this could be implemented in INSTANCE is an interesting question. This will be discussed further in Chapter 7.

9. Adaptable Error Control for Efficient Provision of Reliable Services in ATM Networks

Carle [11] proposes two new ATM adaptation layer protocols for real-time and multicast requirements. The motivation behind this research is to avoid high cell loss rates and to design protocols for an adaptation layer that provides the necessary QoS needed.

The framework introduced by Carle is adaptable for cell loss rate. It should be possible to choose a suitable error correcting scheme for each cell loss rate in this framework. For light traffic and low cell loss rate, Reliable Lightweight Multicast Protocol (RLMCP) is proposed. Error correcting is performed with retransmissions in this protocol. Retransmissions are performed either with the go-back-N or the selective repeat method. It is possible to change the frame size to suit different traffic loads. RLMCP does not have FEC or checksums.

The Reliable Multicast ATM Adaptation layer (RMC-AAL) is proposed for high cell loss rates. This protocol utilizes the benefits of ARQ and FEC. This protocol has three schemes for error recovery. It uses either pure ARQ, ARQ and FEC or only FEC. Carle proposes encoding and decoding that is either done with XOR operations or with an algorithm that is based on Reed-Solomon codes [31]. Carle describes a performance evaluation for simulating multicasts and concludes how the results are useful for selection of an error control scheme and for dimensioning of frame sizes.

Comments: The three possible schemes in RMC-AAL can be compared to other protocols like XTP or TP++. However, Carle points out that TP++ does not have multicast possibilities. The protocols proposed in Carle's paper focus on the adaptivity of the systems. We can conclude with the same comments regarding Schatzmayr et al. [42]. The method introduced by Carle is quite specialized for different purposes that only small parts of it would be appropriate to utilize. It is therefore easier to utilize simpler transmission protocols for our purpose.

10. Priority Encoding Transmission

Albanese et al. [3] introduce a new method, Priority Encoding Transmission (PET), for transmitting data over lossy packet-based networks. The focus in this paper is to find out how to avoid packet losses due to congestion and buffer overflows.

In PET, a message is partitioned into segments. Each segment has a priority value that specifies the fractions of packets sufficient to decode it. For example, in video multicasting, they show how priority values can be used. The example describes how an MPEG stream consists of different types of frames: I, P, and B-frames. A frame that has the highest priority I-frame can be displayed independently. The next priority A-frame needs information from the I-frame to be displayed properly. B-frames refer to the I and A-frames.

Comments: This method is presented only briefly because it does not have direct similarities to the FEC we are looking for. As we can understand, this method includes the coding possibility directly into the packets. If we weight the needs of our system including striping in RAID, the priority based FEC can become complicated. The problem is how to separate different frame types in RAID. Calculations for disk reconstruction can be difficult if different types of frames are stored in their own stripes and are furthermore dependent on each other. This method needs a careful storage plan for the different types of frames. We can conclude that PET is not suitable for our purpose but is more specialized for specific services.

11. A multicast Transport Protocol with Forward Error Correction for Satellite Environments

Linder et al. [29] present a new transmission protocol for multimedia applications in satellite environments called the Restricted Reliable Multicast Protocol (RRMP). This protocol is intented to be used for multimedia applications like video or voice conferences or electronic newspapers, that do not need strict reliability, but do need some level of control of the lost packets. This new protocol is mainly used for satellite communication, but it is also suitable for heterogeneous networks.

The starting-point for this study is packet loss in satellite channels due to noise. They point out how the signal is more error prone to bit failure in longer distances because of long delays and weather conditions. They present various error correction possibilities, like ARQ, that are dependent upon the QoS offered by the transport layer or the network layer.

FEC code in satellite channels is typically convolutional code, such as Viterbi code: however, the algorithm used in this paper is based on RSE code. This code is able to correct as many erasures as the amount of redundant information sent. To recover one lost Transport Data Unit (TPDU), they have proposed an interleaving technique. This algorithm is based on the partitioning of a Transport Service Data Unit (TSDU) into segments and chunks and interleaving them. This allows recovery of the complete TSDU if other TPDUs are received completely. The knowledge of the beginning of the TSDU and the position of the TPDU within the TSDU, is restored in the header of each RRMP TPDU.

This method is capable of specifying the amount of redundant control information and does not have services like ARQ. Therefore, it does not need a backward channel for acknowledgments. It is preferable for FEC to be dynamically adjustable, since changing weather conditions are the major error source for satellite channels. The user is able to adjust the amount of data compared to error correction information.

RRMP was tested with different TSDU sizes and error rates. Loss rates for TPDUs and TSDUs as well as the number of corrected TSDUs were counted. They found out that code with a large ratio of original data/redundant data is not recommended for usage over satellite links. When they increased the TSDUs size, the losses of them increased. This was caused by increased length of RRMP TPDUs sent. Their experiments show that the throughput with RRMP is comparable to TCP on high delay channels: at the same time the redundant information ratio was hold relatively low.

Comments: The method proposed by Linder et al. is specially suited for multimedia, video conferencing, and satellite communication. However, the interests in this thesis remain on protocols that do not use ARQ.

12. The case for FEC-Based Error Control for Packet Audio in the Internet

Bolot and Vega-Garcia [10] describe a method to reconstruct lost audio packets. The environment described in this paper is a network that does not guarantee bandwidth or performance measurements and do not therefore support real-time applications. For example, networks such as Internet do not provide support for real-time applications. The problem is how to distribute real-time audio packets over these networks without packet losses. Bolot et al. is especially interested in the transmission of audio data and minimizing the effect of delay jitter and packet loss. A method in which the application adapts to the best effort service currently provided by the network is proposed in this paper. Therefore, delay variance and loss characteristics are guidelines for audio encoding/decoding adaptation.

Bolot and Vega-Garcia give an overview of several FEC possibilities. The simplest way is to replace lost packets with silence. An even better solution is to copy the previous packet. They also introduce the XOR operation that can solve a single loss in one group of XORed packets. Redundant information was obtained by using LPC, GSM or 2-bit ADPCM coding. For every lost packet, the receiver waits for the next packet and uses this to reconstruct the lost one. However, this method is capable of recovering only isolated errors. To avoid this problem, they propose to add redundant information in packets or packet repetition.

Bolot et al. observed how it is likely to have packet losses due to high traffic at specific time within a twenty-four hour period, and what the length of these loss periods. They found out that loss period is usually one or two packets. Measurements in multicast are essentially the same as those for unicast.

Comments: The method in which Bolot et al. propose coding for audio combines two different types of coding: Linear Predictive Coding (LPC) and Pulse Code Modulation (PCM). PCM is a speech compression and decompression algorithm and LPC is a compression algorithm used for low bit rate (2400 and 4800 bps) speech coding. These coding methods for audio are not related for us because we want to find the most appropriate FEC method for all data types, not only for audio data.

13. Reliable IP Multicast Communication over ATM Networks Using Forward Error Correction Policy

Esaki and Fukuda [16] introduce problems and problem solving issues among IP multicast. They give detailed description of packet error/loss probabilities and how to better QoS with FEC. This paper focuses on the problems that appear when trying to provide error free multicast and how to provide a large scale multicast when there are many receivers, because error loss probability is larger when the number of multicast receivers is large. Esaki and Fukuda also present problems that arise when receivers and senders are widespread.

The issue that is focused on in this paper is the increased probability of observed packet error/loss. With observed packet error/loss they mean accumulated IP packet errors or losses. The probability increases linearly according to the number of destination processes. Esaki and Fukuda propose an end-to-end based FEC policy in the ATM cell level as a solution for this problem.

The remainder of this article discusses the performance evaluation of IP multicast over ATM networks. They found out that applying FEC dramatically reduces the IP packet error or loss probability. They also point out that the FEC frame length should be aligned with the IP packet length. IP multicast with an FEC policy is successful because the probability of retransmissions is sufficiently small.

Comments: This paper shows that FEC with IP is possible and that it is capable of providing reliable transmission for multicast also. The FEC code utilized in this paper is RSE. This means that this code can offer reliability with different protocol combinations. The ideas proposed by Esaki and Fukuda merely show the way what the possibilities are for FEC in the protocol stack. The use of FEC in INSTANCE does not depend on the aspects that were pointed out about multicast. We are not primarily interested in multicast solutions.

14. Error Correcting Codes for Satellite Communication Channels

Chen and Rutledge [12] focused on FEC for satellite channels. The errors they wish to correct are double bit errors, produced by the differential decoder. Because single bit errors are highly unlikely, the code does not correct them. Chen and Rutledge carried out probability calculations from the error patterns to find sufficient code.

Calculations to the theory of modulation are established in this paper.

They also give some information about how modulation techniques are connected with FEC. They preview a Quadri-Phase-Shift Keying modulation (QPSK), that is a modulation technique for satellite communication channels. A class of code that suites this purpose are Convolutional codes, such as codes that are based on the truncated parity check matrix.

Comments: This paper introduces the traditional method for FEC with satellite communications. However, this method utilizes convolutional coding and is out of our interest range.

6.2.2 Summary of Related Work

We can recognize the difference between computer networks and telecommunication systems. FEC is traditionally used in telecommunication and applications for multicast. As Rizzo [39] emphasizes, the telecommunications world was first in developing FEC policies. The interest is to correct relatively short strings of bits and implement them dedicated hardware. The benefits of redundant information can be achieved easily. It is also logical to use FEC in telecommunication, as it is usually comprehended as non-reliable. FEC is added for better QoS.

Related works often point out the basic reasons for the use of FEC. In telecommunication, noise is the main cause for error. Noise that causes packet losses in computer networks can be compared to congestion. ATM networks experience three types of errors: a) bit errors which corrupt the data portion of a the cell header, b) errors due to connected corruption of the cell header, and c) cell loss due to congestion [6].

Many works emphasize how congestion in ATM communications is a significant issue, as it can have a dramatic effect on critical or real-time data. Cell loss due to congestion is the main problem in transmission. Therefore, most of these articles focus on correcting only erasure and not occasional bit errors. Since an ATM type network is usually used for real-time transmissions, ARQ is not suitable as an error correction method. Therefore, among related works, FEC often stands in focus as an error correcting method added to AAL.

One example of the use of FEC in telecommunications is satellite links. The drawback for satellite channels is attenuation. However, this problem can be solved with FEC. Unlike satellite links, analog links do not have the same possibility for error correction. Another usage that the articles focus on is FEC with multicast environment. FEC is in many cases combined with ARQ. This proves to be effective method.

We can find some similarities with all research done in related work in Chapter 6. Much of the research concentrates on ATM and improving the capabilities of AAL5. Reed-Solomon is proposed in many ways to guarantee data integrity. In their works, strategies to avoid unnecessary redundancy with e.g., adaptivity with new methods are proposed [15, 27, 34, 42, 44]. Adaptivity was seen as a questionable feature for the use in this thesis. This is basically because RAID includes FEC parity information.

Parity information in the approach for this thesis is not calculated only for the needs of a receiver at any given point in time. Therefore, parity calculation updating for adaptive needs leads to complicated implementation for RAID. The only possibility for adaptation in integrated FEC in RAID and TP, is either to choose to send redundant data or not. In other words, the receiver has the possibility to choose whether to receive redundant information or not. However, if redundant data is not sent, we are forced to implement a re-sending strategy for transmission. Retransmission of data is not the basic idea in this thesis. Data transmission without FEC is suitable for channels that are reliable enough. Otherwise, FEC is included in data transmission.

Another similarity besides adaptivity is the FEC code proposed in related works. For example, Srinivasan [44] and Stoc [46], among others, have utilized the same algorithm McAuley [31] described in his paper. McAuley's code is designed for burst erasure and has a simplified algorithm. This is originally the code that is designed for the TP++ [18]. RSE coding is also a coding algorithm worth consideration in this thesis. Referring back to Chapter 5, RS is also used for error correction in RAID. The implementation possibilities for RSE and RS are discussed more closely in Chapter 7. The useful methods from the related work we can point out the two works [6, 31] which gives a basic information about the FEC in data transmission.

We can summon from earlier comments that protocols which are designed to be used in certain areas, like adaptivity and service diversity, are not suitable methods for integrated FEC in this thesis. One of the reasons is the data type we wish to store on RAID. One single strictly specialized transmission protocol is not capable of transmitting all data types. In addition, far developed adaptivity prohibits the parity functionality in RAID. When multimedia is in question, we need a transmission protocol that is flexible in handling many types of data or many TPs that are specialized in one data transmission type.

NR	FEC Algorithm	Error to Correct	Environment	Basic Idea	Conclusion	Other Issues
1	Simplified Reed-Solomon erasure correction code RSE	Error detection and correction of burst erasures due to congestion	broadband network TP++	Tries to find out how RSE works in a broad band network.	Points out how to provide adequate service with RSE in broadband network, and also describes weak nesses of this solution.	hybrid scheme: FEC + ARQ
2	Reed-Solomon burst erasure correcting code	burst erasures, cell loss	ATM-type network, video	Original + redundant decoded information is sent to recover cell loss	The loss behaviour depends on the statistics of the source and the traffic scenario	multiplexing,traffic scenarios, congestion
3	CRC for header detects errors, FEC algorithm recovers data part, RS ?	bit and/or cell loss	ATM, AAL level	FEC scheme at cell level instead of packet level. Reduce data transmissions -> CRC. Variable length packets	Sufficient throughput and latency performance with reasonable transmission overhead.	CRC,variable length user data, multicast
4	CREG-VP, a consecutive cell loss recovery method, parity, dummy cells, matrix	cell loss detection , lost cell regeneration	ATM, VPs virtual paths	New cell loss recovery method to be applied to virtual paths, reducing coding/decoding delays and facility sharing.	This method reduces the cell loss rates of virtual paths in ATM networks.	multimedia, B-ISDN, cell discard process
NR	**FEC Algorithm**	**Error to Correct**	**Environment**	**Basic Idea**	**Conclusion**	**Other Issues**
5	Versatile FEC control, based on Reed-Solomon codes	cell loss detection , lost cell regeneration	ATM	FEC that can be adapted to the level of congestion and noise.	Versatile FEC provides an effective throughput higher than with conventional FEC.	real time data
6	Reed-Solomon , RSE ADP, adaptive pushout mechanism: buffer management	cell loss from ATM block, cingle loss from IP datagram	ATM networks, IP	Priority based cell discarding and FEC, How to avoid IP packet losses.	ADP perform remarkably well with FEC. ADP account for correlations between cell losses.	ATM networks block loss rate
7	RAPID transport protocol, depends on network's error characteristics: Reed-Solomon , parity XOR, XOR matrix	Depends on type of environment and QoS	Does not depend on one specific, ATM AAL5	Flexibility to adapt different QoSs	Unidirectional data transfer capability and the ability to adapt its mechanisms profile to different transport user needs and runtime environment facilities.	implementation aspects, protocol architecture
8	XOR /RSE used in Adaptive Layered - FEC scheme.	packet losses	realtime and non-realtime IP multicast	To combine IP multicast with FEC, receiver can choose to have FEC	AL-FEC can adapt dynamically to the current network situation or change user preferences	subgrouping for local error recovery
9	XOR /RSE	packet losses, cell loss on bursty channels	ATM, Reliable Multicast ATM AAL	Adaptable framework for error control.	Performance evaluations are useful for the selection of the error control scheme.	

Table 6.1: Related work, articles 1 - 9

NR	FEC Algorithm	Error to Correct	Environment	Basic Idea	Conclusion	Other Issues
10	Priority Encoding Transmission PET	packet losses,	packet based networks	Each packet has a priority value that determines the fraction of encoding packets sufficient to recover that part. Many priority levels.	The receiver is able to recover the parts of the message if a sufficient fraction of the encoding packets are received	MPEG
11	Restricted Reliable Multicast Protocol (RRMP),based on RSE	packet losses	Multimedia applications in satellite communication but also heterogenous networks ATM and LANs	FEC chunk sizes are selectable among other parameters for adapting different networks	Many advantages, e.g.,FEC used in RRMP gains stability for the QoS parameters, since lost data can be reconstructed without any retransmission.	
12	e.g different XOR methods	lost audio packets	Real-time data and audio transferred over networks that do not quarantee QoS for them. IP / UDP / RTP	Open loop error control mechanism based on forward error correction. Adaptive audio coders/ decoders	Clearly adding redundant information increases the reward.	
13	RSE	packet error and loss	ATM, IP, multicast	The focus is on solving the problem of accumulated packet losses/ errors.	FEC dramatically reduces the IP packet error or loss probability.	error detection and correction mechanisms
NR	**FEC Algorithm**	**Error to Correct**	**Environment**	**Basic Idea**	**Conclusion**	**Other Issues**
14	convolutional codes,viterbi	two bit errors; single or double	satellite channel, DQPSK	Combining FEC with an efficient modulation technique.	FEC works well for satellite channels.	

Table 6.2: Related work, articles 10 - 14

Chapter 7

Integration of FEC into RAID and TP

In this chapter, issues to be considered when implementing integrated FEC for RAID and TP are introduced. Questions about RAID in this context are considered in the second section. We propose approaches to connect FEC in RAID and TP in the last section.

Figure 7.1, represents questions that lead the way to the discussion issues in this chapter. As Figure 7.1 shows, the main topics are written inside the grey ovals. One central discussion area focuses on relations between transmission, TDU size, FEC algorithm, and stripe unit. Another discussion concentrates on RAID. RAID levels in relation to file size, stripe unit, and number of disks is shown in the Figure 7.1.

The questions placed between the “issue ovals”, in Figure 7.1, lead to many approaches for using FEC schemes in INSTANCE. The main discussion topics give rise to further questions like; How reliable systems do we want to build? What kind of services do we wish to use? How large RAID do we wish to have and how effectively does it survive disk crashes? In any case, all of the systems and methods we choose to use have to be capable of working together. This means that whatever transport protocol is implemented, the FEC has to be compatible with the error coding in RAID. Therefore, selecting the right methods from FEC, RAID and transmission protocols is important. An end-to-end protocol where both ends of the communicating system agree which FEC is to be used must be built up. These and many other questions are discussed in the course of this chapter.

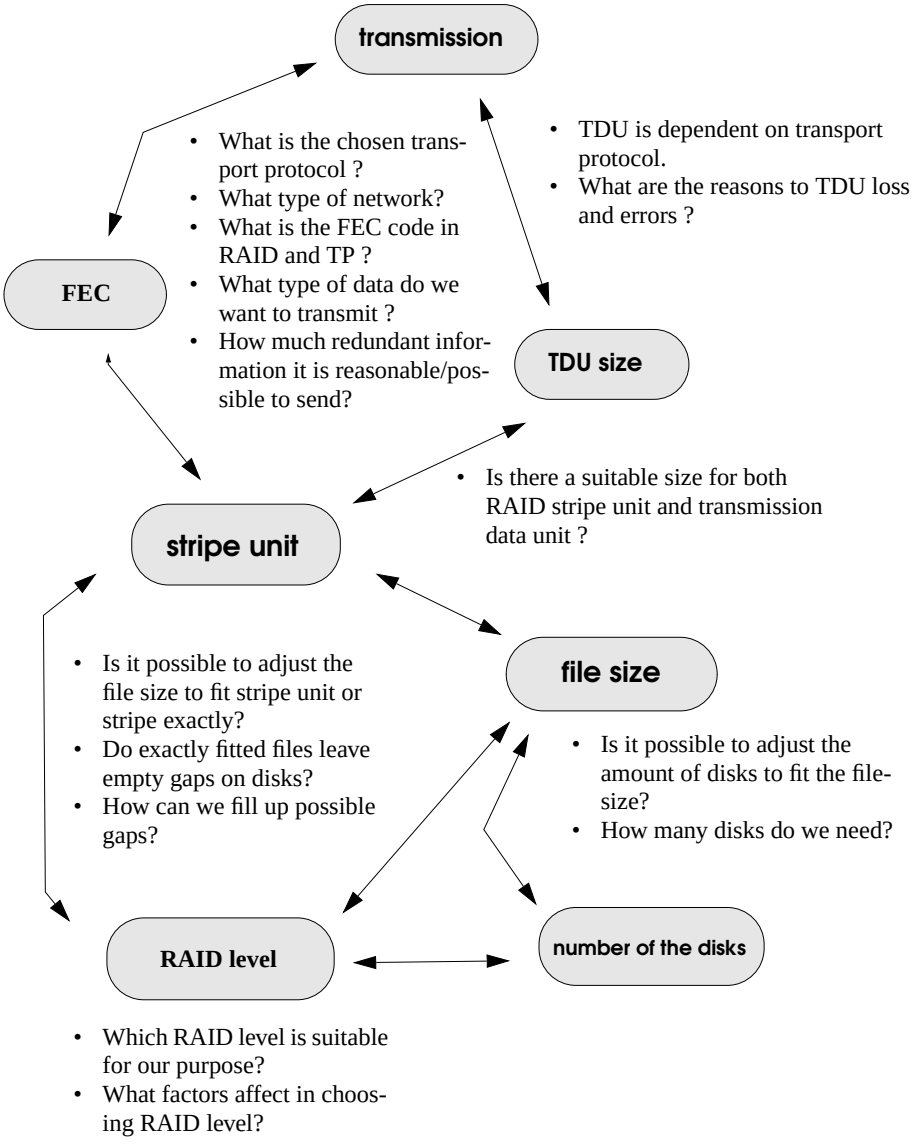

Figure 7.1: Issues for discussion

7.1 Aspects About RAID

There are many factors that influence the selection of the most appropriate RAID level. For example, striping affects main memory requirements, latency, and also throughput that can be achieved. However, in this section, we focus on the basics, such as request size and the type of data stored, which directly affect the choice of stripe unit and RAID level.

The best way to approach the question, "Which RAID level we should choose?", is to find out what type of data is to be stored. The most appropriate RAID level choice gives the best benefits. We can take a look at the Tables in Section 5.2, and use it as a guide in choosing a RAID level. The disadvantages in the table are marked with '-', and the benefits with '+'. As we can deduce from Table 5.1, mirroring in level 1 does not have the parity calculation for redundant data, therefore RAID level 1 is not a valid choice for our purposes.

Including RAID level 1, we do not evaluate RAID level 2 and 4 as disk array solutions for INSTANCE. RAID level 2's weakness is high parity disk overhead. RAID level 2's parity disk overhead is improved on RAID level 3. Unlike RAID level 2, level 3 has only one parity disk. In addition, RAID level 4 has one parity disk. RAID level 4's bottleneck is the parity disk. On RAID level 5, this bottleneck is eliminated by distributing parity across disks. These are the reasons why levels 2 and 4 are not better solutions than RAID level 5, therefore, it is neither advantageous to choose them nor discuss them further in this context. Because levels 1 and 3 can be viewed as subclasses of level 5, the choice of RAID level is guided by the parity group and stripe unit size.

Chen [13] has introduced two guidelines to choose the suitable parity group size and stripe unit for a RAID system:

1. "If a parity group size of two is indicated, then mirroring is adequate." However, this small parity size is not appropriate for INSTANCE, since we also need the proper parity for transmission of data.

2. "A stripe unit much smaller than the size of an average request may indicate the use of level 3 RAID. "

IBM has introduced two guidelines for choosing the suitable RAID level [25]. The motivation for these guidelines is the type of data and how often it is accessed:

1. Select RAID level 5 for applications that manipulate small amounts of data, such as transaction processing applications.

2. Select RAID level 3 for applications that process large blocks of data. RAID level 3 provides redundancy without the high overhead incurred by mirroring in RAID level 1.

These two sets of guidelines are the main criteria for choosing the most appropriate RAID level. We might have to compromise for the best RAID level in our approach. The requirements and problems that transmission protocol set to RAID level selecting are introduced in the following sections.

7.1.1 The Parity Problem

Data is striped over several disks in RAID, and parity information is calculated from all disks that include user data. For example, Biersack [6] proposes that redundant information be sent in transmission on its own units. Here, we might run into a problem with the striping on RAID. This problem, which we call *the parity problem*, is illustrated with the following simplified example in Figure 7.2.

In Figure 7.2, P stands for parity. A, B, and C stand for original data, e.g. movie files striped over several disks. Movie file A consists of data units A1 + A2 + · · ·+ A6 + A7. Movie file B consists of data units B1 + B2 + · · · + B6 + B7. Movie file C consists of data units C1 and C2. The parity is calculated over all disks and stored to a separate parity disk in this example. Parity stripes which include data units from two or several different movie files cause the parity problem. This problem does not occur in conventional RAID systems but only in the context of the FEC integration with TP.

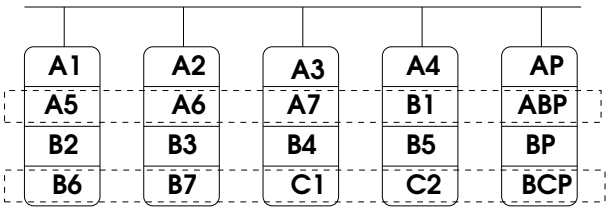

Figure 7.2: Problematic data striping

Parity problems occur in particular when we want to get only a small amount of information from the disks for transmission. For example, when we read movie file “A” for transmission, we need to read the parity information also. Since movie A’s parity calculation does not go even to one single parity stripe, we have some irrelevant parity information on parity unit “ABP”. The parity stripes that cause irrelevant parity information are marked with a dashed line in Figure 7.2. Parity units “ABP” and “BCP” do not consists of

only one movie file information but of two. We transmit only one movie and the relevant parity information for each client in data transmission.

One can interpret figure 7.2 that obtaining the right parity information from the disks is problematic. The parity problem occurs if we want to transmit, e.g., movie A. Movie A's parity unit "ABP" also includes parity information for data unit "B1's" reconstruction. Therefore, this parity includes irrelevant information for us, since we do not want to transmit movie file B. Assume we have a coding method that is capable of correcting one lost TDU in an FEC block. We remember from chapter 4 that an FEC block consists of all original data and parity in one parity stripe. Assume that, "B1" is not send and "A5" is lost in transmission. In this case, we lose the only possibility for encoding the lost packet which had otherwise been possible if the FEC block was sent completely.

We cannot possibly reconstruct lost TDUs with irrelevant parity information that has been sent from parity disks to the receiver. The success of encoding depends on the ratio of redundant data to original data. An FEC block that is not transmitted completely, is not adequate for error correction in case of TDU losses. A more concrete example of a parity problem is given next.

For example, if an MPEG-2 movie video is about two hours long, then the size of the file is about 3,6 Gbytes with MPEG-2, compressed 5 Mbytes/s. The RAID is about 30 Gbytes. We can store about 8 MPEG-2 movies in one 30 Gbyte RAID. If we have striping unit size that is 512 bytes, then one MPEG-2 movie creates 15625 data units plus the redundant parity units. If we have five disk drives, then we have 6 Gbytes stored in each disk, e.g., like in RAID level 5 in Figure 5.7. Since 6 Gbytes is used for parity, we have 4×6 Gbytes for original data. If we stripe one movie over the four disks in 512 bytes stripes, then 15625 data units are not spread evenly on the four disks ($15625/4 = 3906.25$). Hence, we have a similar situation to that shown in Figure 7.2.

The parity problem becomes even worse if we have a large stripe unit and small files, e.g. text files for WWW that are usually in Kbyte size only. This is because small files seldom fill up one parity group in a large RAID that can be greater than 70 disks. To be able to stripe small files over a large RAID requires a small stripe unit. However, small stripe units can be inefficient during reads. How the parity problem is avoided is discussed in the next section.

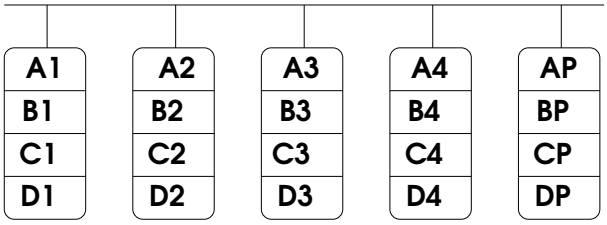

Figure 7.3: Ideal solution for the parity problem

7.1.2 How to Avoid Parity Problems

An ideal solution for striping data could be represented as in Figure 7.3. The files are equal in size in this figure. The stripes are calculated in such a way that each parity stripe has data units from only one file. This means the number of disks is suitable for striping files evenly over all disks. For example, a stripe unit is large enough to divide each file into five stripes. Assume the RAID used here has six disks plus a parity disk. In this case, the stripe size must be adjusted smaller, in a way that each file is divided into six stripes. Or, if it is not appropriate to have a smaller stripe unit, we have to have a RAID with five disks plus a parity disk. Another possibility is to use the virtual parity striping method introduced by Holland et al. [23]. However, files that are stored to a RAID are seldom equal in size in reality. Ideal striping solution is therefore not always possible to achieved.

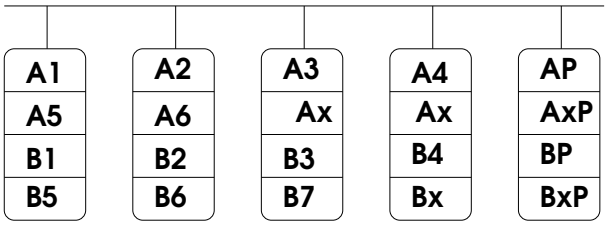

Figure 7.4: Proposal to solve a parity problem

In the following section a solution for the parity problem is proposed. It is illustrated in Figure 7.4. This proposal is based on the fact that movies or text files are not always the same length or size. By defining the maximum size for one file and arranging same type of files in one RAID, we can reduce the effect of the parity problem. This arrangement is useful because it is unlikely that one parity stripe includes information from only one movie. The remain-

der of a parity stripe that would not include the same file is, e.g. filled with 0's. We call this method *padding*. The next example illustrates how padding is done to avoid the parity problem.

In this example, two files are shown in Figure 7.4. The size of the files differs by one stripe unit. The padded data units, i.e. data units filled with 0's, are marked as "Ax" for file A, and "Bx" for file B. There are three data units which do not include any data.

We can avoid extra work by not sending the padded information. This is possible if the receiver knows how much data is coming and calculates the last "missing" TDUs as zeros in the last FEC block. As discussed earlier in this chapter, the receiver has to have a complete FEC block to recover from TDU losses. In other words, the receiver has to receive a minimum amount of TDUs that is equal to the amount of original TDUs in one coded block. If the last received FEC block has more TDU losses than the code is capable of correcting, the lost TDUs are assumed to be the "padded" TDUs. This misinterpretation can be responsible for some wrongly coded data in the end of the data transmission. This can be avoided if the total size of each data transmission is known to the receiver.

Because of the parity problem and padding, some space is wasted on RAID. This is why it is preferable to adjust the RAID size to suit certain amounts of files that are a maximum length. For example, count the average length of movies and store them to one RAID. Much longer and shorter movies are stored on their own RAIDs.

Movies are quite easy to adjust on RAID. Movies usually consists of the same types of data. For example, multimedia includes different types of data. These files can differ greatly in size. However, they are accessed from the RAID for the same data transmission. The selection of the RAID level and stripe unit becomes easier if different types of data are each stored on separate RAIDs.

We have to remember the guidelines given earlier in this section when designing the stripe unit. For example, if we have text files for WWW, the file sizes are small and many small reads are going to take place. In this situation, it is best to choose level 5 and adjust the stripe unit to have as few empty data units, i.e. padding, as possible. Since data is written byte by byte in RAID level 3, one might think it is easier to adjust smaller files on disks. However, level 3 is not perfectly suitable for many small simultaneous reads because it can satisfy only one I/O request at a time.

7.1.3 RAID Level 6 and INSTANCE

If we want to utilize better reliability in RAID and use P&Q parity, we might think that we could use both parities. However, P&Q redundancy is not suitable for this purpose because of the way parities are calculated. Figure 5.8 shows that parity P is calculated *vertically*, whereas Q is calculated *horizontally* [2]. Therefore, parity P is not a copy of parity Q, but a result of different original data than parity P. This can cause the parity problem mentioned earlier on page 7.1.1. Figure 7.5 shows the parity problem in P&Q redundancy.

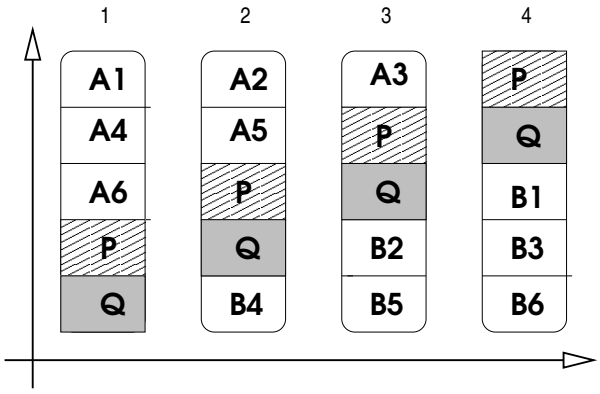

Figure 7.5: P&Q redundancy and parity problem

We have two files “A” and “B” in Figure 7.5, which are striped over disks on an array. To simplify the example, the files here are of equal size. Both files are divided into six stripes: A = A1 + · · · + A6, B = B1 + · · · + B6. Parity stripe for parity P is calculated *horizontally* and parity Q is calculated *vertically*. This method of parity calculating is two-dimensional.

As we see from Figure 7.5, the parity Q, e.g. on disk 2, is calculated vertically. This parity information is not calculated over only one file, e.g. file A, but includes data units from file B also. In this case, the calculated parity includes irrelevant parity information from file B. When we wish to transmit only file A, we also transmit irrelevant parity information from file B. Two dimensional parity is therefore not an appropriate method for INSTANCE because we want to send only one file and the corresponding parity information at a time. P&Q is only useful if one sends for example original data with parity P and use the extra parity Q for RAID security only, presuming that parity P includes only relevant parity information.

7.2 FEC Cooperation in TP and RAID

There are various motivations to choose an algorithm and methods for error correction in data transmission. Issues to be considered are for example, the type of data to be transmitted, and channel capacity. Matching the requirements in RAID and data transmission together is a complicated task. This task is discussed in the next five sections.

7.2.1 Redundant Data Calculations

The algorithm for redundant data calculations can be chosen by the requirements we set. Simple XOR calculations and Hamming coding are adequate for RAID and data transmission if the RAID is not very large or the TDU loss rate due to congestion on the channel is not high. It is possible to have one disk crash/day or a week [22] if RAID is larger than 100 disks. This means that the algorithm that produces more than one parity unit/parity stripe is needed for RAID to be able to survive two simultaneous disk crashes. However, as discussed in the earlier section, P&Q parity is not suitable for this purpose. In addition, stronger calculations are needed if a channel is congestion prone. RS and RSE are examples of codes that are frequently used for better data integrity.

In related work investigations in Chapter 6, RSE code [31] was the most frequently used correcting code. The authors agreed that conventional RS coding is quite complicated to implement. This is the reason why conventional RS is less attractive to use in error correcting. Coding and decoding would take too much effort with RS. In addition, FEC is needed only for TDU loss correcting when bit errors can be corrected with CRC.

Since RSE is adequate for transmission, we can evaluate the possibilities for RSE in RAID also. Chen [13] mentions that in disk arrays, crashed disks are recognized with disk drives. Therefore, the lost data is in a known location. Also, Plank's [36] failure model is erasure in RAID systems. However, instead of RSE, Plank gives a detailed strategy of how to implement Reed-Solomon coding called RS-Raid. The mathematical approach of RS in RAID equals the RS codes introduced in transmission protocols.

Hellerstein et al. [22] and Blaum et al. [9], introduce binary linear codes in a matrix form for more effective coding. Their proposals are based on parity calculations with more than one dimension. The two-dimensional parity calculation is done in a similar manner to RAID level 6. A two-dimensional parity calculation method enables RAID to recover from two simultaneous disk crashes. This is a desired quality in very large disk arrays where a disk crash probability is higher. However, when we consider to combining this

method with transmission protocol, we meet the problem that was discussed earlier in Section 7.1.3. This problem was that each parity which is calculated over one disk, i.e. one column, introduces parity information from data that is irrelevant for data transmission. Irrelevant parity data prohibits the receiver from reconstructing lost data.

The parity calculation must be arranged in another way to avoid the parity problem and still have larger ratios of parity units in one FEC block. The solution is to perform the parity calculation one-dimensionally. Figure 7.6 shows a situation where two parity units are calculated, e.g. with RS coding. The arrows express the one dimensional parity calculation: one parity stripe produces two parity units. These two parities are stored on two separate parity disks: “4P” and “5P”. This method is realized as RAID level 3. However, it is also possible to distribute parity as in RAID level 5. The distribution of parity looks like RAID level 6.

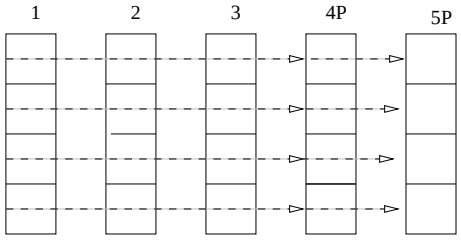

Figure 7.6: Two parity units in one FEC block

As we can see from the Figure 7.6, the double parity information increases the disk place requirement. If no more disks are introduced to the array, parity uses space from the original data. In the solution where two separate parity disks are provided, it is possible to survive two disk crashes. For example, one data disk crash and a parity disk crash simultaneously can be survived, because the parity information on the second parity disk is a copy of the first one. This is a good quality, especially if the disk array is very large. As discussed earlier, large disk arrays have a higher probability of having two disk crashes simultaneously.

We can conclude that RSE coding is available both in RAID and transmission protocols. One-dimensional RS or RSE coding in RAID is also suitable for use in INSTANCE. Plank [36] points out how small disk arrays are more reliable than larger disk arrays, and need only simple coding, such as XOR or Hamming coding to survive one disk crash. These parity calculating methods are also suitable for data transmission if it is certain congestion on the channel is moderate. The congestion tendency in a data transmission channel is

also a guideline for redundant data ratios to original data in one FEC block. Since we also have to be able to avoid congestion, the amount of redundant information has to be reasonably small. The more redundant data that is sent, the higher the risk to cause congestion.

As mentioned earlier, in Chapter 4, one FEC block consists of original data and redundant data. If an FEC block is large, it has a higher probability of losing more TDUs in one block than a small FEC blok. A large FEC block size produces encoding and decoding delays, specially when recovering from errors and decoding missing cells. The cause for delay on the receiver side is due to the time the receiver waits for an entire FEC block for coding. However, this is a smaller disadvantage than the retransmissions delay. These issue has been studied in [6, 31, 34, 44].

7.2.2 Stripe Unit in Integrated FEC

There are many aspects that can be viewed when designing striping for RAID. If we have many clients accessing a disk array containing data, for multimedia purposes e.g., large block sizes are preferable. The total seek time is shorter on large stripes, and faster for heavy traffic than with a small stripe unit. However, a large block size might not always be the most appropriate solution.

A large block size can cause a load imbalance when the most popular files in a disk array are accessed, for example, when a large stripe unit is used to stripe a popular movie file in a VoD system over a disk array. The stripes are located on only a few disks because of the large stripe unit. Because of the popularity of a particular movie, only a few disks in a disk array which contain the movie are accessed frequently. This is the cause for load imbalance. In addition, service time for the most heavily loaded disks increases. However, in this thesis, the aim is to stripe, a movie file, e.g., over parity stripes in such a way that the parity group consists of only this movie file. In addition, we avoid the parity problem presented in Section 7.1.1 by operating this way.

A stripe unit is also connected to different RAID levels. RAID level 5 uses larger data blocks in striping, whereas RAID Level 3 takes the data and writes it bit-by-bit or byte-by-byte in parallel to the data drives, plus one bit or byte to the parity drive. The result is a disk array that works best with large block transfers, but does not work well for transaction processing systems. RAID level 3's bottleneck is the parity drive. Because it is the only access to the parity, the bottleneck is at its worst when small updates are made to the disk array.

7.2.3 An Approach to Bottlenecks in RAID

We can not use existing RAID solutions because we need to read the parity disk also in our approach to implementations. This also changes traditional approaches to RAID bottlenecks. Bottlenecks for each RAID level were also discussed in Chapter 5.

RAID level 3 has all parity information stored on one disk. Unlike in RAID level 5, where parity information is distributed over all of the disks in the disk array, the bottleneck for RAID level 3 is the parity disk. In our approach, the parity disk maintains the bottleneck situation and becomes even worse when the parity disk or disks are accessed for a read also.

A better solution in our approach is similar to RAID level 5 distribution of parity information. The RAID level 3 bottleneck situation is better on RAID level 5 because parity information is distributed over all of the disks and has the possibility for parallel accessing.

7.2.4 Network Loss Behavior and Stripe Units

The loss behavior of a network gives us a guideline for choosing an appropriate FEC in transmission. For example, congestion losses are dominant in ATM type networks. This means that the typical loss unit is a cell. From the example introduced earlier in Section 3.2.2, we can conclude that single-bit error detection is not useful in data transmission because noise impulses are usually long enough to destroy more than one bit. Impulse noise duration is not long enough to destroy voice data, but it is able to erase bits when data transmission speed is high. Usually, a coding strategy is arranged to protect against a particular error that is known to occur. Therefore, before we design a transmission protocol it is important to understand the loss behavior of the network.

Because of different needs in RAID and data transmission, the TDU size in transmission is not always equal to the stripe unit in RAID. We are not dependent on the TDU size when we make the decision of which RAID level to choose. As presented earlier, the guidelines for choosing suitable a RAID level do not directly have an effect on the transmission protocol structure. However, we might have to divide larger stripe units into smaller TDUs for transmission.

Can larger blocks of information and parity information divided into smaller TDUs in transmission increase the possibility that we lose more TDUs from FEC blocks? In other words, we have a situation where large TDUs are divided into basic multiplexing blocks of the network [31] (cells). Figure 7.7 illustrates

the situation.

For example, we have stored movies that are large in size, usually in Gbytes; the stripe units on a disk array are also relatively large. According to the guidelines on page 73, large stripe units are the most appropriate when we have large reads: in this case, one movie at the time. However, if real-time data is transmitted along, e.g., an ATM type network, TDUs are divided into smaller units. This is because an ATM cell size is only 53 bytes long.

Data is usually striped on larger blocks than 53 bytes. Figure 7.8 illustrates the situation where larger TDUs are divided into smaller units. This situation is comparable to the problem area that Srinivasan et al. [44] point out. This is when TCP/IP packets are transmitted through an ATM type network.

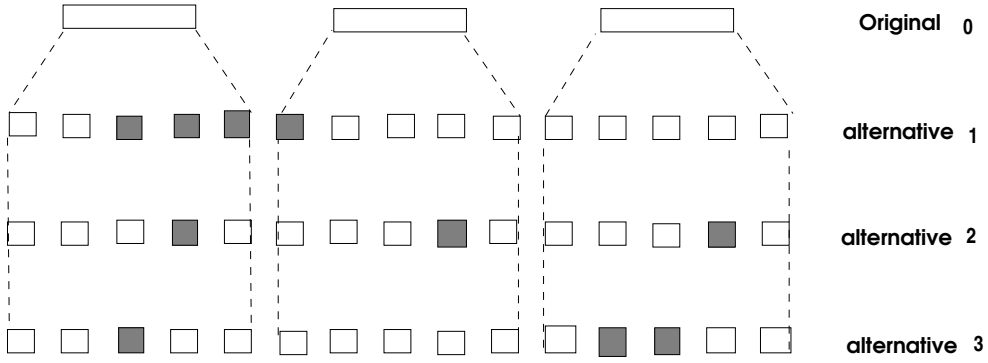

Figure 7.7: Error distribution in fixed size TDUs

Because a TCP/IP packet is larger than an ATM cell, the original packet is divided into smaller units. For example, if one of the cells is lost during data transmission, we are not able to reassemble the original IP packet. This occurs specifically when one unit from the IP header field is lost. This type of problem might be crucial to parity packets that are split into smaller units because the probability of getting error rises each time we divide TDUs into smaller units. The loss of smaller TDU units can cause FEC inefficiency. This situation is illustrated in Figure 7.7 in alternative 2. Here, each original TDU is lost because of losses on smaller TDU units. Lost TDUs are marked with grey.

However, error possibilities are small in optical fibers. Therefore, we can transmit large files without errors that occur in the way Figure 7.7 and al-

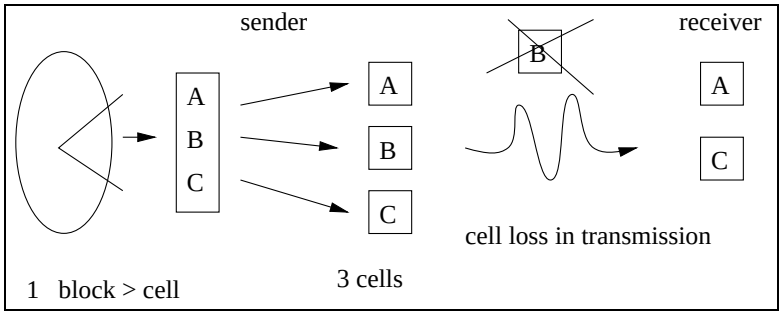

Figure 7.8: Packet Size and Block Size

ternative 2 illustrate. The Bit Error Ratio (BER)[1] on an optic fiber is 10^{-9}. A BER this low does not influence transmission. For example, entire video files that are several Gbytes large, could be transmitted without a single error occurrence. In designing of the transmission protocol for INSTANCE, it is presumed that the BER is low.

A more likely TDU loss scenario is shown in alternatives 1 and 3 in Figure 7.7. Lost TDUs are marked with grey. The first alternative shows a TDU loss due to congestion: consecutive TDUs are discarded. Discarded TDUs can naturally belong to one large original TDU or two. Alternative 3 shows congestion losses with two consecutive TDU losses and one independent loss. Independent losses happen seldom in optic fiber, though this possibility is shown in this figure. We can conclude that even though the original TDUs need to be divided into smaller units, we retain the ability to recover from TDU losses with the help of FEC. The total amount of lost original TDUs is not so high that it could prohibit TDU loss reconstruction at the receiver.

7.2.5 Adaptive Systems in INSTANCE

In computer communications, data traffic over channels is not constant but changes, like rush hour in traffic. Varying channel conditions is the reason why adaptable error correcting systems are attractive.

One point of view in related work investigated in Chapter 6 is to create a method that can adapt to channel conditions or error tendencies. We can utilize these methods in reads from RAID that are reduced to the minimum when collecting redundant information for data transmission. For example, [11] and [42] propose an adaptive FEC method for different requirements in multimedia. Multimedia diversity creates an unique challenge for transport

[1]Bit Error Ratio (BER): The number of erroneous bits divided by the total number of bits transmitted, received, or processed over some stipulated period [17].

protocols. Sources like [11, 15] have weighted also the flexibility in their proposals. This is achieved usually with a possibility to choose whether to use only CRC or FEC or both. This depends on what kind of criteria we wish to have in INSTANCE.

The creation of an adaptive transmission protocol is complicated in our approach, especially if the method does require changing the FEC algorithm or the original-redundant data ratio. This is because it is preferable to have the same original-redundant data ratio already coded on RAID. As discussed earlier in the article investigation comments in Section 6.2.1, frequently changed parity calculations are difficult to accomplish on RAID. In order to avoid the parity problem, parity stripes are arranged with the help of padding. The parity problem occurs if we want to change the amount of data units in parity stripes. This means that each time we wish to change FEC block size or data, we have to update the striping arrangement and padding. In other words, the new striping arrangement in an adaptive FEC has to be parity problem free at any given time.

In a case where adaptivity is attractive, we can decide if we read the redundant data and transmit it or not. Also, FEC integrated with ARQ is suitable if we want to send data to an application that can afford it or requires high data integrity, e.g. file transfer systems for non-real-time systems. However, this solution is not compatible because we wish to have an integrated FEC to avoid retransmissions.

Chapter 8

Implementation and Tests

Simple experiments for FEC in INSTANCE are described in this chapter. In the first section, an explanation of the implemented experiments is given. The second section contains observations which are drawn from experiment results. In the last section issues for further work are presented. The questions are answered: How can the implementations be improved? What observations are interesting for further studies?

8.1 Test Environment

The test environment is Da Capo in this approach. Da Capo is a protocol configuration tool. With Da CaPo, it is possible to have a simplified end-to-end protocol environment that supports a broad range of various application requirements. No unnecessary protocol functions are presented in the test environment. These are the reasons why Da CaPo is used as a test environment in this thesis.

Da CaPo is a communication system that can be viewed in three layers. These three layers are shown in Figure 8.1. Layer A can be understood as a set of distributed applications. Layer C represents the services that the end-to-end communication requires. Layer T represents the transport infrastructure. This layer offers the services that are comparable to the ATM adaptation layer or TCP services. Da CaPo is used for simulating an end-to-end protocol environment in a TCP/IP type network in the experiments.

It is possible to program various protocol functions like error and flow control to layer C. A protocol mechanism specifies the rules for the functions. The protocol mechanisms can be realized by modules in hardware and/or software. We do not go into detail in describing how Da CaPo and the A and T layers are programmed. Instead, we introduce how Da Capo is used in the experiments in this thesis. We also describe the modules that were used

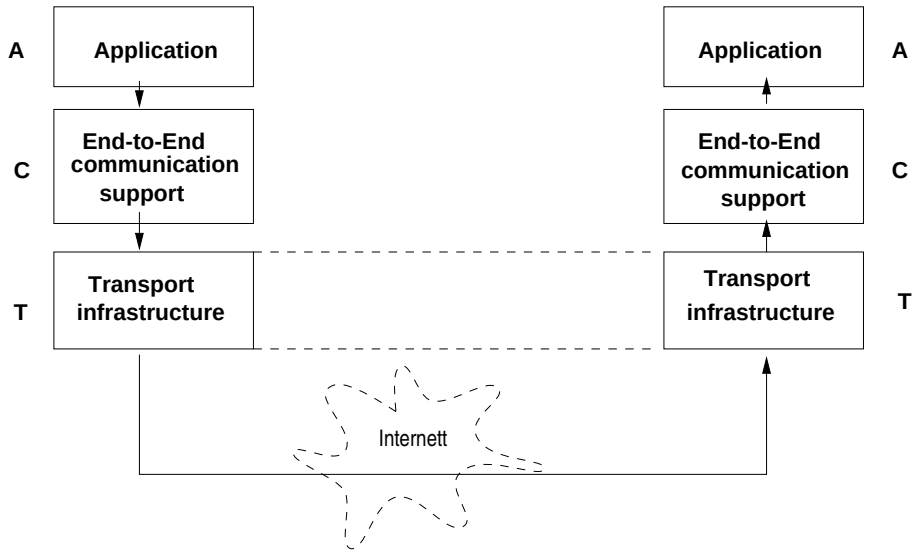

Figure 8.1: Three layer model of Da CaPo

on the C layer. More detailed information about Da CaPo can be found at [20].

The modules that were used in the experiments were the m_killer module and the FEC modules [43]. Layers with the m_killer and FEC modules are shown in Figure 8.2. This figure shows how the modules are arranged in Da CaPo. The arrows show the flow of TDUs from server to client.

We can simulate TDU losses with the m_killer module. The m_killer module eliminates TDUs from the data flow. It is possible to define exactly which TDUs are eliminated with the m_killer module. The user defines into an m_killer file which TDUs are to be eliminated. Each time a TDU arrives from the "A" module to the m_killer module, the m_killer file is read. By defining the TDU sequence numbers on an m_killer file, we can also control the amount and frequency of TDU losses. As Figure 8.2 illustrates, the elimination of TDUs is done before sending the TDUs from the server to the client. We can control the amount of lost TDUs in this manner. In addition, we know how many TDUs are lost and how they are distributed in the data flow. The testing and analyzing of the efficiency and work of FEC explicitly on the client side is easier than in a normal data transmission situation.

FEC modules on the sender and receiver sides do the coding. FEC code produces two redundant TDUs from each two consecutive original TDUs in this experiment. In other words, the FEC block consists of four TDUs, where two TDUs are original, and two TDUs are redundant. Recovery from TDU

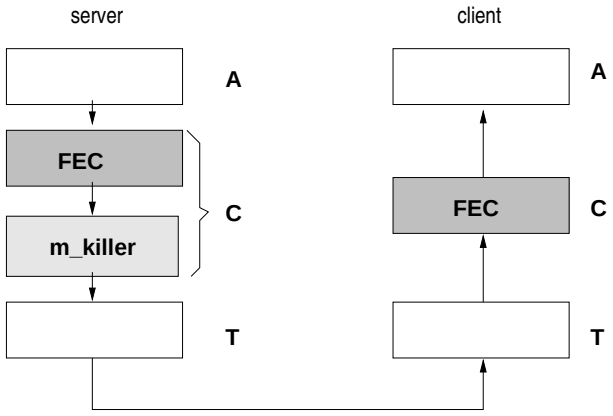

Figure 8.2: Test environment for FEC in Da CaPo

losses is possible if two TDUs are received from one FEC block. The redundant TDUs are calculated by using simple XOR calculation. For example, the first redundant TDU is simply generated by XORing the two original TDUs together. If one of these three TDUs is lost in the data transmission, the lost TDU is regenerated by XORing the two TDUs together. The FEC and m_killer module are available in a Appendix A and A.3.

In addition to modules, some necessary conditions were defined for the tests. During the tests, the test file size was constant. The TDU size was 153 Kbytes in all of the experiments. In addition, tests were performed at the same point of time. Early morning was the most appropriate time for the test, because the data traffic during mornings is usually much alike. Naturally, some variation between different mornings can occur. Testing in the afternoon could have given results that differ from testing executed in the morning. All tests were performed with the same computers, which were located in the same building and connected with 100 Mbit Ethernett using TCP/IP protocol. The sending computer was SUNW, UltraSPARC-IIi, 300MHz and 128 Mbytes memory. The receiving computer was the same type as the sender, UltraSPARC, only slower 167 MHz with 192 Mbytes memory. The sending and receiving machines were the same for each test; therefore, *some* differences in the tests were avoided.

8.2 Experiments and Observations

Various error scenarios were simulated with the help of an m_killer module. With the m_killer module it is possible to eliminate TDUs from data transmission. Since we know the FEC capability of the FEC code we use, it is possible to simulate error scenarios. For example, if more than two TDUs are lost from

one FEC block, an unrecoverable TDU loss occurs. If we want to simulate one TDU loss, because of congestion e.g., we have to eliminate at least three consecutive TDUs from one FEC block, with the m_killer module.

In the test, sending and receiving times were measured. TDUs killed with the m_killer are shown in Tables 8.1, 8.2, 8.3, and 8.4. The row "lost packets" shows how many TDUs were lost, i.e., totally lost after decoding. The row "modules" shows which modules were included in the test. The time measured for sending and receiving is the total time for the sending and receiving process. Time was measured with a *time* function. The time function is started when starting the sender and receiver process and stopped in the end of the sender and receiver process. Because of this, some extra time is used for the time function only, and some inaccuracy in the test results is possible. Tests were executed several times, and an average time was calculated for each test scenario.

killed TDUs	none
lost TDUs	none
modules	none
average send	6.705367 sec
average receive	6.765890 sec

Table 8.1: Results for test 1, without FEC and errors

Experiment 1. The first experiment was a test without modules. With these experiment results, we were able to compare how much more time sending and receiving takes when modules are included. The receiving time is slightly more than the sending time due to slower receiver hardware.

killed TDUs	none
lost TDUs	none
modules	FEC
average send	11.157779 sec
average receive	11.227418 sec

Table 8.2: Results for tests 2, without TDU losses

Experiment 2. In the second experiment, the FEC module was included. Without the m_killer module, no TDU losses occurred. The reason for this experiment was to find out how much more time FEC coding would take than without it. As expected, the execution took slightly more time than in the

first experiment. The difference for execution time was on average five seconds. This was not a surprising result, because FEC code produces redundant TDUs, and transmission takes longer time. The receiving and sending times were in the same ratio as in the first test: receiving took about 0.07 seconds more time than the sending of data.

killed TDUs	90
lost TDUs	none
modules	FEC, killer
average send	11.250913 sec
average receive	11.327733 sec

Table 8.3: Results for test 3, with FEC, no TDU losses

Experiment 3. The third experiment also introduced the m_killer module. The error distribution plan was that no TDU losses should occur despite killed TDUs in the m_killer. As in the second experiment, FEC was able to recover from TDU losses. There was a total of 90 eliminated TDUs in the m_killer. However, all lost TDUs were recreated in decoding. The average sending and receiving times were not more than in the test without the m_killer module. Therefore, we can conclude that the m_killer does not have a big influence on sending times.

killed TDUs	180
lost TDUs	13
modules	FEC, killer
average send	10.876759 sec
average receive	10.875771 sec

Table 8.4: Results for test 4, with FEC, lost TDUs

Experiment 4. The amount of eliminated TDUs is larger, 180 in this experiment. In addition, the TDU loss distribution is arranged in the m_killer in such a way that 13 TDUs are unrecoverable and lost. In other words, the simulation of consecutive TDU losses produce 13 TDU losses despite FEC decoding. The average sending and receiving time in this experiment is even less than in the other experiments with FEC. This can be explained by the fact that the relatively large redundancy (almost a quarter) is reduced with the m_killer. Therefore, transmission does not take as long as in the third experiment, e.g., where larger amounts of data is in transmission.

8.3 Discussion

The experiments show that the test environment and FEC works. The results were not surprising. It was, e.g., expected that data transmission with FEC would use a slightly longer execution time than without the FEC. The FEC at the receiver took slightly longer time if many TDU losses occurred. However, the recovery from TDU losses is a greater benefit than the slight loss of time.

Integration of FEC in RAID and TP needs further studies and implementation of test scenarios. Figure 8.3 introduces a proposal for a test environment structure. If we do not have the possibility to implement RAID for INSTANCE, we can simulate the scenario. The encoding is executed before storing of data to a file in this situation. This simulates the situation when RAID is included. The encoded information is stored and sent through the sender side of Da CaPo. At the sender side, an m_killer module is used, as in earlier experiments. The receiver with the FEC module works in a similar manner as in the earlier experiments and includes only the decoder module. However, we can not utilize the FEC decoder module directly. A new module that is capable of decoding without the encoder module in Da CaPo must be implemented.

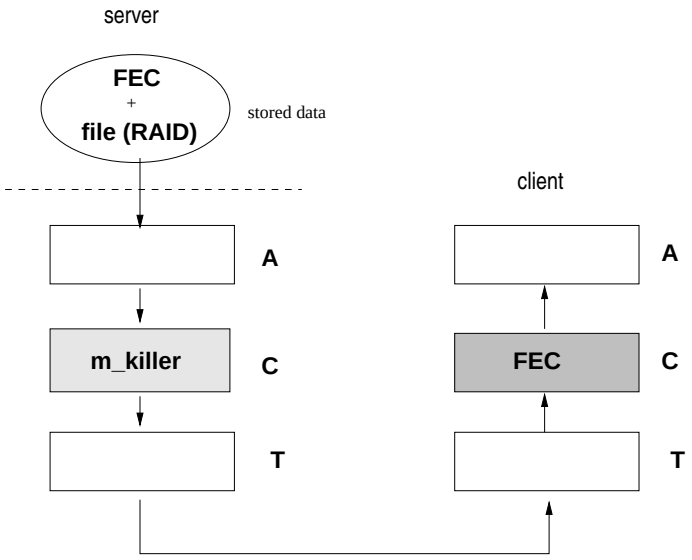

Figure 8.3: Further implementation proposal for FEC in Da CaPo

More than the new layer arrangements in the test environment should be done. In addition, testing of a more appropriate FEC code for INSTANCE is needed. The code used in these experiments produces as many redundant TDUs as the original TDUs. This increases the network load a lot, and the redundant information can cause congestion. This much redundancy is pur-

poseless, especially if the TDU loss ratio is small. Therefore, the original data ratio to redundant data has to be larger.

Since the XOR code was used in this thesis, developing this further should be considered. Simple XOR calculations are also used in transmission and RAID in related work. However, Hamming coding and RS gives stronger coding and a larger scale of variation, e.g. in original and redundant data ratios. As mentioned earlier in Chapter 7, RS or RSE code is one possibility. There are many approaches to consider when we implement RS or RSE code for INSTANCE.

One of the approaches is how to implement RS or RSE for RAID. A one-dimensional solution is appropriate. One likely implementation is to use the typical block code ratios. This means the redundant data ratio to the original data. The most usual symbol size is 8, or coefficient of 8, since one byte is 8 bits long. This was explained in Section 4.5.2. Different original data to redundant data ratios would be interesting to experiment with when trying to find out the limits of an appropriate and useful redundancy ratio with the experiments.

In these tests, we have to consider the striping method of parity information in reality. How do we arrange parity if the FEC algorithm produces more than one unit of parity information to each FEC block? As in our simple experiment, one FEC block includes two parity TDUs and two original data TDUs. We can investigate further whether it is worth having more than one parity unit in one FEC block. This means we have to find out the smallest amount of redundant information necessary to avoid TDU losses.

In addition to channel condition aspects, we have to consider how much disk space it is economically possible to reserve for parity information. We also have to consider how to distribute parity over all disks. An easier way might be to design RAID where parity information is not distributed, as on RAID level 3. This arrangement can eliminate the problems of distributing parity and can be easier in design. However, distinct parity disks can cause bottleneck in each read, write, and parity updating situation. Since each read also accesses parity disks, reads can also be the bottleneck for this type of RAID. These aspects are worth testing and finding out which striping method is most efficient.

Yet another approach to implementation is the analysis of tests. Because the complexity of RS coding is discussed in relevant literature, it would be interesting to measure the differences of some light weighted coding, such as FEC used in earlier experiments: for example, how much effort the coding itself takes in transmission compared to much simpler coding methods like

XOR.

We can conclude that further experiments have many subjects to be investigated. Further work issues include:

* Changes to the modules and the layer structure.
* Which channel conditions do we wish to simulate? If we have a channel with a high congestion level, then we need more redundancy.
* How is striping arranged in RAID? Can we avoid the parity disk bottleneck by distributing redundant data?
* We wish to analyze whether the test scenarios are appropriate for multimedia data.

We can find different approaches to further investigations. We can concentrate on one problem area at a time which can be efficiency or avoidance of one particular bottleneck. Another approach is to try to implement the most well-balanced system where all issues and problem areas are considered simultaneously.

Chapter 9

Summary and Conclusions

A summary of the most important issues in this thesis is given in this chapter. The second section contains conclusions drawn from the research presented in earlier chapters. Areas of further study are explained in the last section.

9.1 Summary

In this thesis, we have studied if it is possible to integrate FEC in RAID and transmission protocols. The motivation for this method was to gain efficiency by reducing redundant FEC functions in the protocol stack. We investigated the typical errors in data storage systems and data transmissions first. We found out that the most significant error in a data storage RAID was a disk crash. RAID recovers from disk crashes with the help of redundant information. It is also possible to recover TDU losses with redundant information in data transmission. The most typical cause for TDU losses in data transmission was congestion. Bit errors were not to be considered in optical fiber links because of the good quality of fiber optics and the low impact of bit errors to the total transmission of data.

Second, error scenarios in RAID and data transmission were evaluated. RAID and TP solutions were introduced, and methods of recovering from their most common errors were studied. Background knowledge and related work studies were the basis for further analysis in this thesis. The most appropriate methods for integrated FEC in RAID and TP were introduced. The problems with some solutions were described and a method to cope with them was presented in Chapter 7.

9.2 Conclusions

We can conclude that the most appropriate methods of integrating FEC in RAID and TP have many starting points. In particular, when the focus is on storing and transmitting multimedia data which includes several types of data like audio and video. Several data types make our task even more complicated because of various QoS requirements for each type of data transmission. The dilemma can be divided into three parts:

1. Which type of RAID level is the most appropriate for the data we have? How is striping arranged?

2. What characteristics do TPs have to have? What is the congestion level on the data transmission channel?

3. What are the most appropriate methods we can choose from 1. and 2. for the integration of FEC in RAID and TP? Which FEC algorithm should be used?

The first point includes the criteria for which striping method we use. For various types of data it is advantageous to store each type of data on its own RAID, because we have to have access to the parity information that is relevant each time. Using this method, the parity problem's effects are also minimized. This means that padding in parity stripes is minimized when each file size closest to average is stored in separate disk arrays.

The preferred RAID level for many small reads is 5, and 3 for large disk accesses. However, parity, striping and FEC algorithm arrangements may differ from the standard RAID levels in our approach. Therefore, the standard RAID levels are only guidelines for evaluating suitable methods. For example, we found out that the traditional bottlenecks for RAID were to be changed because the parity information was also to be read, i.e., parity units are also accessed for data transmission. Therefore, the most appropriate choice for storing parity was to distribute it over all disks like in RAID level 5. By distributing parity, parallelity is gained. In addition, read and write bottlenecks on separate parity disks are avoided.

The second point in our list includes the suitability of TP to transport multimedia data. In this case, we do not require a retransmission possibility from the transmission protocol. Basically the criteria for FEC in TP are based on channel capacity and the amount of congestion on a channel. We found out that the usual TDU loss amounts were about two consecutive TDUs at a time. In addition to the channel error probability, we considered the disk crash frequency. We found out that if disk array is large, greater than 100 disks [22], the probability of having two simultaneous disk crashes in a day

or a week was high.

We can conclude for the third point that the channel conditions and disk array size give a guideline in choosing an error coding algorithm. We agreed that simpler coding algorithms (e.g., XOR) and less redundancy is appropriate for small RAIDs and on channels with low congestion. In large disk arrays and high congestion levels during data transmission, it is preferable to use RS or RSE algorithms for error correcting. However, the amount of redundant information has influence on striping. More than one parity unit produced by one parity stripe in an encoder also requires more disk space for parity. The parity calculations are to be calculated one-dimensionally. Because of the parity problem, two dimensional P&Q redundancy is not directly suitable to be utilized in our approach. However, data transmission can utilize the *horizontally* calculated parity without the parity problem. This is an approach which is acceptable on channels with low congestion rates and disk arrays with a high disk crash probability.

The adaptable FEC methods investigated in the related works for data transmission were not considered suitable for our approach. The reason was the use of parity information also in RAID. The change of FEC block size for TP is complicated to accomplish in RAID. The updating of parity information to agree with the requirements of data transmission, including the requirements for recovery from the disk crashes in RAID, is not considerable. Only some ability to adapt in TP was considerable, like the option of whether to send or not to send redundant information. This is acceptable on data transmissions that do not require reliable transmission.

The study of related work has not pinpointed TPs or RAID solutions that are directly useful in our approach. However, related work shows that the basics of FEC in transmission and RAID can be integrated. The protocol stack has to be build up in a such way that real-time data QoSs are satisfied for multimedia purposes. For example, ATM type networks and real-time data requirements were approved to be functional with FEC in related work studies.

9.3 Outlook

In future experiments we should measure the impact of differences in 1) striping methods, 2) FEC algorithms, 3) amounts of parity information, and 4) types of data. The test results should reveal if the entities of chosen methods are the most appropriate ones. We wish to test a) transmission and receiving times, b) latency, and c) error correcting capabilities in RAID and transmission. d) The error correcting capability in transmission during disk crash is

also one of the test interests.

Further work should give information on whether FEC integrated in RAID and transmission protocol is an effective and functioning system. We should investigate the profits this method may bring. Actual bottlenecks with the system are revealed through tests: for example, pitfalls like data transmission overhead or reduced efficiency in data storage system.

Further research for this thesis includes an implementation of more advanced test scenarios (see Section 8.3) than represented in this thesis. Differences with various FEC algorithms should be studied. If possible, RAID with various striping methods should be investigated. In addition, distribution of the parity stripes has to be considered to avoid read and write bottlenecks. If RAID is cannot be realized, it can be simulated with a conventional file system.

Bibliography

[1] F. El-Guibaly A. Almulhem and T.A. Gulliver. Adaptive error-correction for atm communications using reed-solomon codes. *Proceedings of IEEE SOUTHEASTON*, pages 227–230, 1996.

[2] AC&CN, Raid Technology, 1998. `http://www.raid-storage.com/raid.html` 17.7.1998.

[3] Albanese, Blomer, Edmonds, Luby, and Sudan. Priority encoding transmission. *IEEETIT: IEEE Transactions on Information Theory*, 42, 1996.

[4] Hot Tech, 1998. `http://www.aquanta.com/hot_tech/k3.html` 17.7.1998.

[5] J.W. Atwood, O. Catrina, J. Fenton, and W.T. Strayer. Reliable multicasting in Xpress transport protocol. Technical report, Department of Computer Science, University of Virginia, 1993.

[6] E.W. Biersack. Performance evaluation of forward error correction in an ATM environment. *IEEE J. on sel. areas in commun.*, SAC-11, 4, 1994.

[7] E.W. Biersack and J. Gafsi. Combined RAID 5 and mirroring for cost optimal fault-tolerant video servers. *EURECOM, Sophia-Antipolis, France*, February 1998.

[8] R.E. Blahut. *Theory and Practice of Error Control Codes*. Addison-Wesley, 1984.

[9] M. Blaum, J. Brady, J. Bruck, and J. Menon. EVENODD: an optimal scheme for tolerating double disk failures in RAID architectures. *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 245–254, April 18–21, 1994.

[10] J.-C. Bolot. Characterizing end-to-end packet delay and loss in the internet. *ACMM Multimedia systems*, 1997.

[11] G. Carle. Adaptation layer and group communication server for reliable multipoint services in ATM networks. Workshop on ATM Traffick Management, WATM'95, WG.6.2 Broadband Communication, Paris, 1995.

[12] C.L. Chen and R.A. Rutledge. Error correcting codes for satellite communication channels. *IBM Journal of Research and Development*, 20(2):168–175, March 1976.

[13] P.M. Chen, E.L. Lee, G.A. Gibson, R.H. Katz, and D.A. Patterson. RAID : High-performance, reliable secondary storage. *ACM Computing Surveys*, 26(2):145–185, June 1994.

[14] B.A. Cipra. The ubiquitous reed-solomon codes. *SINEWS: SIAM News*, 26, 1993.

[15] S. Dresler and M. Hoffmann. Adaptive error correction to support heterogenous multicast groups. *Institute of Telematics, University of Karlsruhe, Zirkel 2, 76128 Karlsruhe, Germany*.

[16] H. Esaki and T. Fukuda. Reliable IP multicast communication over ATM networks using forward error correction policy. *IEIEC, Transactions on Communications*, E78-B,12:1622–37, December 1995.

[17] Glossary of telecommunications terms, 1998. `http://www.its.bldrdoc.gov/fs-1037/` 17.7.1998.

[18] D.C. Feldmeier. An overview of the TP++ transport protocol project. *Computer Communications Recearch, Bellcore*.

[19] G.A. Gibson. *Redundant Disk Arrays: Reliable, Parallel Secondary Storage*. MIT Press, 1992.

[20] A. Gotti. The Da CaPo communication system. Technical report, Computer Engineering and Networks Laboratory, Swiss Federal Institute, Zurich, June 1998.

[21] R.P. Grimaldi. *Discrete and Combinatorial Mathematics*. Addison-Wesley, 1994.

[22] L. Hellerstein, G.A. Gibson, R.M. Karp, R.H. Katz, and D.A. Patterson. Coding techniques for handling failures in large disk arrays. *Algorithmica*, 12(2/3):182–208, August/September 1994.

[23] M. Holland and G.A. Gibson. Parity declustering for continuous operation in redundant disk arrays. *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, 27,9:23–35, September 1992.

[24] M. Holland, G.A. Gibson, and D.P. Siewiorek. Fast, on-line failure recovery in redundant disk arrays. *Proceedings of the 23rd Annual International Symposium on Fault-Tolerant Computing (FTCS '93)*, pages 422–431, June 1993.

[25] A practical quide to the ibm 7135 raid array, 1998. http://www.rs6000.ibm.com/resource/aix_resource/Pubs/redbooks/htmlbooks/sg242565.00/rptgch1.html 16.7.1998.

[26] R. Jain. Myths about congestion management in high speed networks. Technical Report 9809088, September 1, 1998.

[27] K. Kanai, R. Grueter, and K. Tsunoda et.al. Forward error correction control on AAL 5:FEC-SSCS. *IEEE International Conference on Communications*, pages 384–91, 1996.

[28] Basic terminology, 1998. http://kwanwoo.postech.ac.kr/~choety/raid/cipark/node3.html 16.7.1998.

[29] H. Linder, I. Miloucheva, and H.D. Clausen. A multicast transport protocol with forward error correction for satellite environments. Madrid, Spain, 1996.

[30] M.E.S. Loomis. *Data Communications*. Prentice Hall, 1983.

[31] A.J. McAuley. Reliable broadband communication using a burst erasure correcting code,. *Proc. ACM SIGCOMM '90; (Special Issue Computer Communication Review)*, pages 297–306, September 1990. Published as Proc. ACM SIGCOMM '90; (Special Issue Computer Communication Review).

[32] A.M. Michelson and A.H. Levesque. *Error Control Techniques For Digital Communication*. Prentice Hall, 1985.

[33] K. Mooi and M. Kitsureqawa. Hot mirroring : A method of hiding parity update penalty and degration during rebuilds for RAID 5. *Sigmod Record*, 25:183–94, 1996.

[34] H. Ohta and T. Kitami. A cell loss recovery method using FEC in ATM networks. *IEEE Journal on selected areas in communications*, 9:1471–83, December 1991.

[35] D.A. Patterson and J.L. Hennessy. *Computer Organization: The Hardware/Software Interface*. Morgan Kaufmann Publishers, 2929 Campus Drive, Suite 260, San Mateo, CA 94403, USA, second edition, 1997.

[36] J.S. Plank. Tutorial on reed-solomon coding for fault-tolerance in RAID-like systems. Technical Report UT-CS-96-332, Department of Computer Science, University of Tennessee, July 1996.

[37] T.R.N. Rao and E. Fujiwara. *Error-Control Coding for Computer Systems*. 1989.

[38] Real-Time encyclopaedia, data striping, 1998. http://www.realtime-info.be/encyc/techno/terms/75/22.htm 17.7.1998.

[39] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM, Computer Communication Review*, 27, 2:24–36, 1997.

[40] D. Rodgers. Commquest multi-function cards offer new features. Technical report, SHF SATCOM Engineering for Advanced Communication Systems, Inc., in Arlington, VA., 1998.

[41] C. Britton Rorabaugh. *Error Coding Cookbook, Practical C/C++ Routines and recipes for Error Detection and Correction*. McGraw Hill, 1996.

[42] R. Schatzmayr and R. Popescu-Zeletin. Providing support for data transfer in a new networking environment. *Lecture Notes in Computer Science*, 882:241–255, 1994.

[43] K. Shamloo. Resource considerations for scalable multicast connections over ATM, Cand. Scient. Thesis, University of Oslo, 1997.

[44] V. Srinivasan, A. Ghanwani, and E. Gelenbe. Block loss reduction in ATM networks. *Lecture Notes in Computer Science*, 919:73–84, 1995.

[45] W. Stallings. *Data and Computer Communications*. Macmillan Pub. Co., NY, NY, 1985.

[46] T. Stock and X. Garcia. On the potentials of forward error correction mechanisms applied to real-time services carried over B-ISDN. *Lecture Notes in Computer Science*, 1044:107–14, 1996.

[47] D. Stodolsky, G. Gibson, and M. Holland. Parity logging overcoming the small write problem in redundant disk arrays. *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 64–75, May 1993.

[48] T.W. Strayer, B.J. Dempsey, and A.C. Weaver. *XTP – The Xpress Transfer Protocol*. Addison-Wesley Publishing Company, 1992.

[49] A.A. Tanenbaum. *Computer Networks*. Prentice Hall, 1996.

[50] A.C. Weaver. The Xpress transfer protocol version 4. Technical report, Department of Computer Science, University of Virginia, 1995.

Appendix A

Da CaPo Modules

The Da CaPo modules this appendix introduces are created by Roland Siposs and are available at UniK. The sender FEC module (Appendix A.1) uses XOR calculations to create two additional TDUs for each consecutive original TDU. This means that one FEC block includes four TDUs; two original and two redundant. The redundancy ratio is therefore high. The receiver FEC module (Appendix A.2) inspects if TDUs are lost and recreates the lost TDUs with XOR calculations. Because two additional TDUs are included in the data transmission, we are able to recover from two TDU losses in one FEC block.

The elimination of TDUs from the data transmission is accomplished with the m_killer module (Appendix A.3). Elimination is accomplished before the TDUs are sent from the sender to the receiver. The user decides which TDUs are to be eliminated. The sequence numbers for these TDUs are defined in the file m_killer.files. In this way, it is possible to control the error scenarios to be simulated in data transmission. As explained earlier, congestion on a channel can be simulated by eliminating consecutive TDUs from the data transmission. One TDU is eliminated by defining three consecutive TDU sequences in the m_killer.files.

A.1 Sender Side FEC

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "rm_types.h"
#include "ms_fec_matrix.h"
#include "m_fec_types.h"

 typedef struct fec_prio1_s_kon{
   int offset;
   int state;
```

```
   buf adata,aheader;
   buf bdata,bheader;
   buf cdata,cheader;
   buf ddata,dheader;
   unsigned char seqnr;
   unsigned char dcba;
 }Fec_prio1_s_desc;

char *fec_prio1_s_init(protocol_attr *pattr, int modnum)
/*******************************************************/
{
   Fec_prio1_s_desc *inst;

   inst = (Fec_prio1_s_desc *) malloc(sizeof(Fec_prio1_s_desc));
   inst->offset = pattr->m[modnum].headerpos[0].pos;
   inst->state  = M_DONE;
   inst->adata = NULL;
   inst->aheader = NULL;
   inst->bdata = NULL;
   inst->bheader = NULL;
   inst->cdata = NULL;
   inst->cheader = NULL;
   inst->ddata = NULL;
   inst->dheader = NULL;
   inst->seqnr = 0;
   inst->dcba = 0;
   return (char *) inst;
}

int fec_prio1_s_exit(char *inst)
/******************************/
{
   free(inst);
   return 0;
}

void fec_prio1_s_request(char *inst,packet pd, int *status)
/*********************************************************/
{
   Fec_prio1_s_desc *ins;
   int datalen, headerlen, i;
   unsigned char *a, *b, *c, *d;
   buf tmp;

   #ifdef deb_fec
      printf("--> fec_prio1_s_request\n");
   #endif
   ins = (Fec_prio1_s_desc *) inst;
   if (ins->state == M_DONE){ /* state ok */
      pd.header->data[ins->offset] = ins->seqnr++;
      pd.header->data[ins->offset+1] = 0;
      if ((ins->seqnr % PRIO_1_SETSIZE) == 1){
         ins->adata = pd.data;
```

```
         ins->aheader = pd.header;
         ins->dcba = E1;
         ins->state = M_DONE;
      }
      else if ((ins->seqnr % PRIO_1_SETSIZE) == 2){
         ins->bdata = pd.data;
         ins->bheader = pd.header;
         datalen = ins->adata->len;
         headerlen = ins->aheader->len;
         /* Check, if packetsize is equal. Headerlength must be equal */
         if (datalen > ins->bdata->len){
            #ifdef deb_fec
               printf("***--> b shorter than a!\n");
            #endif
            tmp = allocate(datalen);
            memcpy(tmp->data,ins->bdata->data,ins->bdata->len);
            memset(tmp->data + ins->bdata->len,0,datalen - ins->bdata-
>len);
            /* Number of dummy bytes */
            ins->bheader->data[ins->offset+1] = datalen - ins->bdata-
>len;
            deallocate(ins->bdata);
            ins->bdata = tmp;
         }
         else if (datalen < ins->bdata->len){
            #ifdef deb_fec
               printf("***--> a shorter than b!\n");
            #endif
            datalen = ins->bdata->len;
            tmp = allocate(datalen);
            memcpy(tmp->data,ins->adata->data,ins->adata->len);
            memset(tmp->data + ins->adata->len,0,datalen - ins->adata-
>len);
            /* Number of dummy bytes */
            ins->aheader->data[ins->offset+1] = datalen - ins->adata-
>len;
            deallocate(ins->adata);
            ins->adata = tmp;
         }
         /* Create new packets c, d */
         ins->cdata = allocate(datalen);
         ins->cheader = allocate(headerlen);
         ins->ddata = allocate(datalen);
         ins->dheader = allocate(headerlen);
         a = ins->adata->data;
         b = ins->bdata->data;
         c = ins->cdata->data;
         d = ins->ddata->data;
         for (i=0; i<datalen; i++){
            *c++ = *a ^ *b;
            *d++ = *a ^ ms_fec_mult_2_8_2[(*b)];
            a++; b++;
         }
```

```
         a = ins->aheader->data;
         b = ins->bheader->data;
         c = ins->cheader->data;
         d = ins->dheader->data;
         for (i=0; i<headerlen; i++){
            *c++ = *a ^ *b;
            *d++ = *a ^ ms_fec_mult_2_8_2[(*b)];
            a++; b++;
         }
         /* Overwrite one byte in c- and dheader to add seqnr. It's is
             not bad, because this byte is not important for a
             reconstructed packet. */
         ins->cheader->data[ins->offset] = ins->seqnr++;
         ins->dheader->data[ins->offset] = ins->seqnr++;
         ins->seqnr = ins->seqnr % PRIO_1_MAXSEQ;
         ins->dcba |= (E2 + E3 + E4);
         ins->state = M_READY;
      }
      else{ /* should never happen */
         deallocate(pd.data);
         deallocate(pd.header);
      }
   }
   else{ /* wrong state => discard   */
      deallocate(pd.data);
      deallocate(pd.header);
   }
}

void fec_prio1_s_indication(char *inst,packet *pd, int *status)
/*************************************************************/
{
   Fec_prio1_s_desc *ins;

   ins = (Fec_prio1_s_desc *) inst;
   if (ins->state == M_READY){ /* state ok */
      if (ins->dcba & E1){ /* send a */
         #ifdef deb_fec
            printf("--> fec_prio1_s_indication:Send a\n");
         #endif
         pd->data = ins->adata;
         pd->header = ins->aheader;
         ins->dcba ^= E1; /* delete bit for a */
         ins->state = M_READY;
         *status = DATA_OK;
         #ifdef deb_fec
            printf("--> fec_prio1_s_indication:DATA_OK\n");
         #endif
      }
      else if (ins->dcba & E2){ /* send b */
         #ifdef deb_fec
            printf("--> fec_prio1_s_indication:Send b\n");
         #endif
```

```
            pd->data = ins->bdata;
            pd->header = ins->bheader;
            ins->dcba ^= E2; /* delete bit for b */
            ins->state = M_READY;
            *status = DATA_OK;
            #ifdef deb_fec
               printf("--> fec_prio1_s_indication:DATA_OK\n");
            #endif
         }
         else if (ins->dcba & E3){ /* send c */
            #ifdef deb_fec
               printf("--> fec_prio1_s_indication:Send c\n");
            #endif
            pd->data = ins->cdata;
            pd->header = ins->cheader;
            ins->dcba ^= E3; /* delete bit for c */
            ins->state = M_READY;
            *status = DATA_OK;
            #ifdef deb_fec
               printf("--> fec_prio1_s_indication:DATA_OK\n");
            #endif
         }
         else if (ins->dcba & E4){ /* send d */
            #ifdef deb_fec
               printf("--> fec_prio1_s_indication:Send d\n");
            #endif
            pd->data = ins->ddata;
            pd->header = ins->dheader;
            ins->dcba = 0; /*reset */
            ins->state = M_DONE;
            *status = DATA_OK;
            #ifdef deb_fec
               printf("--> fec_prio1_s_indication:DATA_OK\n");
            #endif
         }
         else{
            *status = NO_DT;
            #ifdef deb_fec
               printf("--> fec_prio1_s_indication:NO_DT0\n");
            #endif
         }
      } else{
         *status = NO_DT;
            #ifdef deb_fec
               printf("--> fec_prio1_s_indication:NO_DT1\n");
            #endif
      }
   }
```

A.2 Receiver Side FEC

```
#include <stdio.h>
#include <stdlib.h>
#include "rm_types.h"
#include "mr_fec_matrix.h"
#include "m_fec_types.h"

/*#define   deb_fec*/

 typedef struct fec_prio1_r_kon{
   int offset;
   int state;
   int lost_count;
   buf adata,aheader;
   buf bdata,bheader;
   buf cdata,cheader;
   buf ddata,dheader;
   unsigned char seqnr;
   unsigned char setnr;
   unsigned char dcba;
   unsigned char seqcount;
 }Fec_prio1_r_desc;

char *fec_prio1_r_init(protocol_attr *pattr, int modnum)
/******************************************************/
{
   Fec_prio1_r_desc *inst;

   inst = (Fec_prio1_r_desc *) malloc(sizeof(Fec_prio1_r_desc));
   inst->offset = pattr->m[modnum].headerpos[0].pos;
   inst->state  = M_DONE;
   inst->adata = NULL;
   inst->aheader = NULL;
   inst->bdata = NULL;
   inst->bheader = NULL;
   inst->cdata = NULL;
   inst->cheader = NULL;
   inst->ddata = NULL;
   inst->dheader = NULL;
   inst->setnr = 0;
   inst->dcba = 0;
   inst->seqcount = 0;
   return (char *) inst;
}

int fec_prio1_r_exit(char *inst)
/******************************/
{
   free(inst);
   return 0;
}
```

```
void fec_prio1_r_request(char *inst,packet pd, int *status)
/*********************************************************/
{
   Fec_prio1_r_desc *ins;
   int rem;
   int datalen, headerlen, i;
   unsigned char *a, *b, *c, *d;
   buf tmp;

   ins = (Fec_prio1_r_desc *) inst;
   ins->seqnr = pd.header->data[ins->offset];
   #ifdef deb_fec
      printf("--> fec_prio1_r_request:%d\n",ins->seqnr);
   #endif

   if ((ins->state == M_DONE) && (ins->setnr <= (ins->seqnr /
   PRIO_1_SETSIZE)) && !((ins->setnr == 0) && ((ins->seqnr /
   PRIO_1_SETSIZE) == PRIO_1_MAXSET - 1))){

      /* state ok and not a set, which is already through */
      if (ins->seqcount == 0){ /* begin of new set */
         ins->setnr = ins->seqnr / PRIO_1_SETSIZE;
         rem = ins->seqnr % PRIO_1_SETSIZE;
         ins->seqcount++;
         if (rem == 0){
            ins->dcba = E1;
            ins->adata = pd.data;
            ins->aheader = pd.header;
         }
         else if (rem == 1){
            ins->dcba = E2;
            ins->bdata = pd.data;
            ins->bheader = pd.header;
         }
         else if (rem == 2){
            ins->dcba = E3;
            ins->cdata = pd.data;
            ins->cheader = pd.header;
         }
         else{ /* rem == 3 */
            ins->dcba = E4;
            ins->ddata = pd.data;
            ins->dheader = pd.header;
         }
         ins->state = M_DONE;
      }
      else if (ins->seqcount == 1){
         if (ins->setnr == (ins->seqnr / PRIO_1_SETSIZE)){
           /* the same set */
            rem = ins->seqnr % PRIO_1_SETSIZE;
            if (rem == 0){
               ins->dcba |= E1;
```

```
   ins->adata = pd.data;
   ins->aheader = pd.header;
}
else if (rem == 1){
   ins->dcba |= E2;
   ins->bdata = pd.data;
   ins->bheader = pd.header;
}
else if (rem == 2){
   ins->dcba |= E3;
   ins->cdata = pd.data;
   ins->cheader = pd.header;
}
else{ /* rem == 3 */
   ins->dcba |= E4;
   ins->ddata = pd.data;
   ins->dheader = pd.header;
}
if ((ins->dcba & E2) && (ins->dcba & E1)) ;
/* nothing to do */
else if ((ins->dcba & E3) && (ins->dcba & E1)){
  /* reconstruct b */
   #ifdef deb_fec
  printf("***-> Reconsruct b, Case dcba = %d\n", ins->dcba);
   #endif
   datalen = ins->cdata->len;
   headerlen = ins->cheader->len;
   ins->bdata = allocate(datalen);
   ins->bheader = allocate(headerlen);
   a = ins->adata->data;
   b = ins->bdata->data;
   c = ins->cdata->data;
   for (i=0; i<datalen; i++)
      *b++ = *a++ ^ *c++;
   a = ins->aheader->data;
   b = ins->bheader->data;
   c = ins->cheader->data;
   for (i=0; i<headerlen; i++)
      *b++ = *a++ ^ *c++;
   deallocate(ins->cdata);
   deallocate(ins->cheader);
}
else if ((ins->dcba & E3) && (ins->dcba & E2)){
  /* reconstruct a */
   #ifdef deb_fec
  printf("***-> Reconsruct a, Case dcba = %d\n", ins->dcba);
   #endif
   datalen = ins->cdata->len;
   headerlen = ins->cheader->len;
   ins->adata = allocate(datalen);
   ins->aheader = allocate(headerlen);
   a = ins->adata->data;
   b = ins->bdata->data;
```

```
   c = ins->cdata->data;
   for (i=0; i<datalen; i++)
      *a++ = *b++ ^ *c++;
   a = ins->aheader->data;
   b = ins->bheader->data;
   c = ins->cheader->data;
   for (i=0; i<headerlen; i++)
      *a++ = *b++ ^ *c++;
   deallocate(ins->cdata);
   deallocate(ins->cheader);
}
else if ((ins->dcba & E4) && (ins->dcba & E1)){
  /* reconstruct b */
   #ifdef deb_fec
  printf("***-> Reconsruct b, Case dcba = %d\n", ins->dcba);
   #endif
   datalen = ins->ddata->len;
   headerlen = ins->dheader->len;
   ins->bdata = allocate(datalen);
   ins->bheader = allocate(headerlen);
   a = ins->adata->data;
   b = ins->bdata->data;
   d = ins->ddata->data;
   for (i=0; i<datalen; i++)
      *b++ = mr_fec_mult_2_8_142[*a++ ^ *d++];
   a = ins->aheader->data;
   b = ins->bheader->data;
   d = ins->dheader->data;
   for (i=0; i<headerlen; i++)
      *b++ = mr_fec_mult_2_8_142[*a++ ^ *d++];
   deallocate(ins->ddata);
   deallocate(ins->dheader);
}
else if ((ins->dcba & E4) && (ins->dcba & E2)){
  /* reconstruct a */
   #ifdef deb_fec
  printf("***-> Reconsruct a, Case dcba = %d\n", ins->dcba);
   #endif
   datalen = ins->ddata->len;
   headerlen = ins->dheader->len;
   ins->adata = allocate(datalen);
   ins->aheader = allocate(headerlen);
   a = ins->adata->data;
   b = ins->bdata->data;
   d = ins->ddata->data;
   for (i=0; i<datalen; i++)
      *a++ = *d++ ^ mr_fec_mult_2_8_2[*b++];
   a = ins->aheader->data;
   b = ins->bheader->data;
   d = ins->dheader->data;
   for (i=0; i<headerlen; i++)
      *a++ = *d++ ^ mr_fec_mult_2_8_2[*b++];
   deallocate(ins->ddata);
```

```
   deallocate(ins->dheader);
}
else if ((ins->dcba & E4) && (ins->dcba & E3)){
  /* reconstruct a,b */
   #ifdef deb_fec
  printf("***-> Reconsruct a,b, Case dcba = %d\n", ins->dcba);
   #endif
   datalen = ins->ddata->len;
   headerlen = ins->dheader->len;
   ins->adata = allocate(datalen);
   ins->aheader = allocate(headerlen);
   ins->bdata = allocate(datalen);
   ins->bheader = allocate(headerlen);
   a = ins->adata->data;
   b = ins->bdata->data;
   c = ins->cdata->data;
   d = ins->ddata->data;
   for (i=0; i<datalen; i++){
      *b = mr_fec_mult_2_8_244[*c ^ *d++];
      *a++ = *b++ ^ *c++;
   }
   a = ins->aheader->data;
   b = ins->bheader->data;
   c = ins->cheader->data;
   d = ins->dheader->data;
   for (i=0; i<headerlen; i++){
      *b = mr_fec_mult_2_8_244[*c ^ *d++];
      *a++ = *b++ ^ *c++;
   }
   deallocate(ins->cdata);
   deallocate(ins->cheader);
   deallocate(ins->ddata);
   deallocate(ins->dheader);
}
ins->dcba = E2 + E1;
/* check a and b for dummy bits */
if (ins->aheader->data[ins->offset+1] != 0){
   tmp = ins->adata;

   ins->adata = allocate_piece(tmp,0,tmp->len -
           ins->aheader->data[ins->offset+1]);

   deallocate(tmp);
}
if (ins->bheader->data[ins->offset+1] != 0){
   tmp = ins->bdata;
   ins->bdata = allocate_piece(tmp,0,tmp->len -
            ins->bheader->data[ins->offset+1]);
   deallocate(tmp);
}
/* a and b are now available in their original size */
ins->setnr++;
/* all setnumbers < setnr are ignored from now on */
```

```
    ins->setnr = ins->setnr % PRIO_1_MAXSET;
    ins->seqcount = 0;
    ins->state = M_READY;
}
else{ /* two different sets -> have to forget first */
   ins->lost_count++;
   printf("Packet_lost_count: %d\n" , ins->lost_count);
    #ifdef deb_fec
   printf("***-> Lost at least 1 Set
       (More than 2 packets per set are lost) !\n");
    #endif
    /* remove old packet */
    if (ins->dcba & E1){
       deallocate(ins->adata);
       deallocate(ins->aheader);
    }
    else if (ins->dcba & E2){
       deallocate(ins->bdata);
       deallocate(ins->bheader);
    }
    else if (ins->dcba & E3){
       deallocate(ins->cdata);
       deallocate(ins->cheader);
    }
    else if (ins->dcba & E4){
       deallocate(ins->ddata);
       deallocate(ins->dheader);
    }
    ins->dcba = 0;
    /* begin of new set */
    ins->setnr = ins->seqnr / PRIO_1_SETSIZE;
    rem = ins->seqnr % PRIO_1_SETSIZE;
    /* ins->seqcount stays 1 */
    if (rem == 0){
       ins->dcba = E1;
       ins->adata = pd.data;
       ins->aheader = pd.header;
    }
    else if (rem == 1){
       ins->dcba = E2;
       ins->bdata = pd.data;
       ins->bheader = pd.header;
    }
    else if (rem == 2){
       ins->dcba = E3;
       ins->cdata = pd.data;
       ins->cheader = pd.header;
    }
    else{ /* rem == 3 */
       ins->dcba = E4;
       ins->ddata = pd.data;
       ins->dheader = pd.header;
    }
```

```
            ins->state = M_DONE;
         }
      }
      else{ /* should never happen */
         deallocate(pd.data);
         deallocate(pd.header);
      }
   }
   else{
/* wrong state or packet from a set which is already done */
      deallocate(pd.data);
      deallocate(pd.header);
   }
}

void fec_prio1_r_indication(char *inst,packet *pd, int *status)
/*************************************************************/
{
   Fec_prio1_r_desc *ins;

   ins = (Fec_prio1_r_desc *) inst;
   if (ins->state == M_READY){ /* state ok */
      if (ins->dcba & E1){ /* send a */
         #ifdef deb_fec
            printf("--> fec_prio1_r_indication:Send a\n");
         #endif
         pd->data = ins->adata;
         pd->header = ins->aheader;
         ins->dcba ^= E1; /* delete bit for a */
         ins->state = M_READY;
         *status = DATA_OK;
         #ifdef deb_fec
         printf("--> fec_prio1_r_indication:DATA_OK\n");
         #endif
      }
      else if (ins->dcba & E2){ /* send b */
         #ifdef deb_fec
                  printf("--> fec_prio1_r_indication:Send b\n");
         #endif
         pd->data = ins->bdata;
         pd->header = ins->bheader;
         ins->dcba = 0; /*reset */
         ins->state = M_DONE;
         *status = DATA_OK;
         #ifdef deb_fec
          printf("--> fec_prio1_r_indication:DATA_OK\n");
         #endif
      }
      else{
         *status = NO_DT;
         #ifdef deb_fec
           printf("--> fec_prio1_r_indication:NO_DT0\n");
         #endif
```

```
      }
   } else{
      *status = NO_DT;
         #ifdef deb_fec
      printf("--> fec_prio1_r_indication:NO_DT1\n");
         #endif
   }
}
```

A.3 M-killer Module

```
#include <stdio.h>
#include <stdlib.h>
#include "rm_types.h"

#define M_DONE  2
#define M_READY 8
#define K_FILE "m_killer.files"

/*******************************************
  This module kills all the packets with the
  sequencenumbers given in the
   file "m_killer.files".
*******************************************/

 typedef struct killer_m_kon{
   int state;
   int kilcount;
   buf data,header;
   unsigned char seqnr;
 }Killer_m_desc;

char *killer_init(protocol_attr *pattr, int modnum)
/*************************************************/
{
   Killer_m_desc *inst;

   inst = (Killer_m_desc *) malloc(sizeof(Killer_m_desc));
   inst->state  = M_DONE;
   inst->data = NULL;
   inst->header = NULL;
   inst->seqnr = 0;
   return (char *) inst;
}

int killer_exit(char *inst)
/**************************/
{
   free(inst);
   return 0;
}

int found(unsigned char seqnr)
/*****************************/
{
   FILE *killfile;
   int nr;

   killfile = fopen(K_FILE,"r");
   while (fscanf(killfile,"%d",&nr) != EOF){
      if (nr == seqnr){
```

```
         fclose(killfile);
         return 1;
      }
   }
   fclose(killfile);
   return 0;
}

void killer_request(char *inst,packet pd, int *status)
/*******************************************************/
{
   Killer_m_desc *ins;

   ins = (Killer_m_desc *) inst;
   if (ins->state == M_DONE){ /* state ok */
      if (!found(ins->seqnr)){
         ins->data = pd.data;
         ins->header = pd.header;
         ins->state = M_READY;
      }
      else{
         deallocate(pd.data);
         deallocate(pd.header);
   printf("***--> killer_request: Killed packet with seqnr %d\n",ins-
>seqnr);
         ins ->kilcount++;
         printf("Killed packets: %d\n", ins->kilcount);
         ins->state = M_DONE;
      }
      ins->seqnr++;
      ins->seqnr = ins->seqnr % 252;
   }
   else{ /* wrong state => discard */
      deallocate(pd.data);
      deallocate(pd.header);
   }
}

void killer_indication(char *inst,packet *pd, int *status)
/*********************************************************/
{
   Killer_m_desc *ins;

   ins = (Killer_m_desc *) inst;
   if (ins->state == M_READY){ /* state ok */
      pd->data = ins->data;
      pd->header = ins->header;
      ins->state = M_DONE;
      *status = DATA_OK;
   } else{
      *status = NO_DT;
   }
}
```